ndg@mit.edu

Concept learning as inductive programming

Noah D. Goodman
Computational Cognitive Science Group,
MIT

Machine learning meets human learning,
NIPS, 12 December 2008

Motivation

- Three views on concepts:
  - Categorization and inference.
  - Formal semantics.
  - Development.
- Formal apparatus used to capture these aspects of concepts looks very different!

Motivation

- Three views on concepts:
  - Categorization and inference. “statistics”
  - Formal semantics.
  - Development.
- Formal apparatus used to capture these aspects of concepts looks very different!

Motivation

- Three views on concepts:
  - Categorization and inference. “statistics”
  - Formal semantics. “composition”
  - Development.
- Formal apparatus used to capture these aspects of concepts looks very different!

Motivation

- Three views on concepts:
  - Categorization and inference. "statistics"
  - Formal semantics. "composition"
  - Development. "abstract theories"
- Formal apparatus used to capture these aspects of concepts looks very different!

Our approach

- Concepts as functions in a stochastic lambda calculus (a.k.a. probabilistic programs).
  - The meaning of a stochastic function is probabilistic.
  - Stochastic functions compose (subject to type constraints).
  - Abstraction via higher-order functions; theories ("inter-related systems of concepts") are programs (sets of functions).
- Concept *learning* is then inductive (probabilistic) programming.

Function learning

- Can view categorization as inductive learning of a classifier function. (Goodman, et al, 2008)

```
(define (Start) (list 'lambda '(x) (Disj)))
(define (Disj) (if (flip 0.3)
                   (list 'or (Disj) (Conj))
                   (Conj)))
(define (Conj) (if (flip 0.3)
                   (list 'and (Conj) (Feat))
                   (Feat)))
(define (Feat) (list 'feat (sample-integer nfeat) 'x))
```

```
(lex-query
 '((Label-expression (Start))
   (Label-procedure
     (noisify (eval Label-expression) b)))
 'Label-expression
 '(equal? (map Label-procedure obs-objects) obs-labels))
```

Function learning

- Can view categorization as inductive learning of a classifier function. (Goodman, et al, 2008)

```
(define (Start) (list 'lambda '(x) (Disj)))
(define (Disj) (if (flip 0.3)
                   (list 'or (Disj) (Conj))
                   (Conj)))
(define (Conj) (if (flip 0.3)
                   (list 'and (Conj) (Feat))
                   (Feat)))
(define (Feat) (l
```

For example, could generate:

```
(lambda (x)
   (and (feat 1 x)
        ((not (feat 2 x)))))
```

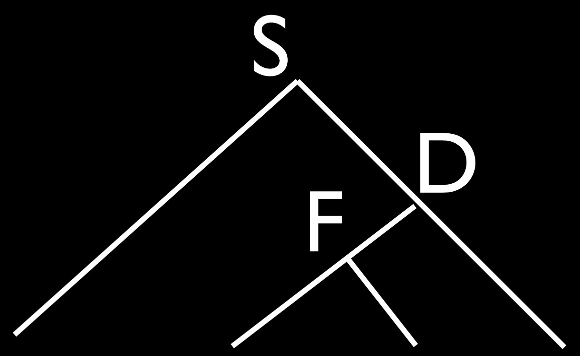
```
(lex-query
 '((Label-expression (Start))
   (Label-procedure
     (noisify (eval Label-expression) b)))
 'Label-expression
 '(equal? (map Label-procedure obs-objects) obs-labels))
```

Category learning

- Inference: MCMC by subtree-regeneration proposals.
  - Select a subtree of the parse tree at random, re-generate from the grammar.
  - Accept/reject according to MH rule.
  - (Cf. Church inference algorithm.)

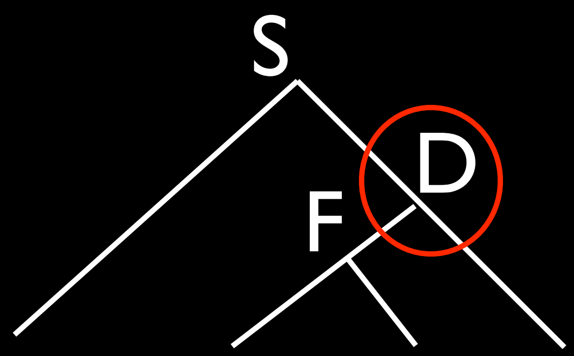
Category learning

- Inference: MCMC by subtree-regeneration proposals.
  - Select a subtree of the parse tree at random, re-generate from the grammar.
  - Accept/reject according to MH rule.
  - (Cf. Church inference algorithm.)



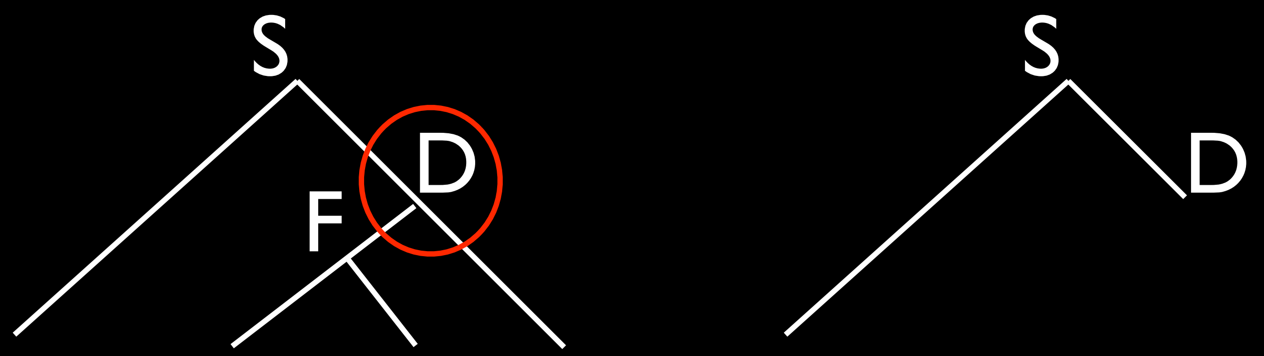
Category learning

- Inference: MCMC by subtree-regeneration proposals.
  - Select a subtree of the parse tree at random, re-generate from the grammar.
  - Accept/reject according to MH rule.
  - (Cf. Church inference algorithm.)



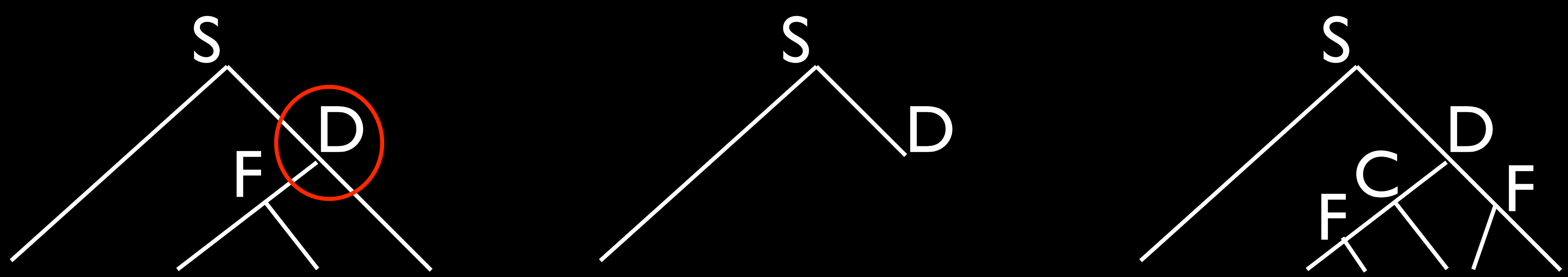
Category learning

- Inference: MCMC by subtree-regeneration proposals.
  - Select a subtree of the parse tree at random, re-generate from the grammar.
  - Accept/reject according to MH rule.
  - (Cf. Church inference algorithm.)



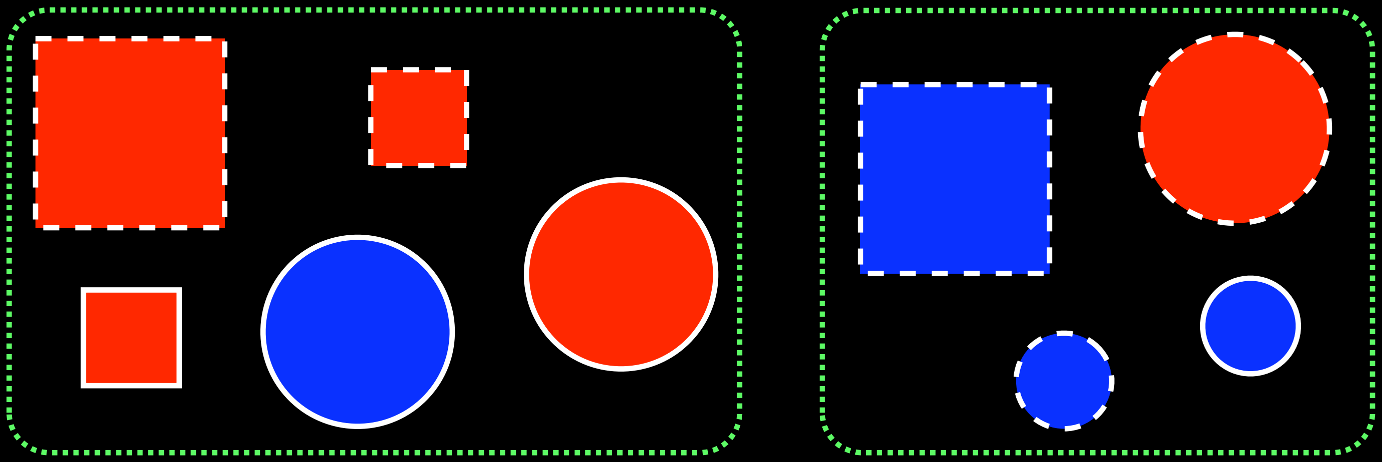
Category learning

- Inference: MCMC by subtree-regeneration proposals.
  - Select a subtree of the parse tree at random, re-generate from the grammar.
  - Accept/reject according to MH rule.
  - (Cf. Church inference algorithm.)

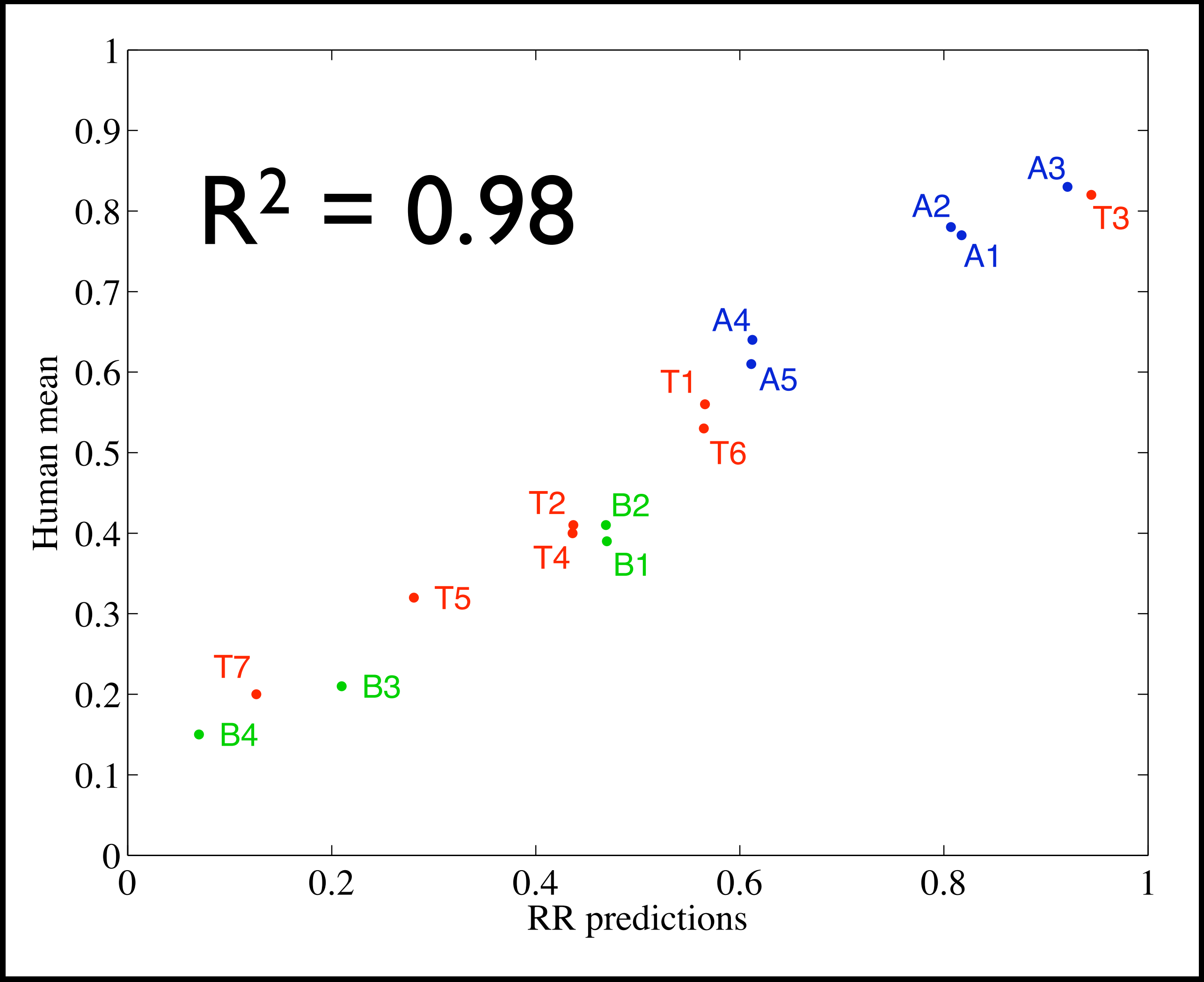


Results

Medin & Schaffer, 1978:



Object	Feature Values
A1	0001
A2	0101
A3	0100
A4	0010
A5	1000
B1	0011
B2	1001
B3	1110
B4	1111
T1	0110
T2	0111
T3	0000
T4	1101
T5	1010
T6	1100
T7	1011



Predicts human performance in several other categorization experiments:

- Medin, Altom, Edelson, & Freko (1982).
- Medin & Schwanenflugel (1981),
- Shepard, Hovland, Jenkins (1961),
- Nosofsky, Clark, & Shin (1989),
- Kruschke (1993),
- Less constrained categories....

Theory learning

- Move from learning a function to a set of inter-related functions -- a program.
  - A set of simple (stochastic) classifier functions that depend on each other gives a Bayes net.
  - Inductive programming gives Bayesian structure learning.
    - Learn dependencies and CPDs together.
    - Extends naturally to learn types and grounding. (Cf. causal schemata and grounded causal models.)

Bayes net learning

```
(define (S) (list 'mem (list 'lambda '(trial) (D))))
(define (D) (if (flip 0.3) (list 'or (D) (C)) (C)))
(define (C) (if (flip 0.3) (list 'and (C) (F)) (F)))
(define (F) (if (flip)
                (list (Var) 'trial)
                (list 'not (list (Var) 'trial))))
(define (Var) (list 'get-proc (uniform-draw '(A B C))))
```

```
(lex-query
 '((get-expr (mem (lambda (var) (S))))
   (get-proc (lambda (var) (eval (get-expr var))))
   (A (get-proc 'A))
   (B (get-proc 'B))
   (C (get-proc 'C)))
 '(map get-expr '(A B C))
 '(and (A 'trial1) (B 'trial1) (not (C 'trial1))
       (A 'trial2) (not (B 'trial2)) (not (C 'trial2))))
```

Bayes net learning

```
(define (S) (list 'mem (list 'lambda '(trial) (D))))
(define (D) (if (flip 0.3) (list 'or (D) (C)) (C)))
(define (C) (if (flip 0.3) (list 'and (C) (F)) (F)))
(define (F) (if (flip)
              (list (Var) 'trial)
              (list 'not (list (Var) 'trial))))
(define (Var) (list 'get-proc (uniform-draw '(A B C))))
```

For example, procedure A could be:

```
(mem (lambda (trial)
       (and ((get-proc C) trial)
            ((get-proc B) trial))))
```

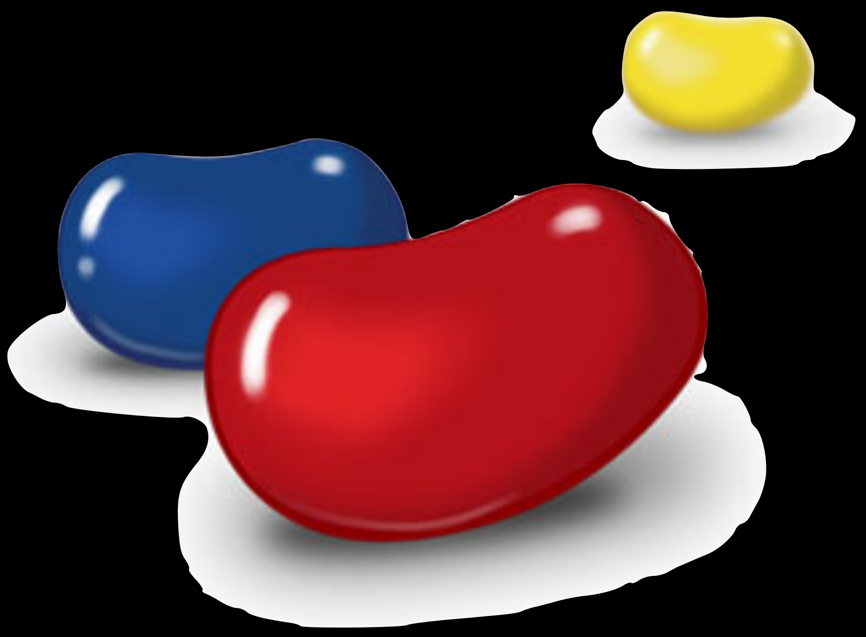
```
(lex-query
 '((get-expr (mem (lambda (var) (S))))
   (get-proc (lambda (var) (eval (get-expr var))))
   (A (get-proc 'A))
   (B (get-proc 'B))
   (C (get-proc 'C)))
 '(map get-expr '(A B C))
 '(and (A 'trial1) (B 'trial1) (not (C 'trial1))
       (A 'trial2) (not (B 'trial2)) (not (C 'trial2))))
```

Theory learning

- Can imagine learning richer more abstract theories in the same way.
  - The functions become more complex and manipulate more complex values.
- Let’s look at a standard test case from cognitive development:
  acquiring natural number concepts....

Learning number

How many jelly beans?



Can you give me two jelly beans?

(Piantadosi, Goodman, Tenenbaum, in prep.)

Learning number

Approx. age	Level	Meaning	Give N task	Highest Count (Wynn 1992)
< 2	**No-knower**	No meanings	Gives a handful	
2 – 2;6	**One-knower**	"One" means one	Correct for "one", Handful for anything else	4.8
2;6 – 3;3	**Two-knower**	"One" means one, "two" means two	Correct for "one" and "two", Else handful	5.7
3;3 – 3;6	**Three-knower**	"One" means one; "Two" mean two", "three" means three	Correct for "one" , "two" "three"; else handful	5.6
> 3;6	**CP-knower**	All numbers	Use counting to give any number	

(Spelke 2003; Wynn 1990, 1992)

(Piantadosi, Goodman, Tenenbaum, in prep.)

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?

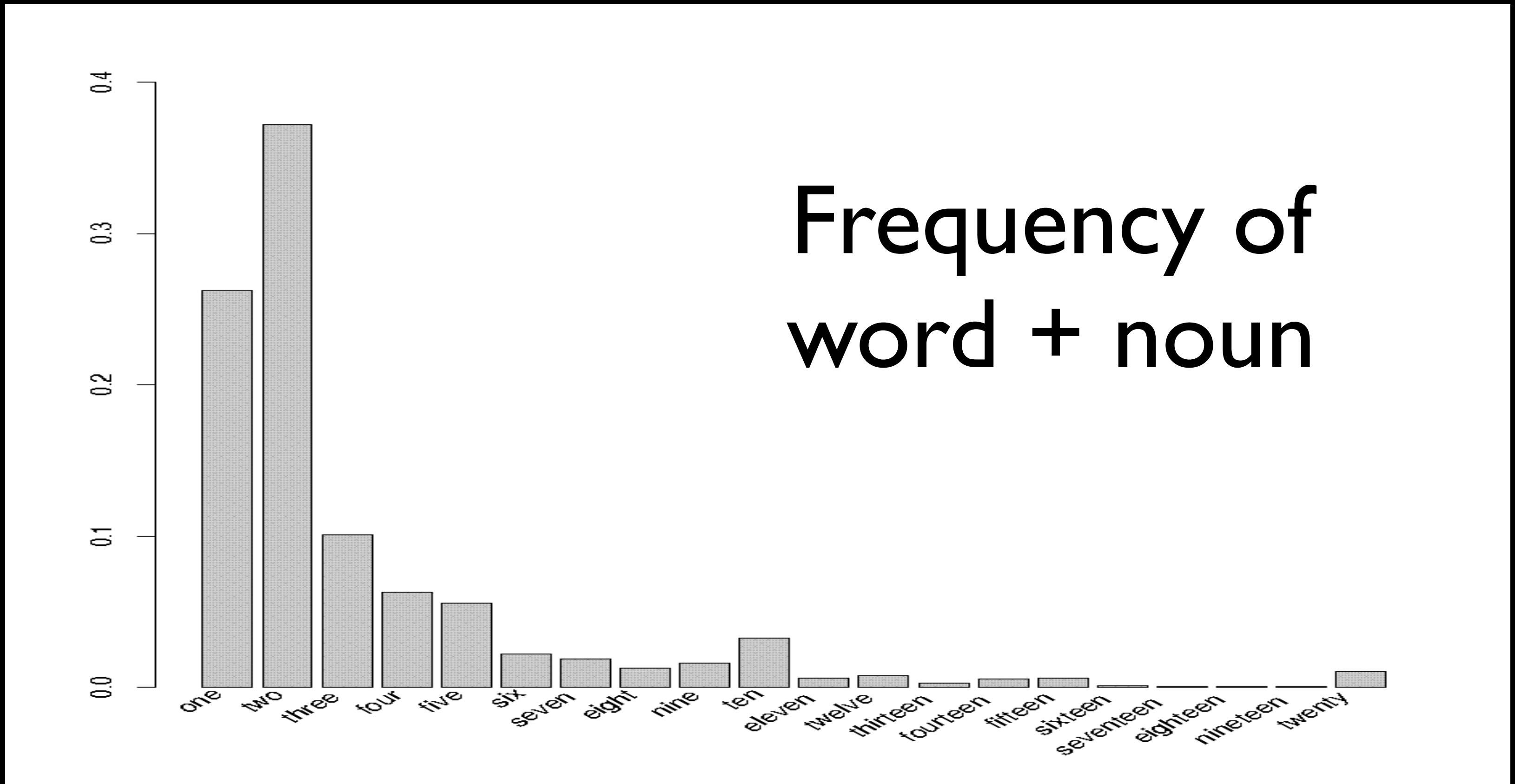
Learning number

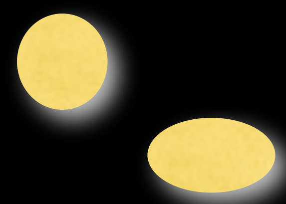
- Our language is (limited) lambda calculus with primitives:
  - `empty?`: is this set of objects empty?
  - `dec`: remove a random object from this set.
  - `prev`: previous word in the count-list (a content-free order on the count words).
  - `C`: get the function for a word.
  - (`next`, `and`, `or`, ...).
- For example "two":

```
(lambda (x) (empty? (dec (dec x))))
```

Learning number

- Learning data: assume situations for each number word occur with the frequency of these words in CHILDES corpus.



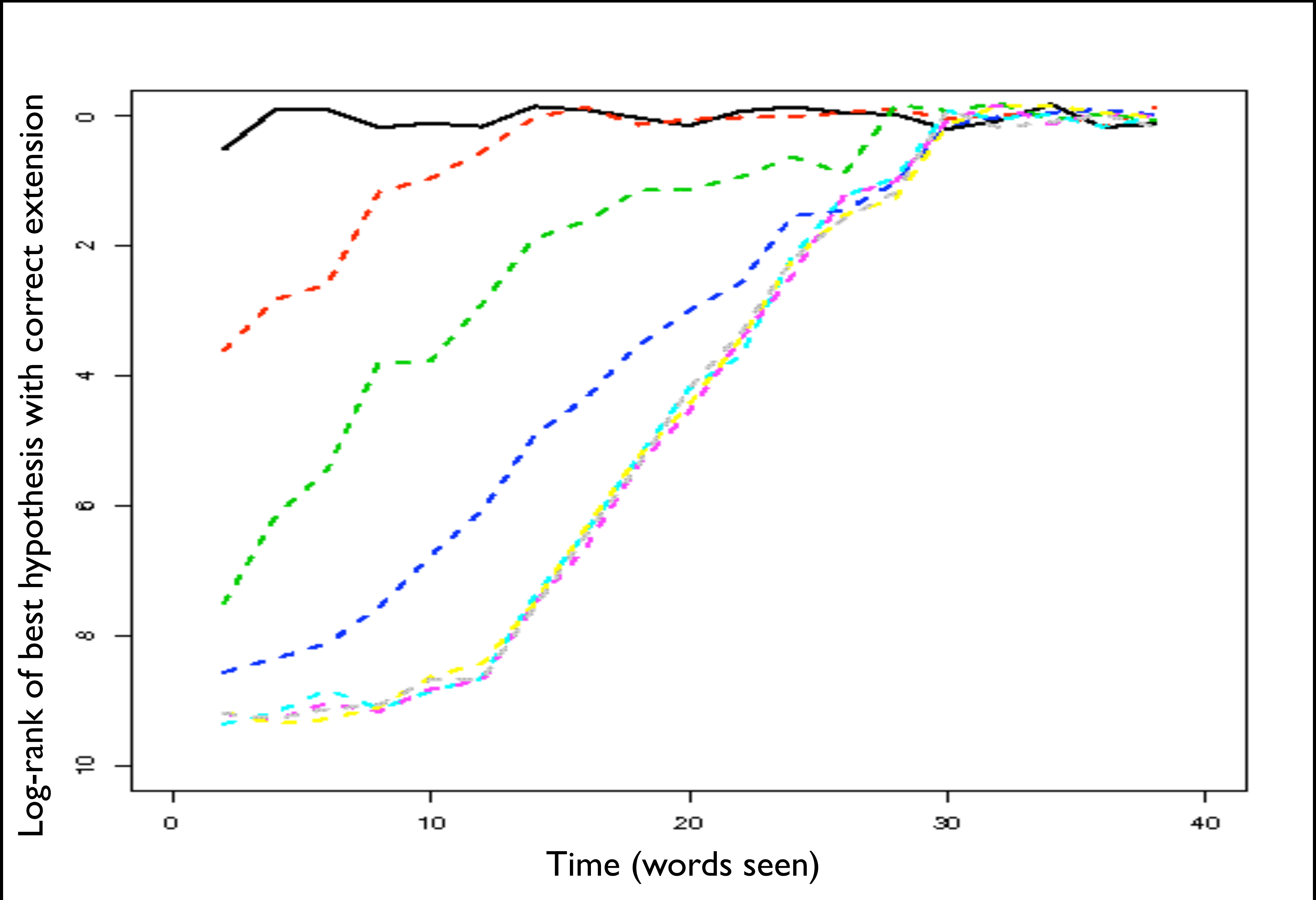


"Look at the two blobs!"

Learning number

- Likelihood: speaker will use a word that is true in the current situation.
  - Uniform choice amongst true words,
  - Some probability of saying a word at random.
    Cf. Frank, Goodman, Tenenbaum (in press).
- Search for best program using MCMC:
  - Same algorithm as used for categorization and causal learning earlier (MH with subtree-regeneration proposals).

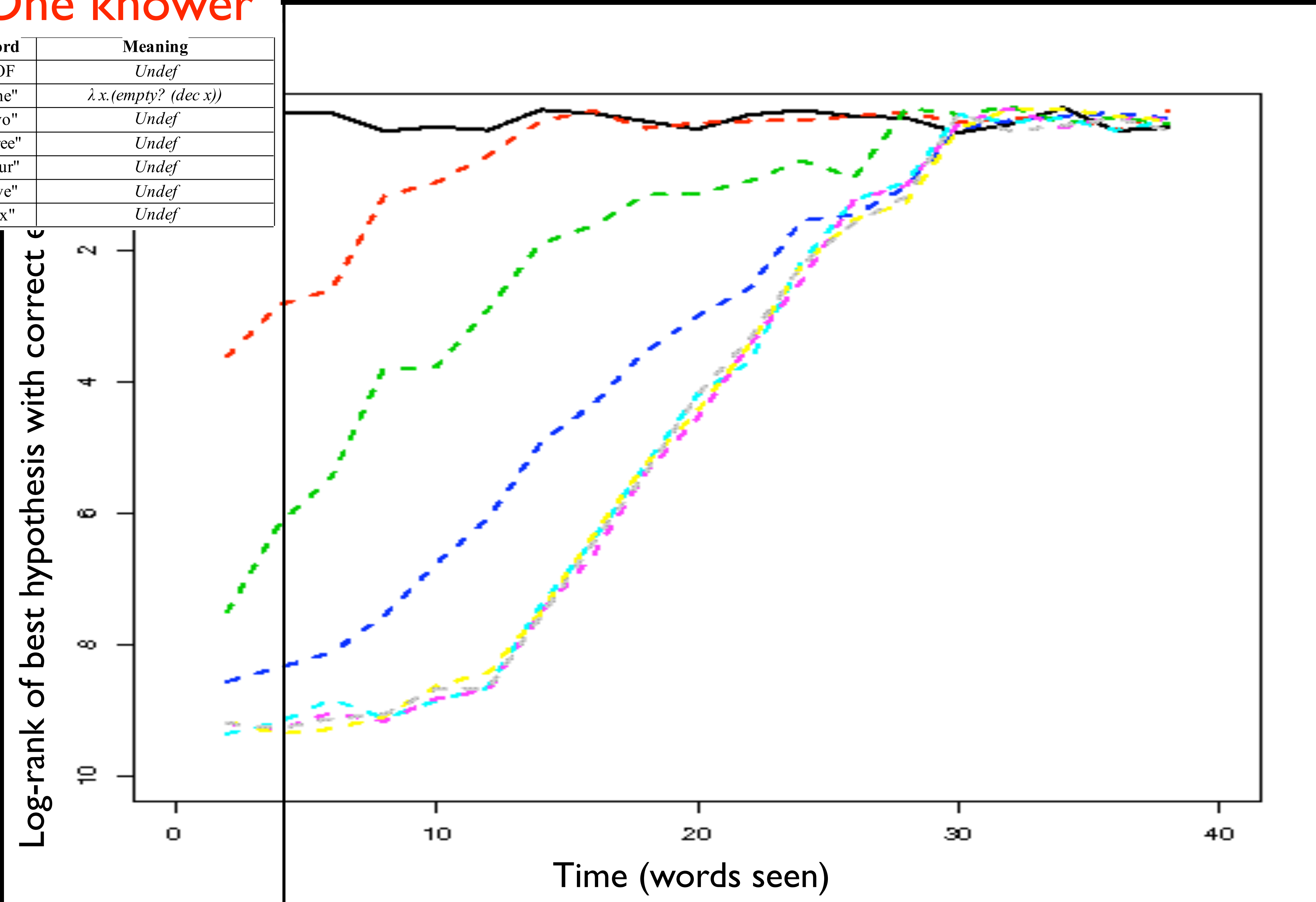
Learning number



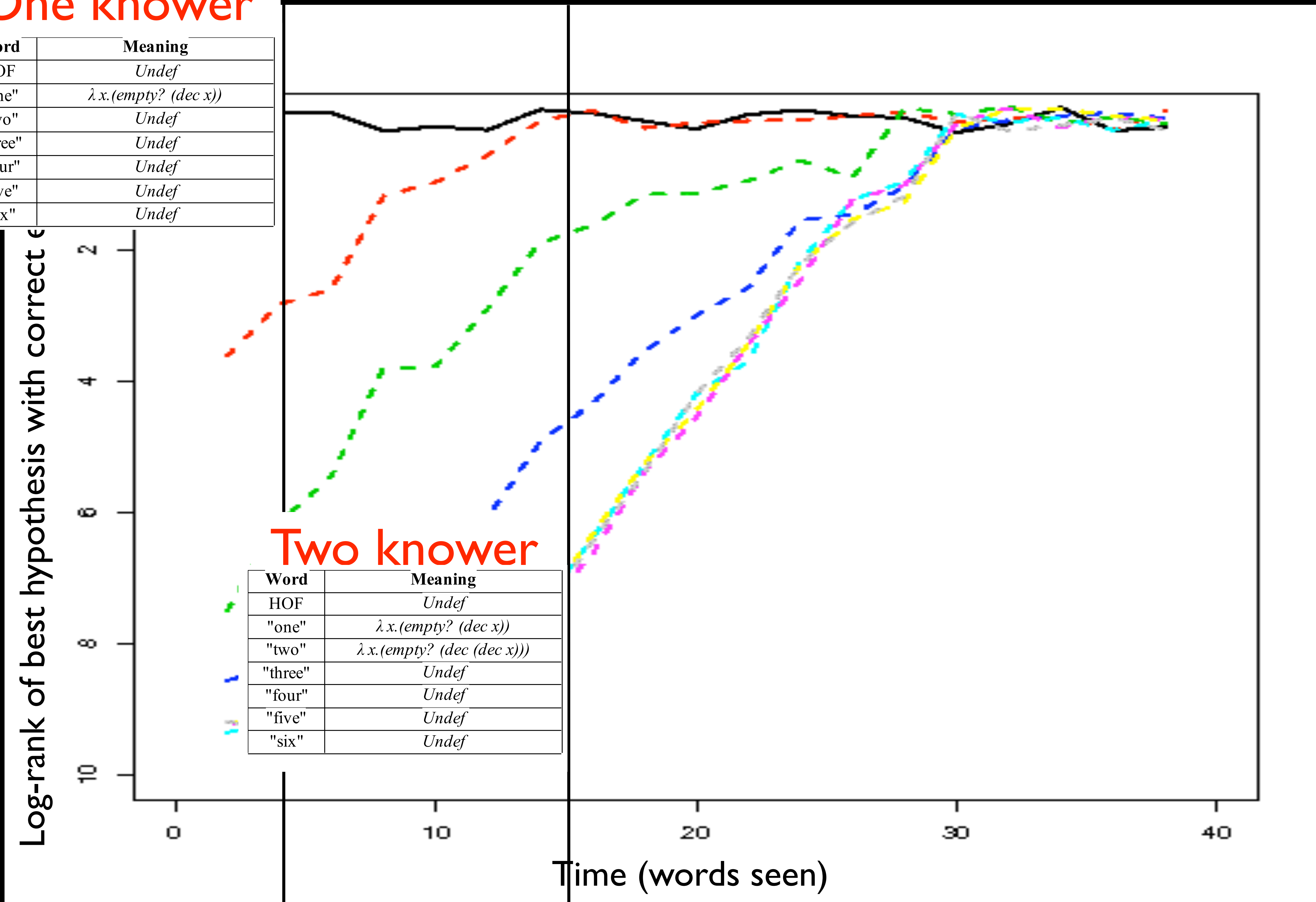
Learning number

One knower

Word	Meaning
HOF	*Undef*
"one"	*λx.(empty? (dec x))*
"two"	*Undef*
"three"	*Undef*
"four"	*Undef*
"five"	*Undef*
"six"	*Undef*



Learning number



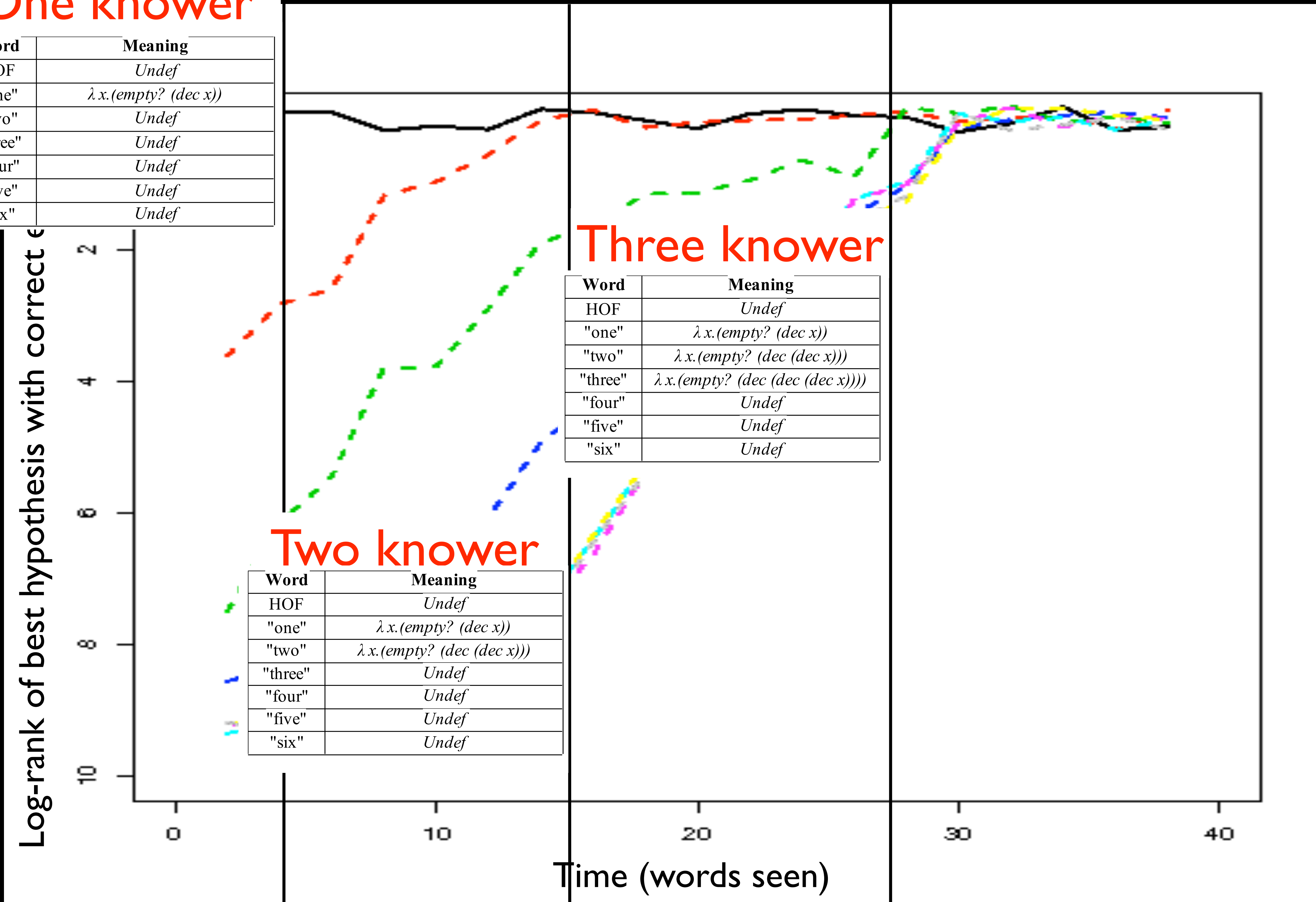
One knower

Word	Meaning
HOF	*Undef*
"one"	*λx.(empty? (dec x))*
"two"	*Undef*
"three"	*Undef*
"four"	*Undef*
"five"	*Undef*
"six"	*Undef*

Two knower

Word	Meaning
HOF	*Undef*
"one"	*λx.(empty? (dec x))*
"two"	*λx.(empty? (dec (dec x)))*
"three"	*Undef*
"four"	*Undef*
"five"	*Undef*
"six"	*Undef*

Learning number



One knower

Word	Meaning
HOF	*Undef*
"one"	$\lambda x.(empty?\ (dec\ x))$
"two"	*Undef*
"three"	*Undef*
"four"	*Undef*
"five"	*Undef*
"six"	*Undef*

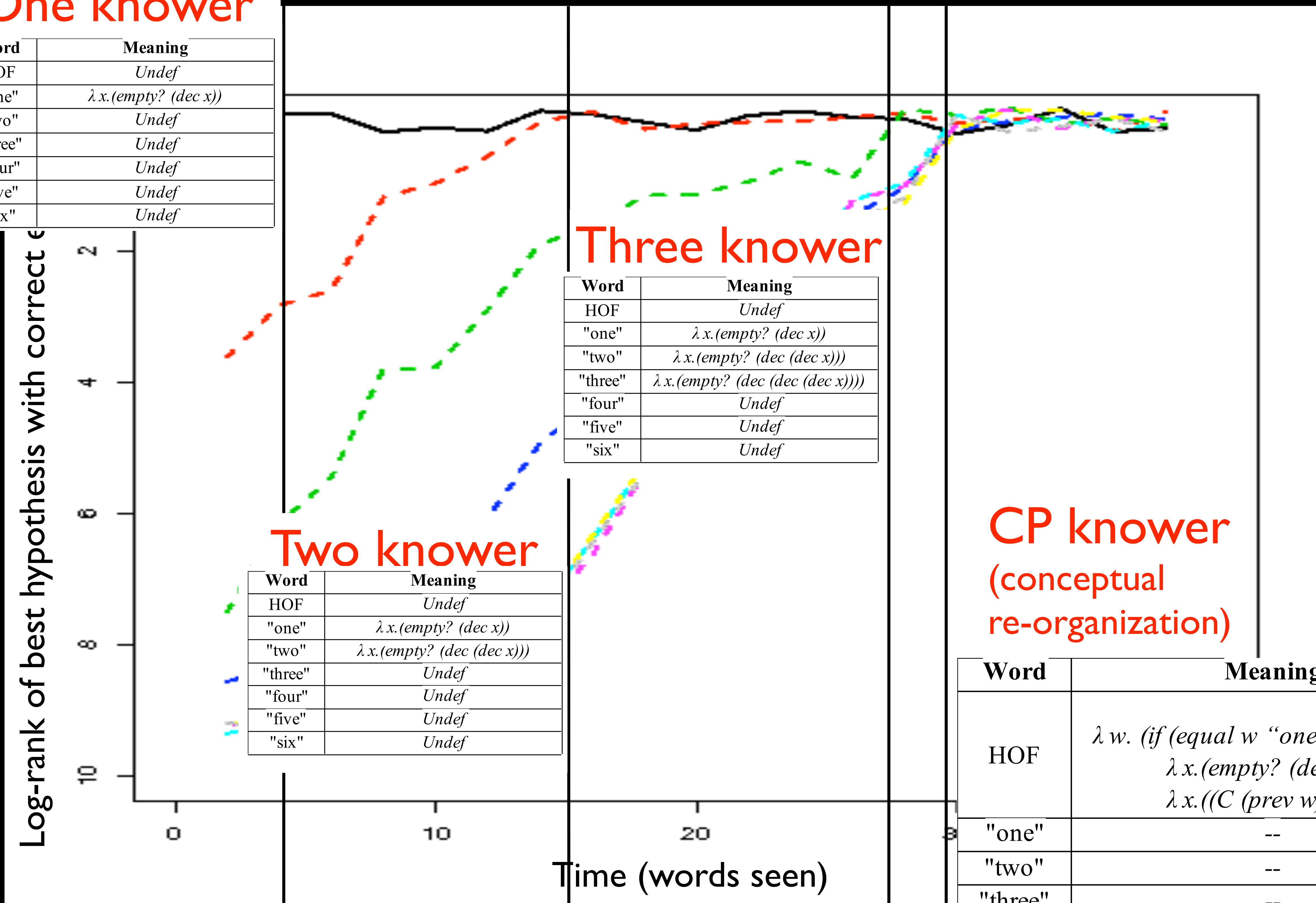
Two knower

Word	Meaning
HOF	*Undef*
"one"	$\lambda x.(empty?\ (dec\ x))$
"two"	$\lambda x.(empty?\ (dec\ (dec\ x)))$
"three"	*Undef*
"four"	*Undef*
"five"	*Undef*
"six"	*Undef*

Three knower

Word	Meaning
HOF	*Undef*
"one"	$\lambda x.(empty?\ (dec\ x))$
"two"	$\lambda x.(empty?\ (dec\ (dec\ x)))$
"three"	$\lambda x.(empty?\ (dec\ (dec\ (dec\ x))))$
"four"	*Undef*
"five"	*Undef*
"six"	*Undef*

Learning number



One knower

Word	Meaning
HOF	*Undef*
"one"	*λx.(empty? (dec x))*
"two"	*Undef*
"three"	*Undef*
"four"	*Undef*
"five"	*Undef*
"six"	*Undef*

Two knower

Word	Meaning
HOF	*Undef*
"one"	*λx.(empty? (dec x))*
"two"	*λx.(empty? (dec (dec x)))*
"three"	*Undef*
"four"	*Undef*
"five"	*Undef*
"six"	*Undef*

Three knower

Word	Meaning
HOF	*Undef*
"one"	*λx.(empty? (dec x))*
"two"	*λx.(empty? (dec (dec x)))*
"three"	*λx.(empty? (dec (dec (dec x))))*
"four"	*Undef*
"five"	*Undef*
"six"	*Undef*

CP knower (conceptual re-organization)

Word	Meaning
HOF	*λw. (if (equal w "one")* *λx.(empty? (dec x))* *λx.((C (prev w)) (dec x)))*
"one"	--
"two"	--
"three"	--
"four"	--

So...

- Recursive number concepts are formed from more primitive operations.
- Inductive programming explains
  - the order of acquisition,
  - the conceptual re-organization giving rise to the CP transition.
- Learning is dependent on linguistic "placeholder structure" (the count list), suggesting new ways to learn programs.

Conclusion

- Viewing concepts as probabilistic programs entails concept learning as inductive programming.
  - A uniform vision for many concept and theory learning tasks.
    - Extends the reach of Bayesian methods.
    - Explains conceptual re-organization, etc.
  - Poses novel machine learning problems and techniques.
    (See "Probabilistic Programming" workshop for more.)