Learning list functions through program induction

Joshua Rule

Learning as Program Induction Workshop
CogSci 2018, 25 July 2018

Computational Cognitive Science Group
brain+cognitive sciences
Center for Brains Minds+ Machines

This talk

- learning as programming
- bootstrapping the LOT with term rewriting
- toward a model of conceptual change

This talk

- learning as programming
- bootstrapping the LOT with term rewriting
- toward a model of conceptual change

Build causal theories from sparse evidence

Build towers, sandcastles, & Lego cars

Navigate complex environments

Use light switches, door knobs, & smartphones

Recognize objects, reason cross-modally

Talk about dinosaurs, trucks, and fairy tales

Tie shoes, make bed, set table

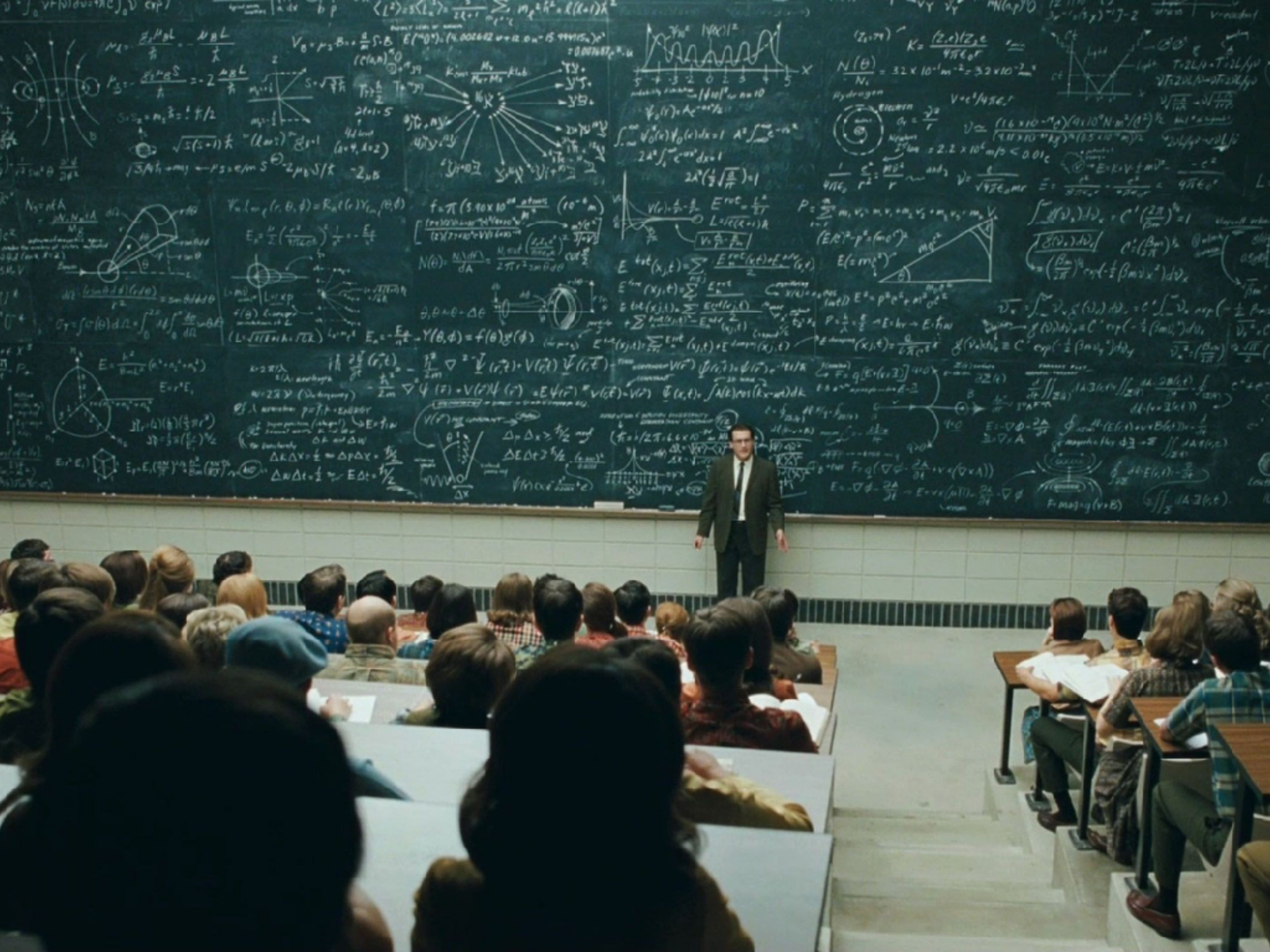
Play with others, share, determine ownership

Introspect on beliefs and desires

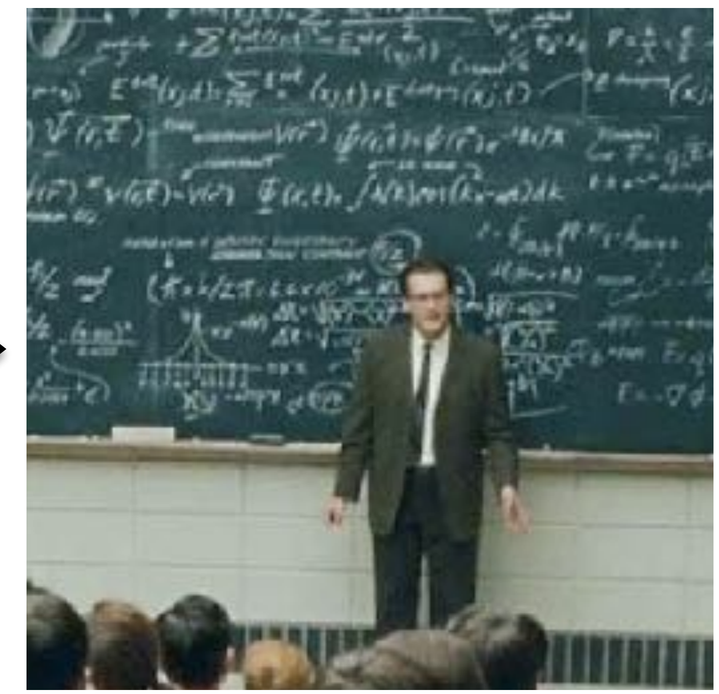
Walk, run, skip, dance, somersault

whisper, shout, sing, joke

Use natural language



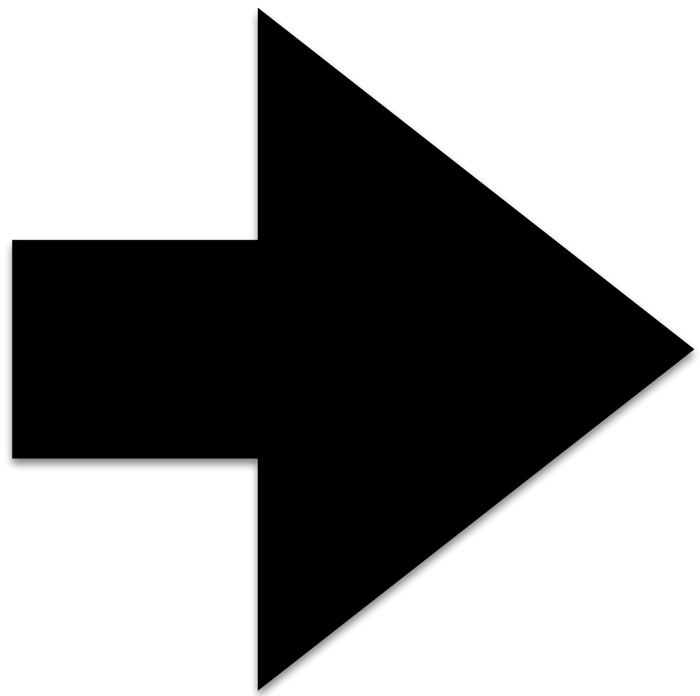
The Puzzle of Learning & Cognitive Development



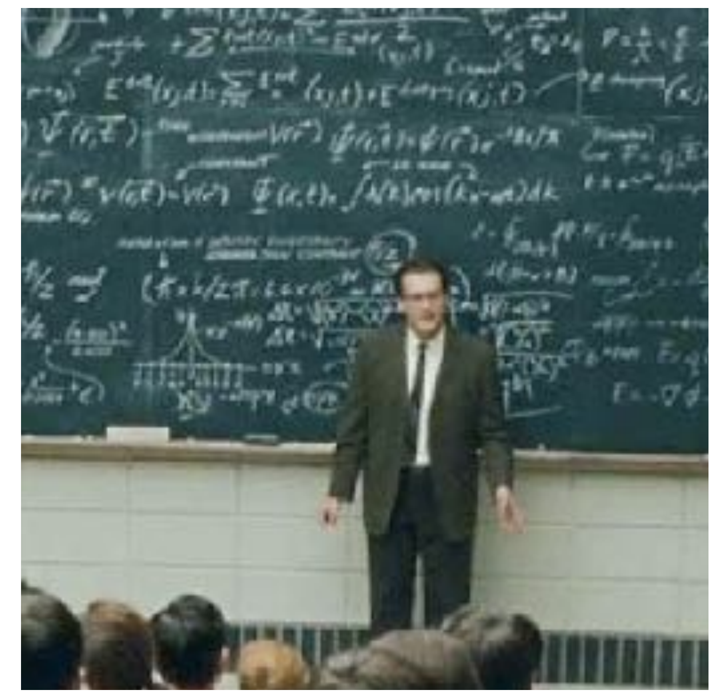
(see Tenenbaum, Kemp, Griffiths, Goodman, 2011; Carey, 2009)

The Puzzle of Learning & Cognitive Development

Initial State



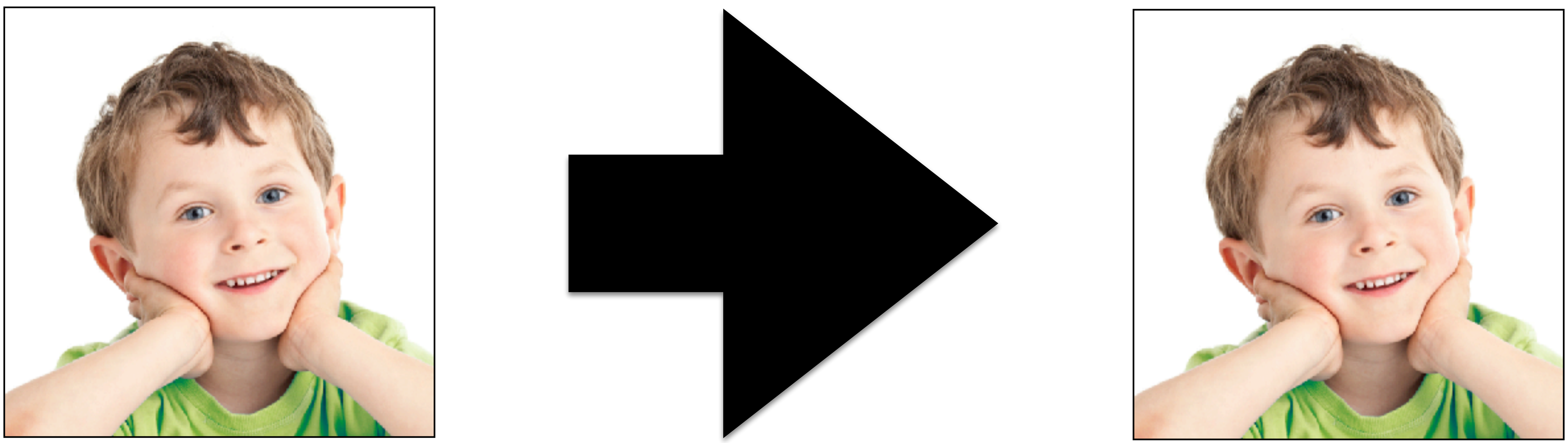
Learning Mechanism



Final State

(see Tenenbaum, Kemp, Griffiths, Goodman, 2011; Carey, 2009)

The Puzzle of Learning & Cognitive Development



Initial State

Learning Mechanism

Final State

(see Tenenbaum, Kemp, Griffiths, Goodman, 2011; Carey, 2009)

The Puzzle of Learning & Cognitive Development

Initial State

LOT?

Learning Mechanism

inductive bootstrapping?

Final State

LOT′?

(Carey, 1985, 2009; Carey, Spelke, 1994)
(Fodor, 1975; Turing, 1936; Fodor & Pylyshyn, 1988; Goodman, Tenenbaum, & Gerstenberg, 2015)

Three questions about learning in the LOT

1. How are concepts represented?

2. How are changes proposed?

3. How are proposals assessed?

(Newell, Shaw, Simon, 1959; Sussman, 1973; Miller, Johnson-Laird, 1976; Lenat & Brown, 1984; Lenat, 1983a,b, 1982)

Three questions about learning in the LOT

1. How are concepts represented?
    - **programs in some language**

2. How are changes proposed?

3. How are proposals assessed?

(Goodman, Tenenbaum, Feldman, & Griffiths, 2008; Piantadosi, Tenenbaum, & Goodman, 2012, 2016; Lake, Salakhutdinov, & Tenenbaum, 2015; Kemp & Tenenbaum, 2008; Dechter, Malmaud, Adams, & Tenenbaum, 2013; Rule, Dechter, & Tenenbaum, 2015; Piantadosi, unpub.; Ullman, Goodman, & Tenenbaum, 2012; Goodman, Ullman, & Tenenbaum, 2011)

Three questions about learning in the LOT

1. How are concepts represented?
   - programs in some language
2. How are changes proposed?
   - **small, random syntactic changes to a concept definition**
3. How are proposals assessed?

(Goodman, Tenenbaum, Feldman, & Griffiths, 2008; Piantadosi, Tenenbaum, & Goodman, 2012, 2016; Lake, Salakhutdinov, & Tenenbaum, 2015; Kemp & Tenenbaum, 2008; Dechter, Malmaud, Adams, & Tenenbaum, 2013; Rule, Dechter, & Tenenbaum, 2015; Piantadosi, unpub.; Ullman, Goodman, & Tenenbaum, 2012; Goodman, Ullman, & Tenenbaum, 2011)

Three questions about learning in the LOT

1. How are concepts represented?
   - programs in some language

2. How are changes proposed?
   - small, random syntactic changes to a concept definition

3. How are proposals assessed?
   - **accuracy & description length (& sometimes efficiency)**

(Goodman, Tenenbaum, Feldman, & Griffiths, 2008; Piantadosi, Tenenbaum, & Goodman, 2012, 2016; Lake, Salakhutdinov, & Tenenbaum, 2015; Kemp & Tenenbaum, 2008; Dechter, Malmaud, Adams, & Tenenbaum, 2013; Rule, Dechter, & Tenenbaum, 2015; Piantadosi, unpub.; Ullman, Goodman, & Tenenbaum, 2012; Goodman, Ullman, & Tenenbaum, 2011)

Three questions about learning in the LOT

1. How are concepts represented?
    - programs in some language

2. How are changes proposed?
    - small, random syntactic changes to a concept definition

3. How are proposals assessed?
    - accuracy & description length (& sometimes efficiency)

(Goodman, Tenenbaum, Feldman, & Griffiths, 2008; Piantadosi, Tenenbaum, & Goodman, 2012, 2016; Lake, Salakhutdinov, & Tenenbaum, 2015; Kemp & Tenenbaum, 2008; Dechter, Malmaud, Adams, & Tenenbaum, 2013; Rule, Dechter, & Tenenbaum, 2015; Piantadosi, unpub.; Ullman, Goodman, & Tenenbaum, 2012; Goodman, Ullman, & Tenenbaum, 2011)

This talk

- learning as programming
- bootstrapping the LOT with term rewriting
- toward a model of conceptual change

A nagging problem

LOT + bootstrapping = LOT′

A nagging problem

LOT + bootstrapping = LOT′

modeled as

Prog. Lang. & Library + stochastic search = Prog. Lang. & Library′

Three questions about learning in the LOT

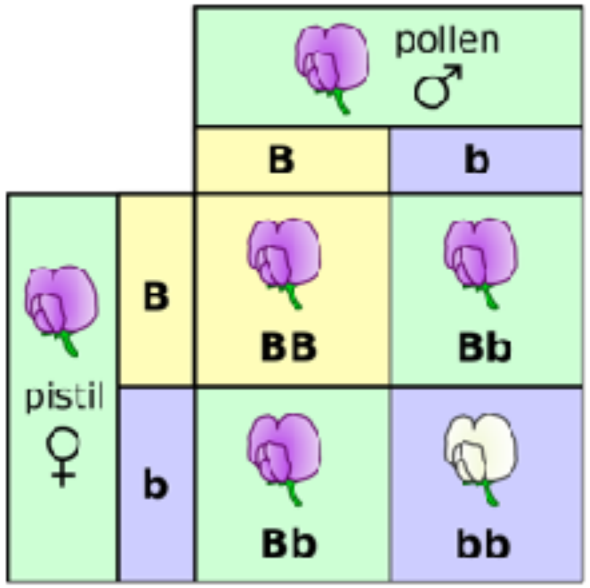
1. How are concepts represented?
   - **programs in some language**
2. How are changes proposed?
   - small, random syntactic changes to a concept definition
3. How are proposals assessed?
   - accuracy & description length (& sometimes efficiency)

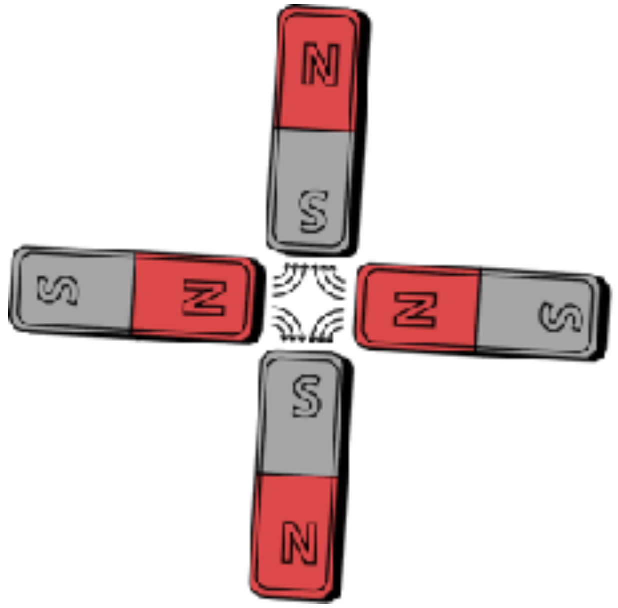
(Goodman, Tenenbaum, Feldman, & Griffiths, 2008; Piantadosi, Tenenbaum, & Goodman, 2012, 2016; Lake, Salakhutdinov, & Tenenbaum, 2015; Kemp & Tenenbaum, 2008; Dechter, Malmaud, Adams, & Tenenbaum, 2013; Rule, Dechter, & Tenenbaum, 2015; Piantadosi, unpub.; Ullman, Goodman, & Tenenbaum, 2012; Goodman, Ullman, & Tenenbaum, 2011)

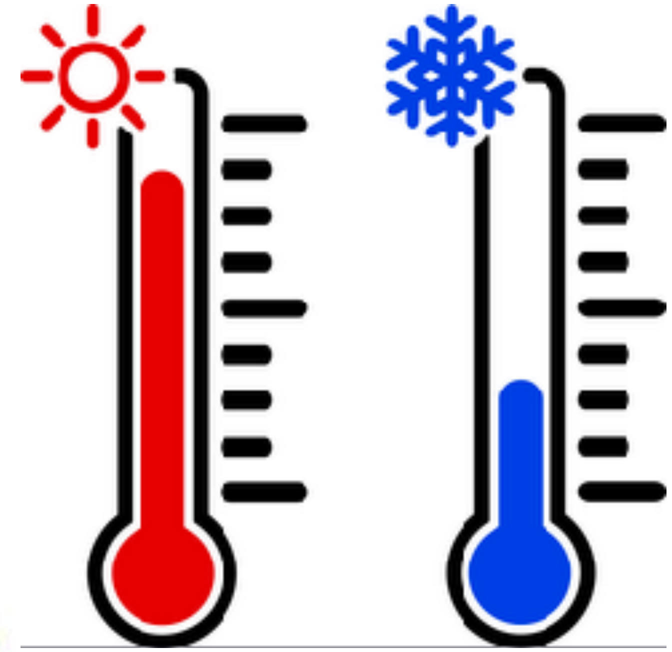
Three questions about learning in the LOT

1. How are concepts represented?
    - **programs in some *fixed* language**
2. How are changes proposed?
    - small, random syntactic changes to a concept definition
3. How are proposals assessed?
    - accuracy & description length (& sometimes efficiency)

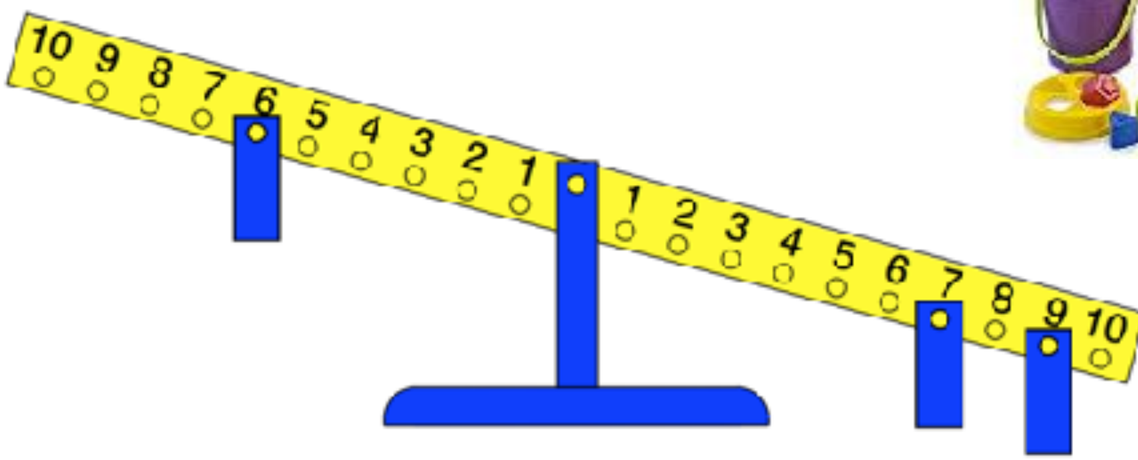
(Goodman, Tenenbaum, Feldman, & Griffiths, 2008; Piantadosi, Tenenbaum, & Goodman, 2012, 2016; Lake, Salakhutdinov, & Tenenbaum, 2015; Kemp & Tenenbaum, 2008; Dechter, Malmaud, Adams, & Tenenbaum, 2013; Rule, Dechter, & Tenenbaum, 2015; Piantadosi, unpub.; Ullman, Goodman, & Tenenbaum, 2012; Goodman, Ullman, & Tenenbaum, 2011)



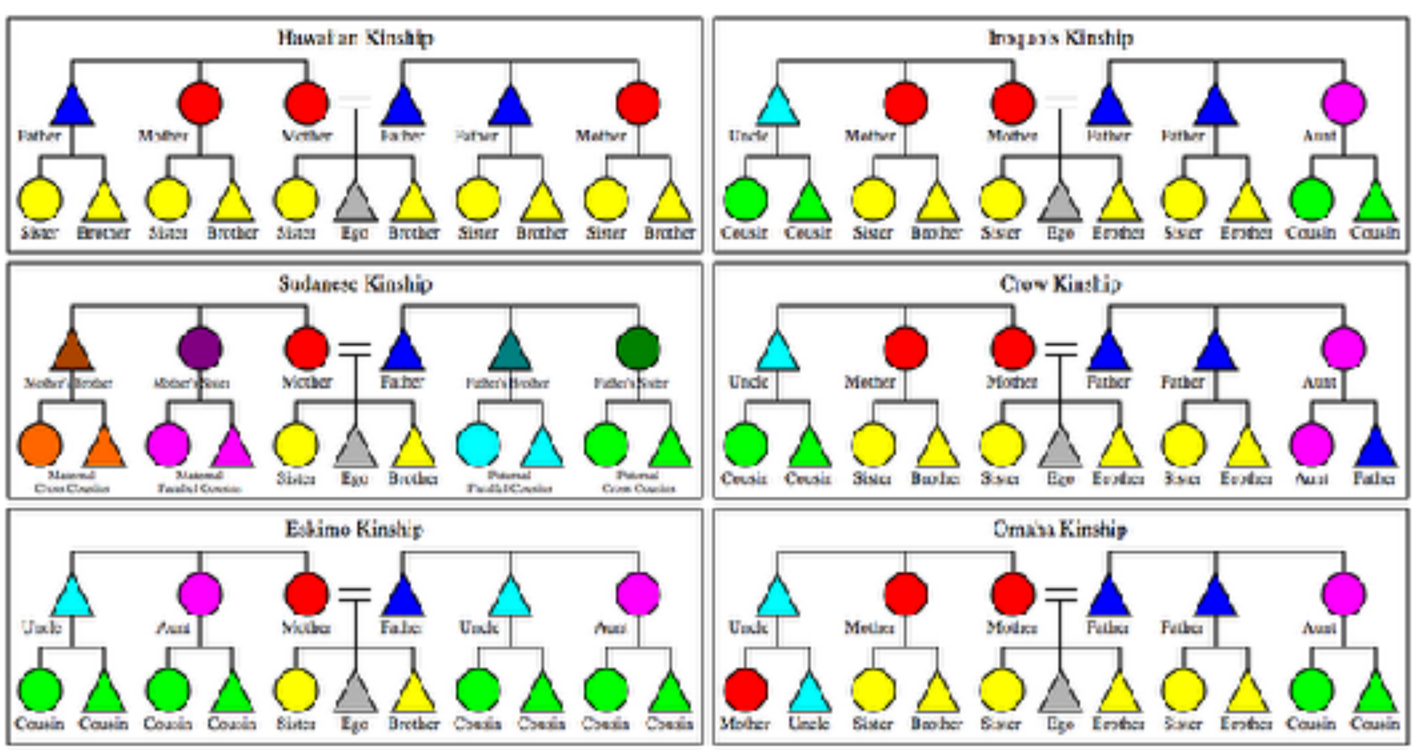




AABABA
ABAAAB
AABAAB
ABAAAB
ABAABA



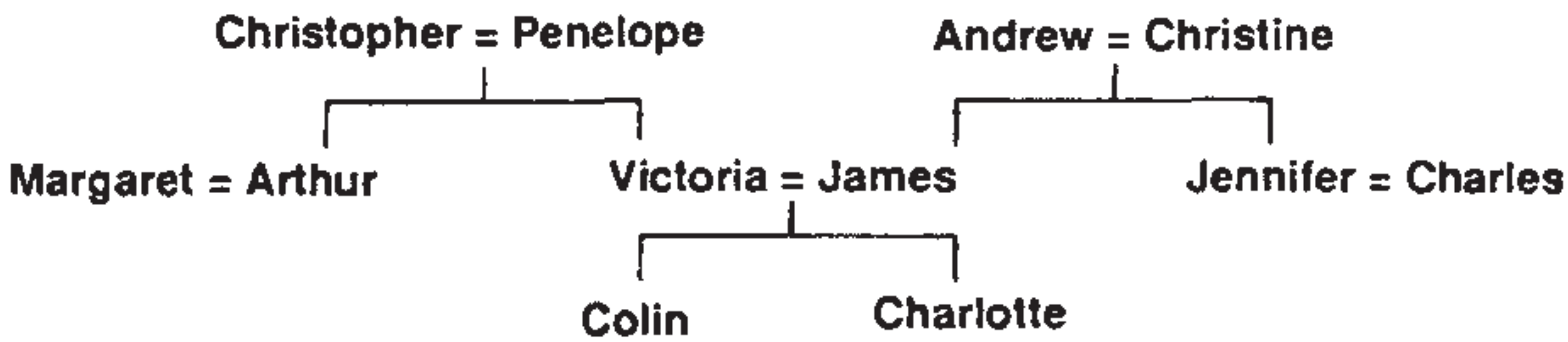
six-hundred-forty-seven-thousand-nine-hundred-sixteen

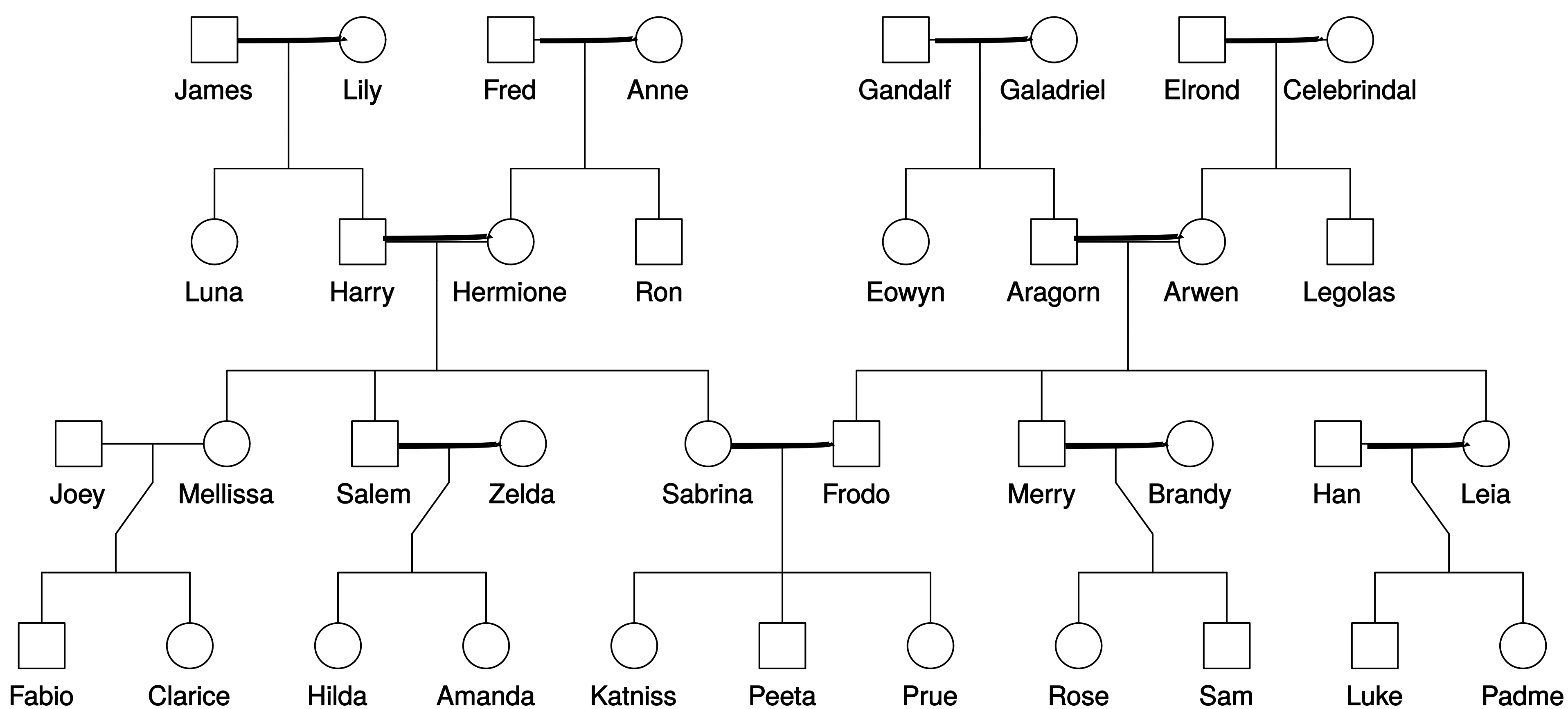


Kinship is a great space for studying conceptual change

Definite Gender	boy, girl, man, woman
Generic Gender	male, female
Definite Nuclear	brother, sister, mother, father, husband, wife, son, daughter
Generic Nuclear	sibling, spouse, parent, child
Definite Extended	aunt, uncle, nephew, niece, grandmother, grandfather, granddaughter, grandson, grandnephew, grandniece
Generic Extended	grandparent, grandchild, cousin
Structurally Recursive	great-aunt, great-uncle, great-grandfather, great-grandmother, great-grandparent, great-granddaughter, great-grandson, great-grandchild, great-great-, great-great-great, . . .
Linearly Recursive	ancestor, descendant
Nonlinearly Recursive	relative, blood relative, in-law, m^{th} cousin n^{th} removed, step-relations

typical kinship data





(Rumelhart, Hinton, Williams, 1986; Mollica, Piantadosi, 2015, sub; Katz, Goodman, Kersting, Kemp, Tenenbaum, 2008)

potential kinship data

```
⋮
true → husband(Christopher, Penelope)
true → cousin(Rose, Luke)
true → uncle(Arthur, Colin)
true → brother(Arthur, Victoria)
true → man(Arthur)
true → girl(Charlotte)
true → dad(Joey, Clarice)
true → brother(Sam, Rose)
true → great-uncle(Ron, Katniss)
true → sister(Katniss, Prue)
true → sister(Prue, Katniss)
true → husband(James, Victoria)
true → sister(Rose, Sam)
false → sister(Sam, Rose)
⋮
```

(Keil & Batterman, 1984; Keil, 1989; Landau, 1982)

potential kinship grammar

potential kinship grammar

```
male
female
spouse
parent
```

potential kinship grammar

```
male(Aragorn)
female(Arwen)
spouse(Aragorn, Arwen)
parent(Elrond, Arwen)
```

potential kinship grammar

```
male(Aragorn)
female(Arwen)
spouse(Aragorn, Arwen)
parent(Elrond, Arwen)
⋮
and(male(x), spouse(x, y)) → husband(x, y)
and(female(x), spouse(x, y)) → wife(y, y)
and(female(x), sibling(x, y)) → sister(x, y)
and(male(x), sibling(x, y)) → brother(x, y)
and(male(x), parent(x, y)) → father(x, y)
and(female(x), parent(x, y)) → mother(x, y)
and(male(x), parent(y, x)) → son(x, y)
and(female(x), parent(y, x)) → daughter(x, y)
and(parent(z,y), parent(z,x)) → sibling(x, y)
⋮
parent(x, y) → ancestor(x, y)
and(parent(x, y), ancestor(y, z)) → ancestor(x, y)
⋮
and(ancestor(x, y), ancestor(x, z)) → blood_relative(y, z)
```

Three questions about learning in the LOT

1. How are concepts represented?
   - **programs in some *fixed* language**

2. How are changes proposed?
   - small, random syntactic changes to a concept definition

3. How are proposals assessed?
   - accuracy & description length (& sometimes efficiency)

(Goodman, Tenenbaum, Feldman, & Griffiths, 2008; Piantadosi, Tenenbaum, & Goodman, 2012, 2016; Lake, Salakhutdinov, & Tenenbaum, 2015; Kemp & Tenenbaum, 2008; Dechter, Malmaud, Adams, & Tenenbaum, 2013; Rule, Dechter, & Tenenbaum, 2015; Piantadosi, unpub.; Ullman, Goodman, & Tenenbaum, 2012; Goodman, Ullman, & Tenenbaum, 2011)

Three questions about learning in the LOT

1. How are concepts represented?
   - **programs in some *adaptive* language**

2. How are changes proposed?
   - small, random syntactic changes to a concept definition

3. How are proposals assessed?
   - accuracy & description length (& sometimes efficiency)

(Goodman, Tenenbaum, Feldman, & Griffiths, 2008; Piantadosi, Tenenbaum, & Goodman, 2012, 2016; Lake, Salakhutdinov, & Tenenbaum, 2015; Kemp & Tenenbaum, 2008; Dechter, Malmaud, Adams, & Tenenbaum, 2013; Rule, Dechter, & Tenenbaum, 2015; Piantadosi, unpub.; Ullman, Goodman, & Tenenbaum, 2012; Goodman, Ullman, & Tenenbaum, 2011)

Term Rewriting Systems

$$TRS = (\Sigma, R)$$

(Baader & Nipkow, 1999; Bezem, Klop, & de Vrijer, 2003)

Term Rewriting Systems

Signature:
- a set of primitives
- what things exist
- syntax

$$TRS = (\Sigma, R)$$

(Baader & Nipkow, 1999; Bezem, Klop, & de Vrijer, 2003)

Term Rewriting Systems

Signature:

- a set of primitives
- what things exist
- syntax

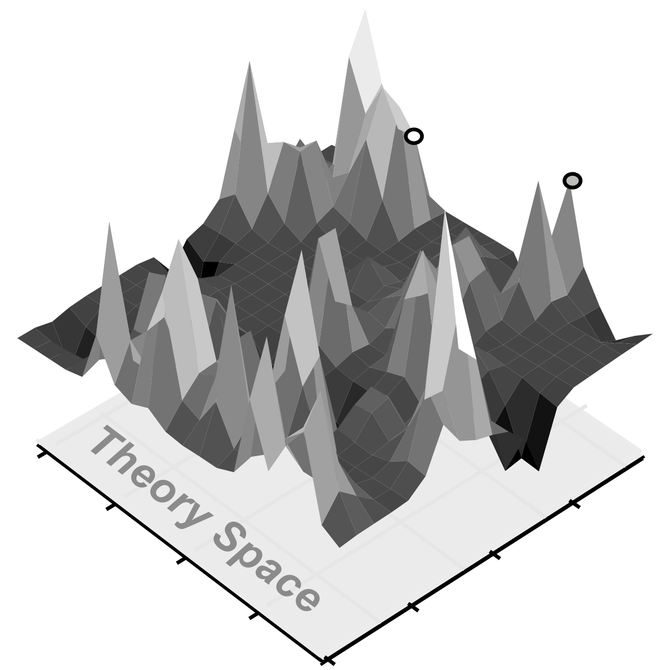
$$TRS = (\Sigma, R)$$

Rules:

- a list of rewrite rules
- how things behave
- semantics

(Baader & Nipkow, 1999; Bezem, Klop, & de Vrijer, 2003)

Stochastic search over TRSs



- remove a symbol s from Σ_{i-1} and all rules involving s from R_{i-1}
- add a symbol s to Σ_i
- generate a new rule r and add it to R_i
- remove a rule r from R_{i-1}

One solution: models LOTs as Term Rewriting Systems (TRSs)

LOT* + *bootstrapping* = *LOT′

modeled as

TRS* + *stochastic search* = *TRS′

This talk

- learning as programming
- bootstrapping the LOT with term rewriting
- toward a model of conceptual change

Learning list concepts through program induction

Joshua Rule,[1*] Eric Schulz,[2*] Steven T. Piantadosi,[3] & Joshua B. Tenenbaum[1]
[1]Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology
[2]Department of Psychology, Harvard University
[3]Department of Brain and Cognitive Sciences, University of Rochester
*Contributed equally

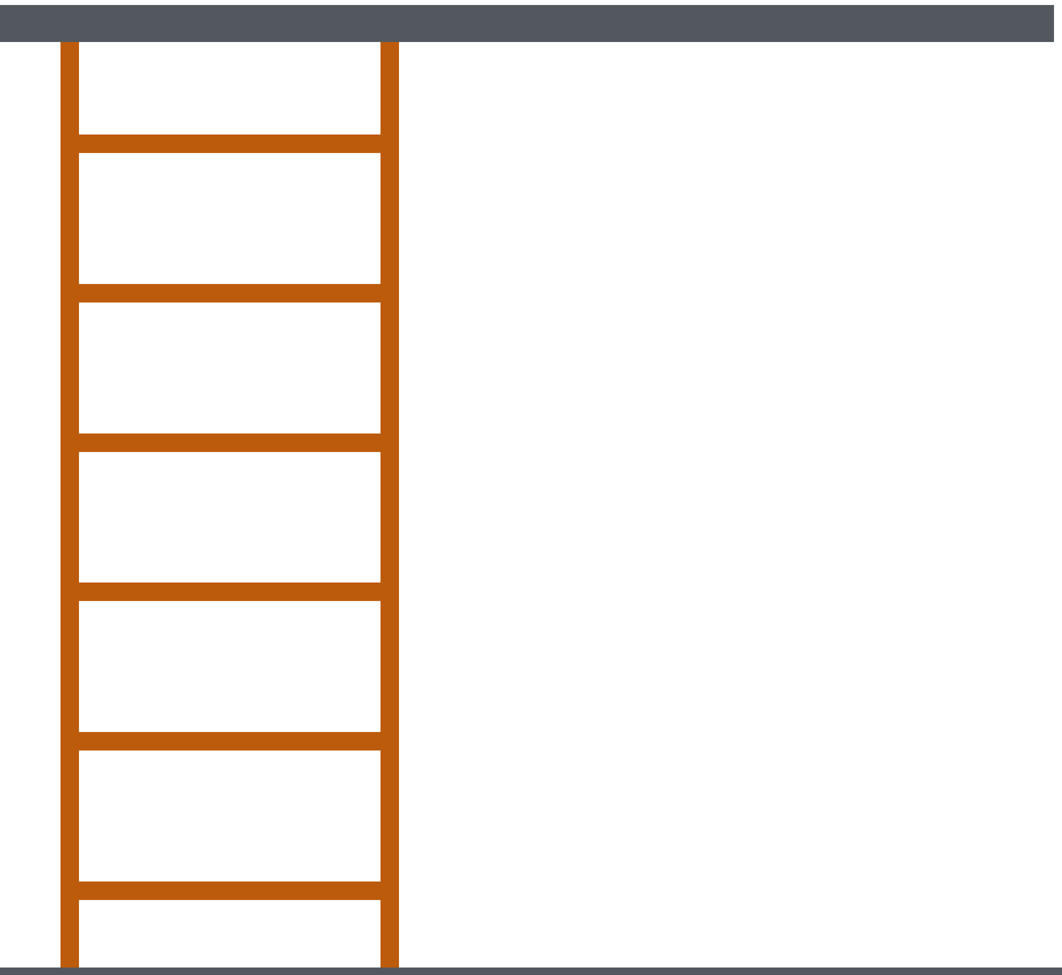
Josh Tenenbaum

Steve Piantadosi

Eric Schulz

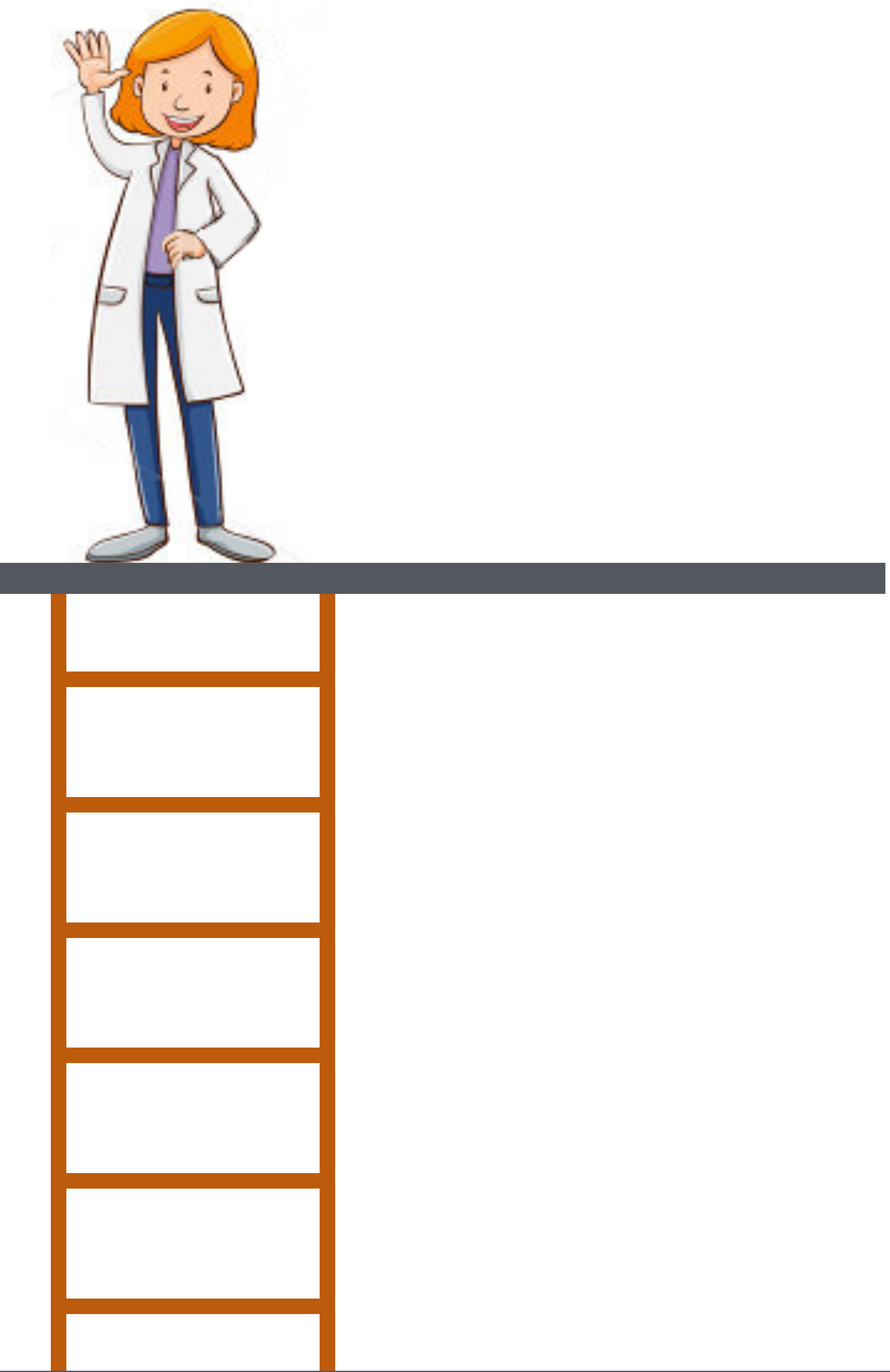
(Rule, Schulz, Piantadosi, Tenenbaum, 2018)

Martha’s Magical Machines



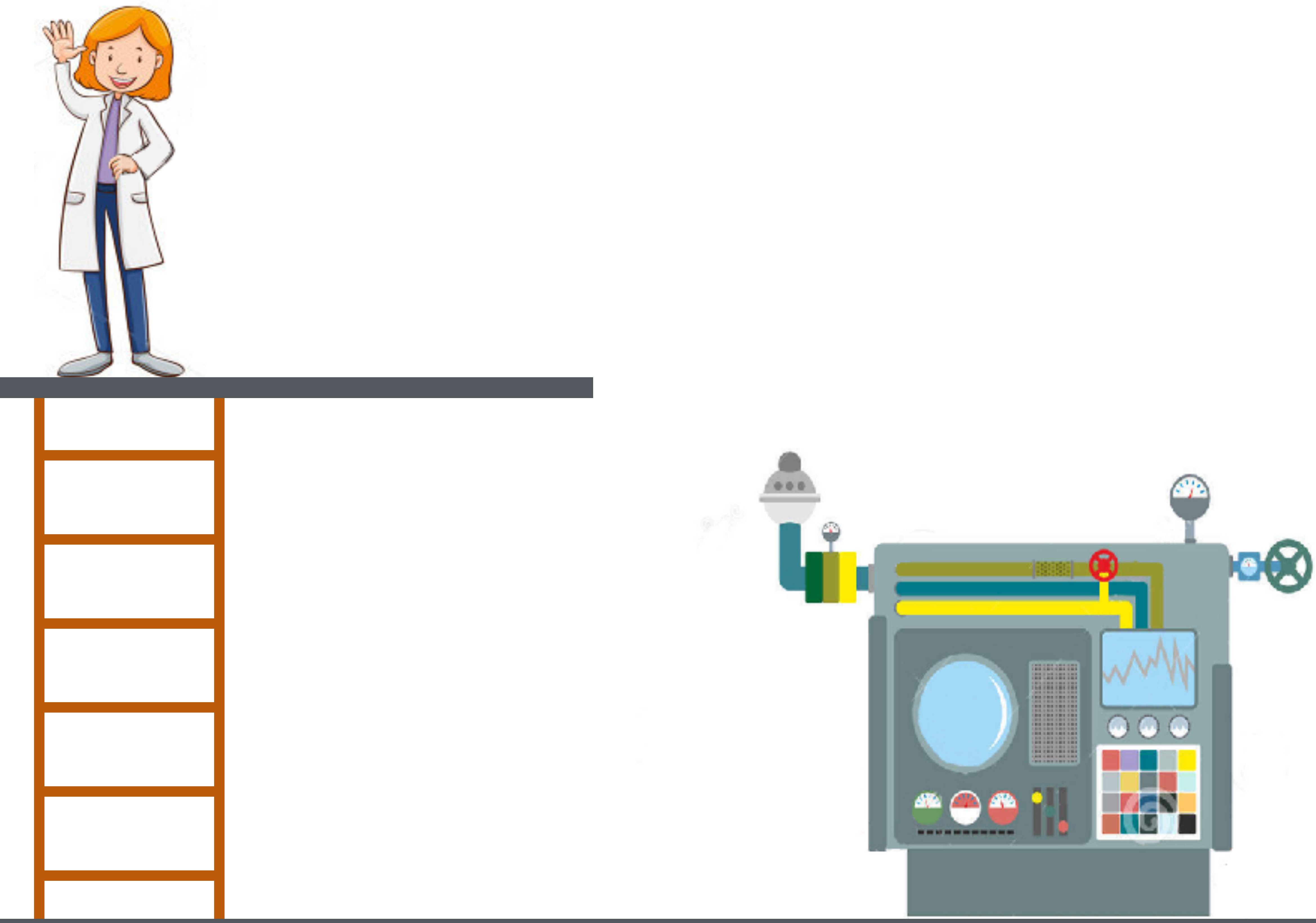
(cf. Piantadosi, Tenenbaum, Goodman, 2016)

Martha’s Magical Machines



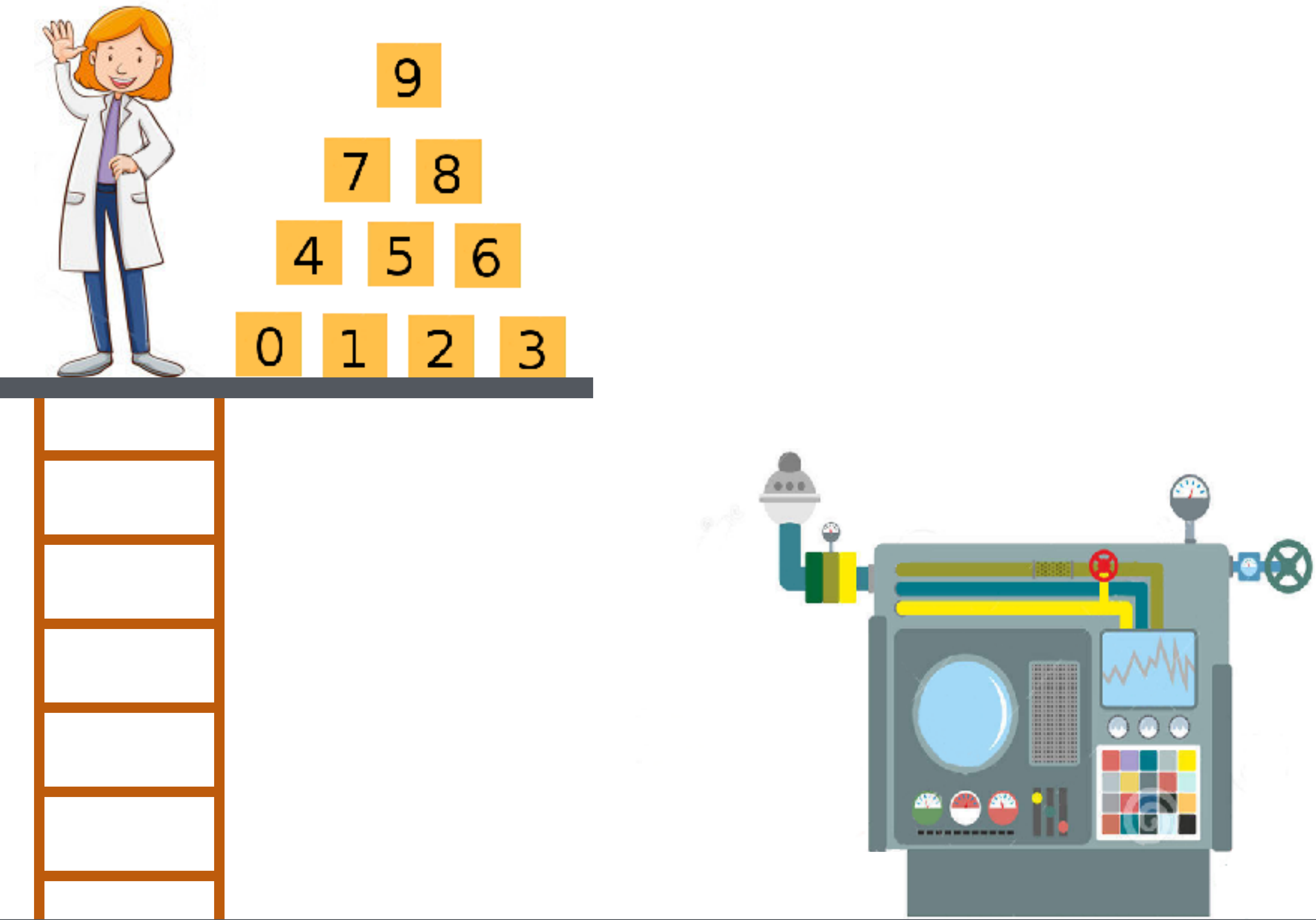
(cf. Piantadosi, Tenenbaum, Goodman, 2016)

Martha's Magical Machines



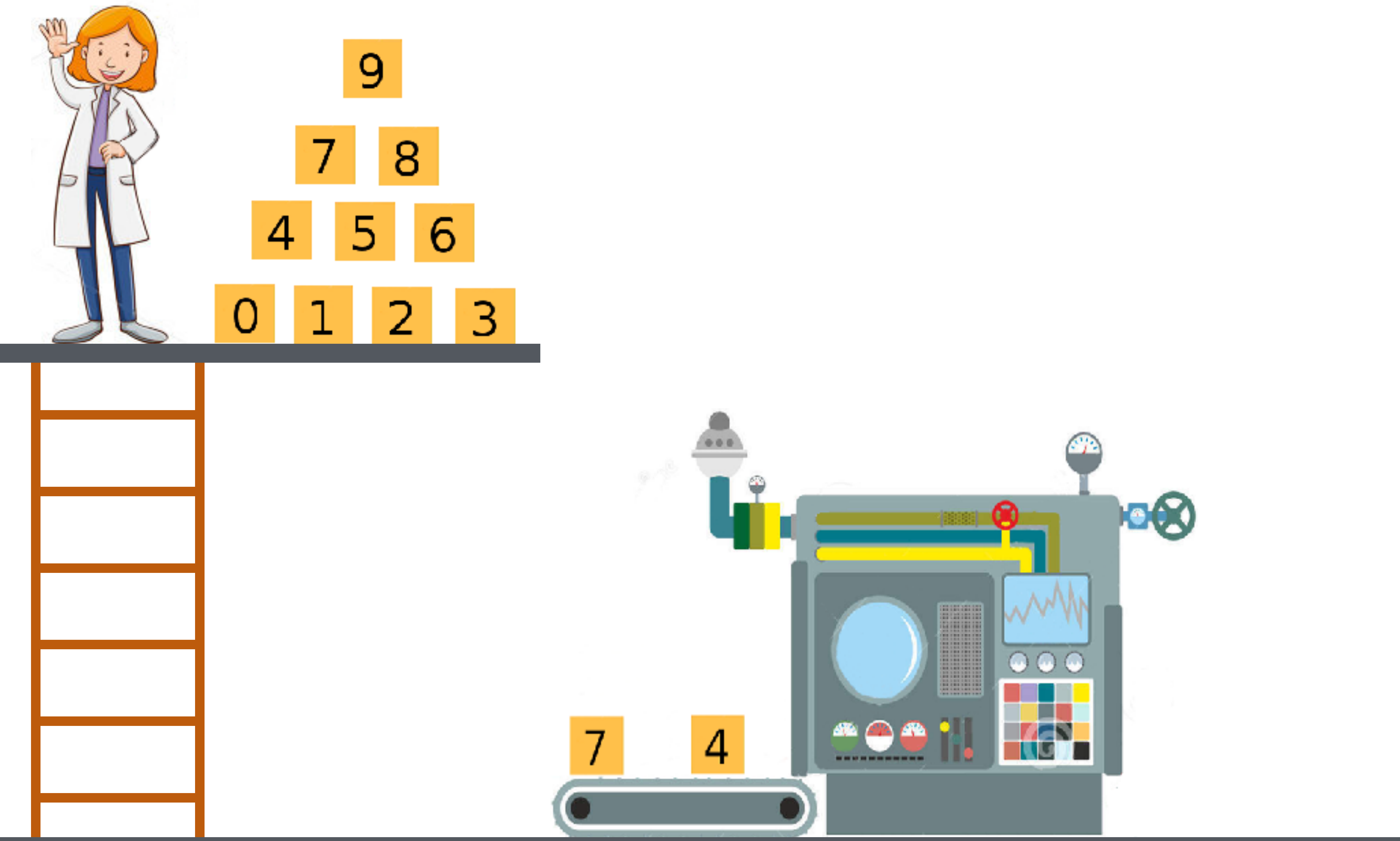
(cf. Piantadosi, Tenenbaum, Goodman, 2016)

Martha’s Magical Machines



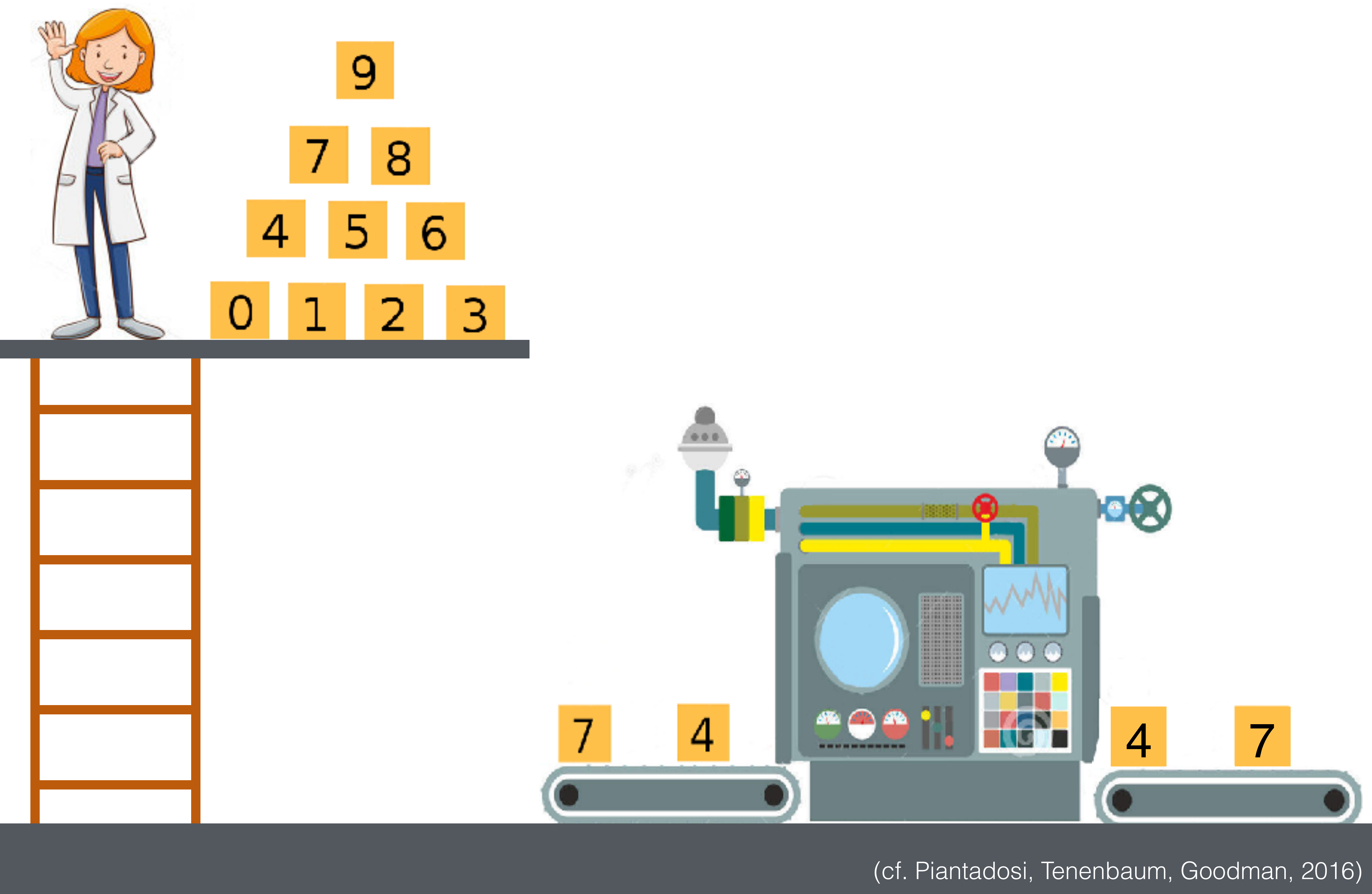
(cf. Piantadosi, Tenenbaum, Goodman, 2016)

Martha’s Magical Machines



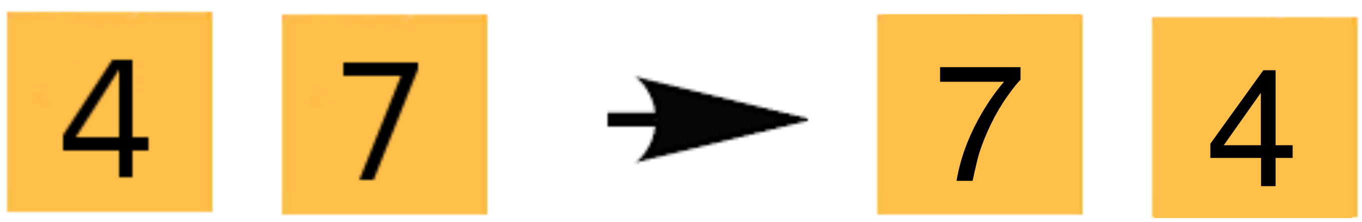
(cf. Piantadosi, Tenenbaum, Goodman, 2016)

Martha’s Magical Machines

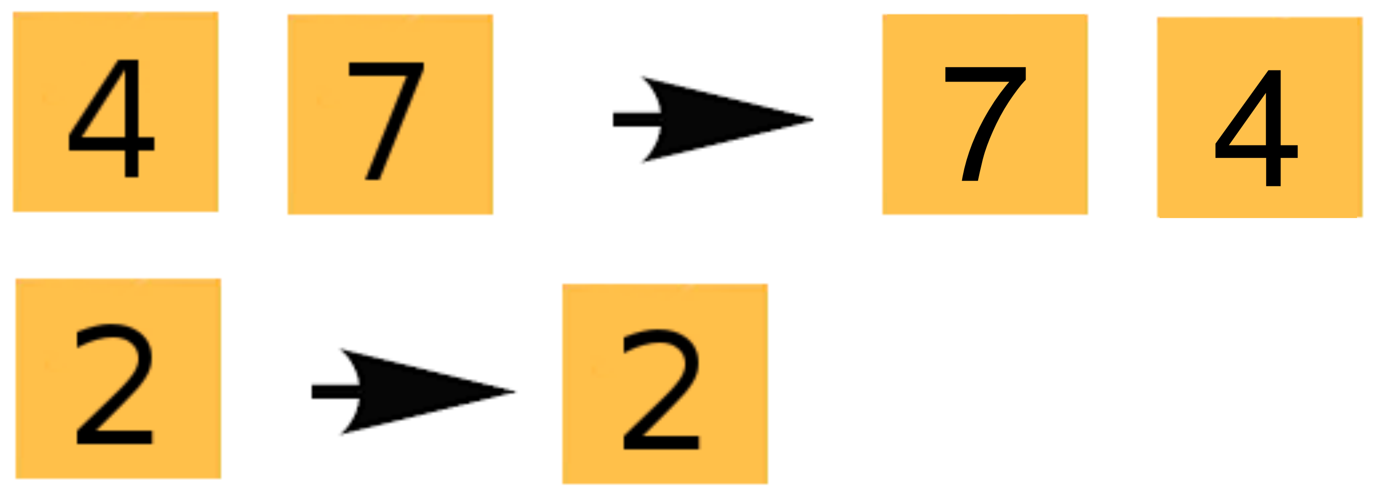


(cf. Piantadosi, Tenenbaum, Goodman, 2016)

Martha’s Magical Machines



Martha’s Magical Machines



Martha's Magical Machines

4 7 → 7 4

2 → 2

9 6 5 → 5 6 9

Stochastic search over TRSs

```
def search(data, h0, N=1500, n_top=10, n_steps=50, confidence=2/3):
    dataset = []
    h, score = h0, score(h0)
    hs = heap([(h, score)])
    for (i, o) in data:
        for _ in range(N):
            h_next = propose(h)
            score_next = score(h_next)
            h, score = metropolis(h, score, h_next, score_next)
            hs.insert((h, score))
        best_hs = hs.take_top(n_top)
        o_hat = most_likely_output(i, n_steps, best_hs)
        data.append((i, o))
        N *= (confidence if o_hat == o else 1/confidence)
    return hs
```

Model Primitives

Name & Input/Output Pair	**Description**
`0, 1, 2`	constant natural numbers
`[]`	the empty list
`succ(0)`	the successor of x
`cons(1, [2,3]) = [1,2,3]`	prepend x to y
`sum([1,2,3]) = [6]`	sum x
`add(3, [1,2,3]) = [4,5,6]`	add x to the elements of y
`insert(4, [3,5]) = [3,4,5]`	insert x into y in sorted order
`remove(1, [6,1,4]) = [6,4]`	remove every x in y
`count(7, [7,1,7] = [2])`	count every x in y
`even(5) = false`	`true` if x is even else `false`
`greater(8, 2) = true`	`true` if $x > y$ else `false`
`if(true, [7], [2,5]) = [7]`	if x then y else z
`nth(3, [9,5,8]) = [8]`	the x^{th} element of y

Model Primitives

$$h_0 = (\Sigma_0^{*}, R_0)$$

Name & Input/Output Pair	**Description**
`0, 1, 2`	constant natural numbers
`[]`	the empty list
`succ(0)`	the successor of x
`cons(1, [2,3]) = [1,2,3]`	prepend x to y
`sum([1,2,3]) = [6]`	sum x
`add(3, [1,2,3]) = [4,5,6]`	add x to the elements of y
`insert(4, [3,5]) = [3,4,5]`	insert x into y in sorted order
`remove(1, [6,1,4]) = [6,4]`	remove every x in y
`count(7, [7,1,7] = [2])`	count every x in y
`even(5) = false`	`true` if x is even else `false`
`greater(8, 2) = true`	`true` if $x > y$ else `false`
`if(true, [7], [2,5]) = [7]`	if x then y else z
`nth(3, [9,5,8]) = [8]`	the x^{th} element of y

*plus the target concept

Experiment 1

- 149 participants (61 female, mean age=36.93, SD=12.20)
- 5 concepts/participant (out of 12)
- 10 trials/concept

Experiment 1

Experiment 1

```
# const xs: return 3
# Example: const([1,2,4]) = [3]
const(x_) = 3;
```

Experiment 1

```
# const xs: return 3
# Example: const([1,2,4]) = [3]
const(x_) = 3;
```

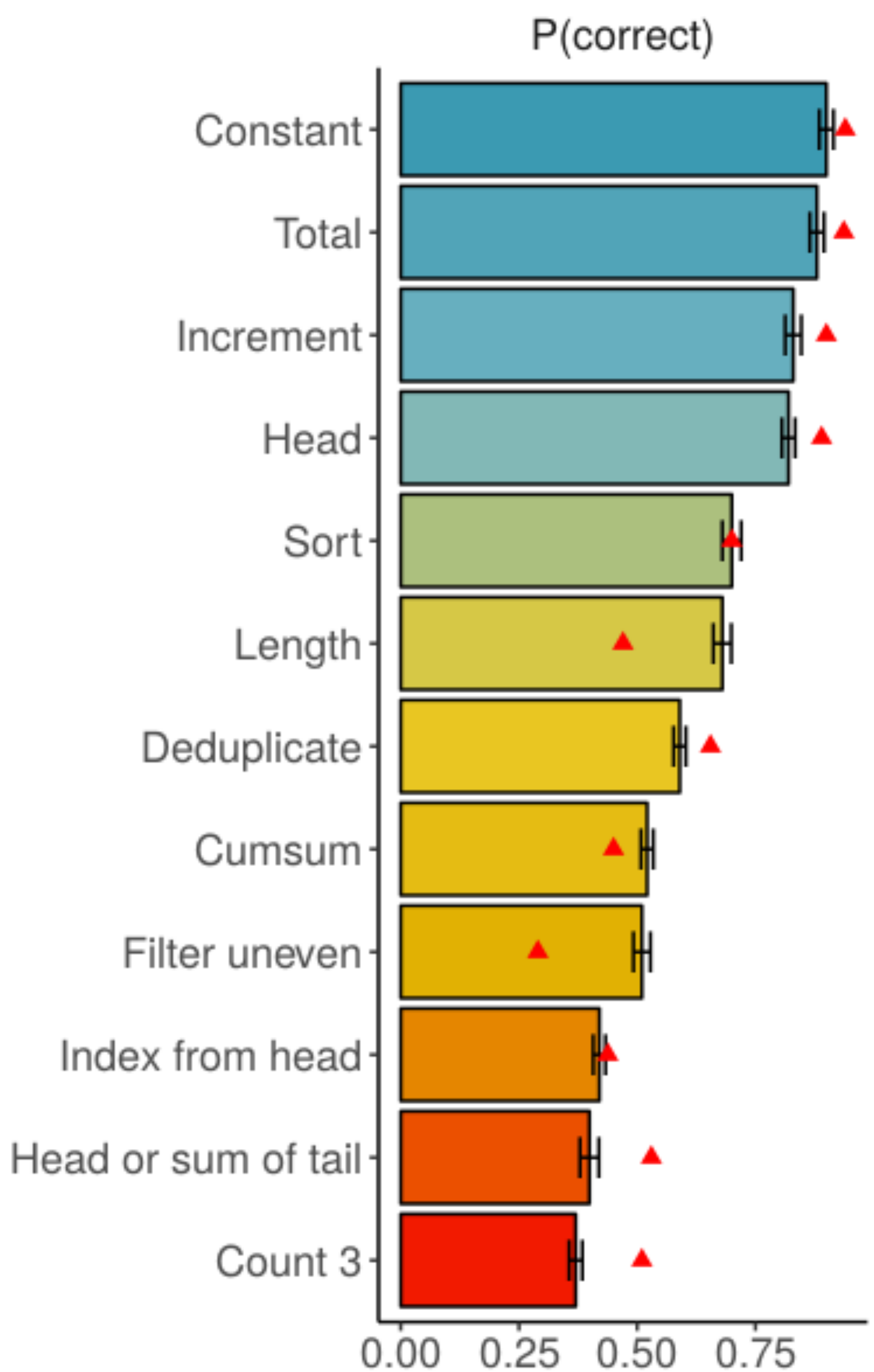
```
# index-in-head xs: return the headth element of the xs
# Example: index_in_head([2,3]) = [3]
index-in-head(cons(0 y_)) = 0
index-in-head(cons(succ(x_) y_)) = nth(x_ y_);
```

Experiment 1

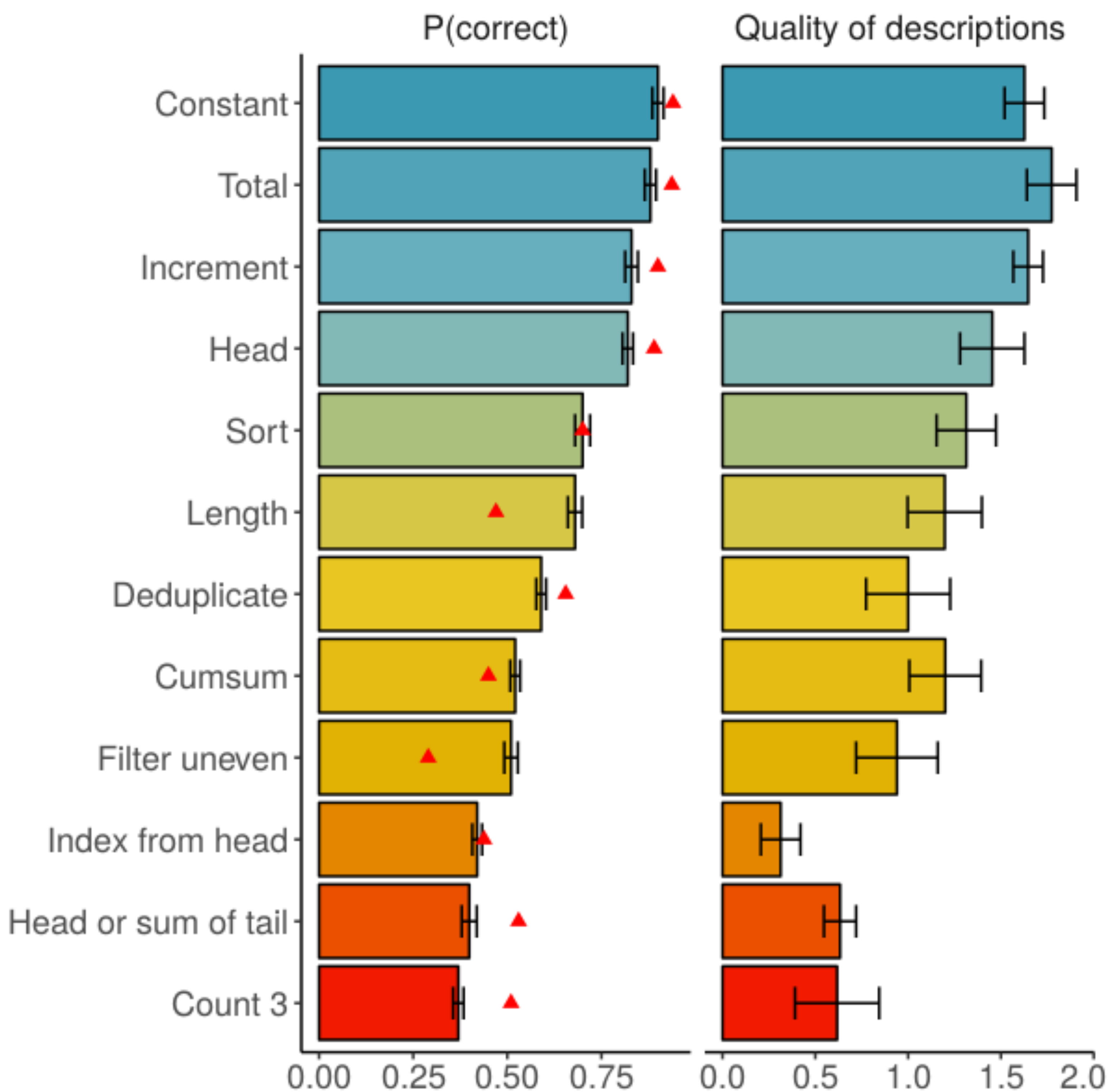
```
# const xs: return 3
# Example: const([1,2,4]) = [3]
const(x_) = 3;

# total xs: sum all the elements of xs
# Example: total([1,2,3]) = [6]
total(x_) = sum(x_);

# increment xs: add 1 to each element of xs
# Example: increment([1,2]) = [2,3]
increment(x_) = add(1 x_);

# head xs: return the first element of xs
# Example: head([2,3,1]) = [2]
head(cons(x_ y_)) = x_;

# length xs: compute the length of xs
# Example: length([2,3,1]) = [3]
length([]) = 0;
length(cons(x_ y_)) = succ(length(y_));

# sort xs: sort xs
# Example: sort([3,1]) = [1,3]
sort([]) = [];
sort(cons(x_ y_)) = insert(x_ sort(y_));

# deduplicate xs: remove all duplicates from xs
# Example: deduplicate([2,1,2,2,1]) = [2,1]
deduplicate([]) = [];
deduplicate(cons(x_ y_)) =
    cons(x_ deduplicate(remove(x_ y_)))
```

```
# cumsum xs: cumulatively sum the elements of xs
# Example: cumsum([2,3,1]) = [2,5,6]
cumsum([]) = [];
cumsum(cons(x_ y_)) = cons(x_ cumsum(add(x_ y_)))

# filter_odd xs: remove the odd numbers from xs
# Example: filter_odd([2,3,1,4]) = [2,4]
filter_odd([]) = [];
filter_odd(cons(x_ y_)) =
    if(even?(x_) cons(x_ filter_odd(y_)) filter_odd(y_));


# index-in-head xs: return the headth element of the xs
# Example: index_in_head([2,3]) = [3]
index-in-head(cons(0 y_)) = 0
index-in-head(cons(succ(x_) y_)) = nth(x_ y_);

# head-or-tail: return the larger of head or sum-of-tail
# Example: head_or_tail([2,3,1]) = [4]
head-or-tail([]) = 0;
head-or-tail(cons(x_ y_)) =
    if(greater(x_ sum(y_)) x_ sum(y_));

# count3 xs: how often does 3 appear in xs?
# Example: count3([2,3,3]) = [2]
count3(x_) = count(succ(succ(succ(0))) x_);
```

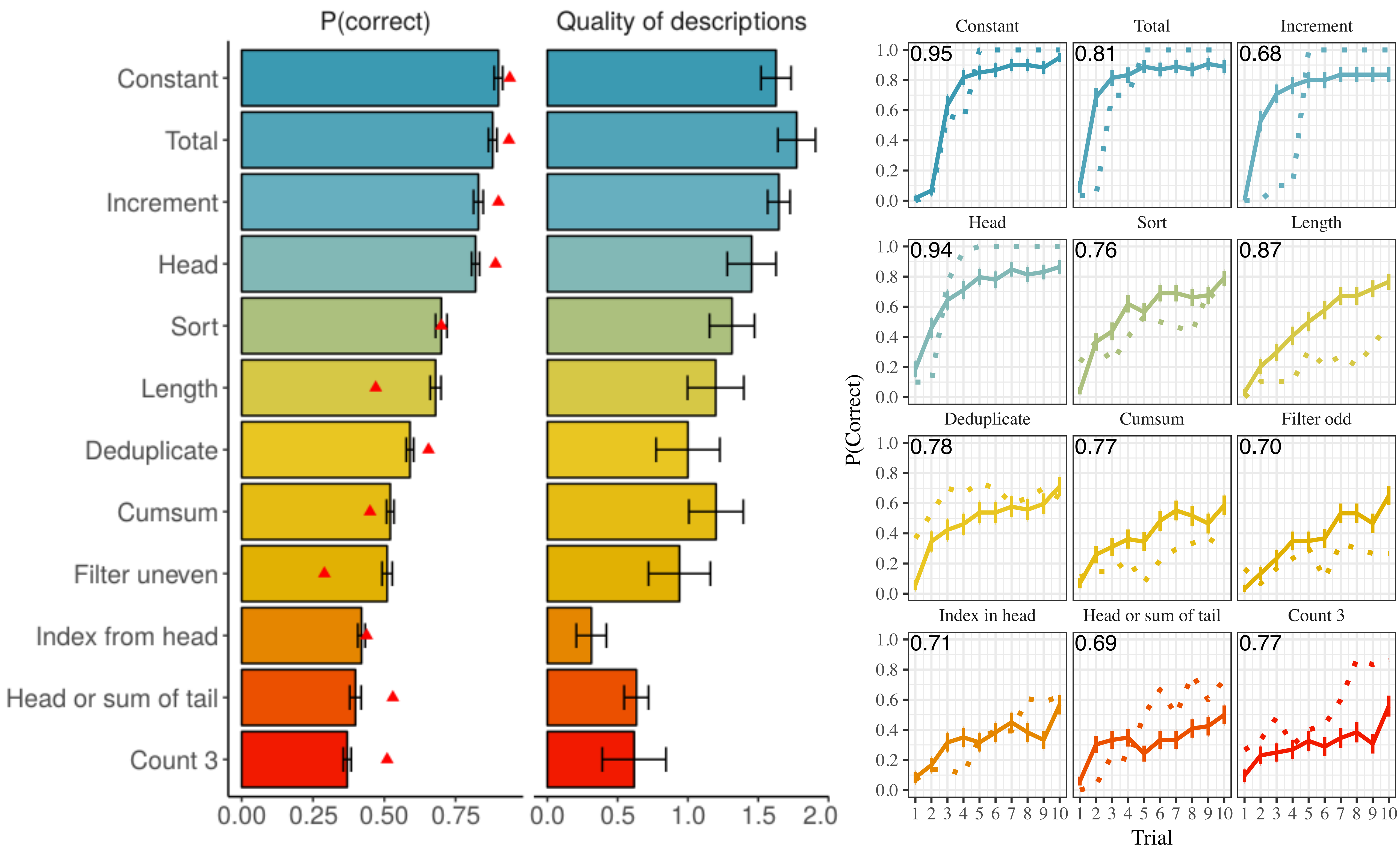
Experiment 1



Experiment 1



Experiment 1



Experiment 2

- 91 participants (46 males, mean age=34.51, SD=10.57)
- randomly assigned condition:
  relevant curriculum or random curriculum
- 4 concepts/condition
- 10 trials/concept

Experiment 2

```
# head xs: return the first element of xs
# Example: head([2,3,1]) = [2]
head(cons(x_ y_)) = x_;

# tail xs: return all but the first element of xs
# Example: tail([2,3,3]) = [3,3]
tail([]) = [];
tail(cons(x_ y_)) = y_;
```

```
# count3 xs: how often does 3 appear in xs?
# Example: count3([3,2,3]) = [2]
count3(x_) = count(succ(succ(succ(0))) x_);

# count-head-in-tail xs: how often is head in the tail?
# Example: count-head-in-tail([2,3,2]) = [1]
count-head-in-tail([]) = 0;
count-head-in-tail(x_) = count(head(x_) tail(x_));
```

Experiment 2

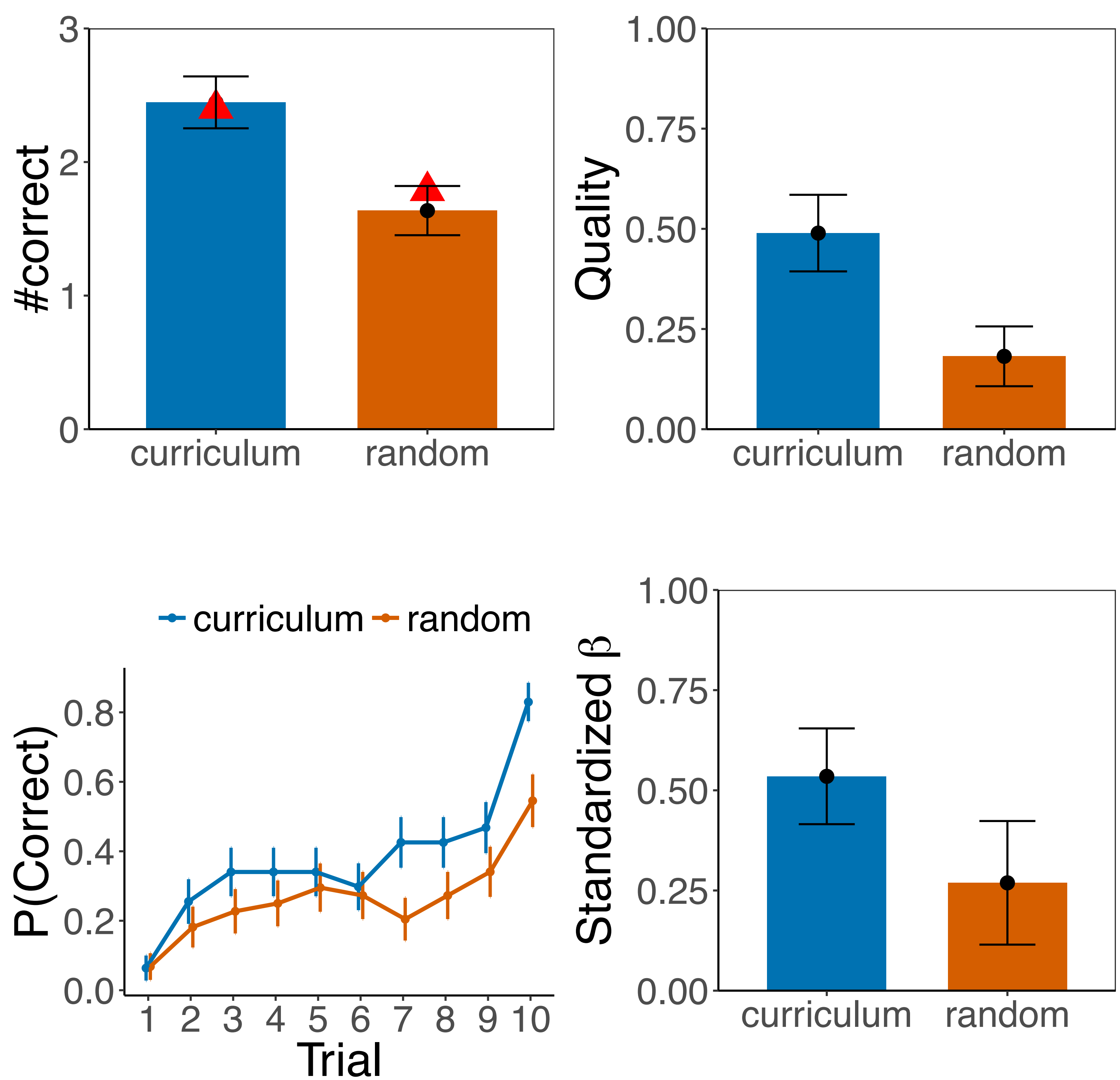
```
# head xs: return the first element of xs
# Example: head([2,3,1]) = [2]
head(cons(x_ y_)) = x_;

# tail xs: return all but the first element of xs
# Example: tail([2,3,3]) = [3,3]
tail([]) = [];
tail(cons(x_ y_)) = y_;
```

```
# count3 xs: how often does 3 appear in xs?
# Example: count3([3,2,3]) = [2]
count3(x_) = count(succ(succ(succ(0))) x_);

# count-head-in-tail xs: how often is head in the tail?
# Example: count-head-in-tail([2,3,2]) = [1]
count-head-in-tail([]) = 0;
count-head-in-tail(x_) = count(head(x_) tail(x_));
```

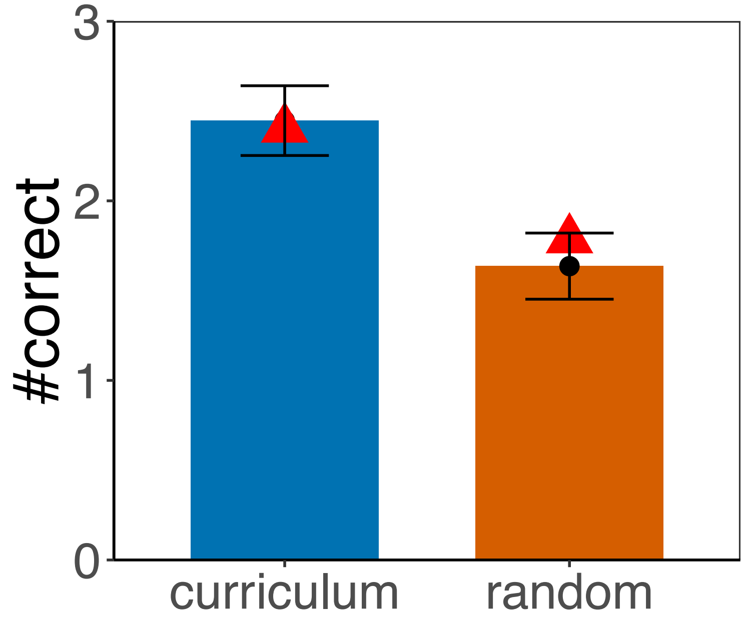
```
# const xs: return 3
# Example: const([1,2,4]) = [3]
const(x_) = 3;

# total xs: sum all the elements of xs
# Example: total([1,2,3]) = [6]
total(x_) = sum(x_);

# increment xs: add 1 to each element of xs
# Example: increment([1,2]) = [2,3]
increment(x_) = add(1 x_);

# length xs: compute the length of xs
# Example: length([2,3,1]) = [3]
length([]) = 0;
length(cons(x_ y_)) = succ(length(y_));

# sort xs: sort xs
# Example: sort([3,1]) = [1,3]
sort([]) = [];
sort(cons(x_ y_)) = insert(x_ sort(y_));

# deduplicate xs: remove all duplicates from xs
# Example: deduplicate([2,1,2,2,1]) = [2,1]
deduplicate([]) = [];
deduplicate(cons(x_ y_)) =
    cons(x_ deduplicate(remove(x_ y_)))
```

```
# cumsum xs: cumulatively sum the elements of xs
# Example: cumsum([2,3,1]) = [2,5,6]
cumsum([]) = [];
cumsum(cons(x_ y_)) = cons(x_ cumsum(add(x_ y_)))

# filter_odd xs: remove the odd numbers from xs
# Example: filter_odd([2,3,1,4]) = [2,4]
filter_odd([]) = [];
filter_odd(cons(x_ y_)) =
    if(even?(x_) cons(x_ filter_odd(y_)) filter_odd(y_));

# index-in-head xs: return the headth element of the xs
# Example: index_in_head([2,3]) = [3]
index-in-head(cons(0 y_)) = 0
index-in-head(cons(succ(x_) y_)) = nth(x_ y_);

# head-or-tail: return the larger of head or sum-of-tail
# Example: head_or_tail([2,3,1]) = [4]
head-or-tail([]) = 0;
head-or-tail(cons(x_ y_)) =
    if(greater(x_ sum(y_)) x_ sum(y_));
```

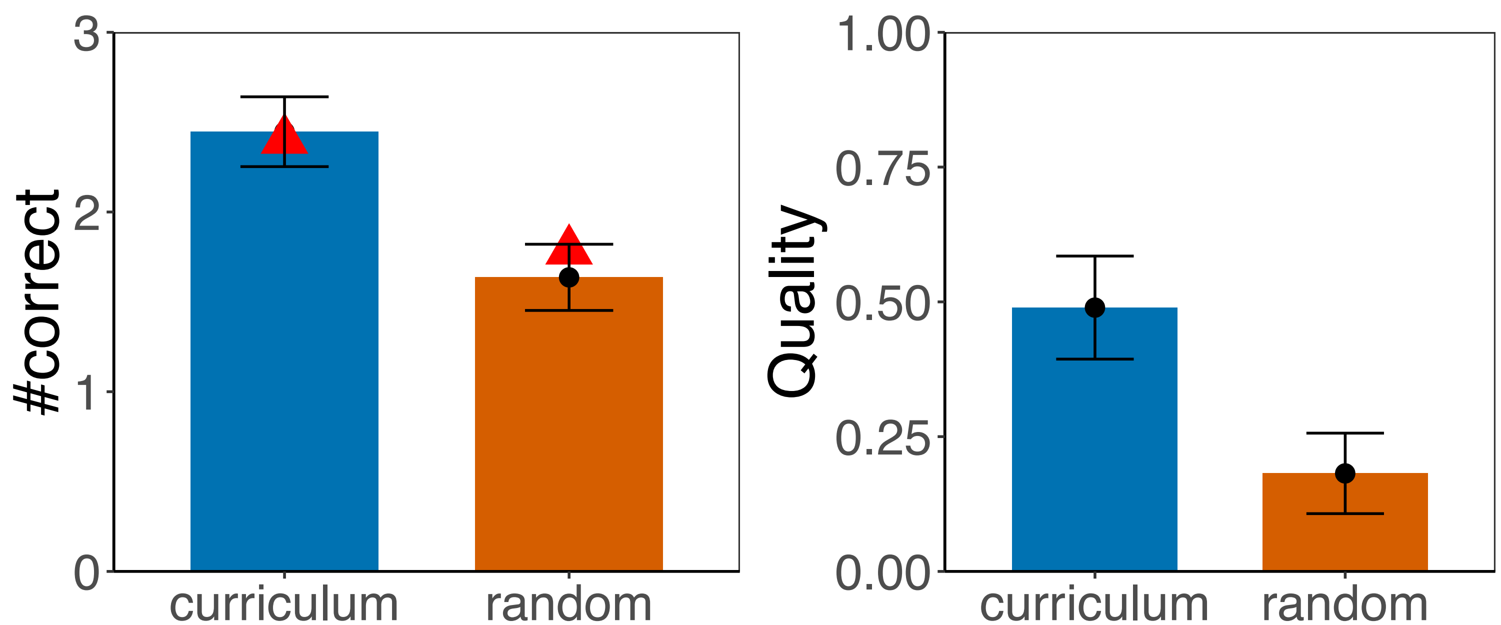
Experiment 2



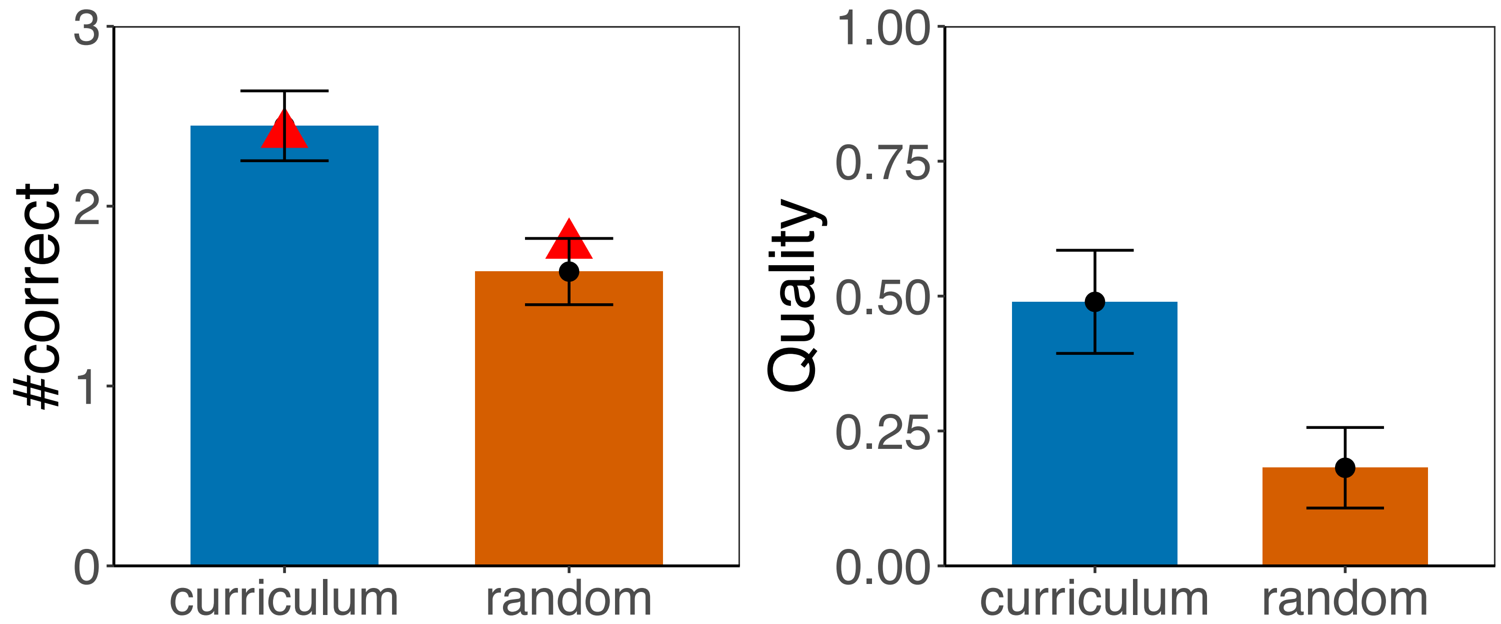
Experiment 2

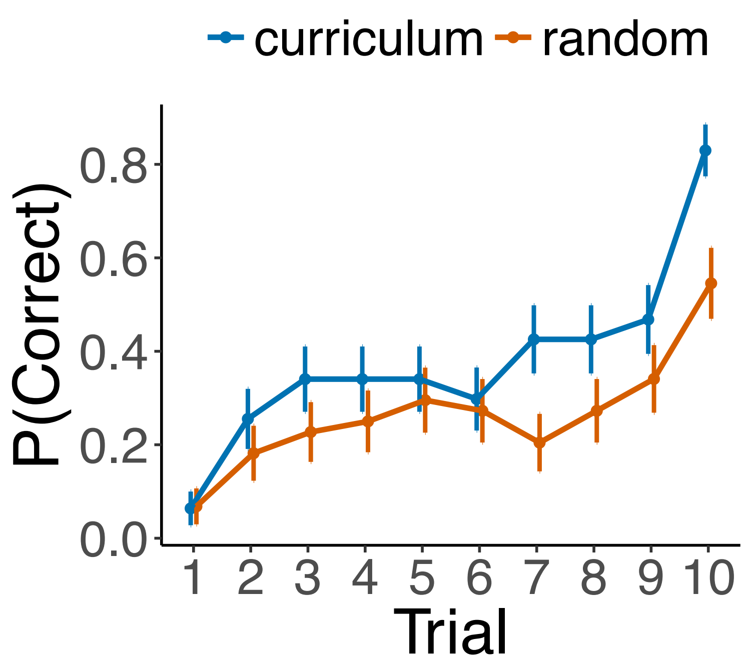


Experiment 2

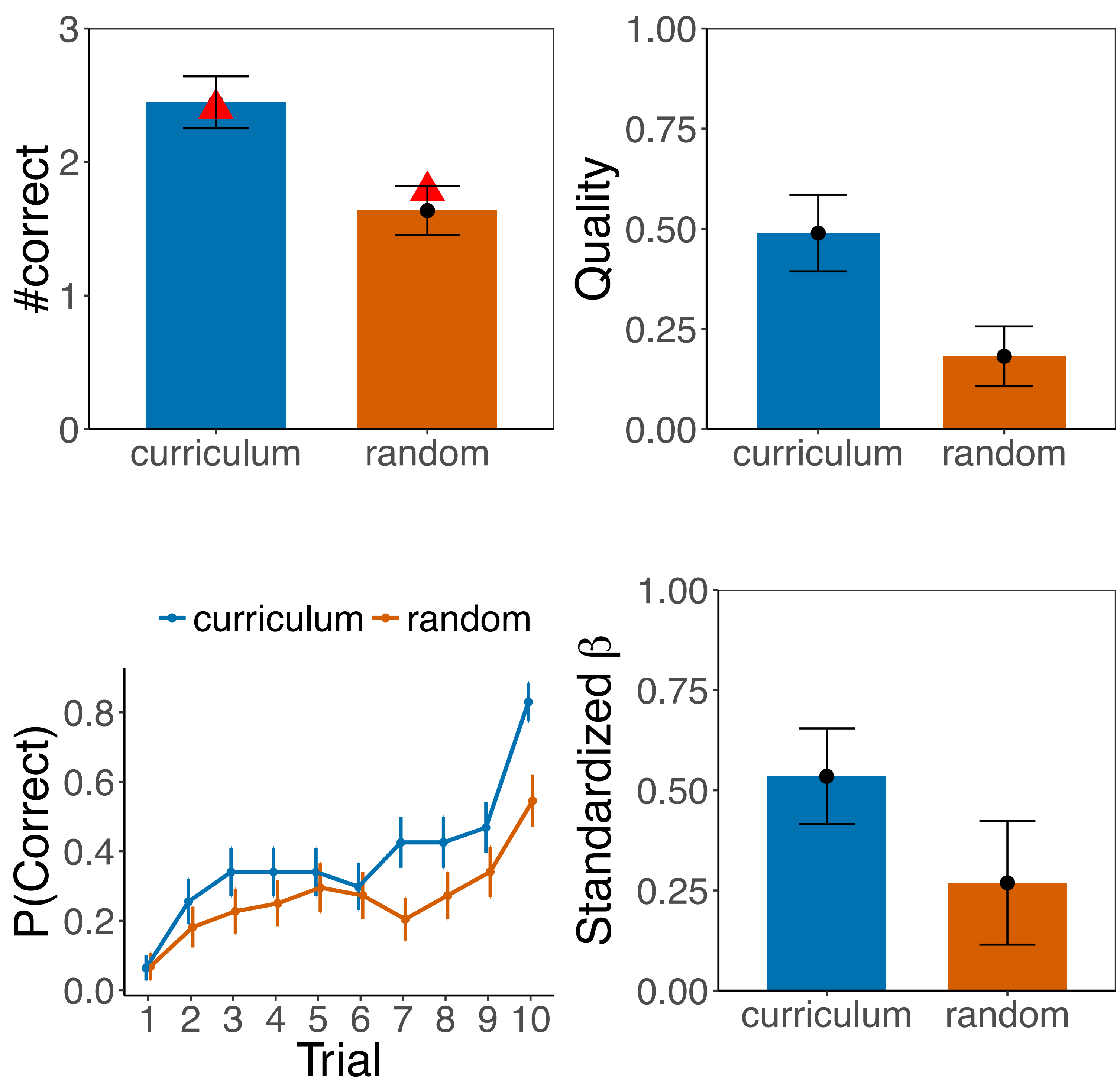


Experiment 2





Experiment 2



This talk

- learning as programming
- bootstrapping the LOT with term rewriting
- toward a model of conceptual change

This talk

- learning as programming
- bootstrapping the LOT with term rewriting
- toward a model of conceptual change

Thank you!