Learning list functions through program induction

Joshua Rule
Learning as Program Induction Workshop
CogSci 2018, 25 July 2018
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
Navigate complex environments
Recognize objects, reason cross-modally
Tie shoes, make bed, set table
Introspect on beliefs and desires
whisper, shout, sing, joke

Build towers, sandcastles, & Lego cars
Use light switches, door knobs, & smartphones
Talk about dinosaurs, trucks, and fairy tales
Play with others, share, determine ownership
Walk, run, skip, dance, somersault
Use natural language
The Puzzle of Learning & Cognitive Development

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

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?

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)
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

\[ \text{LOT} + \text{bootstrapping} = \text{LOT'} \]
A nagging problem

\[ \text{LOT} + \text{bootstrapping} = \text{LOT'}. \]

modeled as

\[ \text{Prog. Lang.} & \text{Library} + \text{stochastic search} = \text{Prog. Lang.} & \text{Library'}. \]
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)
six-hundred-forty-seven-thousand-nine-hundred-sixteen
Kinship is a great space for studying conceptual change

<table>
<thead>
<tr>
<th>Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definite Gender</td>
<td>boy, girl, man, woman</td>
</tr>
<tr>
<td>Generic Gender</td>
<td>male, female</td>
</tr>
<tr>
<td>Definite Nuclear</td>
<td>brother, sister, mother, father, husband, wife, son, daughter</td>
</tr>
<tr>
<td>Generic Nuclear</td>
<td>sibling, spouse, parent, child</td>
</tr>
<tr>
<td>Definite Extended</td>
<td>aunt, uncle, nephew, niece, grandmother, grandfather, granddaughter,</td>
</tr>
<tr>
<td></td>
<td>grandson, grandnephew, grandniece</td>
</tr>
<tr>
<td>Generic Extended</td>
<td>grandparent, grandchild, cousin</td>
</tr>
<tr>
<td>Structurally Recursive</td>
<td>great-aunt, great-uncle, great-grandfather, great-grandmother, great-</td>
</tr>
<tr>
<td></td>
<td>grandparent, great-granddaughter, great-grandson, great-grandchild, great-</td>
</tr>
<tr>
<td></td>
<td>great-, great-great-, ...</td>
</tr>
<tr>
<td>Linearly Recursive</td>
<td>ancestor, descendant</td>
</tr>
<tr>
<td>Nonlinearly Recursive</td>
<td>relative, blood relative, in-law, $m^{th}$ cousin $n^{th}$ removed, step-relations</td>
</tr>
</tbody>
</table>
Nothing to suggest the model could not handle it with the was included in the grammar as a potential set. The context were potential sets. Additionally, the speaker X complement, and primitives specific to the kinship domain, theoretical primitives, union, intersection, set-difference and our assumed semantic primitives and entities in the context well any hypothesized representation explains the observed representations and a size principle likelihood specifying how & Goodman, 2013, 2012): a simplicity prior over semantic Griffiths, 2008; Ullman et al., 2012; Piantadosi, Tenenbaum, particular parent (e.g. “Brandy”). The learner must aggregate information across us-occur with only one particular model is cross-situational because any instance of implicitly define this set. For instance, a word like word; however, the model also allows logical hypotheses that hypotheses explicitly “memorize” the set of referents for each

-Christopher = Penelope
-Andew = Christine
-Margaret = Arthur
-Victoria = James
-Jennifer = Charles
-Colin
-Charlotte

Christiana violenta

<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Christopher</td>
<td>male</td>
<td>Father of Penelope</td>
</tr>
<tr>
<td>Andrew</td>
<td>male</td>
<td>Father of Christine</td>
</tr>
<tr>
<td>Margaret</td>
<td>female</td>
<td>Mother of Arthur</td>
</tr>
<tr>
<td>Victoria</td>
<td>female</td>
<td>Mother of James</td>
</tr>
<tr>
<td>Jennifer</td>
<td>female</td>
<td>Mother of Charles</td>
</tr>
<tr>
<td>Colin</td>
<td>male</td>
<td>Son of Christopher</td>
</tr>
<tr>
<td>Charlotte</td>
<td>female</td>
<td>Daughter of Andrew</td>
</tr>
<tr>
<td>Luna</td>
<td>female</td>
<td>Wife of James</td>
</tr>
<tr>
<td>Lily</td>
<td>female</td>
<td>Wife of Andrew</td>
</tr>
<tr>
<td>Fred</td>
<td>male</td>
<td>Father of Anne</td>
</tr>
<tr>
<td>Anne</td>
<td>female</td>
<td>Wife of Fred</td>
</tr>
<tr>
<td>Gandalf</td>
<td>male</td>
<td>Father of Galadriel</td>
</tr>
<tr>
<td>Galadriel</td>
<td>female</td>
<td>Wife of Gandalf</td>
</tr>
<tr>
<td>Aragorn</td>
<td>male</td>
<td>Father of Arwen</td>
</tr>
<tr>
<td>Arwen</td>
<td>female</td>
<td>Wife of Aragorn</td>
</tr>
<tr>
<td>Joey</td>
<td>male</td>
<td>Son of James</td>
</tr>
<tr>
<td>Mellissa</td>
<td>female</td>
<td>Daughter of James</td>
</tr>
<tr>
<td>Salem</td>
<td>male</td>
<td>Son of Luna</td>
</tr>
<tr>
<td>Zelda</td>
<td>female</td>
<td>Daughter of Luna</td>
</tr>
<tr>
<td>Sabrina</td>
<td>female</td>
<td>Daughter of Fred</td>
</tr>
<tr>
<td>Frodo</td>
<td>male</td>
<td>Son of Sabrina</td>
</tr>
<tr>
<td>Merry</td>
<td>male</td>
<td>Son of Aragorn</td>
</tr>
<tr>
<td>Brandy</td>
<td>female</td>
<td>Daughter of Aragorn</td>
</tr>
<tr>
<td>Han</td>
<td>male</td>
<td>Son of Gandalf</td>
</tr>
<tr>
<td>Leia</td>
<td>female</td>
<td>Daughter of Gandalf</td>
</tr>
<tr>
<td>Fabio</td>
<td>male</td>
<td>Son of Joey</td>
</tr>
<tr>
<td>Clarice</td>
<td>female</td>
<td>Daughter of Mellissa</td>
</tr>
<tr>
<td>Hilda</td>
<td>female</td>
<td>Daughter of Salem</td>
</tr>
<tr>
<td>Amanda</td>
<td>female</td>
<td>Daughter of Zelda</td>
</tr>
<tr>
<td>Katniss</td>
<td>female</td>
<td>Daughter of Sabrina</td>
</tr>
<tr>
<td>Peeta</td>
<td>male</td>
<td>Son of Katniss</td>
</tr>
<tr>
<td>Prue</td>
<td>male</td>
<td>Son of Merry</td>
</tr>
<tr>
<td>Rose</td>
<td>female</td>
<td>Daughter of Frodo</td>
</tr>
<tr>
<td>Sam</td>
<td>male</td>
<td>Son of Aragorn</td>
</tr>
<tr>
<td>Luke</td>
<td>male</td>
<td>Son of Gandalf</td>
</tr>
<tr>
<td>Padme</td>
<td>female</td>
<td>Daughter of Gandalf</td>
</tr>
</tbody>
</table>

(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)

\[
\begin{align*}
\text{and}(\text{male}(x), \text{spouse}(x, y)) & \rightarrow \text{husband}(x, y) \\
\text{and}(\text{female}(x), \text{spouse}(x, y)) & \rightarrow \text{wife}(y, y) \\
\text{and}(\text{female}(x), \text{sibling}(x, y)) & \rightarrow \text{sister}(x, y) \\
\text{and}(\text{male}(x), \text{sibling}(x, y)) & \rightarrow \text{brother}(x, y) \\
\text{and}(\text{male}(x), \text{parent}(x, y)) & \rightarrow \text{father}(x, y) \\
\text{and}(\text{female}(x), \text{parent}(x, y)) & \rightarrow \text{mother}(x, y) \\
\text{and}(\text{male}(x), \text{parent}(y, x)) & \rightarrow \text{son}(x, y) \\
\text{and}(\text{female}(x), \text{parent}(y, x)) & \rightarrow \text{daughter}(x, y) \\
\text{and}(\text{parent}(z, y), \text{parent}(z, x)) & \rightarrow \text{sibling}(x, y) \\
\end{align*}
\]

\[
\begin{align*}
\text{parent}(x, y) & \rightarrow \text{ancestor}(x, y) \\
\text{and}(\text{parent}(x, y), \text{ancestor}(y, z)) & \rightarrow \text{ancestor}(x, y) \\
\text{and}(\text{ancestor}(x, y), \text{ancestor}(x, z)) & \rightarrow \text{blood_relative}(y, z)
\end{align*}
\]
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 $\Sigma_{i-1}$ and all rules involving $s$ from $R_{i-1}$
• add a symbol $s$ to $\Sigma_i$
• 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)

\[ \text{LOT} + \text{bootstrapping} = \text{LOT'} \]

modeled as

\[ \text{TRS} + \text{stochastic search} = \text{TRS'} \]
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

1Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology
2Department of Psychology, Harvard University
3Department of Brain and Cognitive Sciences, University of Rochester

⇤Contributed equally

Abstract

Humans create complex systems of interrelated concepts like mathematics and natural language. Previous work suggests conceptual learning as learning program-like structures from observations, also known as program induction (Lake, Salakhutdinov, & Tenenbaum, 2015). It seems to do so by compositionally recombining smaller parts (e.g. Lake, Salakhutdinov, & Tenenbaum, 2015). Program induction algorithms have been used to model unsupervised learning and sequence learning (Ellis, Dechter, & Tenenbaum, 2015; Romano, Salles, Amalric, Dehaene, Sigman, & Figueria, 2017), to support one-shot inferences (Lake et al., 2015), and to investigate the primitives of thought (Piantadosi et al., 2016).

Existing models of concept learning as program induction have had less success at explaining larger-scale aspects of human learning and conceptual development. These approaches typically learn by stochastically searching through a (possibly infinite) space of possible programs to find good candidates. To help constrain this search, they usually make two limiting assumptions. First, they focus on learning one concept at a time. Humans, however, do not learn this way. We instead learn multiple concepts as interrelated systems. Second, they focus on learning one concept at a time. Humans, however, do not learn this way. We instead learn multiple concepts as interrelated systems.

To help constrain this search, they usually make two limiting assumptions. First, they focus on learning one concept at a time. Humans, however, do not learn this way. We instead learn multiple concepts as interrelated systems. Second, they focus on learning one concept at a time. Humans, however, do not learn this way. We instead learn multiple concepts as interrelated systems.

The first contribution is to introduce and use a fixed semantics, often based on combinatory logic (CL; Dechter et al., 2013; Piantadosi, 2017), λ-calculus (LC; Piantadosi, Tenenbaum, & Goodman, 2012), or first-order logic (FO; Green, Feldman, & Tenenbaum, 2016). The second contribution is to introduce Term Rewriting Systems (TRSs) as a model for conceptual representations. TRSs, like CL and LC, were originally developed as an abstract model of computation. Two features of TRSs make them particularly suitable for concept learning: 1) unlike CL or LC, the set of primitives can be easily revised; and 2) the meaning of concepts is entirely determined by a set of re-usable rewrite rules describing how terms execute over time.

The third contribution is to introduce the idea of using a meta-language to guide learning. We propose a computational model of concept learning that represents not merely different definitions of a concept within a system, but changes to a program-like structure. This allows a single, universal Language of Thought (LOT; Dechter et al., 2013; Piantadosi, 2017), λ-calculus (LC; Piantadosi, Tenenbaum, & Goodman, 2012), or first-order logic (FO; Green, Feldman, & Tenenbaum, 2016).

The second contribution is to introduce Term Rewriting Systems (TRSs) as a model for conceptual representations. TRSs, like CL and LC, were originally developed as an abstract model of computation. Two features of TRSs make them particularly suitable for concept learning: 1) unlike CL or LC, the set of primitives can be easily revised; and 2) the meaning of concepts is entirely determined by a set of reversible rewrite rules describing how terms execute over time.

The third contribution is to introduce the idea of using a meta-language to guide learning. We propose a computational model of concept learning that represents not merely different definitions of a concept within a system, but changes to a program-like structure. This allows a single, universal Language of Thought (LOT; Dechter et al., 2013; Piantadosi, 2017), λ-calculus (LC; Piantadosi, Tenenbaum, & Goodman, 2012), or first-order logic (FO; Green, Feldman, & Tenenbaum, 2016).
Martha’s Magical Machines

(cf. Piantadosi, Tenenbaum, Goodman, 2016)
Martha’s Magical Machines
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

4 7 → 7 4
Martha’s Magical Machines

4 7 → 7 4
2 → 2
9 6 5 → 5 6 9
Stochastic search over TRSs

```python
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/conference)
    return hs
```
## Model Primitives

<table>
<thead>
<tr>
<th>Name &amp; Input/Output Pair</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1, 2</td>
<td>constant natural numbers</td>
</tr>
<tr>
<td>[]</td>
<td>the empty list</td>
</tr>
<tr>
<td>succ(0)</td>
<td>the successor of x</td>
</tr>
<tr>
<td>cons(1, [2, 3]) = [1, 2, 3]</td>
<td>prepend x to y</td>
</tr>
<tr>
<td>sum([1, 2, 3]) = [6]</td>
<td>sum x</td>
</tr>
<tr>
<td>add(3, [1, 2, 3]) = [4, 5, 6]</td>
<td>add x to the elements of y</td>
</tr>
<tr>
<td>insert(4, [3, 5]) = [3, 4, 5]</td>
<td>insert x into y in sorted order</td>
</tr>
<tr>
<td>remove(1, [6, 1, 4]) = [6, 4]</td>
<td>remove every x in y</td>
</tr>
<tr>
<td>count(7, [7, 1, 7] = [2])</td>
<td>count every x in y</td>
</tr>
<tr>
<td>even(5) = false</td>
<td>true if x is even else false</td>
</tr>
<tr>
<td>greater(8, 2) = true</td>
<td>true if x &gt; y else false</td>
</tr>
<tr>
<td>if(true, [7], [2, 5]) = [7]</td>
<td>if x then y else z</td>
</tr>
<tr>
<td>nth(3, [9, 5, 8]) = [8]</td>
<td>the $x^{th}$ element of y</td>
</tr>
</tbody>
</table>
the log likelihood of an input/output pair is the log probability of a TRS is the sum over all rules of the number of focus away from program induction and toward one, describing a conceptual system for lists. This switch in in each simulation was entirely determined by learned rewrite operators, and thus the set of primitives, was fixed, and some we fixed the set of operators and provided rules fixing the dosi et al., 2012; Ullman et al., 2012). We specifically con-model learning as stochastic search (Lake et al., 2015; Pianta-

Learning Concepts with Stochastic Search

Relating terms with rewrite rules gives them definite mean-

tution to the RHS. Consider these rules for unary addition:

<Model Primitives>

\[ h_0 = (\Sigma^*, R_0) \]

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

\*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

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

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

Example

# 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([2,1,2,1]) = [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

![Bar chart showing results for various categories: P(correct) and Quality of descriptions. The categories include Constant, Total, Increment, Head, Sort, Length, Deduplicate, Cumsum, Filter uneven, Index from head, Head or sum of tail, and Count 3. Each category has bars indicating the values with error bars.]
Participants learned the concepts, with additional trials on average to complete. Participants were randomly assigned to one of two learner groups: random learners (total of 90 participants) or curriculum learners (total of 91 participants) from Amazon Mechanical Turk and paid a flat fee of $1. The task took 12 minutes on average to complete. Participants were males, mean age=34.51, SD=10.57.

Participants and Design

Participants might benefit by reserving harder problems for the later rounds of the experiment. We examine how curriculum design affects performance in Experiment 2. In particular, we compare two learners: random learners and curriculum learners. The curriculum is designed to make difficult concepts more learnable with a curriculum from which learners are taught. Larry and his colleagues (2020) trained a model and collected performance data on a set of human learners. These performance scores (human) and scores from the model (model) are shown in Figure 1. The x-axis shows different concepts, and the y-axis shows the average probability of a successful prediction. The error bars represent the standard error of the mean. Solid curves are human learners, and dashed curves are model learners. Human learners show more graded progress, and for the most difficult concepts (e.g., count3), participants only needed 1-2 examples to perform near ceiling. Other concepts (e.g., odd, even) are more difficult. Error bars give the standard error. Solid curves are human learners, and dashed curves are model learners.

We recruited 91 participants (46 males, mean age=34.51, SD=10.57) from Amazon Mechanical Turk and paid a flat fee of $1. The task took 12 minutes on average to complete. Participants were randomly assigned to one of two learner groups: random learners (total of 90 participants) or curriculum learners (total of 91 participants) from Amazon Mechanical Turk and paid a flat fee of $1. The task took 12 minutes on average to complete. Participants were males, mean age=34.51, SD=10.57.

We also analyzed the verbal descriptions provided for each concept. Participants learned the concepts, with additional trials on average to complete. Participants were randomly assigned to one of two learner groups: random learners (total of 90 participants) or curriculum learners (total of 91 participants) from Amazon Mechanical Turk and paid a flat fee of $1. The task took 12 minutes on average to complete. Participants were males, mean age=34.51, SD=10.57.

Participants and Design

Participants might benefit by reserving harder problems for the later rounds of the experiment. We examine how curriculum design affects performance in Experiment 2. In particular, we compare two learners: random learners and curriculum learners. The curriculum is designed to make difficult concepts more learnable with a curriculum from which learners are taught. Larry and his colleagues (2020) trained a model and collected performance data on a set of human learners. These performance scores (human) and scores from the model (model) are shown in Figure 1. The x-axis shows different concepts, and the y-axis shows the average probability of a successful prediction. The error bars represent the standard error of the mean. Solid curves are human learners, and dashed curves are model learners. Human learners show more graded progress, and for the most difficult concepts (e.g., count3), participants only needed 1-2 examples to perform near ceiling. Other concepts (e.g., odd, even) are more difficult. Error bars give the standard error. Solid curves are human learners, and dashed curves are model learners.

We recruited 91 participants (46 males, mean age=34.51, SD=10.57) from Amazon Mechanical Turk and paid a flat fee of $1. The task took 12 minutes on average to complete. Participants were males, mean age=34.51, SD=10.57.

Participants and Design

Participants might benefit by reserving harder problems for the later rounds of the experiment. We examine how curriculum design affects performance in Experiment 2. In particular, we compare two learners: random learners and curriculum learners. The curriculum is designed to make difficult concepts more learnable with a curriculum from which learners are taught. Larry and his colleagues (2020) trained a model and collected performance data on a set of human learners. These performance scores (human) and scores from the model (model) are shown in Figure 1. The x-axis shows different concepts, and the y-axis shows the average probability of a successful prediction. The error bars represent the standard error of the mean. Solid curves are human learners, and dashed curves are model learners. Human learners show more graded progress, and for the most difficult concepts (e.g., count3), participants only needed 1-2 examples to perform near ceiling. Other concepts (e.g., odd, even) are more difficult. Error bars give the standard error. Solid curves are human learners, and dashed curves are model learners.

We recruited 91 participants (46 males, mean age=34.51, SD=10.57) from Amazon Mechanical Turk and paid a flat fee of $1. The task took 12 minutes on average to complete. Participants were males, mean age=34.51, SD=10.57.
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

```haskell
# 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_;

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

# 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_));

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

# 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_));

# head-x tail: return the 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_));
Six participants correctly described the concept; four were Fig. 4b), although both sets of descriptions scored weakly.

Curriculum learners generally wrote better descriptions of the main concepts (matched in complexity and excluding the curriculum) before attempting the same final target concept. Error bars represent the standard error.

We first analyzed performance during the last 5 trials. Curricula learners (#) scored significantly with performance on the target round (#)(#). Combined with an unboundedly large set of unique variables, the nature of unary addition might be the following:

\[
\begin{align*}
\text{succ}((x_0 \text{ head} x_0 \text{ tail} x_0)) &= x_0 \\
\text{how}((x_1 \text{ count} x_0)) &= x_1 \\
\text{count}(x_0) &= x_0 \\
\text{count}(x_1 \text{ head} x_2 \text{ tail} x_0) &= x_1 \text{ count} x_0 \text{ head} x_2 \text{ tail} x_0 \\
\text{count}(x_1 \text{ head} x_2 \text{ tail} x_1) &= x_1 \text{ count} x_0 \text{ head} x_2 \text{ tail} x_1 \\
\text{count}(x_1 \text{ head} x_2 \text{ tail} x_1) &= x_1 \text{ count} x_0 \text{ head} x_2 \text{ tail} x_1 \\
\text{count}(x_1 \text{ head} x_2 \text{ tail} x_1) &= x_1 \text{ count} x_0 \text{ head} x_2 \text{ tail} x_1 \\
\text{count}(x_1 \text{ head} x_2 \text{ tail} x_1) &= x_1 \text{ count} x_0 \text{ head} x_2 \text{ tail} x_1 \\
\text{count}(x_1 \text{ head} x_2 \text{ tail} x_1) &= x_1 \text{ count} x_0 \text{ head} x_2 \text{ tail} x_1 \\
\end{align*}
\]

Listing 2: Rewrite rules for the concepts in Experiment 2.

Training code is available at: https://git.io/vNbK6
Six participants correctly described the concept; four were Fig. 4b), although both sets of descriptions scored weakly.

The quality of participant descriptions for the final machine, curriculum learners performed significantly better than random learners (i.e. the target concept) (Fig. 4a). Curdictions over trials).

Figure 4: Comparison of the quality of participant descriptions for the final machine, between curriculum learners and random learners. The bar graph shows the average number of correct descriptions for curriculum learners (blue) and random learners (orange), with error bars indicating the standard deviation.

Curriculum learners scored significantly higher than random learners, with a mean of 2.5 correct descriptions compared to 1.8 for random learners. The difference is statistically significant (t-test, p < 0.05).

Title: Experiment 2

Graph: Experiment 2

Y-axis: Number of correct descriptions
X-axis: Condition (Curriculum, Random)
Bar heights: Curriculum learners vs. Random learners
Error bars: Standard deviation

Legend: Curriculum, Random

Analysis:
- Curriculum learners performed significantly better than random learners.
- The difference in performance is statistically significant.
- This suggests that a structured curriculum can enhance learning compared to random exposure to tasks.
Six participants correctly described the concept; four were Fig. 4b), although both sets of descriptions scored weakly.

Curriculum learners (using the same coding scheme as in Experiment 1. Curriculum learners benefited by learning earlier concepts. Curricula of the final round (i.e. the target concept) (Fig. 4a). Curricula learners (Listing 2) before attempting a fourth and final target concept, curriculum learners saw three fixed curriculum concepts. Magical Machines.

We first analyzed performance during the last 5 trials. Learning curves (i.e. mean proportion of correct pre-

Quality of participant descriptions for the final machine, learners than random learners (Fig. 4c) suggest that curriculum learners scored more strongly by learning earlier concepts. More curriculum learners scored 1 or 2 than random learners used to construct the target concept. For the curriculum learners, performance on the first three rounds correlated significantly with past performance, because the curriculum concepts can be associated with a type to help constrain search. For example, we could define a binary operator for addition, associated with a type like programming languages boil down to trees of symbols, which are called terms, and rules for how those terms compute. TRSs formalize the idea that symbolic forms of computation are represented as trees and rules for how those trees compute. Given a set of trees, the problem of proving properties about a term rewriting system is solvable using techniques from formal language theory. We introduce them as a model of conceptual representations and model learning as a search through the space of possible languages for those well-suited to the problems at hand. Term Rewriting Systems (TRS), developed and studied in the literature, are a natural model for representing and manipulating symbolic structures. We consider TRSs formalized as an abstract model of computation, provide such a meta-

Model code is available at: https://git.io/vNbK6
Six participants correctly described the concept; four were more accurate than random learners (tally learners generally wrote better descriptions of the main concepts). The performance of curriculum learners was better than random learners during the last round (Fig. 4a). Curriculum learners often had better predictions during the last 5 trials.

**Figure 4:** Experiment 2, by condition.

We first analyzed performance during the last 5 trials. Error bars represent the standard error. The participants played as in Experiment 1. However, whereas curriculum learners saw three fixed curriculum concepts before attempting the same final target concept (Magical Machines before attempting the same final target concept), the random learners interacted with three randomly chosen concepts (Magical Machines before attempting the same final target concept). The curriculum learners performed better than the random learners in the final round. Error bars represent the standard error. The difference in performance between the curriculum learners and the random learners was significant (p < 0.01).

**Training Language and Procedure**: Participants played a game where they had to learn concepts through trial and error. The concepts were defined by a probabilistic grammar over TRS rules. We introduced them as a model of conceptual representation, associating with a type to help constrain search. For example, we could define a binary operator for addition, represented as `Nat -> Nat -> Nat`. Variables are recursively subterms, where an operator is associated with a type to help constrain search. The successor function, which is the next state of a concept, is valid only if the concept is valid. For example, `(x) = x; xs; cons; return; count-head-in-tail; xs cons` is a valid operator.

**Listing 2:** Rewrite rules for the concepts in Experiment 2.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Right-hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(x) = x; xs cons</code></td>
<td><code>count-head-in-tail</code></td>
</tr>
<tr>
<td><code>xs cons</code></td>
<td><code>count3</code></td>
</tr>
<tr>
<td><code>count3</code></td>
<td><code>count-head-in-tail</code></td>
</tr>
<tr>
<td><code>count-head-in-tail</code></td>
<td><code>xs cons</code></td>
</tr>
<tr>
<td><code>xs cons</code></td>
<td><code>return</code></td>
</tr>
<tr>
<td><code>return</code></td>
<td><code>count3</code></td>
</tr>
<tr>
<td><code>count3</code></td>
<td><code>count-head-in-tail</code></td>
</tr>
<tr>
<td><code>count-head-in-tail</code></td>
<td><code>xs cons</code></td>
</tr>
</tbody>
</table>

**Standardized Trial Number**

In that case, the following are valid rewrite rules: `xs cons`, `count3`, `count-head-in-tail`, `xs cons`, `return`, `count3`, `count-head-in-tail`, `xs cons`, `return`. A signature for a simple term rewriting system `sNat Nat -> Nat` is the arity of the operator. Variables are recursively subterms, where an operator is associated with a type to help constrain search. For example, we could define a binary operator for addition, represented as `Nat -> Nat -> Nat`. Variables are recursively subterms, where an operator is associated with a type to help constrain search. The successor function, which is the next state of a concept, is valid only if the concept is valid. For example, `(x) = x; xs cons; return; count-head-in-tail; xs cons` is a valid operator.
Six participants correctly described the concept; four were using the same coding scheme as in Experiment 1. Curriculum learners (als of the final round (i.e. the target concept) (Fig. 4a). Correct predictions in the first 3 rounds onto total correct predictions in predictions over trials).

Curriculum learners saw three fixed curriculum concepts (matched in complexity and excluding the curriculum) before attempting the same final target concept. However, whereas random learners interacted with three randomly chosen concepts (Listing 2) before attempting a fourth and final target concept, curriculum learners benefited by learning earlier concepts. Performance during the last 5 trials of Experiment 2 are shown in Fig. 4b. The number of correct predictions in the last five trials increased over time for both curriculum and random learners. Some participants were completely incorrect; 38 were random learners. 66 participants were completely correct; 15 were curriculum learners. In particular during later trials. Finally, we analyzed how the past performance, because the curriculum concepts can be used to construct the target concept. For the curriculum learners, performance on the first three rounds correlated significantly with performance on the target round (Fig. 4c) suggest that curriculum learners benefited by learning earlier concepts.

Training Learning curves (i.e. mean proportion of correct predictions) over trials).

We first analyzed performance during the last 5 trials. Participant learning curves during the last 5 trials are shown in Fig. 4d. The number of correct predictions in the last five trials increased over time for both curriculum and random learners. Some participants were completely incorrect; 38 were random learners. 66 participants were completely correct; 15 were curriculum learners. In particular during later trials. Finally, we analyzed how the past performance, because the curriculum concepts can be used to construct the target concept. For the curriculum learners, performance on the first three rounds correlated significantly with performance on the target round (Fig. 4c) suggest that curriculum learners benefited by learning earlier concepts.
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!