Dreamcoder: Bootstrapping Inductive Program Synthesis With Wake-Sleep Library Learning

Kevin Ellis
2021

Synthesis of Models and Systems
1. Represent knowledge as programs: as symbolic code
1. Represent knowledge as programs: as symbolic code

2. Learning = adding to that body of knowledge = making new programs = program synthesis
Why program induction?

strong generalization
+ data efficiency
interpretability
universal expressivity

\( f(x) = (x-1)^2 - 0.5 \) vs 3
Why program induction?

strong generalization
+ data efficiency

\[ f(x) = (x-1)^2 - 0.5 \]
Why program induction?

strong generalization
+data efficiency

\[ f(x) = (x-1)^2 - 0.5 \]
Why program induction?

- strong generalization
- data efficiency

\[ f(x) = (x-1)^2 - 0.5 \]

interpretability

universal expressivity

VS
**EXAMPLE 3** (Directory Name Extraction). Consider the following example taken from an excel online help forum.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company\Code\index.html</td>
<td>Company\Code\</td>
</tr>
<tr>
<td>Company\Docs\Spec\specs.doc</td>
<td>Company\Docs\Spec\</td>
</tr>
</tbody>
</table>

**String Program:**

\[ \text{SubStr}(v_1, CPos(0), Pos(\text{SlashTok}, \epsilon, -1)) \]
EXAMPLE 3 (Directory Name Extraction). Consider the following example taken from an excel online help forum.

<table>
<thead>
<tr>
<th>Input $v_1$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company\Code\index.html</td>
<td>Company\Code\</td>
</tr>
<tr>
<td>Company\Docs\Spec\specs.doc</td>
<td>Company\Docs\Spec\</td>
</tr>
</tbody>
</table>

String Program:

$$\text{SubStr}(v_1, \text{CPos}(0), \text{Pos}(\text{SlashTok, }ε, -1))$$

Szalinski (Nandi 2020)

(a) CAD model of ship’s wheel

(b) Caddy program
**FlashFill (Gulwani 2012)**

EXAMPLE 3 (Directory Name Extraction). Consider the following example taken from an excel online help forum.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company\Code\index.html</td>
<td>Company\Code\</td>
</tr>
<tr>
<td>Company\Docs\Spec\specs.doc</td>
<td>Company\Docs\Spec\</td>
</tr>
</tbody>
</table>

String Program:
\[\text{SubStr}(v_1, \text{CPos}(0), \text{Pos}(	ext{SlashTok}, \epsilon, -1))\]

**Szalinski (Nandi 2020)**

---

\[
\begin{align*}
\text{String expr } P & := \text{Switch}((b_1, e_1), \ldots, (b_n, e_n)) \\
\text{Bool b} & := d_1 \lor \cdots \lor d_n \\
\text{Conjunct d} & := \pi_1 \land \cdots \land \pi_n \\
\text{Predicate } \pi & := \text{Match}(v_i, r, k) \lor \neg \text{Match}(v_i, r, k) \\
\text{Trace expr } e & := \text{Concatenate}(f_1, \ldots, f_n) \\
\text{Atomic expr } f & := \text{SubStr}(v_i, p_1, p_2) \\
& \quad | \quad \text{ConstStr}(s) \\
& \quad | \quad \text{Loop}(\lambda w : e) \\
\text{Position } p & := \text{CPos}(k) \lor \text{Pos}(r_1, r_2, c) \\
\text{Integer expr } c & := k | k_1 w + k_2 \\
\text{Regular Expression } r & := \text{TokenSeq}(T_1, \ldots, T_m) \\
\text{Token } T & := C + | \neg C + \\
& \quad | \quad \text{SpecialToken}
\end{align*}
\]

\[
\begin{align*}
\text{op} & := + | - | \times | / \\
\text{num} & := \mathbb{R} | \langle \text{var} \rangle | \langle \text{num} \rangle \langle \text{op} \rangle \langle \text{num} \rangle \\
\text{vec2} & := [\langle \text{num} \rangle, \langle \text{num} \rangle] \\
\text{vec3} & := [(\langle \text{num} \rangle, \langle \text{num} \rangle, \langle \text{num} \rangle)] \\
\text{affine} & := \text{Translate} \mid \text{Rotate} \mid \text{Scale} \mid \text{TranslateSpherical} \\
\text{binop} & := \text{Union} \mid \text{Difference} \mid \text{Intersection} \\
\text{cad} & := (\text{Cuboid} \langle \text{vec3} \rangle) \mid (\text{Sphere} \langle \text{num} \rangle) \\
& \quad | (\text{Cylinder} \langle \text{vec2} \rangle) \mid (\text{HexPrism} \langle \text{vec2} \rangle) \mid \ldots \\
& \quad | (\langle \text{affine} \rangle \langle \text{vec3} \rangle \langle \text{cad} \rangle) \\
& \quad | (\langle \text{binop} \rangle \langle \text{cad} \rangle \langle \text{cad} \rangle) \\
& \quad | (\text{Fold} \langle \text{binop} \rangle \langle \text{cad-list} \rangle) \\
\text{cad-list} & := (\text{List} \langle \text{cad} \rangle)+ \\
& \quad | (\text{Concat} \langle \text{cad-list} \rangle+) \\
& \quad | (\text{Tabulate} \langle (\text{var}) \varepsilon \rangle+ \langle \text{cad} \rangle) \\
& \quad | (\text{Map2} \langle \text{affine} \rangle \langle \text{vec3-list} \rangle \langle \text{cad-list} \rangle) \\
\text{vec3-list} & := (\text{List} \langle \text{vec3} \rangle)+ \\
& \quad | (\text{Concat} \langle \text{vec3-list} \rangle+) \\
& \quad | (\text{Tabulate} \langle (\text{var}) \varepsilon \rangle+ \langle \text{vec3} \rangle) 
\end{align*}
\]
Visual programs

Mao*, Zhang*, et al 2019

Ellis*, Nye*, Pu*, Sosa*, et al 2019

Young et al 2019

Tian et al 2019
Where does this language come from?

Mao*, Zhang*, et al 2019

Ellis*, Nye*, Pu*, Sosa*, et al 2019

Young et al 2019

Tian et al 2019
Program Induction and learning to learn

- learning a DSL
- learning to synthesize
- synergy between DSL+learned synthesizer
Goal: acquire domain-specific knowledge needed to induce a class of programs

- Library of abstractions (domain specific language)
- Inference strategy (synthesis algorithm)
Library learning

Sample Problem: sort list

\[
\begin{align*}
[9 & 2 & 7 & 1] & \rightarrow & [1 & 2 & 7 & 9] \\
\end{align*}
\]

...
Library learning

**Initial Primitives**

```
: ...
map
fold
if
cons
> :
```

**Sample Problem: sort list**

```
[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
...
```
Library learning

Initial Primitives
::
map
fold
if
cons
>
::

Sample Problem: sort list

| [9 2 7 1]  | [1 2 7 9] |
| [3 8 9 4 2] | [2 3 4 8 9] |
| [6 2 2 3 8 5] | [2 2 3 5 6 8] |

Library learning

Initial Primitives

 Learned Library of Concepts

Sample Problem: sort list

[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
...

Library learning

Initial Primitives

... map fold if cons > ...

Learned Library of Concepts

concept_4

(λ(L P)(fold L nil (λ(z u) (if (P z) (cons z u) u)))))

[filter]

Sample Problem: sort list

[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2]→ [2 3 4 8 9]
[6 2 2 3 8 5]++ [2 2 3 5 6 8]
...

Library learning

Solution rewritten in initial primitives:

\[
\text{(lambda (x) (map (lambda (y) (car (fold (fold x nil (lambda (z u) (if (gt? (+ y 1) (length (fold x nil (lambda (v w) (if (gt? z v) (cons v w) w))))) (cons z u) u))) nil (lambda (a b) (if (nil? (fold (fold x nil (lambda (c d) (if (gt? (+ y 1) (length (fold x nil (lambda (e f) (if (gt? c e) (cons e f) f))))) (cons c d) d))) nil (lambda (g h) (if (gt? g a) (cons g h) h)))) a b) b))) (range (length x))))}
\]

induced sort program found in \(\leq 10\) min. Brute-force search without learned library would take \(\approx 10^{73}\) years.

Library learning

Solution rewritten in initial primitives:

\[
\text{\lambda (x) (map (\lambda (y) (\text{car (fold (fold x nil (\lambda (z u) (\text{if (gt? (+ y 1) (length (fold x nil (\lambda (v w) (\text{if (gt? z v) (cons v w) w))))) (cons z u) u))) nil (\lambda (a b) (\text{if (nil? (fold (fold x nil (\lambda (c d) (\text{if (gt? (+ y 1) (length (fold x nil (\lambda (e f) (\text{if (gt? c e) (\text{cons e f) f))))) (cons c d) d))) nil (\lambda (g h) (\text{if (gt? g a) (\text{cons g h) h))))) (\text{cons a b) b}))))) (range (length x))))}}
\]

induced sort program found in \(\leq 10\) min. Brute-force search without learned library would take \(\approx 10^{73}\) years.

Solution rewritten in initial primitives:

\[
\lambda (x) \ (\text{map}\ (\lambda (y)\ (\text{car}\ (\text{fold}\ (\text{fold}\ x\ \text{nil}\ (\lambda (z\ u)\ (\text{if}\ (\text{gt}\ (+\ y\ 1)\ (\text{length}\ (\text{fold}\ x\ \text{nil}\ (\lambda (e\ f)\ (\text{if}\ (\text{gt}\ c\ e)\ (\text{cons}\ e\ f)\ f))))\ (\text{cons}\ c\ d)\ d)\ \lambda (g\ h)\ (\text{if}\ (\text{gt}\ g\ a)\ (\text{cons}\ g\ h)\ h))))\ (\text{range}\ (\text{length}\ x))))
\]

Induced sort program found in \(\leq 10\) min. Brute-force search without learned library would take \(\approx 10^{73}\) years.

---

**Sample Problem:** sort list

- \([9\ 2\ 7\ 1]\rightarrow\ [1\ 2\ 7\ 9]\)
- \([3\ 8\ 9\ 4\ 2]\rightarrow\ [2\ 3\ 4\ 8\ 9]\)
- \([6\ 2\ 2\ 3\ 8\ 5]\rightarrow\ [2\ 2\ 3\ 5\ 6\ 8]\)
- ...

---

Library learning

Solution rewritten in initial primitives:

\[
\begin{align*}
\lambda (x) & \mapsto (\lambda (y) (\text{car} (\text{fold} (\text{fold} x \, \text{nil} \, (\lambda (z \, u) (\text{if} (P \, z) (\text{cons} \, z \, u) \, u)))))) \\
\text{induced sort program found in } \leq 10 \text{ min. Brute-force search without learned library would take } \approx 10^{73} \text{ years.}
\end{align*}
\]

Library learning

Solution rewritten in initial primitives:

\[
\lambda (x) \left( \text{map} \left( \lambda (y) \left( \text{car} \left( \text{fold} \left( \text{fold} \left( x \ \text{nil} \ \left( \lambda (z \ u) \ (\text{if} \ (P \ z) \ (\text{cons} \ z \ u) \ u)) \right) \right) \right) \right) \right) \end{equation}

induced sort program found in \( \leq 10 \text{min.} \) Brute-force search without learned library would take \( \approx 10^{73} \) years.

Library learning

Library learning

Solution rewritten in initial primitives:

\[
\text{map} \quad \text{fold} \quad \text{if} \quad \text{cons} \quad > \\
\frac{\lambda(P)}{(\lambda(z \ u) \ (\text{if} \ (P \ z) \ \text{cons} \ z \ u \ u)))} \quad \frac{\lambda(L \ P) (\text{fold} \ L \ \text{nil} \ (\lambda(z \ u) \ (\text{if} \ (P \ z) \ \text{cons} \ z \ u \ u)))} \quad \frac{\text{filter}} \quad \frac{\text{maximum}} \quad \frac{\text{nth largest element}} \quad \frac{\text{concept}_4} \quad \frac{\text{concept}_13} \quad \frac{\text{concept}_15}
\]

Induced sort program found in \( \leq 10 \text{ min} \). Brute-force search without learned library would take \( \approx 10^{73} \) years.


Sample Problem: sort list

\[
[9 \ 2 \ 7 \ 1] \rightarrow [1 \ 2 \ 7 \ 9] \\
[3 \ 8 \ 9 \ 4 \ 2] \rightarrow [2 \ 3 \ 4 \ 8 \ 9] \\
[6 \ 2 \ 2 \ 3 \ 8 \ 5] \rightarrow [2 \ 2 \ 3 \ 5 \ 6 \ 8] \\
...
\]

Solution to sort list discovered in learned language:

\[
\begin{align*}
\text{map} \quad & (\lambda \ (n) \\
\text{concept}_15 \quad & L \ (+ \ 1 \ n)) \\
\text{range} \quad & (\text{length} \ L))
\end{align*}
\]
Library learning

Solution rewritten in initial primitives:

\[
(\lambda (x) (\text{map} (\lambda (y) (\text{car} (\text{fold} (\text{fold} x \text{nil} (\lambda (z \ u) (\text{if} (P \ z) (\text{cons} z u) u))))))))
\]

induced sort program found in \(\leq 10\) min. Brute-force search without learned library would take \(\approx 10^{73}\) years.

Solution rewritten in initial primitives:

\[
\text{sort list} = (\lambda (x) (\text{map} (\lambda (y) (\text{car} (\text{fold} (\text{fold} x \text{nil} (\lambda (z u) (\text{if} (\text{gt?} (+ y 1) (\text{length} (\text{fold} x \text{nil} (\lambda (e f) (\text{if} (\text{gt?} c e) (\text{cons} e f) f)))) (\text{cons} c d) d)) \text{nil} (\lambda (g h) (\text{if} (\text{gt?} g a) (\text{cons} g h) h)))) (\text{cons} a b) b)))) (\text{range} (\text{length} x))))
\]

Solution rewritten in initial primitives:

\[
\text{(lambda (x) (map (lambda (y) (car (fold (fold x nil (lambda (z u) (if (gt? (+ y 1) (length (fold x nil (lambda (c d) (if (gt? c e) (cons e f) f)))) (cons c d) d))) nil (lambda (a b) (if (nil? (fold (fold x nil (lambda (c d) (if (gt? (+ y 1) (length (fold x nil (lambda (e f) (if (gt? e g) (cons e f) f)))) (cons e f) d))) nil (lambda (g h) (if (gt? g a) (cons g h) h)))) (cons a b) b)))) (range (length x))))
}\]

induced sort program found in $\leq 10\text{min}$. Brute-force search without learned library would take $\approx 10^{73}$ years

• **Wake:** Solve problems by writing programs
• **Sleep:** Improve library and neural recognition model:
  - **Abstraction sleep:** Improve library
  - **Dream sleep:** Improve neural recognition model

cf. Helmholtz machine, wake/sleep neural network training algorithms
Library learning as Bayesian inference

dark: observed
light: unobserved

[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]
Library learning as Bayesian inference

dark: observed
light: unobserved

[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]
Library learning as Bayesian inference

dark: observed
light: unobserved

[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]
Library learning as Bayesian inference

dark: observed
light: unobserved

[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]
Library learning as **neurally-guided** Bayesian inference

Library learning via program analysis +
new neural inference network for program synthesis +
better program representation (Lisp + polymorphic types [Milner 1978])

Library

\[ f_1(x) = (+ x 1) \]
\[ f_2(z) = (\text{fold cons} \ (\text{cons } z \ \text{nil})) \]

Task

\[ [7 \ 2 \ 3] \rightarrow [4 \ 3 \ 8] \]
\[ [4 \ 3 \ 2] \rightarrow [3 \ 4 \ 5] \]

Recognition model

Programs for task:

\[(\text{map } f_1 \ (\text{fold } f_2 \ \text{nil } x))\]

SLEEP: ABSTRACTION

SLEEP: DREAMING

Wake
Wake

**Library**

\[ f_1(x) = (+ x 1) \]

\[ f_2(z) = (\text{fold cons} \ (\text{cons} z \text{ nil})) \]

**Tasks**

\[
\begin{align*}
[7 & 2 & 3] & \rightarrow [4 & 3 & 8] \\
[4 & 3 & 2] & \rightarrow [3 & 4 & 5]
\end{align*}
\]

**Programs for task:**

\[(\text{map } f_1 \ (\text{fold } f_2 \text{ nil } x))\]

**Sleep: Abstraction**

**Sleep: Dreaming**

- **Fantasies**
  - Library
  - Sample program
- **Replays**
  - Library
  - Sample program

Train recognition model

Loss

Task

Program

Sample

Run
S/l.sceep: Abstraction

Library

\[ f_1(x) = (+ x 1) \]
\[ f_2(z) = (\text{fold cons} \ (\text{cons} z \text{ nil})) \]

………

Task

[7 2 3] → [4 3 8]
[4 3 2] → [3 4 5]

S/l.sceep: Dreaming

Wake

Library

\[ f_1(x) = (+ x 1) \]
\[ f_2(z) = (\text{fold cons} \ (\text{cons} z \text{ nil})) \]

………

Programs for task:

\( (\text{map} \ f_1 \ (\text{fold} \ f_2 \ \text{nil} \ x)) \)

………

SLEEP: ABSTRACTION

progs. for task 1:

\( (+ \ (\text{car} \ z) \ 1) \)

progs. for task 2:

\( (\text{cons} \ (+ 1 \ 1)) \)

Refactoring Algorithm:

version spaces

new Library w/ \( (+ x 1) \):

SLEEP: DREAMING

Fantasies

Replays

Library

progs. for task

Library

\( \text{sample} \)

program

\( \text{sample} \)

program

Train recognition model

program

run

task

Loss
Wake

Library
$f_1(x) = (+ x 1)$
$f_2(z) = (\text{fold cons (cons z nil)})$

Task
$[7 \ 2 \ 3] \rightarrow [4 \ 3 \ 8]$
$[4 \ 3 \ 2] \rightarrow [3 \ 4 \ 5]$

Neurally-Guided Search

Programs for task:
$(\text{map } f_1 \ (\text{fold } f_2 \ \text{nil } x))$

Sleep: Abstraction

progs. for task 1:
$(+ (\text{car } z) 1)$
progs. for task 2:
$(\text{cons } (+ 1 1))$

Refactoring Algorithm: version spaces

new Library w/ $(+ x 1)$:

Sleep: Dreaming

Fantasies
Replays

Library
progs. for task

Train recognition model

Loss

sample program
run task

sample program
Program Induction and learning to learn
learning a DSL
learning to synthesize
synergy between DSL+learned synthesizer
## Abstraction Sleep: Growing the library via refactoring

<table>
<thead>
<tr>
<th>Task: [1 2 3] → [2 4 6]</th>
<th>Task: [1 2 3] → [0 1 2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[4 3 4] → [8 6 8]</td>
<td>[4 3 4] → [3 2 3]</td>
</tr>
</tbody>
</table>

#### Wake: Program Search

\[
\text{Wake: Program Search} = (\lambda f \text{. } (Y (\lambda r l \text{. } \text{if } (\text{nil?} l) \text{ nil } \text{cons } (f (\text{car} l)) (r (\text{cdr} l)))))) (\lambda z \text{. } (+ z z))
\]

\[
\text{refactor} = (\lambda z \text{. } (- z 1))
\]

\[
\text{refactor} = (\lambda z \text{. } (+ z z))
\]

\[
\text{map} = (\lambda f \text{. } (Y (\lambda r l \text{. } \text{if } (\text{nil?} l) \text{ nil } \text{cons } (f (\text{car} l)) (r (\text{cdr} l)))))
\]

\[
\text{Compress (MDL/Bayes objective)} = -1654 - 1989 + 20
\]
Abstraction Sleep: Growing the library via refactoring

Task: \([1 \ 2 \ 3] \rightarrow [2 \ 4 \ 6] \\
[4 \ 3 \ 4] \rightarrow [8 \ 6 \ 8]

Wake: program search

\((Y \ (\lambda \ (r \ l) \ (\text{if} \ (\text{nil} \ ? \ l) \ \text{nil} \ \\
\text{cons} \ (+ \ (\text{car} \ l) \ (\text{car} \ l)) \ \\
(r \ (\text{cdr} \ l))))))\)

Task: \([1 \ 2 \ 3] \rightarrow [0 \ 1 \ 2] \\
[4 \ 3 \ 4] \rightarrow [3 \ 2 \ 3]

Wake: program search

\((Y \ (\lambda \ (r \ l) \ (\text{if} \ (\text{nil} \ ? \ l) \ \text{nil} \ \\
\text{cons} \ (- \ (\text{car} \ l) \ 1) \ \\
(r \ (\text{cdr} \ l))))))\)
Abstraction Sleep: Growing the library via refactoring

Task: $[1 \ 2 \ 3] \rightarrow [2 \ 4 \ 6]$
$[4 \ 3 \ 4] \rightarrow [8 \ 6 \ 8]$

Wake: program search

$((\lambda (f) \ (Y \ (\lambda (r \ l) \ (if \ (\text{nil?} \ l) \ \text{nil} \n\ (\text{cons} \ (+ \ (\text{car} \ l) \ (\text{car} \ l))\n\ (r \ (\text{cdr} \ l)))))\n\ (\lambda (z) \ (+ \ z \ z)))$)

refactor
($10^{14}$ refactorings)

Sleep: Abstraction

$((\lambda (f) \ (Y \ (\lambda (r \ l) \ (if \ (\text{nil?} \ l) \ \text{nil} \n\ (\text{cons} \ (f \ (\text{car} \ l))\n\ (r \ (\text{cdr} \ l)))))\n\ (\lambda (z) \ (+ \ z \ z)))$)

Task: $[1 \ 2 \ 3] \rightarrow [0 \ 1 \ 2]$
$[4 \ 3 \ 4] \rightarrow [3 \ 2 \ 3]$

Wake: program search

$((\lambda (f) \ (Y \ (\lambda (r \ l) \ (if \ (\text{nil?} \ l) \ \text{nil} \n\ (\text{cons} \ (- \ (\text{car} \ l) \ 1)\n\ (r \ (\text{cdr} \ l)))))\n\ (\lambda (z) \ (+ \ z \ z)))$)

refactor
($10^{14}$ refactorings)

Sleep: Abstraction

$((\lambda (f) \ (Y \ (\lambda (r \ l) \ (if \ (\text{nil?} \ l) \ \text{nil} \n\ (\text{cons} \ (f \ (\text{car} \ l))\n\ (r \ (\text{cdr} \ l)))))\n\ (\lambda (z) \ (+ \ z \ z)))$)
Abstraction Sleep: Growing the library via refactoring

Task: \([1 \ 2 \ 3] \rightarrow [2 \ 4 \ 6]\\n[4 \ 3 \ 4] \rightarrow [8 \ 6 \ 8]\\n
Wake: program search

\[(Y (\lambda (r \ l) (if (nil? \ l) nil (cons (+ (car \ l) (car \ l)) (r \ (cdr \ l))))))\\n\]

refactor \((10^{14} \text{ refactorings})\\n
\[((\lambda (f) (Y (\lambda (r \ l) (if (nil? \ l) nil (cons (f (car \ l)) (r \ (cdr \ l))))))\\n(\lambda \ (z) (+ \ z \ z)))\\n\]

Sleep: Abstraction

Refactor \((10^{14} \text{ refactorings})\\n
\[((\lambda (f) (Y (\lambda (r \ l) (if (nil? \ l) nil (cons (f (car \ l)) (r \ (cdr \ l))))))\\n(\lambda \ (z) (+ \ z \ z)))\\n\]

Compress (MDL/Bayes objective)

\[(\text{MAP} (\lambda (z) (+ z z))) \quad \text{MAP} (\lambda (z) (- z 1))\\n\]

\[\text{MAP} = (\lambda (f) (Y (\lambda (r \ l) (if (nil? \ l) nil (cons (f (car \ l)) (r \ (cdr \ l)))))))\\n\]
Abstraction Sleep: Growing the library via refactoring

Task: \[1\ 2\ 3\] \rightarrow \[2\ 4\ 6\]
\[4\ 3\ 4\] \rightarrow \[8\ 6\ 8\]

Task: \[1\ 2\ 3\] \rightarrow \[0\ 1\ 2\]
\[4\ 3\ 4\] \rightarrow \[3\ 2\ 3\]

Wake: program search

\((\lambda (r\ l)\ (\text{if}\ (\text{nil?}\ l)\ \text{nil}\
\ (\text{cons}\ (+\ (\text{car}\ l)\ (\text{car}\ l))\ 
\ (r\ (\text{cdr}\ l))))))\)

\((\lambda (r\ l)\ (\text{if}\ (\text{nil?}\ l)\ \text{nil}\
\ (\text{cons}\ (-\ (\text{car}\ l)\ 1)\ 
\ (r\ (\text{cdr}\ l))))))\)

Compress (MDL/Bayes objective)

\((\text{map}\ ((\lambda (z)\ (+\ z\ z)))\ ((\lambda (z)\ (-\ z\ 1))))\)

\text{map} = (\lambda (f)\ (Y\ (\lambda (r\ l)\ (\text{if}\ (\text{nil?}\ l)\ \text{nil}\
\ (\text{cons}\ (f\ (\text{car}\ l))\ 
\ (r\ (\text{cdr}\ l)))))))\)

these \(10^{14}\) refactorings represented in exponentially more efficient refactoring data structure: equivalence graphs + version spaces using \(10^6\) nodes, calculated in under 5min c.f. [Tate et al 2009], [Gulwani 2012]
Program Induction and learning to learn
learning a DSL
learning to synthesize
synergy between DSL+learned synthesizer
\[ f_1(x) = (+ x 1) \]
\[ f_2(z) = (\text{fold cons (cons z nil)}) \]

\[ \text{Recognition model} \]

\[ \text{Task} \]
\[ [7, 2, 3] \rightarrow [4, 3, 8] \]
\[ [4, 3, 2] \rightarrow [3, 4, 5] \]

\[ \text{Library} \]

\[ \text{Programs for task:} \]
\[ (\text{map } f_1 (\text{fold } f_2 \text{ nil } x)) \]

\[ \text{SLEEP: ABSTRACTION} \]

\[ \text{progs. for task 1:} \]
\[ (+ (\text{car } z) 1) \]

\[ \text{progs. for task 2:} \]
\[ (\text{cons } (+ 1 1)) \]

\[ \text{Refactoring Algorithm: version spaces} \]

\[ \text{new Library w/ } (+ x 1): \]

\[ \text{SLEEP: DREAMING} \]

\[ \text{Fantasies} \]

\[ \text{Replays} \]

\[ \text{Library} \]

\[ \text{progs. for task} \]

\[ \text{Train recognition model} \]

\[ \text{Loss} \]

\[ \text{task} \]
**Library**

\[ f_1(x) = (+ x 1) \]

\[ f_2(z) = (\text{fold cons} \ (\text{cons} \ z \ \text{nil})) \]

**Task**

\[ [7 \ 2 \ 3] \rightarrow [4 \ 3 \ 8] \]

\[ [4 \ 3 \ 2] \rightarrow [3 \ 4 \ 5] \]

**Programs for task:**

\[(\text{map} \ f_1 \ (\text{fold} \ f_2 \ \text{nil} \ x))\]

**Recognition model**

**Refactoring Algorithm:**

Version spaces

**SLEEP: ABSTRACTION**

progs. for task 1:

\((+ (\text{car} \ z) \ 1)\)

progs. for task 2:

\((\text{cons} \ (+ \ 1 \ 1))\)

**Refactoring Algorithm:**

version spaces

**new Library w/ (+ x 1):**

\[
\begin{align*}
\text{cons} & \rightarrow + \\
\text{+} & \rightarrow 1 \\
\text{car} & \rightarrow z \\
\text{1} & \rightarrow + 1 \\
\end{align*}
\]

**SLEEP: DREAMING**

**Fantasies**

**Replays**

- Library
  - sample program
- progs. for task
  - sample program
- Train recognition model
  - program run task
- Loss

**WAKE**
Neural recognition model guides search

task → program
Neural recognition model guides search

Task $\rightarrow$ Distribution $\rightarrow$ Sample $\rightarrow$ Program
Neural recognition model guides search

is a... recurrent network (Devlin et al 2017)
unigram model (Menon et al 2013; Balog et al 2016)
Neural recognition model guides search

task → distribution → sample → program

P(child|parent, arg)

is a “bigram” model over syntax trees
Neural recognition model guides search

\[ P(\text{child} | \text{parent}, \text{arg}) \]
Neural recognition model guides search

task → distribution → sample ∼→ program

\[ \text{P(child|parent, arg)} \]

\[ \text{P(·| +, arg=left)} \]

9
Neural recognition model guides search
Neural recognition model guides search

P(child|parent, arg)

Task \rightarrow \text{distribution} \rightarrow \text{sample} \rightarrow \text{program}
Neural recognition model guides search

Advantages:
neural net runs once per task,
so CPU bottlenecks instead of GPU
Neural recognition model guides search

Advantages:
neural net runs once per task,
so CPU bottlenecks instead of GPU
learns to break syntactic symmetries:
P(1|*,arg=left)=0.0
“do not multiply by one”
Program Induction and learning to learn
learning a DSL
learning to synthesize
synergy between DSL + learned synthesizer
DreamCoder Domains

List Processing

**Sum List**
- \([1 \ 2 \ 3] \rightarrow 6\)
- \([4 \ 6 \ 8 \ 1] \rightarrow 17\)

**Double**
- \([1 \ 2 \ 3] \rightarrow [2 \ 4 \ 6]\)
- \([4 \ 5 \ 1] \rightarrow [8 \ 10 \ 2]\)

Text Editing

**Abbreviate**
- Allen Newell \(\rightarrow\) A.N.
- Herb Simon \(\rightarrow\) H.S.

**Drop Last Three**
- shrdlu \(\rightarrow\) shr
- shaky \(\rightarrow\) sha

Regexes

**Phone numbers**
- (555) 867-5309
- (650) 555-2368

**Currency**
- $100.25
- $4.50

LOGO Graphics

Block Towers

Symbolic Regression

\[ y = f(x) \]

Recursive Programming

**Filter Red**
- \([\text{red} \ \text{red} \ \text{red}] \rightarrow [\text{red} \ \text{red}]\)
- \([\text{red} \ \text{red} \ \text{red} \ \text{green}] \rightarrow [\text{red} \ \text{red} \ \text{red} \ \text{green}]\)
- \([\text{red} \ \text{green} \ \text{red}] \rightarrow [\text{red} \ \text{red} \ \text{green}]\)

Physical Laws

\[ \vec{a} = \frac{1}{m} \sum_i \vec{F}_i \]
\[ \vec{F} \propto \frac{q_1 q_2}{|\vec{r}|^2} \hat{r} \]
DreamCoder Domains

List Processing

| Sum List   | [1 2 3] → 6  |
|           | [4 6 8 1] → 17  |
| Double    | [1 2 3] → [2 4 6]  |
|           | [4 5 1] → [8 10 2]  |

Text Editing

| Abbreviate  | Allen Newell → A.N. |
|            | Herb Simon → H.S.    |
| Drop Last Three | shrdlu → shr |
|              | shaky → sha          |

Regexes

| Phone numbers   | (555) 867–5309 |
|                | (650) 555–2368 |
| Currency        | $100.25        |
|                | $4.50          |

LOGO Graphics

Block Towers

Symbolic Regression

| y = f(x) |

Recursive Programming

| Filter Red |
| [■■■■■■■] → [■■■■■] |
| [■■■■■■■■■] → [■■■■■■■] |
| [■■■■■■■■■■■] → [■■■■■■■■] |

Physical Laws

| $\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$ |
| $\vec{F} \propto \frac{q_1 q_2}{|\vec{r}|^2} \hat{r}$ |

30 out of 160 tasks
LOGO Turtle Graphics – learning an interpretable library

LOGO Turtle Graphics – learning an interpretable library

LOGO Turtle Graphics – learning an interpretable library

LOGO Turtle Graphics – learning an interpretable library

LOGO Turtle Graphics – learning an interpretable library

radial symmetry \( (n, \text{body}) \)

LOGO Turtle Graphics – learning an interpretable library

circle$(r)$

arc$(n, ℓ, θ)$

What does DreamCoder dream of? (before learning)
What does DreamCoder dream of? (after learning)
example tasks (112 total)
Planning to build towers

example tasks (112 total)

learned library routines (∼ 20 total)

Dreams before learning
Dreams after learning
Learning dynamics

Learning dynamics

![Graph showing learning dynamics](image)

Learning dynamics

baselines: Exploration-Compression, EC [Dechter et al. 2013]
neural program synthesis, RobustFill [Devlin et al. 2017]
24 hours of brute-force enumeration

Learning dynamics

Synergy between recognition model and library learning

Problem-solving

Library

Recognition model

Synergy between recognition model and library learning

Synergy between recognition model and library learning

Evidence for dreaming bootstrapping better libraries

Darker: Early in learning
Brighter: Later in learning

Evidence for dreaming bootstrapping better libraries

Darker: Early in learning

Brighter: Later in learning

Evidence for dreaming bootstrapping better libraries

From learning libraries, to learning languages
From learning libraries, to learning languages

modern functional programming $\rightarrow$ physics
From learning libraries,
to learning languages

1950’s Lisp → modern functional programming → physics
Physics Formula Sheet

**Mechanics**

\[ x = x_0 + v_0 t + \frac{1}{2} a t^2 \quad \text{a}_x = \frac{v_x}{r} \quad |\vec{F}_{\text{spring}}| = k |\vec{x}| \]

\[ v = v_0 + at \quad \theta = \theta_0 + \omega_0 t + \frac{1}{2} \alpha t^2 \quad \text{PE}_{\text{spring}} = \frac{1}{2} k x^2 \]

\[ v_x^2 - v_{x0}^2 = 2a(x - x_0) \quad \omega = \omega_0 + \alpha t \quad T_{\text{spring}} = 2\pi \sqrt{\frac{m}{k}} \]

\[ \ddot{a} = \frac{\sum F}{m} = \frac{F_{\text{net}}}{m} \quad T = \frac{2\pi}{\omega} = \frac{1}{f} \quad T_{\text{pendulum}} = 2\pi \sqrt{\frac{l}{g}} \]

\[ |\vec{F}_{\text{friction}}| \leq \mu |\vec{F}_{\text{Normal}}| \quad \nu = f \lambda \]

\[ \ddot{p} = m \ddot{v} \quad x = A \cos(2\pi ft) \quad |\vec{F}_{\text{gravity}}| = G \frac{m_1 m_2}{r^2} \]

\[ \Delta \ddot{p} = \vec{F} \Delta t \quad \ddot{x} = \frac{\Delta \vec{x}}{\Delta t} \quad |\vec{F}_{\text{gravity}}| = m \ddot{g} \]

\[ KE = \frac{1}{2} mv^2 \quad \ddot{\tau} = r \times \vec{F} \quad \text{PE}_{\text{gravity}} = -G \frac{m_1 m_2}{r} \]

\[ \Delta PE = mg\Delta y \quad L = I \omega \quad \rho = \frac{m}{V} \]

\[ \Delta E = W = Fdcos\theta \quad \Delta L = \tau \Delta t \quad KE = \frac{1}{2} I \omega^2 \]

**Electricity**

\[ |\vec{E}| = k \frac{|q_1 q_2|}{r^2} \quad \Delta V = IR \quad R = \frac{\rho l}{A} \]

\[ l = \frac{\Delta q}{\Delta t} \quad P = l \Delta V \]

\[ R_{\text{series}} = R_1 + R_2 + \ldots + R_n \quad \frac{1}{R_{\text{parallel}}} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots + \frac{1}{R_n} \]

**Geometry**

**Rectangle** \( A = bh \)

**Rectangular Solid** \( V = lwh \)

**Triangle** \( A = \frac{1}{2} bh \)

**Circle** \( A = \pi r^2 \)

**Cylinder** \( V = \pi r^2 h \)

**Sphere** \( V = \frac{4}{3} \pi r^3 \)

\( C = 2\pi r \)

\( S = 2\pi r l + 2\pi r^2 \)

\( S = 4\pi r^2 \)

**Trigonometry**

\[ \sin \theta = \frac{a}{c} \quad \cos \theta = \frac{b}{c} \quad \tan \theta = \frac{a}{b} \]
### Physics Equations

**Newton’s Second Law**

\[ \ddot{a} = \frac{1}{m} \sum F_i \]

**Parallel Resistors**

\[ R_{\text{total}} = \left( \sum \frac{1}{R_i} \right)^{-1} \]

**Work**

\[ U = \vec{F} \cdot \vec{d} \]

**Force in a Magnetic Field**

\[ |\vec{F}| = q |\vec{v} \times \vec{B}| \]

**Kinetic Energy**

\[ KE = \frac{1}{2} m |\vec{v}|^2 \]

**Coulomb’s Law**

\[ \vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \frac{\vec{r}_1}{|\vec{r}_1 - \vec{r}_2|} \frac{\vec{r}_2}{|\vec{r}_1 - \vec{r}_2|} \]
Growing languages for vector algebra and physics

Growing languages for vector algebra and physics

Growing languages for vector algebra and physics

**Learned Library of Concepts**

- \( \vec{u} - \vec{v} \): subtract vectors
- \( \vec{u} + \vec{v} \): add vectors
- \( |\vec{v}|^2 \): magnitude of \( \vec{v} \)
- \( a\vec{v} \): scalar multiplication
- \( \sum_j \vec{v}_j \): sum many vectors
- \( \frac{ab}{|\vec{v}|^2} \): division by magnitude
- \( \sqrt{x} \): square root
- \( \sum_j [j] \): sum components
- \( \vec{u} \cdot \vec{v} \): dot product
- \( \frac{1}{x} \): reciprocal
- \( \frac{2\pi}{ab} \): period

**Physics Equations**

- **Newton's Second Law**
  \[ \vec{a} = \frac{1}{m} \sum_i \vec{F}_i \]

- **Parallel Resistors**
  \[ R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1} \]

- **Work**
  \[ U = \vec{F} \cdot \vec{d} \]

- **Force in a Magnetic Field**
  \[ |\vec{F}| = q|\vec{v} \times \vec{B}| \]

- **Kinetic Energy**
  \[ KE = \frac{1}{2} m|\vec{v}|^2 \]

- **Coulomb's Law**
  \[ \vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \]

Growing languages for vector algebra and physics

Growing languages for vector algebra and physics

Growing a language for recursive programming

Growing a language for recursive programming

Growing a language for recursive programming

Growing a language for recursive programming

Growing a language for recursive programming


Origami Programming: Jeremy Gibbons, 2003
Growing a language for recursive programming


1 year of compute. 5 days on 64 CPUs.

Origami Programming: Jeremy Gibbons, 2003
Library learning interacts synergistically with neural synthesis: bootstrapping, more than sum of parts
Lessons

Library learning interacts synergistically with neural synthesis: bootstrapping, more than sum of parts

Symbols aren’t necessarily interpretable. Grow the language based on experience to make it more powerful and more human understandable
Library learning interacts synergistically with neural synthesis: bootstrapping, more than sum of parts

Symbols aren’t necessarily interpretable. Grow the language based on experience to make it more powerful and more human understandable

Learning-from-scratch is possible in principle. Don’t do it. But program induction makes it convenient to build in what we know how to build in, and then learn on top of that
the end.
Collaborators

Josh Tenenbaum
Armando Solar-Lezama
Max Nye
Cathy Wong
Mathias Sable-Meyer
Lucas Morales

thank you
3D program induction

Challenge: combinatorial search!
Branching factor: $> 1.3$ million per line of code, $\approx 20$ lines of code
Search space size: $(1.3 \text{ million})^{20} \approx 10^{122}$ programs

*equal contribution
Solution: stochastic **tree search** + learn **policy** that writes code + learn **value** function that assesses execution of program so far; analogous to *AlphaGo* [Silver et al. 2016]

**Input:**
Solution: stochastic tree search + learn policy that writes code + learn value function that assesses execution of program so far; analogous to AlphaGo [Silver et al. 2016]
Solution: stochastic **tree search** + learn **policy** that writes code + learn **value** function that assesses execution of program so far; analogous to **AlphaGo** [Silver et al. 2016]
Solution: stochastic **tree search** + learn **policy** that writes code + learn **value** function that assesses execution of program so far; analogous to **AlphaGo** [Silver et al. 2016]
3D program induction

Input (voxels)

Rendered program


*equal contribution
3D program induction

same architecture learns to synthesize text editing programs (FlashFill, Gulwani 2012)

DreamCoder learns libraries for FlashFill-style text editing [Gulwani 2012]

Library structure: Generating Text

Libraries for probabilistic generative models over text: data from crawling web for CSV files
150 random dreams before learning
150 random dreams after learning