

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

Kevin Ellis

Collaborators: Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer,
Lucas Morales, Armando Solar-Lezama, Joshua B. Tenenbaum

2021

Synthesis of Models and Systems

The premise of program induction

1. Represent knowledge as programs: as symbolic code

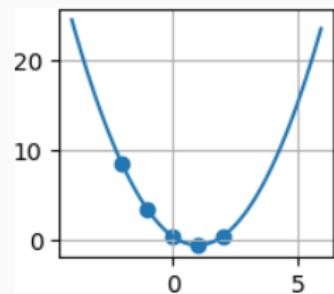
The premise of program induction

1. Represent knowledge as programs: as symbolic code
2. Learning=adding to that body of knowledge=
making new programs=program synthesis

Why program induction?

Why program induction?

strong generalization
+data efficiency

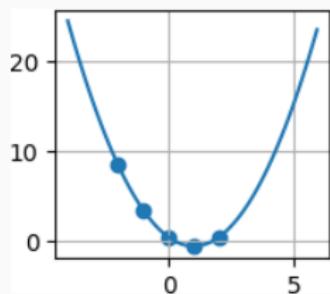


$$f(x) = (x-1)^{**2} - 0.5$$

Why program induction?

strong generalization
+data efficiency

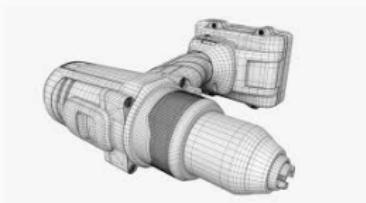
interpretability



$$f(x) = (x-1)^{\star\star 2} - 0.5$$

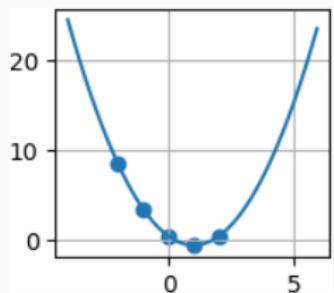


VS



Why program induction?

strong generalization
+data efficiency

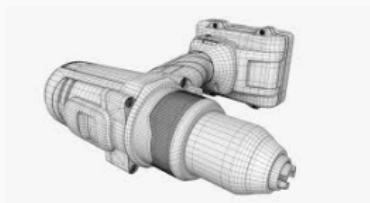


$$f(x) = (x-1)^{**2} - 0.5$$

interpretability



universal expressivity



FlashFill (Gulwani 2012)

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

Input v_1	Output
Company\Code\index.html	Company\Code\
Company\Docs\Spec\specs.doc	Company\Docs\Spec\

String Program:

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

FlashFill (Gulwani 2012)

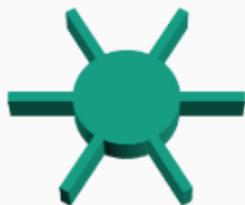
EXAMPLE 3 (Directory Name Extraction). Consider the following example taken from an excel online help forum.

Input v_1	Output
Company\Code\index.html	Company\Code\
Company\Docs\Spec\specs.doc	Company\Docs\Spec\

String Program:

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

Szalinski (Nandi 2020)



(a) CAD model of ship's wheel

```
(Union  
  (Cylinder [1, 5, 5])  
  (Fold Union  
    (Tabulate (i 6)  
      (Rotate [0, 0, 60i]  
        (Translate [1, -0.5, 0]  
          (Cuboid [10, 1, 1]))))))
```

(b) Caddy program

FlashFill (Gulwani 2012)

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

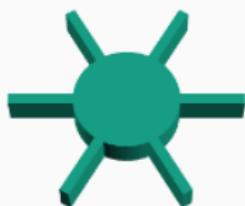
Input v_1	Output
Company\Code\index.html	Company\Code\
Company\Docs\Spec\specs.doc	Company\Docs\Spec\

String Program:

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

String expr P	$\text{Switch}((b_1, e_1), \dots, (b_n, e_n))$
Bool b	$d_1 \vee \dots \vee d_n$
Conjunct d	$\pi_1 \wedge \dots \wedge \pi_n$
Predicate π	$\text{Match}(v_i, r, k) \mid \neg \text{Match}(v_i, r, k)$
Trace expr e	$\text{Concatenate}(f_1, \dots, f_n)$
Atomic expr f	$\text{SubStr}(v_i, p_1, p_2)$ $\text{ConstStr}(s)$ $\text{Loop}(\lambda w : e)$
Position p	$\text{CPos}(k) \mid \text{Pos}(r_1, r_2, c)$
Integer expr c	$k \mid k_1 w + k_2$
Regular Expression r	$\text{TokenSeq}(T_1, \dots, T_m)$
Token T	$C + \mid [\neg C] + \mid \text{SpecialToken}$

Szalinski (Nandi 2020)



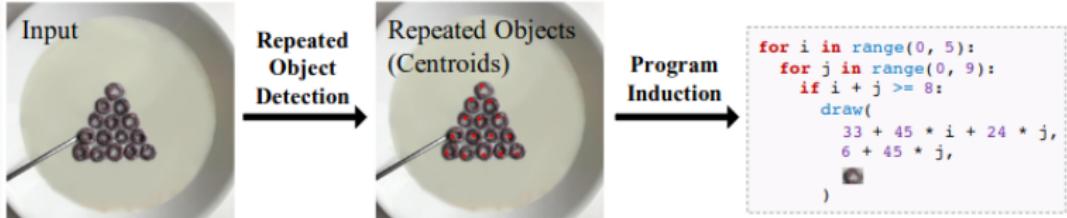
(a) CAD model of ship's wheel

```
(Union
  (Cylinder [1, 5, 5])
  (Fold Union
    (Tabulate (i 6)
      (Rotate [0, 0, 60i]
        (Translate [1, -0.5, 0]
          (Cuboid [10, 1, 1]))))))
```

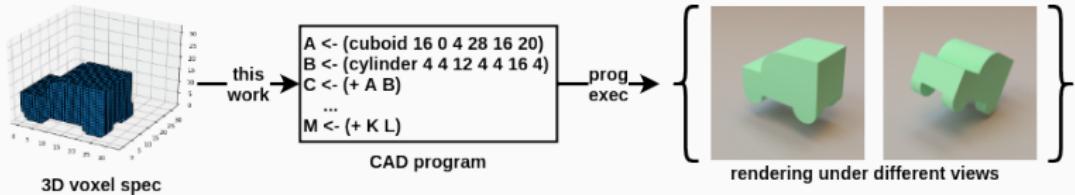
(b) Caddy program

op	$\text{::= } + \mid - \mid \times \mid / \mid \text{num} \text{ ::= } \mathbb{R} \mid \langle \text{var} \rangle \mid \langle \text{num} \rangle \langle \text{op} \rangle \langle \text{num} \rangle$
vec2	$\text{::= } [\langle \text{num} \rangle, \langle \text{num} \rangle] \mid \text{vec3} \text{ ::= } [\langle \text{num} \rangle, \langle \text{num} \rangle, \langle \text{num} \rangle]$
affine	$\text{::= } \text{Translate} \mid \text{Rotate} \mid \text{Scale} \mid \text{TranslateSpherical}$
binop	$\text{::= } \text{Union} \mid \text{Difference} \mid \text{Intersection}$
cad	$\text{::= } (\text{Cuboid } \langle \text{vec3} \rangle) \mid (\text{Sphere } \langle \text{num} \rangle)$ $(\text{Cylinder } \langle \text{vec2} \rangle) \mid (\text{HexPrism } \langle \text{vec2} \rangle) \mid \dots$ $((\text{affine}) \langle \text{vec3} \rangle \langle \text{cad} \rangle)$ $((\text{binop}) \langle \text{cad} \rangle \langle \text{cad} \rangle)$ $(\text{Fold } \langle \text{binop} \rangle \langle \text{cad-list} \rangle)$
cad-list	$\text{::= } (\text{List } \langle \text{cad} \rangle^+)$ $(\text{Concat } \langle \text{cad-list} \rangle^+)$ $(\text{Tabulate } (\langle \text{var} \rangle \ Z^+)^+ \langle \text{cad} \rangle)$ $(\text{Map2 } \langle \text{affine} \rangle \langle \text{vec3-list} \rangle \langle \text{cad-list} \rangle)$
vec3-list	$\text{::= } (\text{List } \langle \text{vec3} \rangle^+)$ $(\text{Concat } \langle \text{vec3-list} \rangle^+)$ $(\text{Tabulate } (\langle \text{var} \rangle \ Z^+)^+ \langle \text{vec3} \rangle)$

Visual programs



Mao*, Zhang*, et al 2019



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



partial image x_{part}

```
for i = 1..3
    for j = 1..1
        draw(i*2, j*1, [color])
    ...

```

synthesized program P_{part}



```
Draw("Top", "Circle", position, geometry)

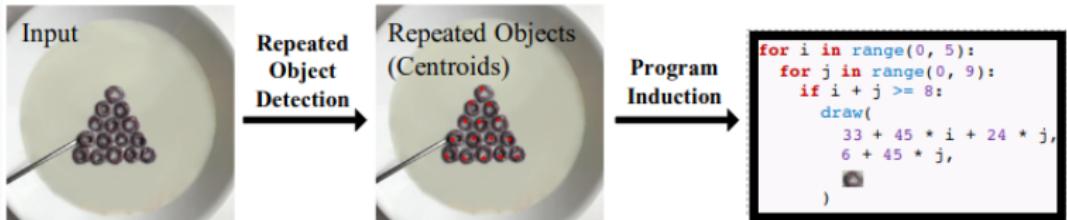
for(i < 2, "translation", a)
    for(j < 2, "translation", b)
        Draw("Leg", "Cub", position + i*a + j*b, )

for(i < 2, "translation", c)
    Draw("Layer", "Rec", position + i*c, geomet
```

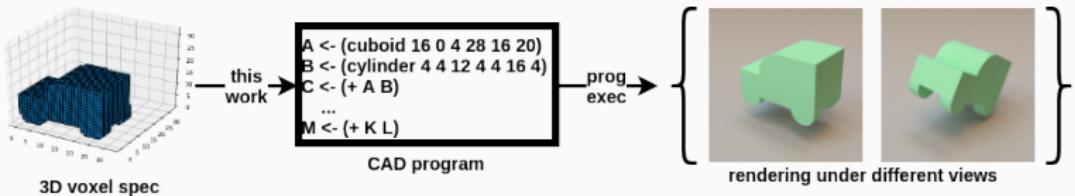
Young et al 2019

Tian et al 2019

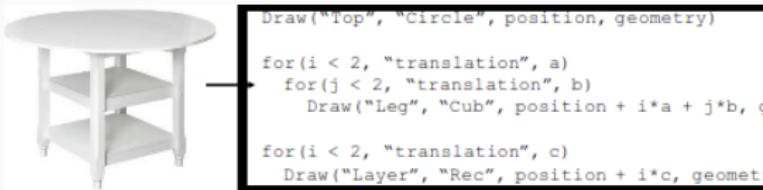
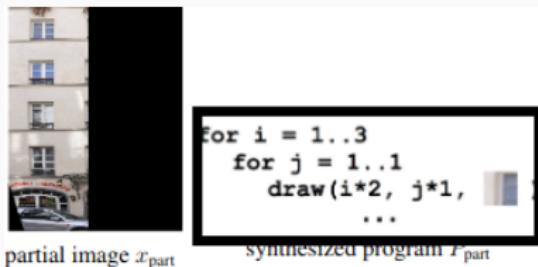
Where does this language come from?



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

Learning to write code

Goal: acquire domain-specific knowledge needed to induce a class of programs

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

Library learning

Initial Primitives

: value
:

map

fold func

if

cons

>

: value
:

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

```
:
```

```
:
```

```
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]
```

```
...
```

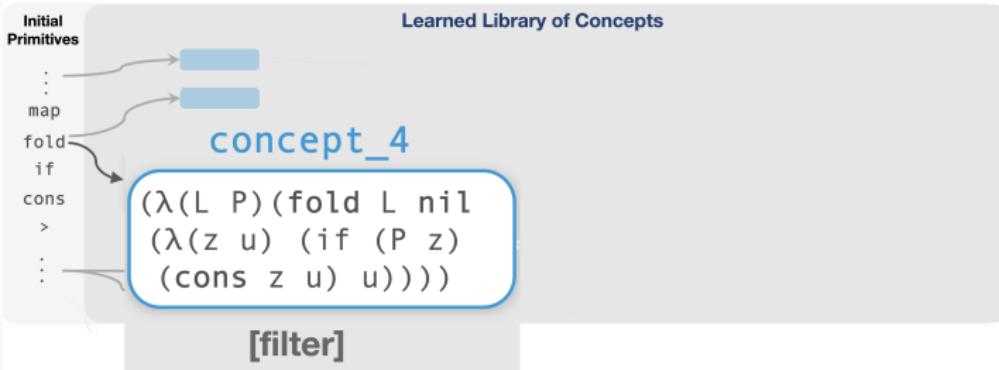
Library learning



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



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]
...

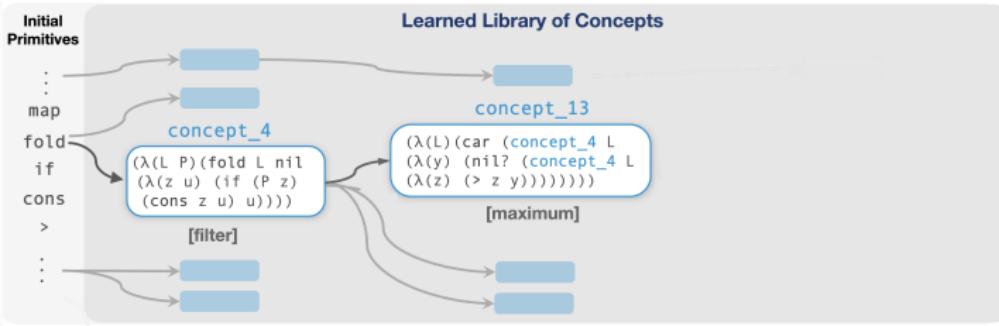
Library learning



Sample Problem: sort list

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

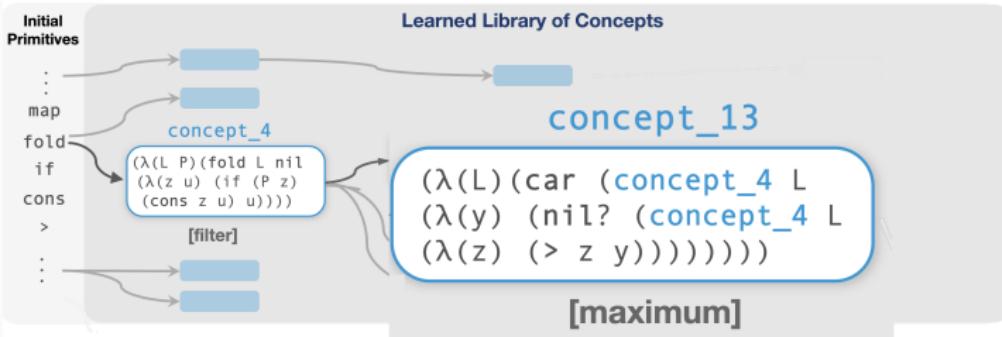
Library learning



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]
...

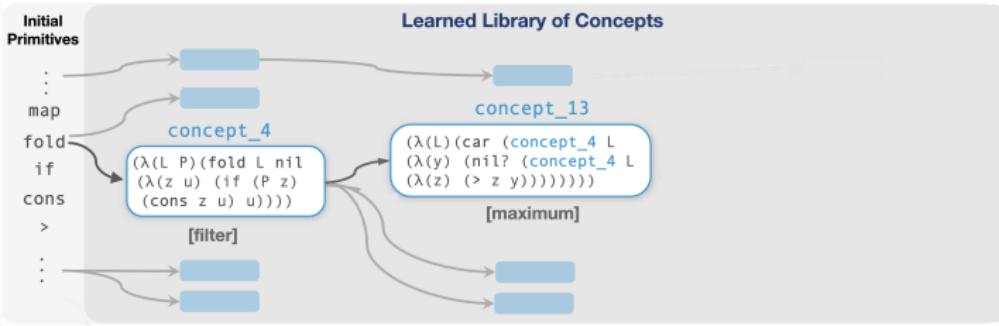
Library learning



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]
...

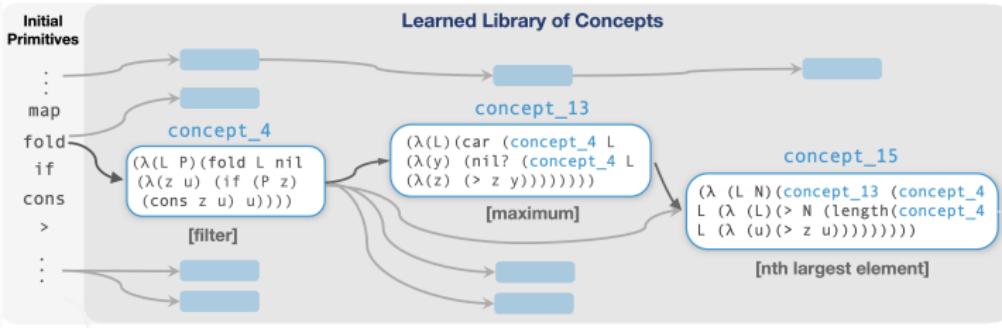
Library learning



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]
...

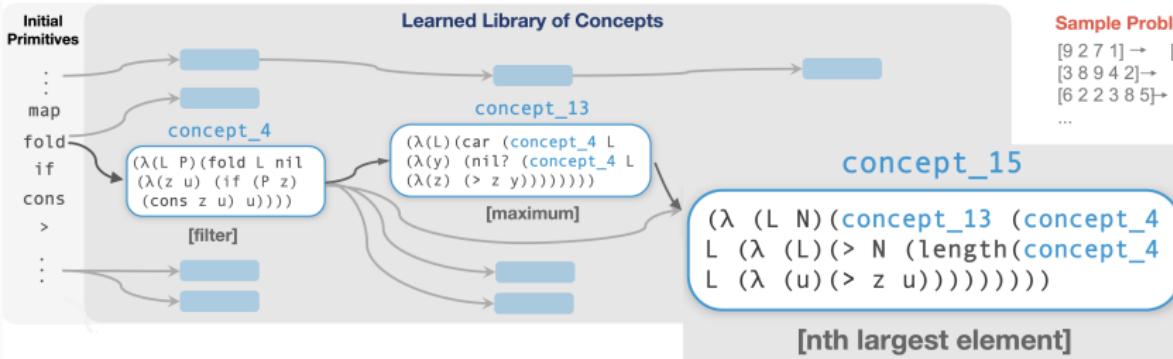
Library learning



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]
...

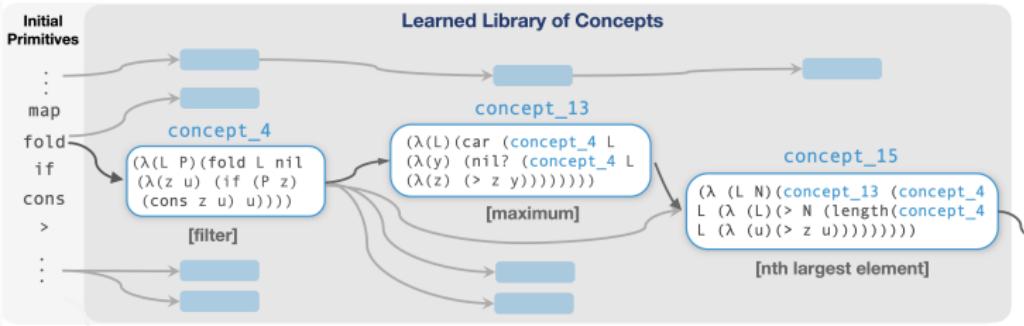
Library learning



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



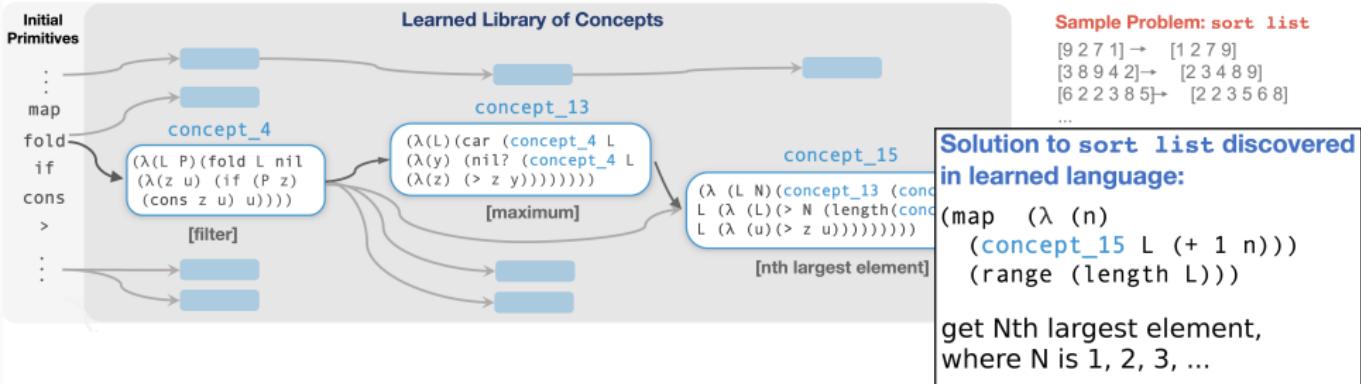
Sample Problem: sort list

[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]
...

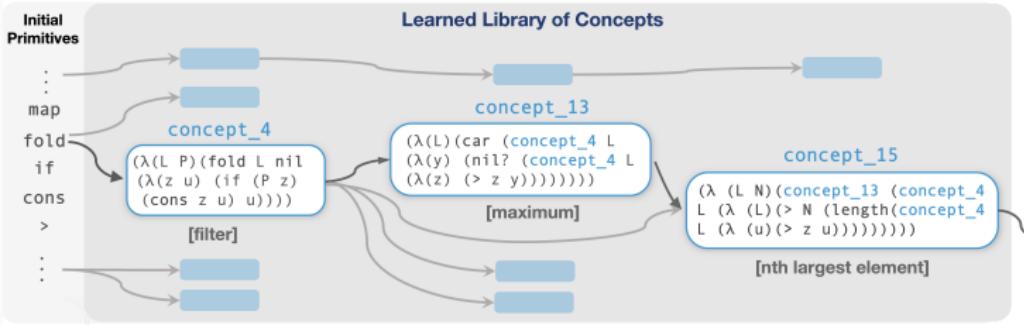
Solution to sort list discovered in learned language:

(map (λ (n)
 (concept_15 L (+ 1 n)))
 (range (length L)))

Library learning



Library learning



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]
...

Solution to sort list discovered in learned language:

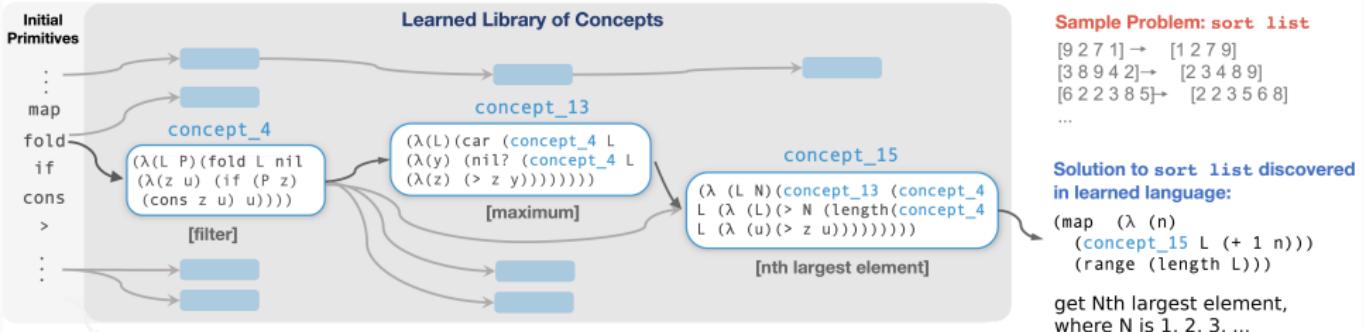
```
(map (λ (n)
  (concept_15 L (+ 1 n)))
  (range (length L)))
```

get Nth largest element,
where N is 1, 2, 3, ...

Solution rewritten in initial primitives:

```
(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))) (cons a b) b)))))) (range (length x))))
```

Library learning



Solution rewritten in initial primitives:

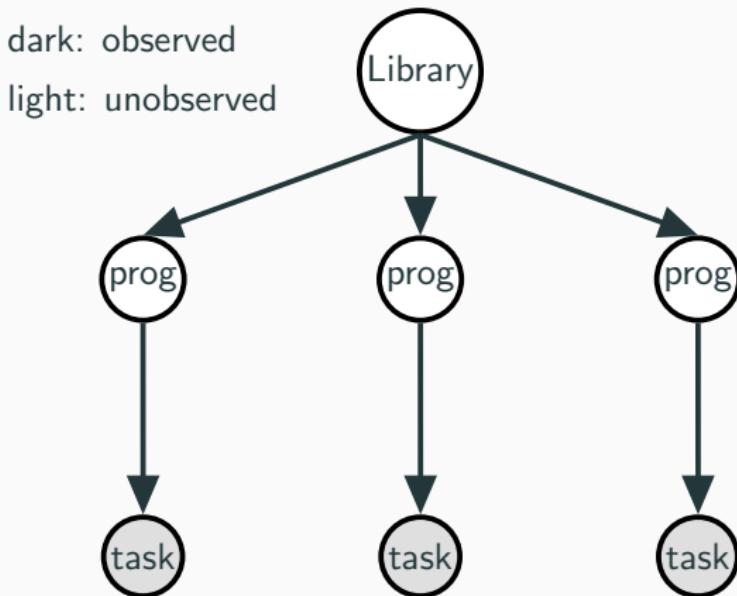
```
(lambda (x) (map (lambda (y) (car (fold (fold x nil (lambda (z u) (if (gt? (+ y 1) (length (fold x nil (lambda (v w) (if (gt? z v) (cons v w) w)))) (cons z u) u))) nil (lambda (a b) (if (nil? (fold (fold x nil (lambda (c d) (if (gt? (+ y 1) (length (fold x nil (lambda (e f) (if (gt? c e) (cons e f) f)))) (cons c d) d))) nil (lambda (g h) (if (gt? g a) (cons g h) h))) (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

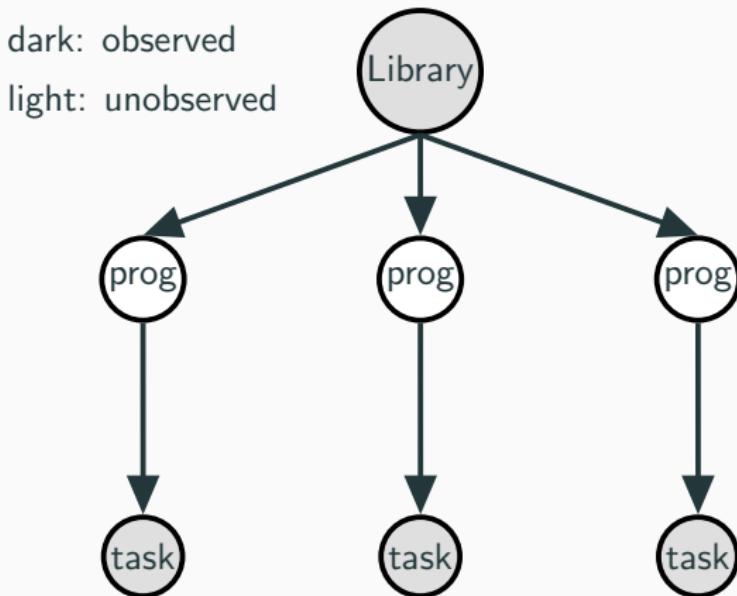
cf. Helmholtz machine, wake/sleep neural network training algorithms

Library learning as Bayesian inference



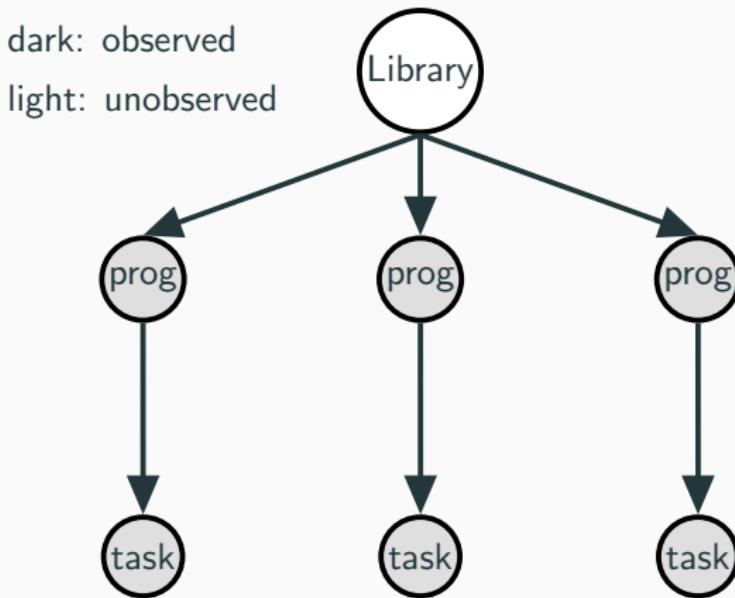
[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

Library learning as Bayesian inference



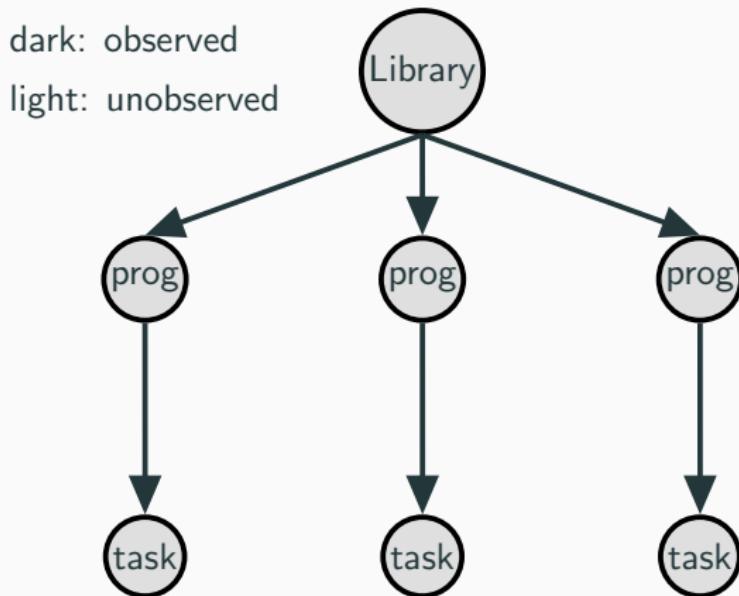
[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

Library learning as Bayesian inference



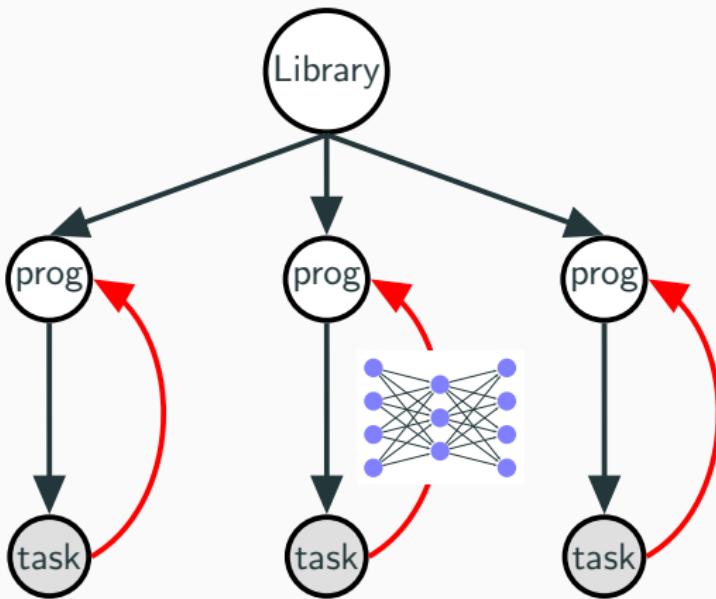
[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

Library learning as Bayesian inference

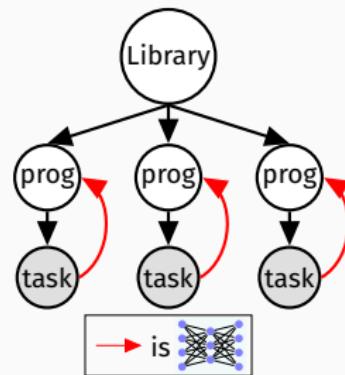


[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

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])



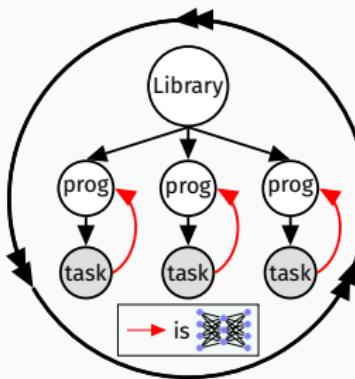
WAKE

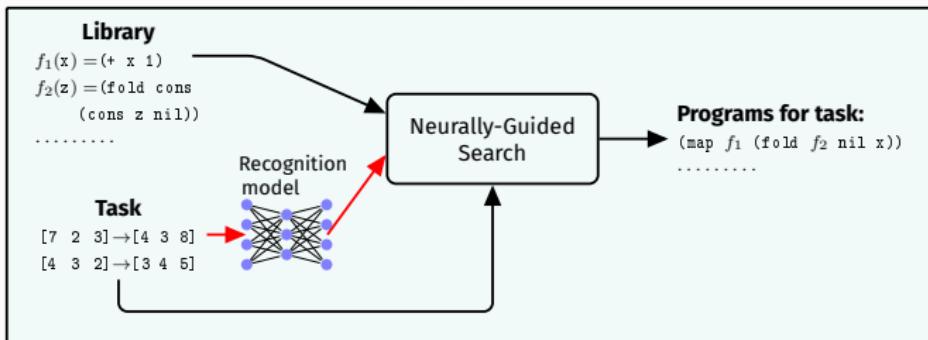
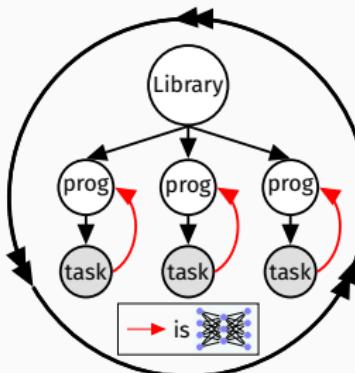


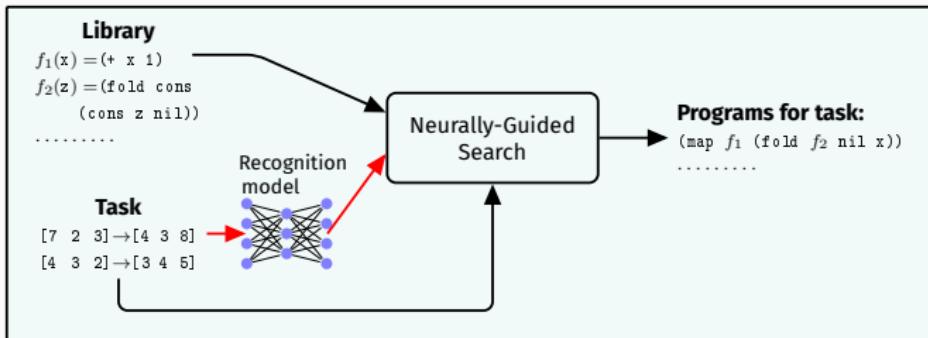
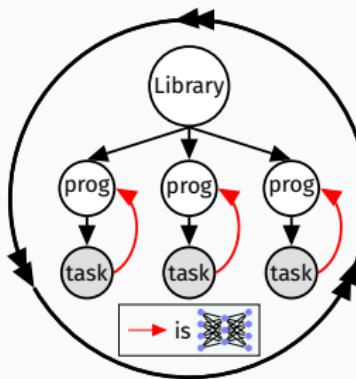
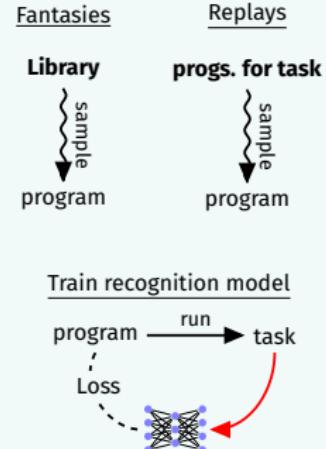
SLEEP: ABSTRACTION



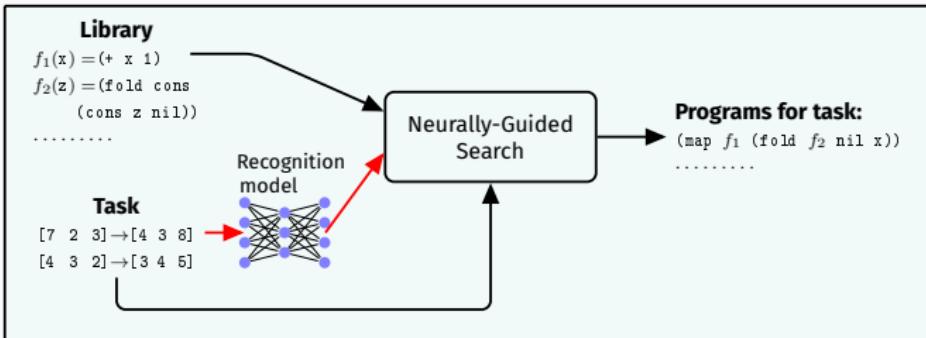
SLEEP: DREAMING



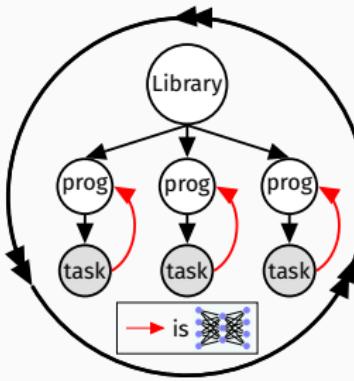
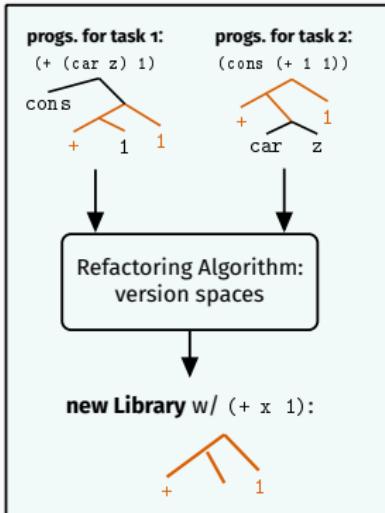
WAKE**SLEEP: ABSTRACTION****SLEEP: DREAMING**

WAKE**SLEEP: ABSTRACTION****SLEEP: DREAMING**

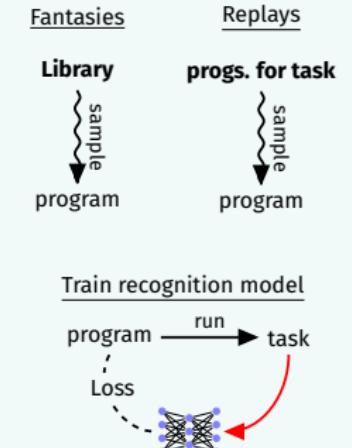
WAKE



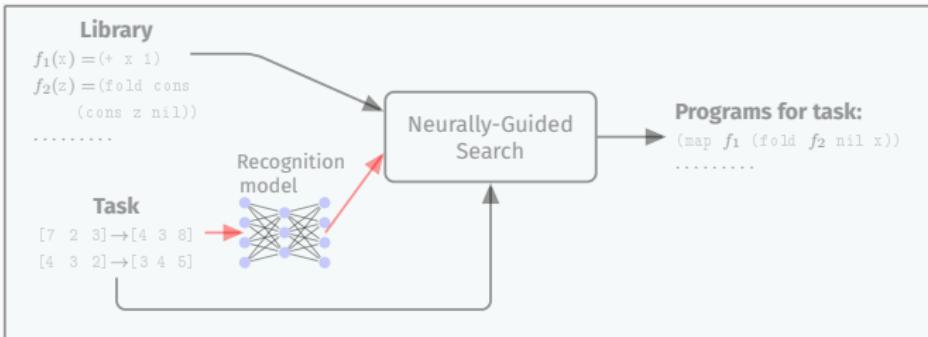
SLEEP: ABSTRACTION



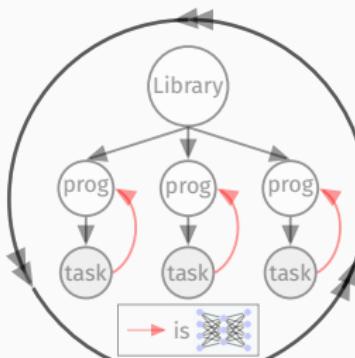
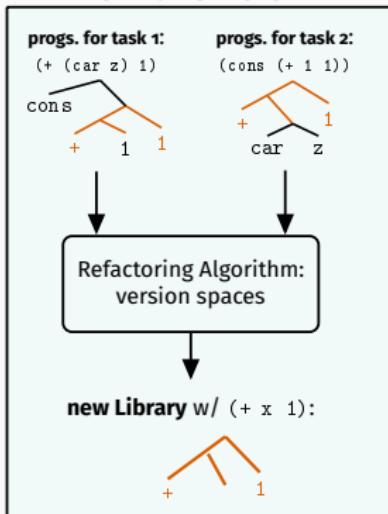
SLEEP: DREAMING



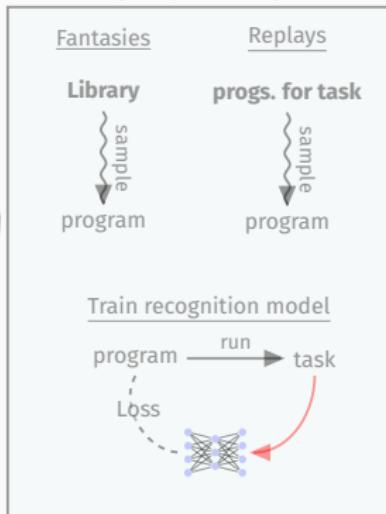
WAKE



SLEEP: ABSTRACTION



SLEEP: DREAMING



Program Induction and learning to learn
learning a DSL
learning to synthesize
synergy between DSL+learned synthesizer

Abstraction Sleep: Growing the library via refactoring

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

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

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 (λ (r 1) (if (nil? 1) nil  
           (cons (+ (car 1) (car 1))  
                 (r (cdr 1)))))))
```

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

Wake: program search

```
(Y (λ (r 1) (if (nil? 1) nil  
           (cons (- (car 1) 1)  
                 (r (cdr 1)))))))
```

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 (λ (r 1) (if (nil? 1) nil  
           (cons (+ (car 1) (car 1))  
                  (r (cdr 1)))))))
```

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

Wake: program search

```
(Y (λ (r 1) (if (nil? 1) nil  
           (cons (- (car 1) 1)  
                  (r (cdr 1)))))))
```

refactor

$(10^{14}$ refactorings)

```
((λ (f) (Y (λ (r 1) (if (nil? 1)  
                           nil  
                           (cons (f (car 1))  
                                 (r (cdr 1)))))))  
  (λ (z) (+ z z)))
```

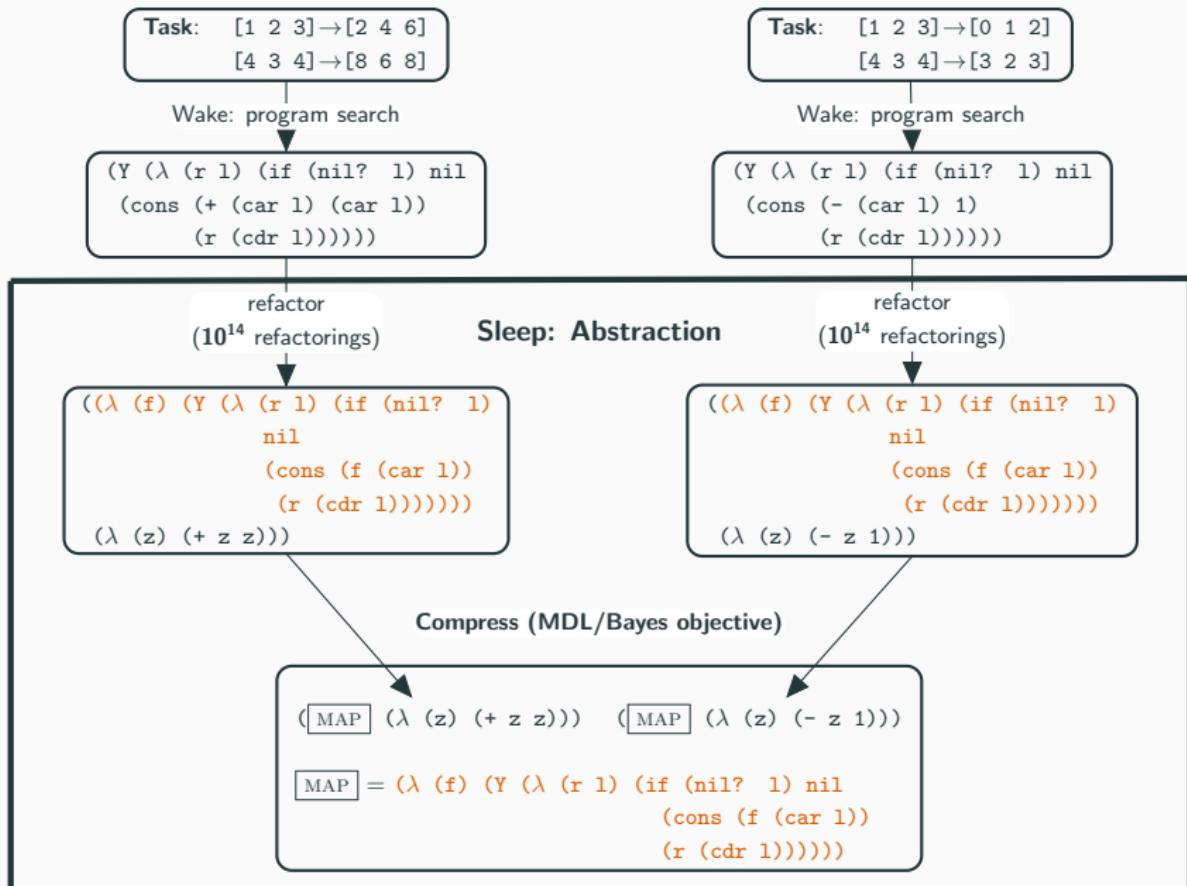
Sleep: Abstraction

refactor

$(10^{14}$ refactorings)

```
((λ (f) (Y (λ (r 1) (if (nil? 1)  
                           nil  
                           (cons (f (car 1))  
                                 (r (cdr 1)))))))  
  (λ (z) (- z 1)))
```

Abstraction Sleep: Growing the library via refactoring



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 (λ (r 1) (if (nil? 1) nil  
           (cons (+ (car 1) (car 1))  
                  (r (cdr 1)))))))
```

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

Wake: program search

```
(Y (λ (r 1) (if (nil? 1) nil  
           (cons (- (car 1) 1)  
                  (r (cdr 1)))))))
```

these 10^{14} refactorings represented in exponentially more efficient refactoring data structure:

$(\lambda$ equivalence graphs+version spaces using 10^6 nodes,
calculated in under 5min

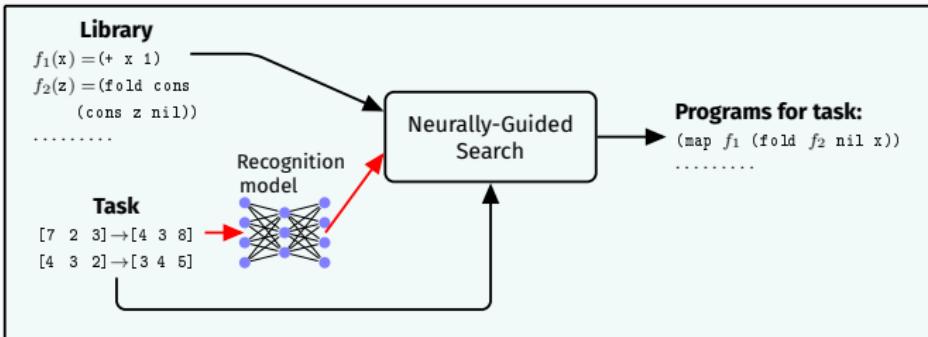
c.f. [Tate et al 2009], [Gulwani 2012]

Compress (MDL/Bayes objective)

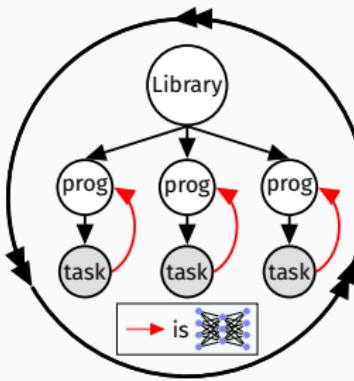
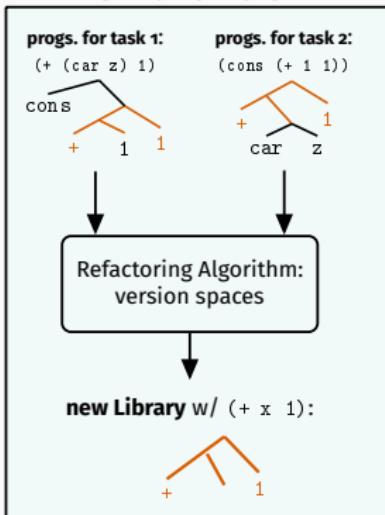
```
(MAP (λ (z) (+ z z))) (MAP (λ (z) (- z 1)))  
MAP = (λ (f) (Y (λ (r 1) (if (nil? 1) nil  
                           (cons (f (car 1))  
                                 (r (cdr 1)))))))
```

Program Induction and learning to learn
learning a DSL
learning to synthesize
synergy between DSL+learned synthesizer

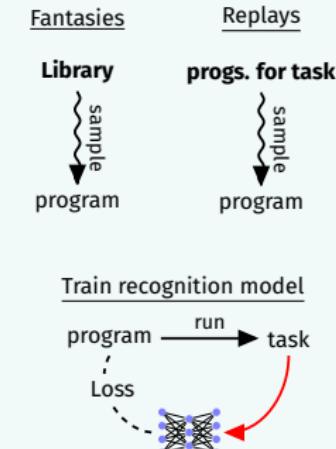
WAKE



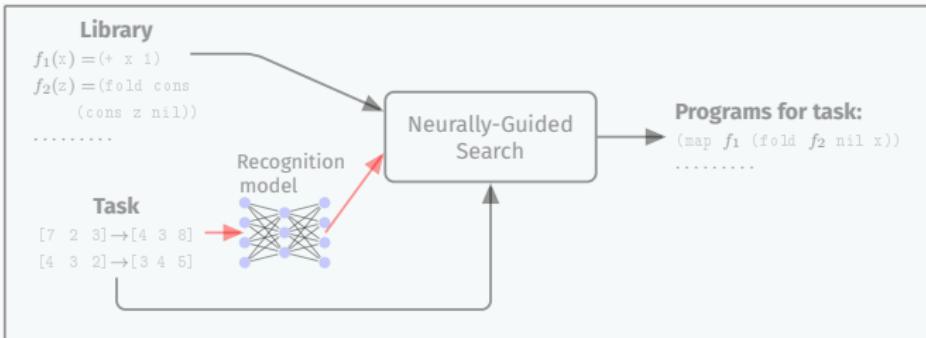
SLEEP: ABSTRACTION



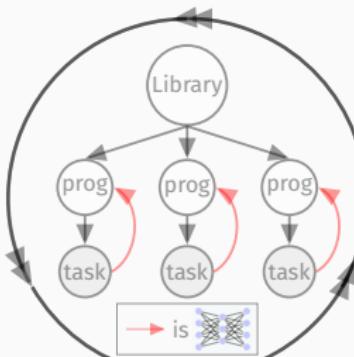
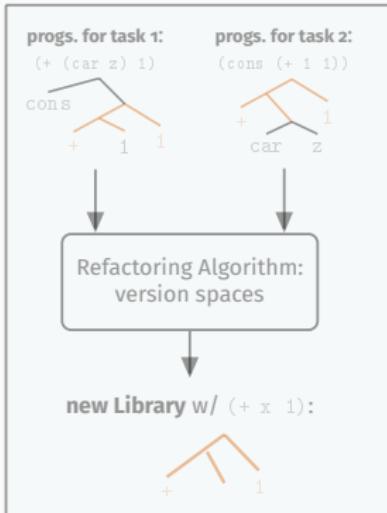
SLEEP: DREAMING



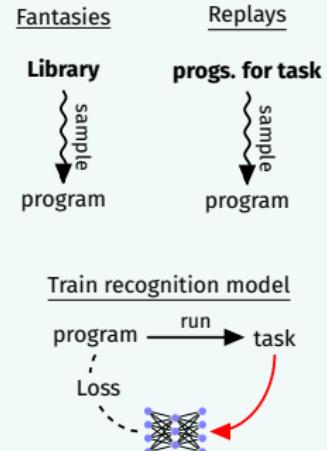
WAKE



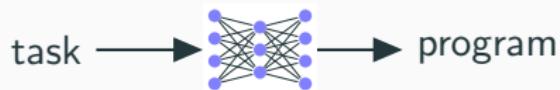
SLEEP: ABSTRACTION



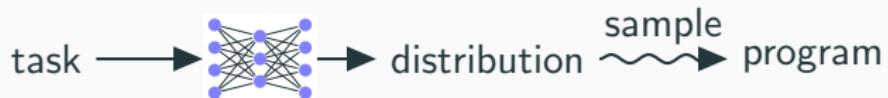
SLEEP: DREAMING



Neural recognition model guides search



Neural recognition model guides search



Neural recognition model guides search

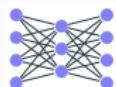


is a...

recurrent network (Devlin et al 2017)

unigram model (Menon et al 2013; Balog et al 2016)

Neural recognition model guides search

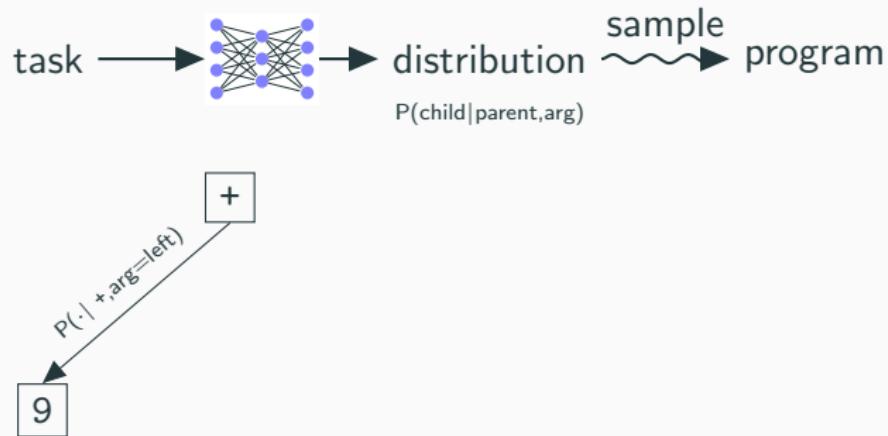


is a “**bigram**” model over syntax trees

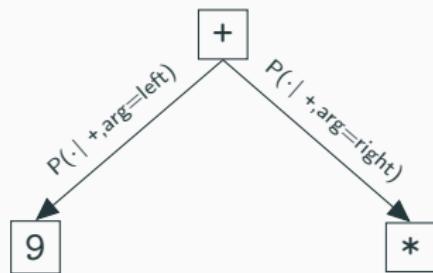
Neural recognition model guides search



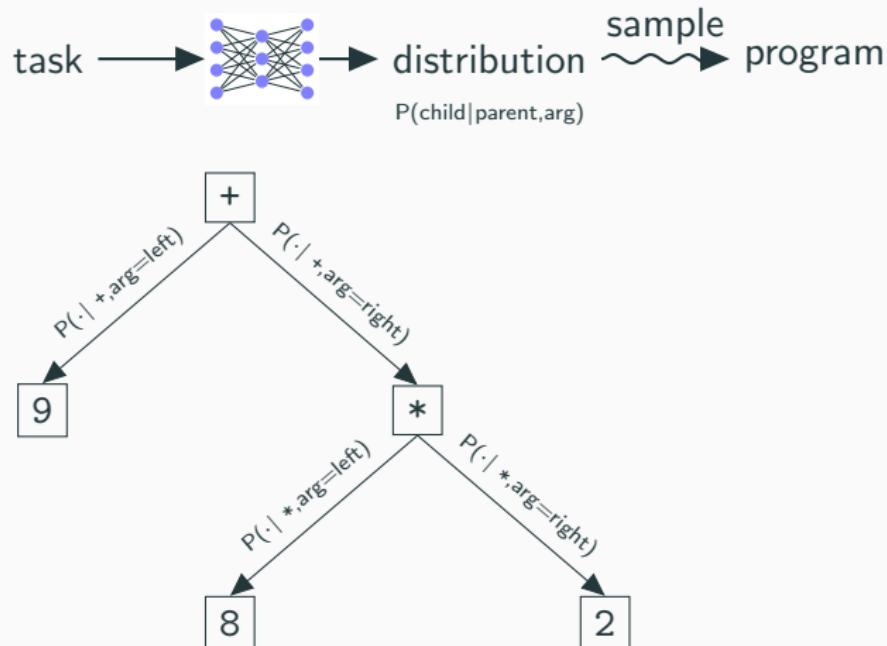
Neural recognition model guides search



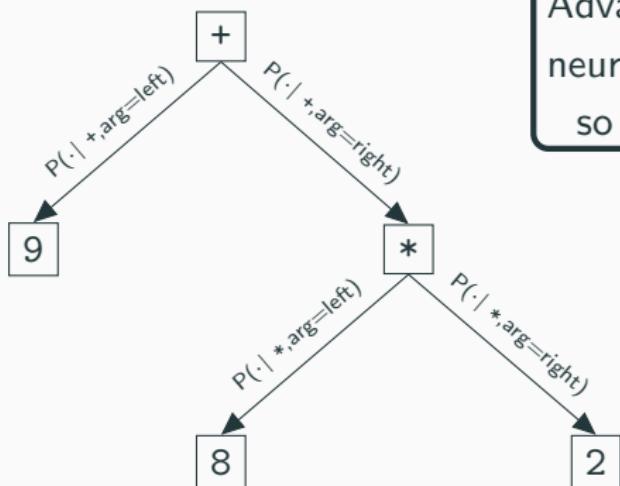
Neural recognition model guides search



Neural recognition model guides search



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:
 $P(1*,\text{arg}=left)=0.0$
“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] → 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

shakey → sha

Regexes

Phone numbers

(555) 867-5309

(650) 555-2368

Currency

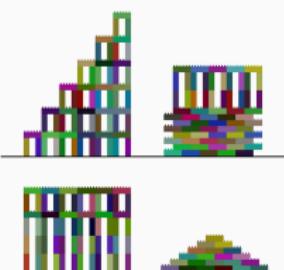
\$100.25

\$4.50

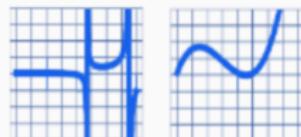
LOGO Graphics



Block Towers



Symbolic Regression



$$y = f(x)$$

Recursive Programming

Filter Red

[■■■■■■■■] → [■■■■]

[■■■■■■■■■■] → [■■■■■■■■]

[■■■■■■■■■■■■] → [■■■■■■■■■■]

Physical Laws

$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}|^2} \hat{r}$$

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

shakey → sha

Regexes

Phone numbers

(555) 867-5309

(650) 555-2368

Currency

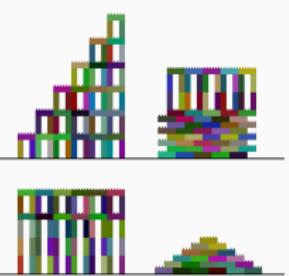
\$100.25

\$4.50

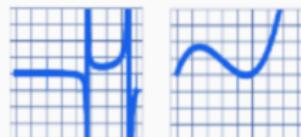
LOGO Graphics



Block Towers



Symbolic Regression



$$y = f(x)$$

Recursive Programming

Filter Red

[■■■■■■■■] → [■■■■■■■]

[■■■■■■■■■■] → [■■■■■■■■■]

[■■■■■■■■■■■] → [■■■■■■■■■]

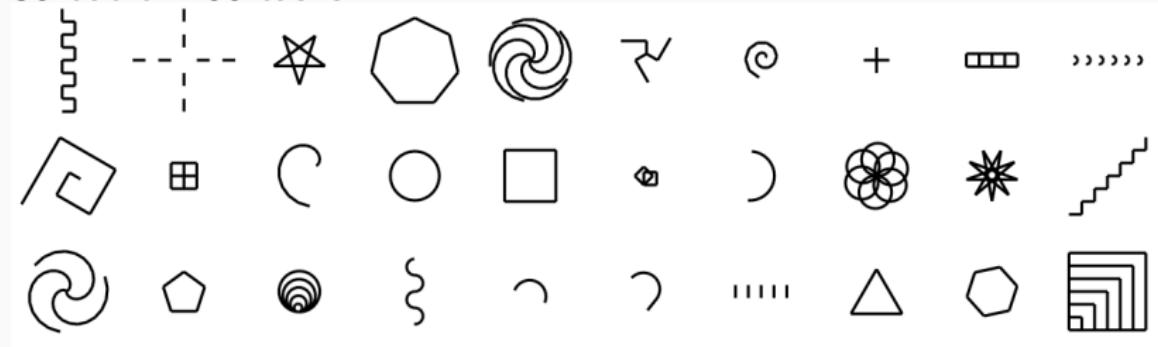
Physical Laws

$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

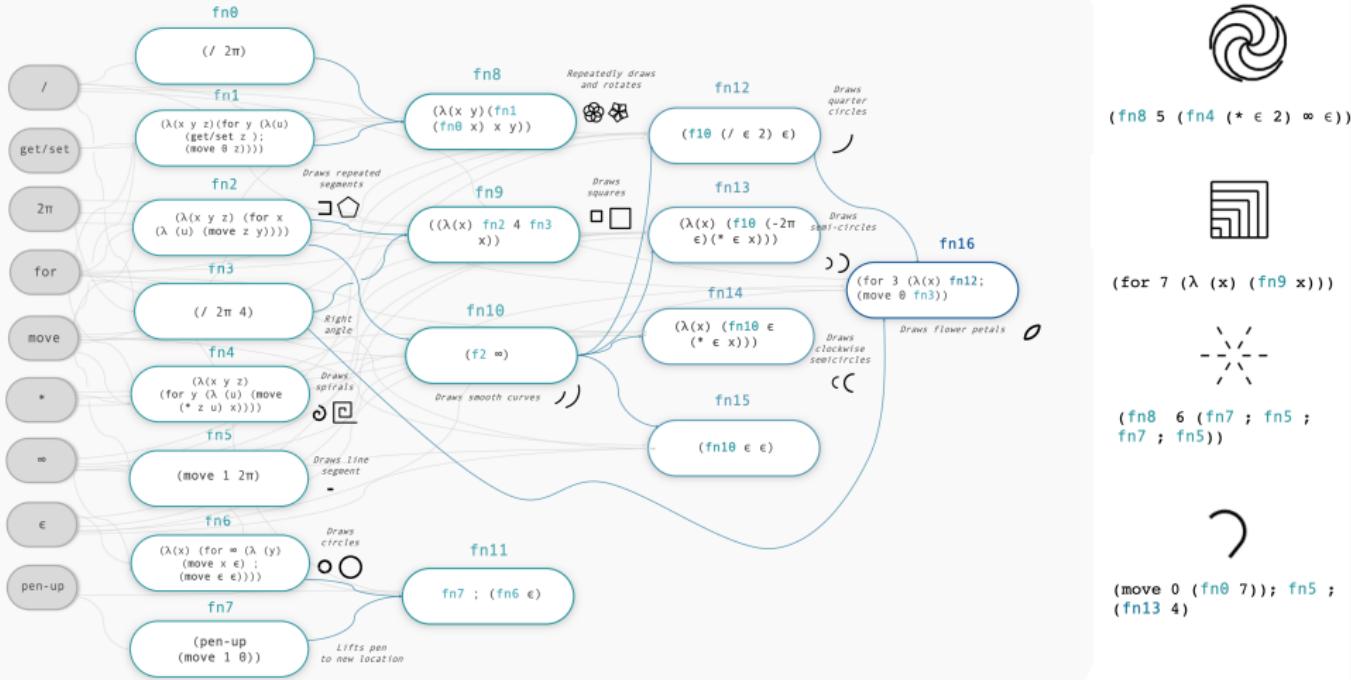
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}|^2} \hat{r}$$

LOGO Turtle Graphics

30 out of 160 tasks



LOGO Turtle Graphics – learning an interpretable library



(fn8 5 (fn4 (* ε 2) ∞))



(for 7 (λ (x) (fn9 x)))

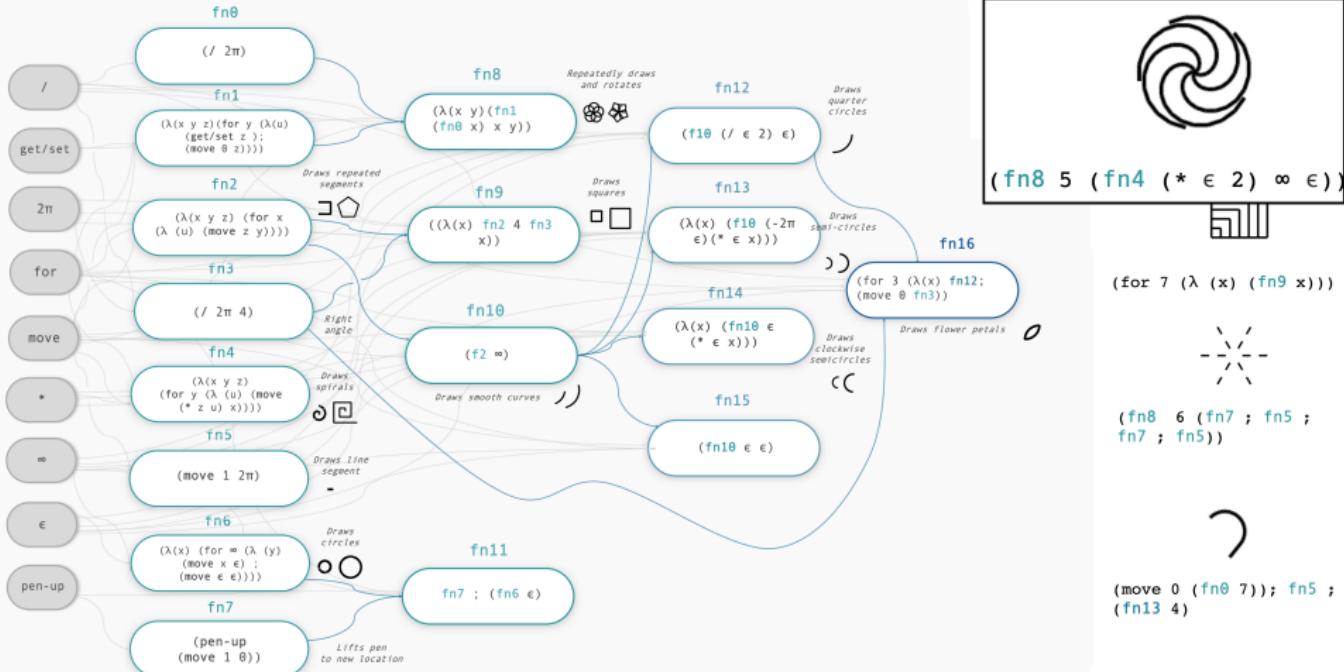


(fn8 6 (fn7 ; fn5 ; fn7 ; fn5))

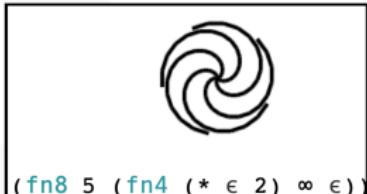
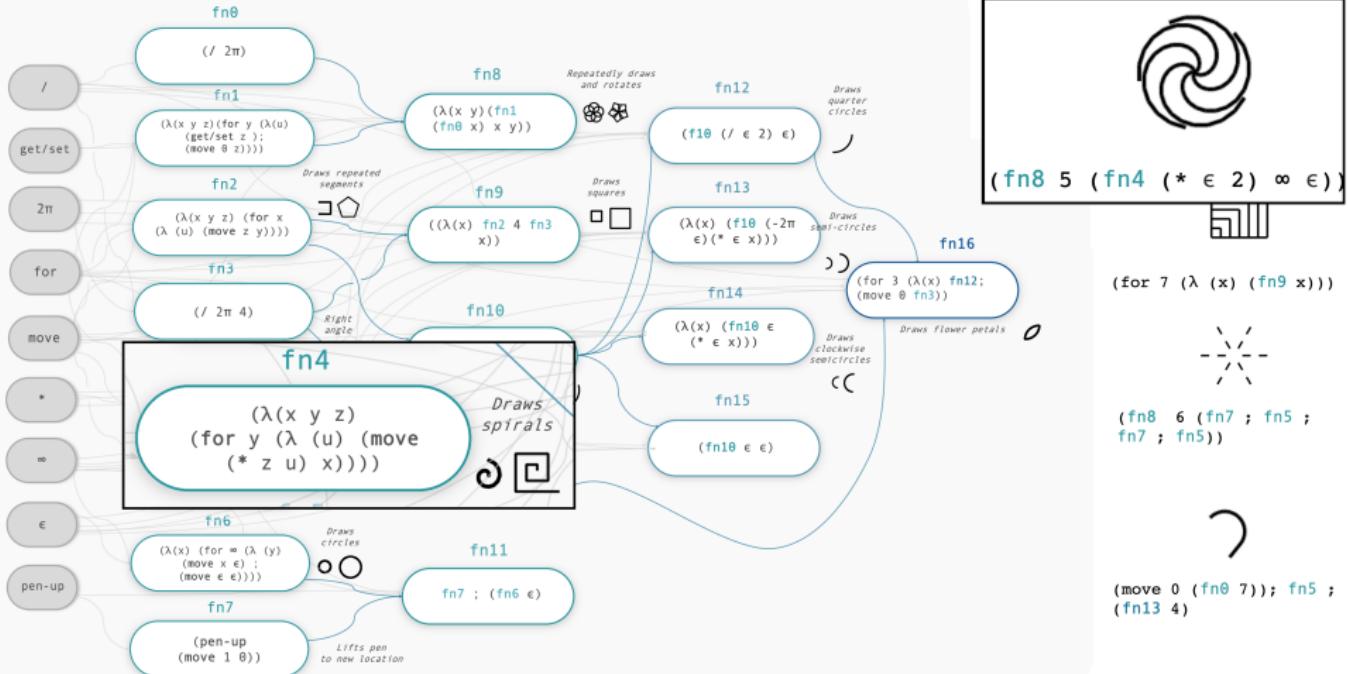


(move 0 (fn0 7)); fn5 ; (fn13 4)

LOGO Turtle Graphics – learning an interpretable library



LOGO Turtle Graphics – learning an interpretable library



$(\text{fn8}\ 5\ (\text{fn4}\ (*\ \epsilon\ 2)\ \infty\ \epsilon))$

$(\text{for } 7\ (\lambda(x)\ (\text{fn9}\ x)))$

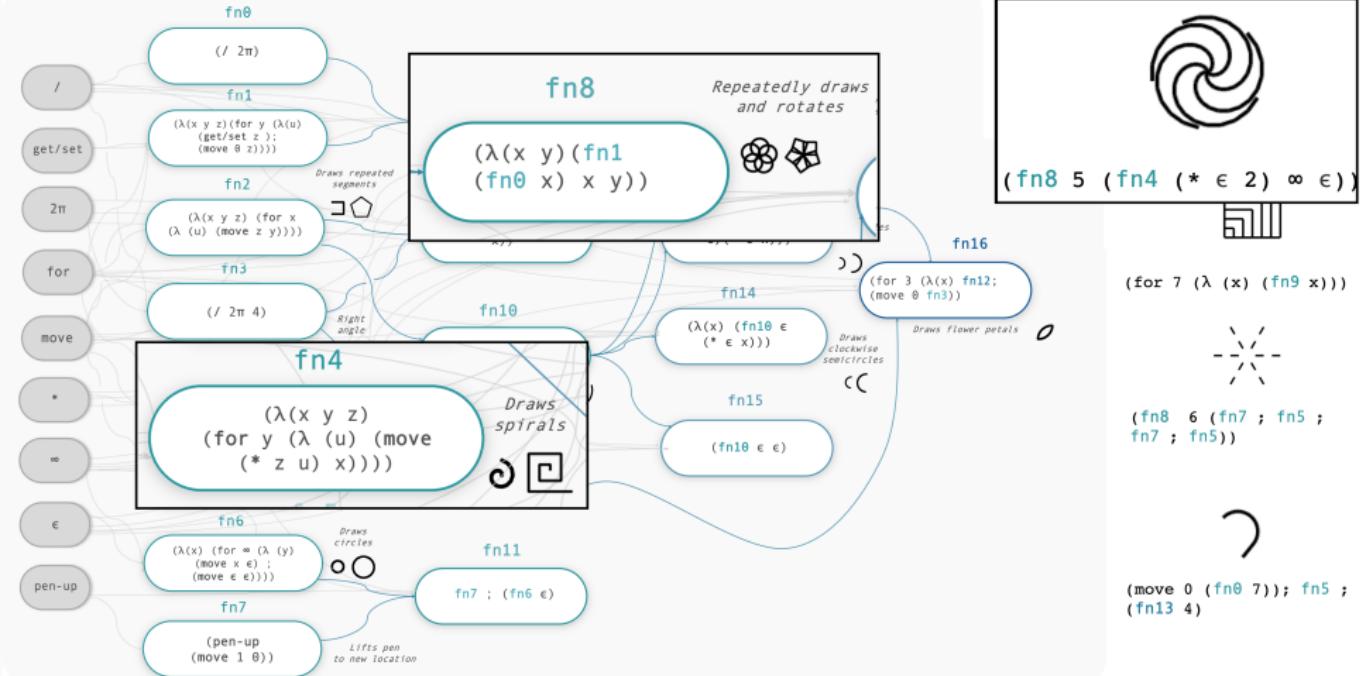


$(\text{fn8}\ 6\ (\text{fn7}\ ;\ \text{fn5}\ ;\ \text{fn7}\ ;\ \text{fn5}))$

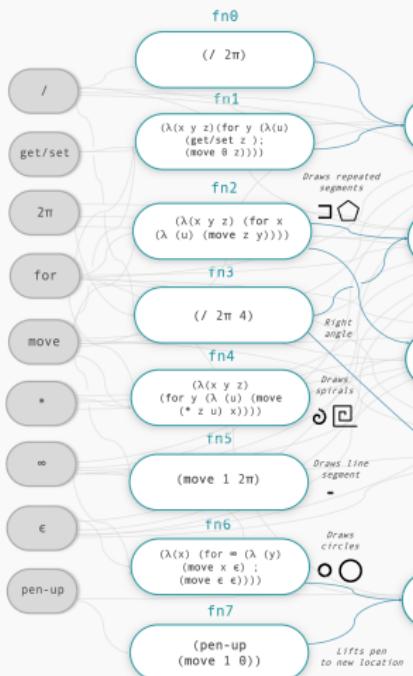


$(\text{move}\ 0\ (\text{fn0}\ 7));\ \text{fn5}\ ;\ (\text{fn13}\ 4)$

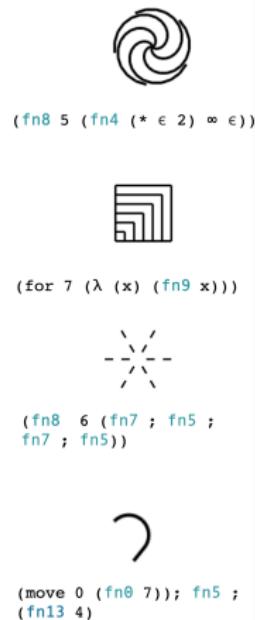
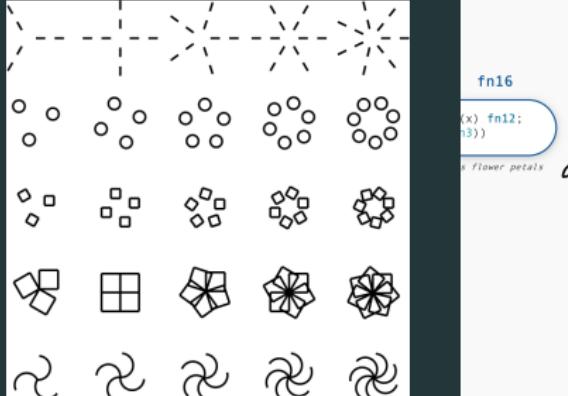
LOGO Turtle Graphics – learning an interpretable library



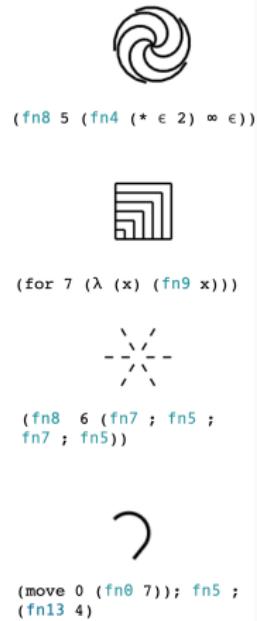
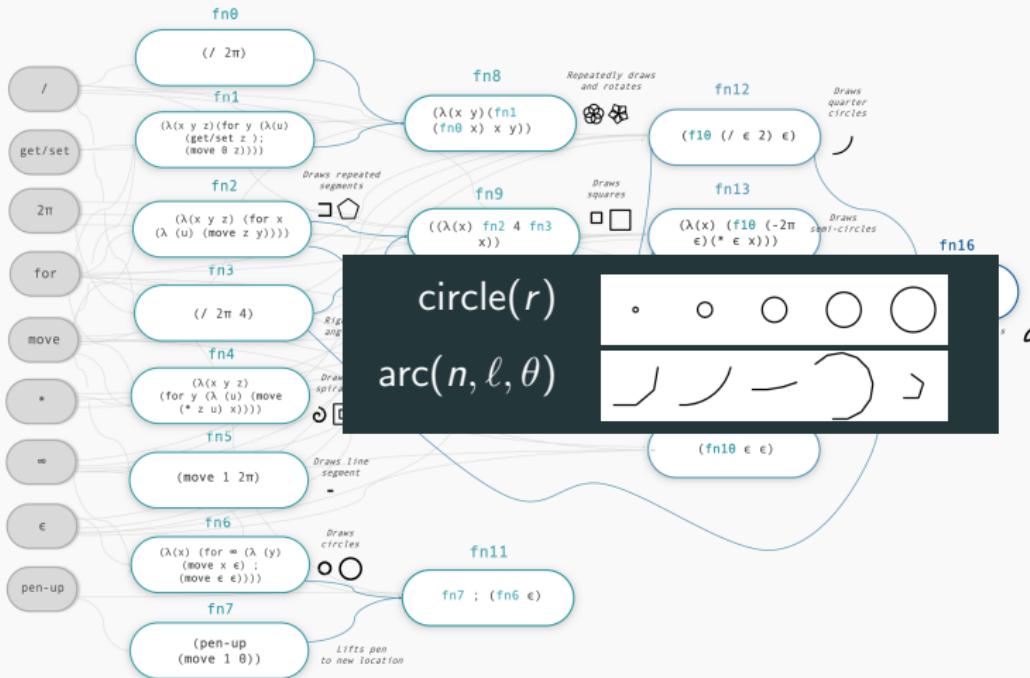
LOGO Turtle Graphics – learning an interpretable library



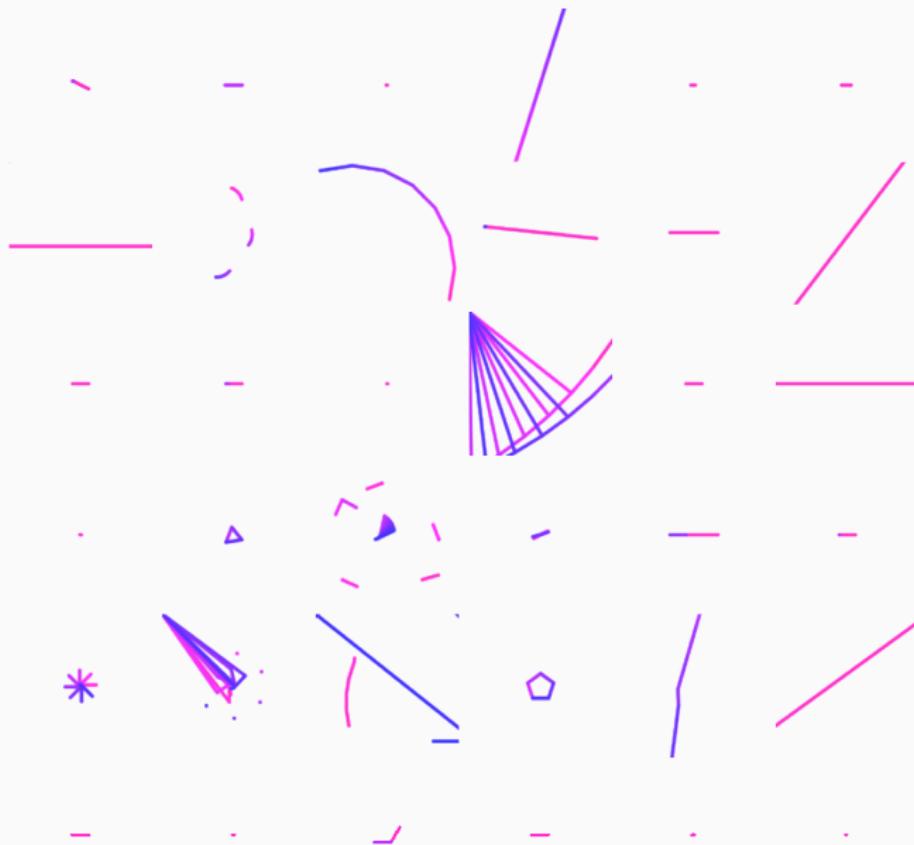
radial symmetry(n , body)



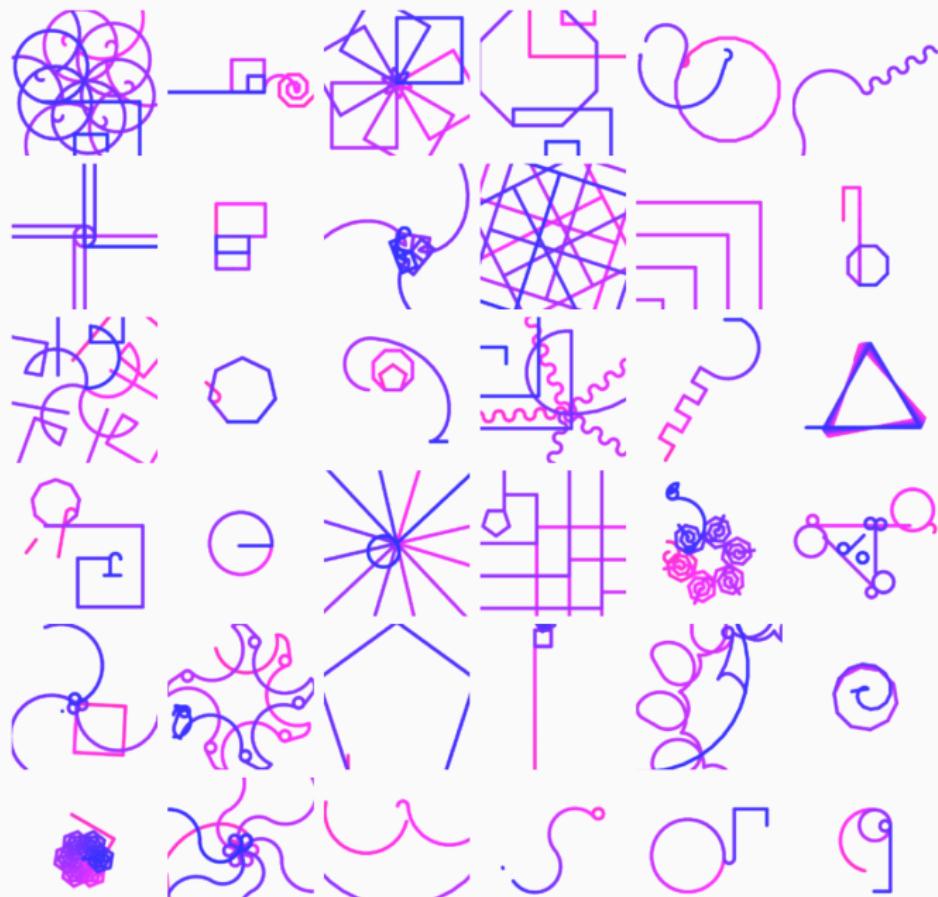
LOGO Turtle Graphics – learning an interpretable library



What does DreamCoder dream of? (before learning)

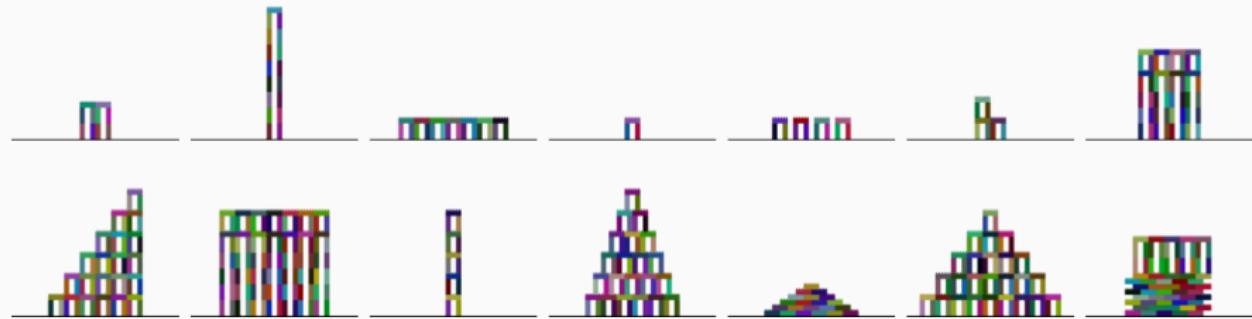


What does DreamCoder dream of? (after learning)



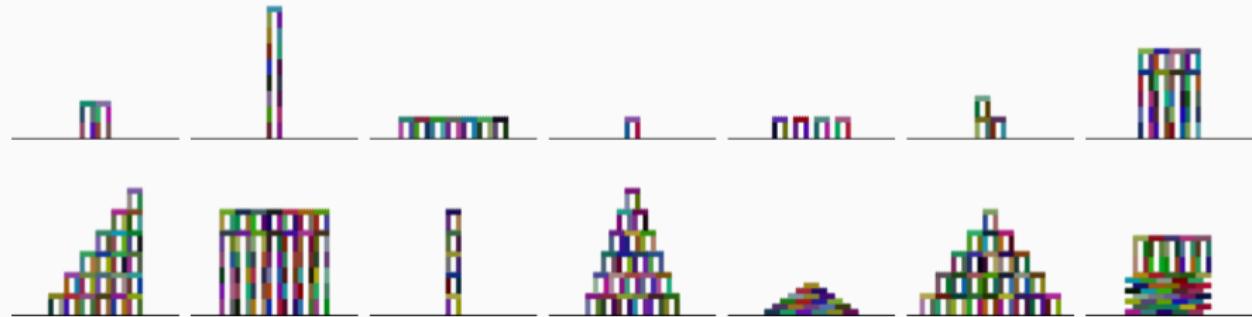
Planning to build towers

example tasks (112 total)



Planning to build towers

example tasks (112 total)

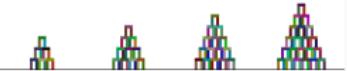


learned library routines (≈ 20 total)

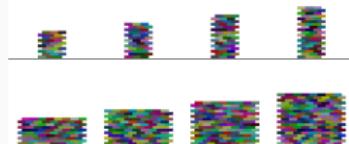
$\text{arch}(h)$



$\text{pyramid}(h)$



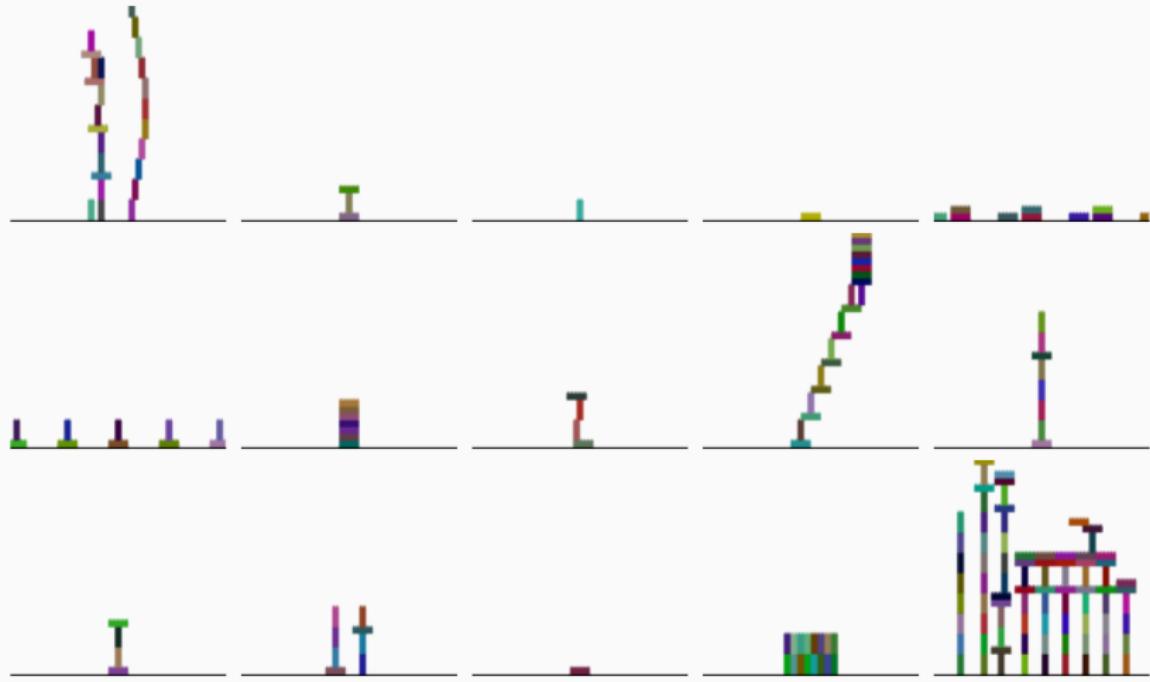
$\text{wall}(w, h)$



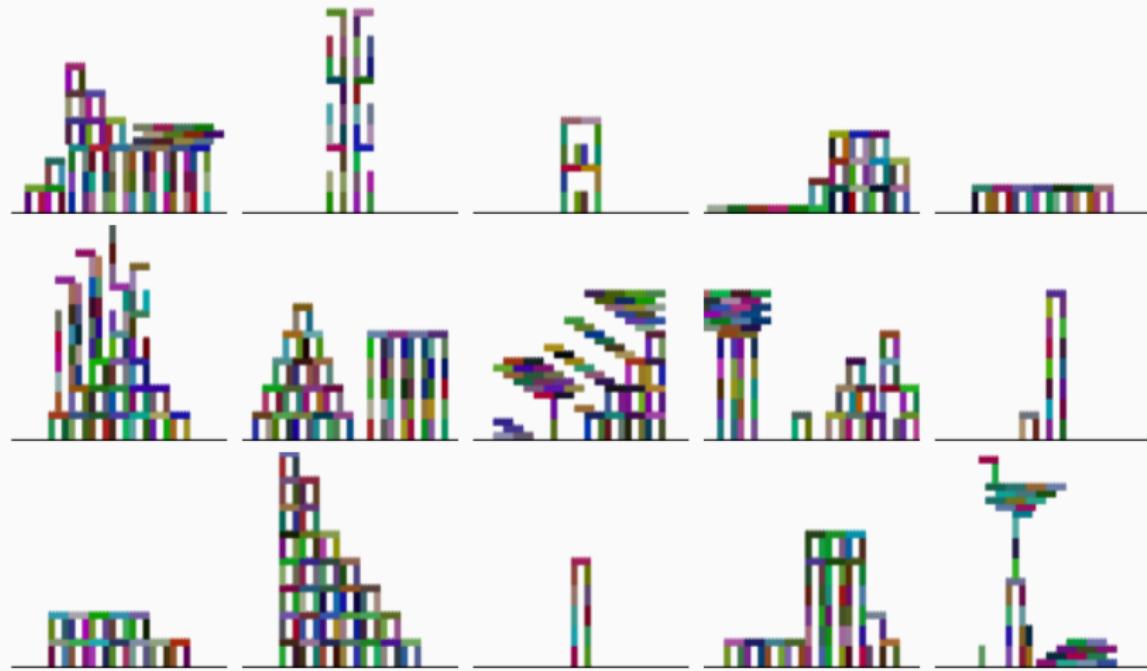
$\text{bridge}(w, h)$



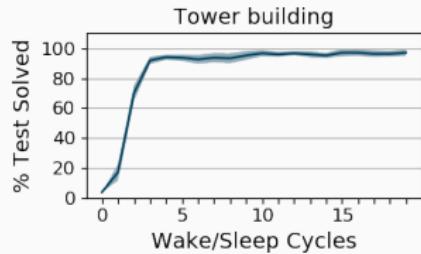
Dreams before learning



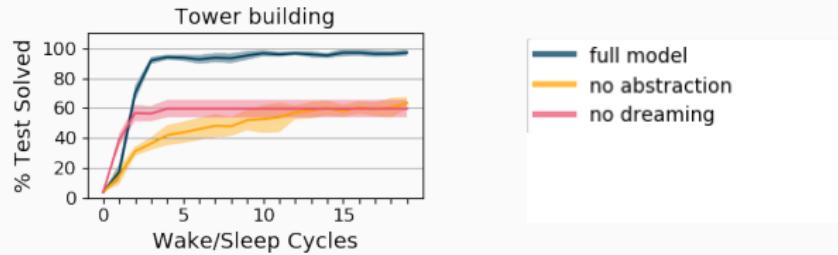
Dreams after learning



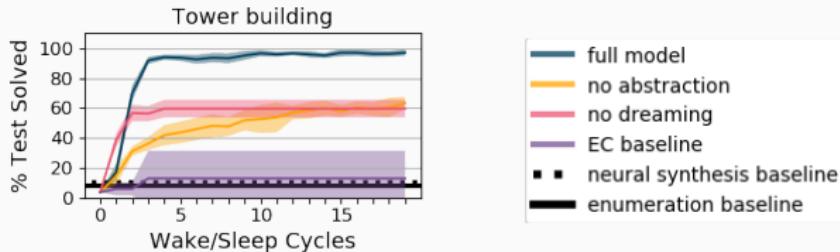
Learning dynamics



Learning dynamics

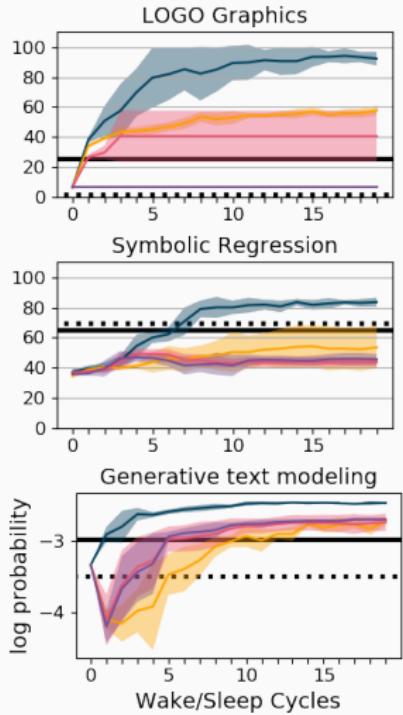
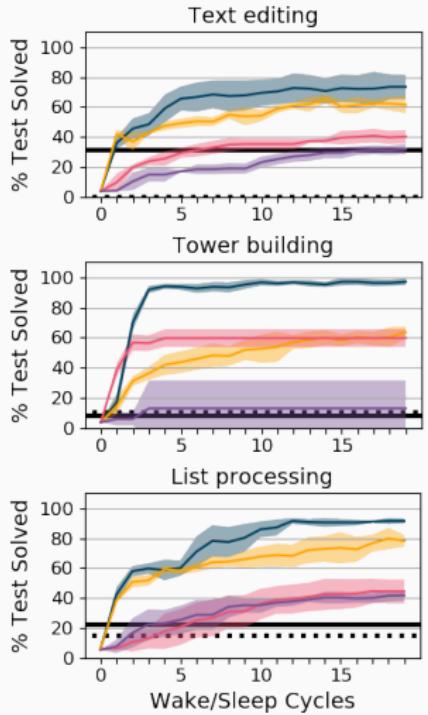


Learning dynamics



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

Learning dynamics



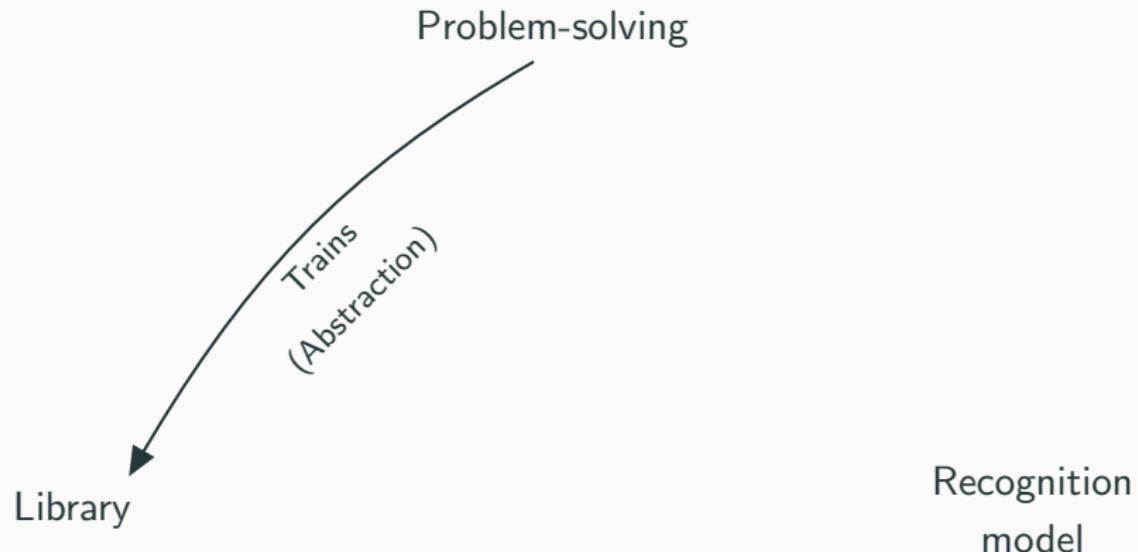
Synergy between recognition model and library learning

Problem-solving

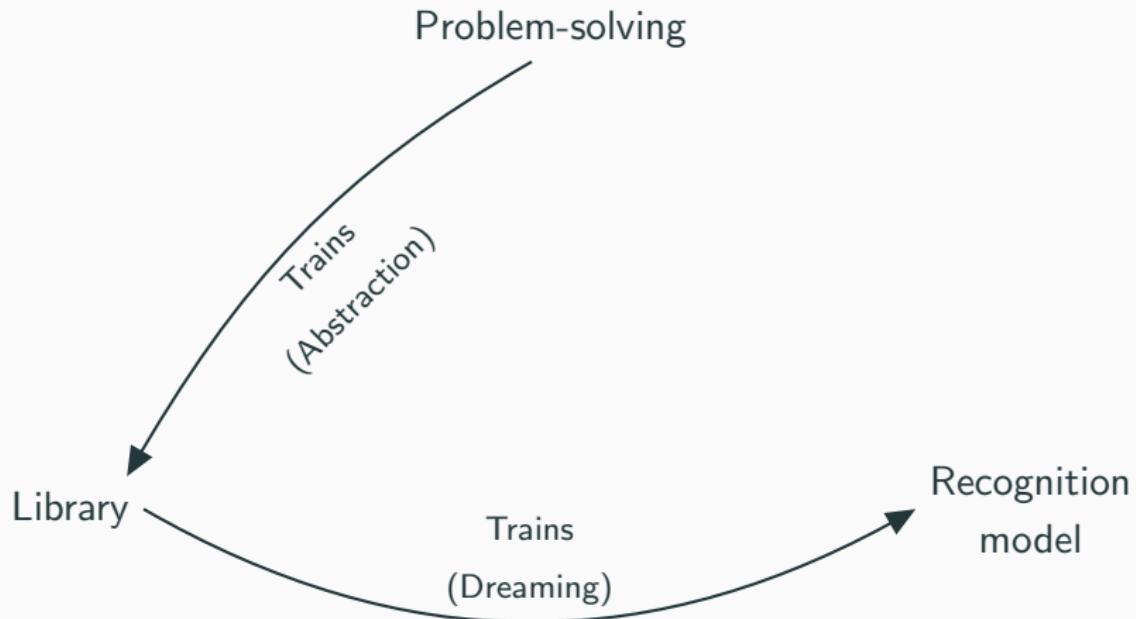
Library

Recognition
model

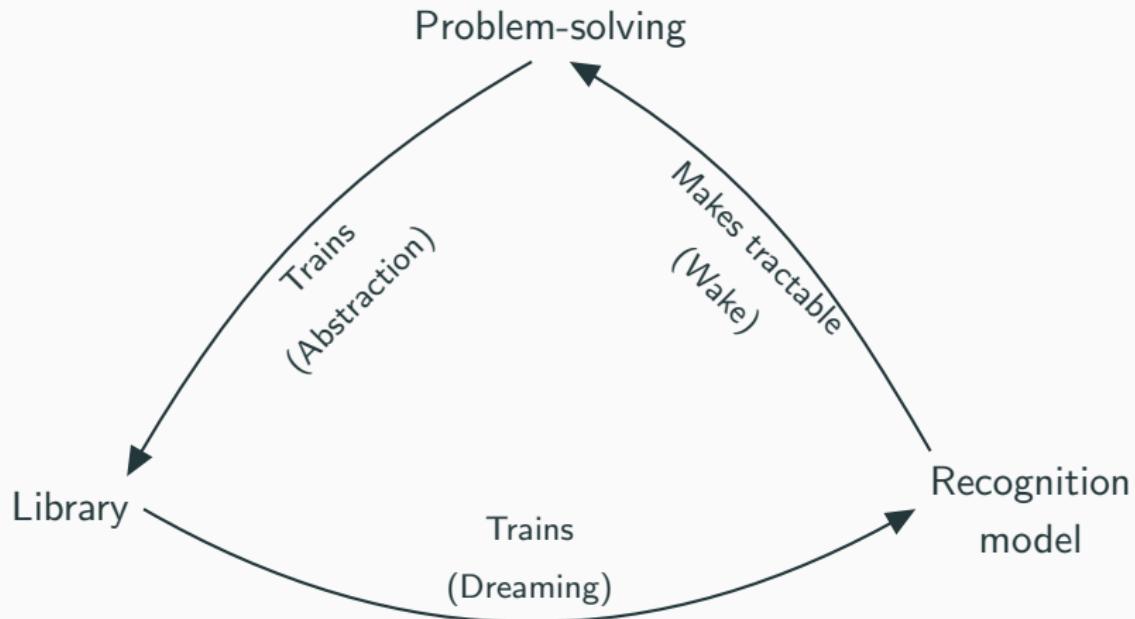
Synergy between recognition model and library learning



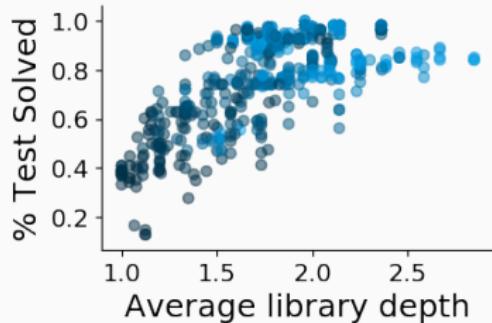
Synergy between recognition model and library learning



Synergy between recognition model and library learning



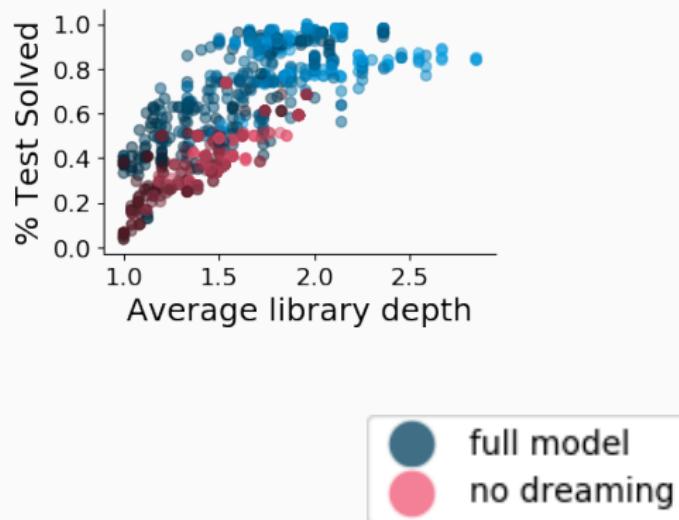
Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Brighter: Later in learning

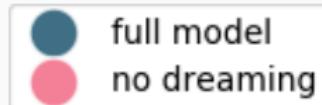
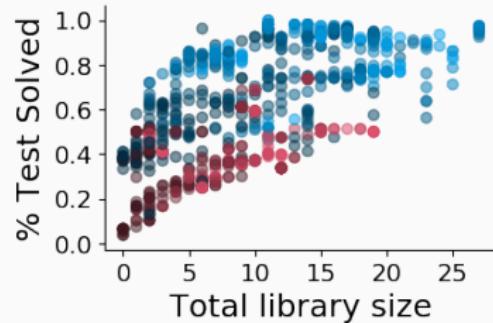
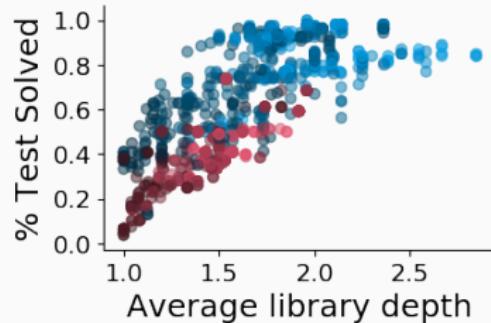
Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Brighter: Later in learning

Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Brighter: Later in learning

From learning libraries,
to learning languages

From learning libraries,
to learning languages

modern functional programming → physics

From learning libraries,
to learning languages

1950's Lisp → modern functional programming → physics

Physics Formula Sheet

Mechanics

$x = x_0 + v_{x0}t + \frac{1}{2}a_xt^2$	$a_t = \frac{v^2}{r}$	$ \vec{F}_{\text{spring}} = k \vec{x} $
$v = v_0 + at$	$\theta = \theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2$	$\text{PE}_{\text{spring}} = \frac{1}{2}kx^2$
$v_s^2 - v_{s0}^2 = 2a(x - x_0)$	$\omega = \omega_0 + \alpha t$	$T_{\text{spring}} = 2\pi \sqrt{\frac{m}{k}}$
$\bar{a} = \frac{\sum \vec{F}}{m} = \frac{\vec{F}_{\text{net}}}{m}$	$T = \frac{2\pi}{\omega} = \frac{1}{f}$	$T_{\text{pendulum}} = 2\pi \sqrt{\frac{L}{g}}$
$ \vec{F}_{\text{friction}} \leq \mu \vec{F}_{\text{Normal}} $	$v = f\lambda$	
$\bar{p} = m\bar{v}$	$x = A\cos(2\pi ft)$	$ \vec{F}_{\text{gravity}} = G \frac{m_1 m_2}{r^2}$
$\Delta \bar{p} = \vec{F} \Delta t$	$\bar{a} = \frac{\sum \vec{F}}{I} = \frac{\vec{F}_{\text{net}}}{I}$	$ \vec{F}_{\text{gravity}} = m\bar{g}$
$KE = \frac{1}{2}mv^2$	$\vec{r} = r \times F$	$\text{PE}_{\text{gravity}} = -G \frac{m_1 m_2}{r}$
$\Delta PE = mg\Delta y$	$L = I\omega$	$p = \frac{m}{V}$
$\Delta E = W = Fd\cos\theta$	$\Delta L = \tau \Delta t$	$KE = \frac{1}{2}I\omega^2$

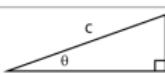
Electricity

$ \vec{F}_E = k \left \frac{q_1 q_2}{r^2} \right $	$\Delta V = IR$	$R = \frac{\rho l}{A}$
$I = \frac{\Delta q}{\Delta t}$		$P = I\Delta V$
$R_{\text{series}} = R_1 + R_2 + \dots + R_n$	$\frac{1}{R_{\text{parallel}}} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}$	

Geometry

Rectangle	$A = bh$	Rectangular Solid	$V = lwh$	Triangle	$A = \frac{1}{2}bh$
Circle	$A = \pi r^2$	Cylinder	$V = \pi r^2 l$	Sphere	$V = \frac{4}{3}\pi r^3$
	$C = 2\pi r$		$S = 2\pi rl + 2\pi r^2$		$S = 4\pi r^2$

Trigonometry



$$\begin{array}{ccccccc} & c & & a & & & \\ & \diagdown & & \square & & & \\ & \theta & & & & & \end{array}$$

$$c^2 = a^2 + b^2 \quad \sin\theta = \frac{a}{c} \quad \cos\theta = \frac{b}{c} \quad \tan\theta = \frac{a}{b}$$

Variables

a = acceleration
 A = amplitude
 A = Area
 b = base length
 C = circumference
 d = distance
 E = energy
 f = frequency
 F = force
 h = height
 I = current
 I = rotational inertia
 KE = kinetic energy
 k = spring constant
 L = angular momentum
 l = length
 m = mass
 P = power
 p = momentum
 q = charge
 r = radius
 R = resistance
 S = surface area
 T = period
 t = time
 PE = potential energy
 V = electric potential
 V = volume
 v = velocity
 w = width
 W = work
 x = position
 y = height
 α = angular acceleration
 λ = wavelength
 μ = coefficient of friction

Growing languages for vector algebra and physics

Initial Primitives

map
zip

cons

empty

cdr

power

fold

car

+

-

*

/

0

1

π

Physics Equations

Newton's Second Law

$$\vec{a} = \frac{1}{m} \sum_l \vec{F}_l$$

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

$$|\vec{F}| = q |\vec{v} \times \vec{B}|$$

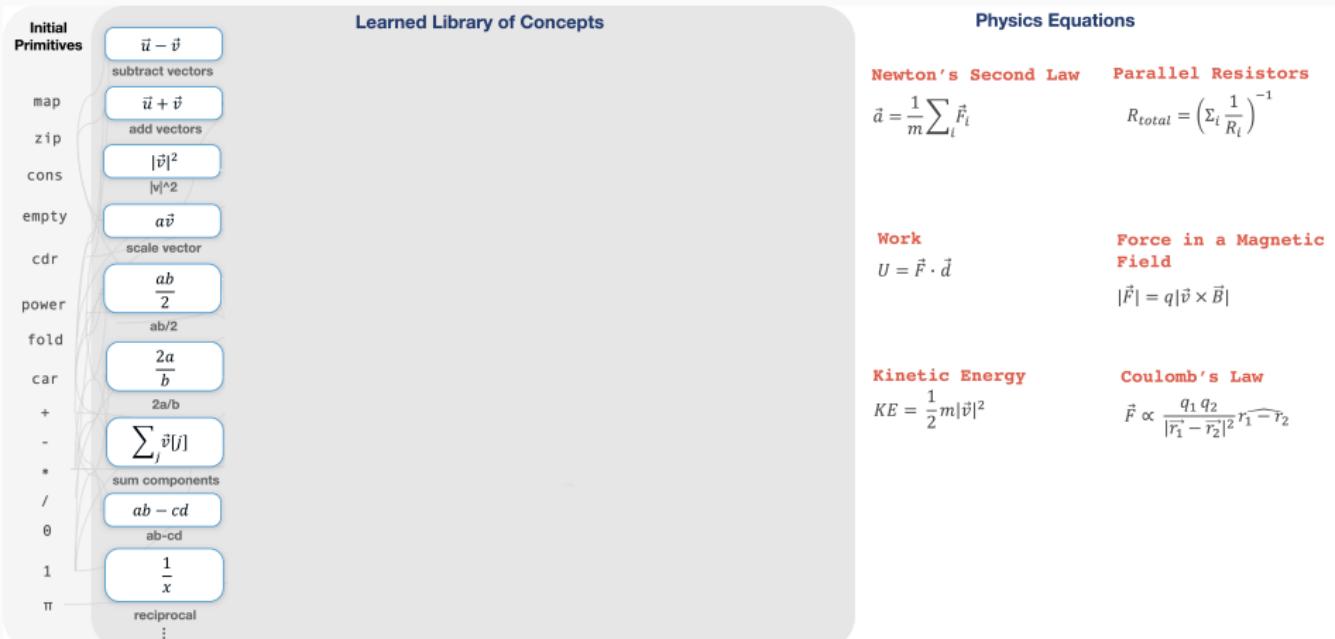
Kinetic Energy

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

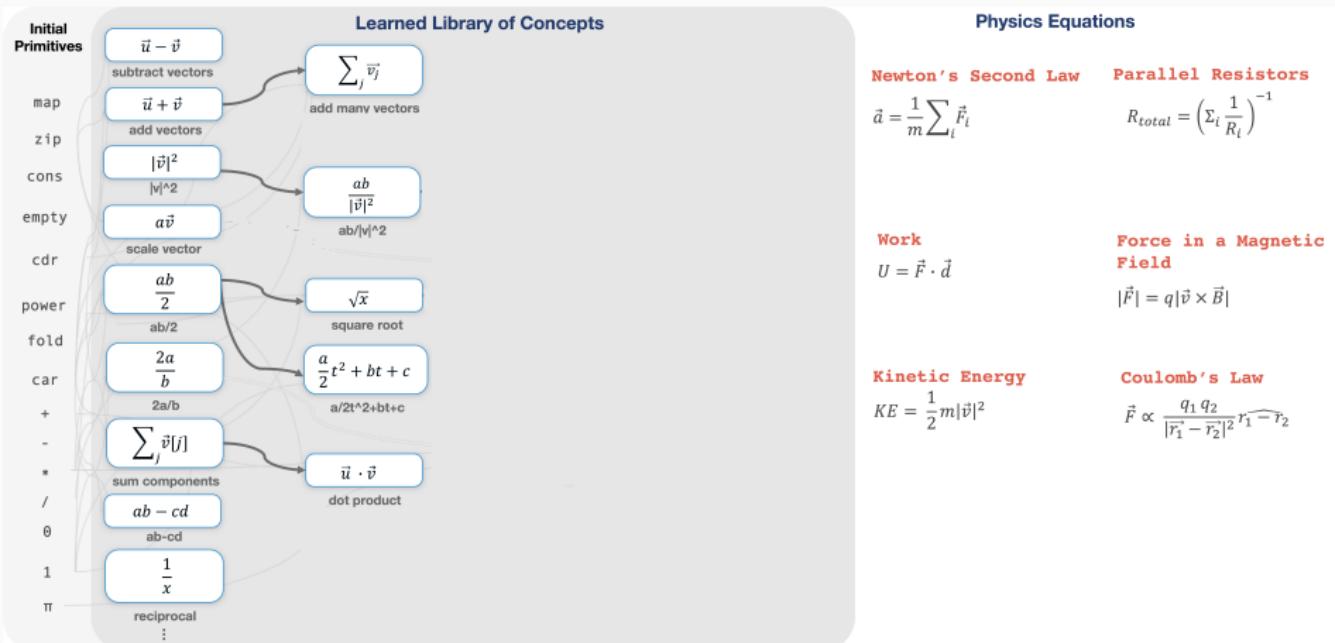
Coulomb's Law

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{r}_1 - \hat{r}_2$$

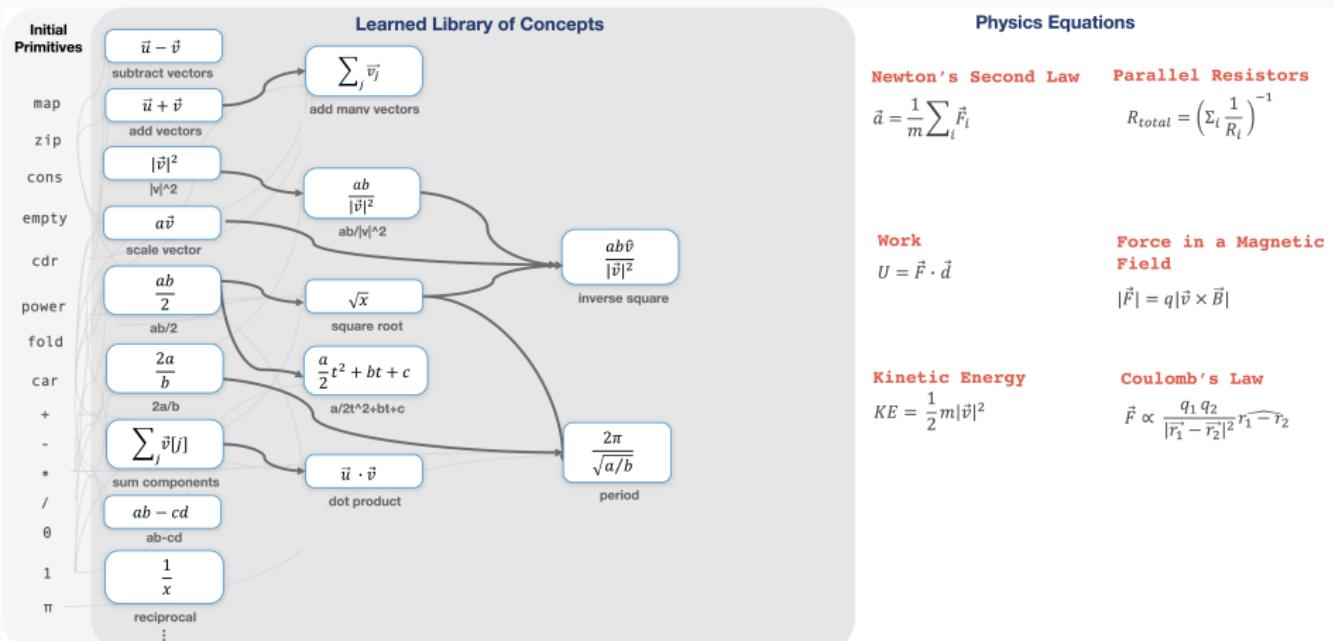
Growing languages for vector algebra and physics



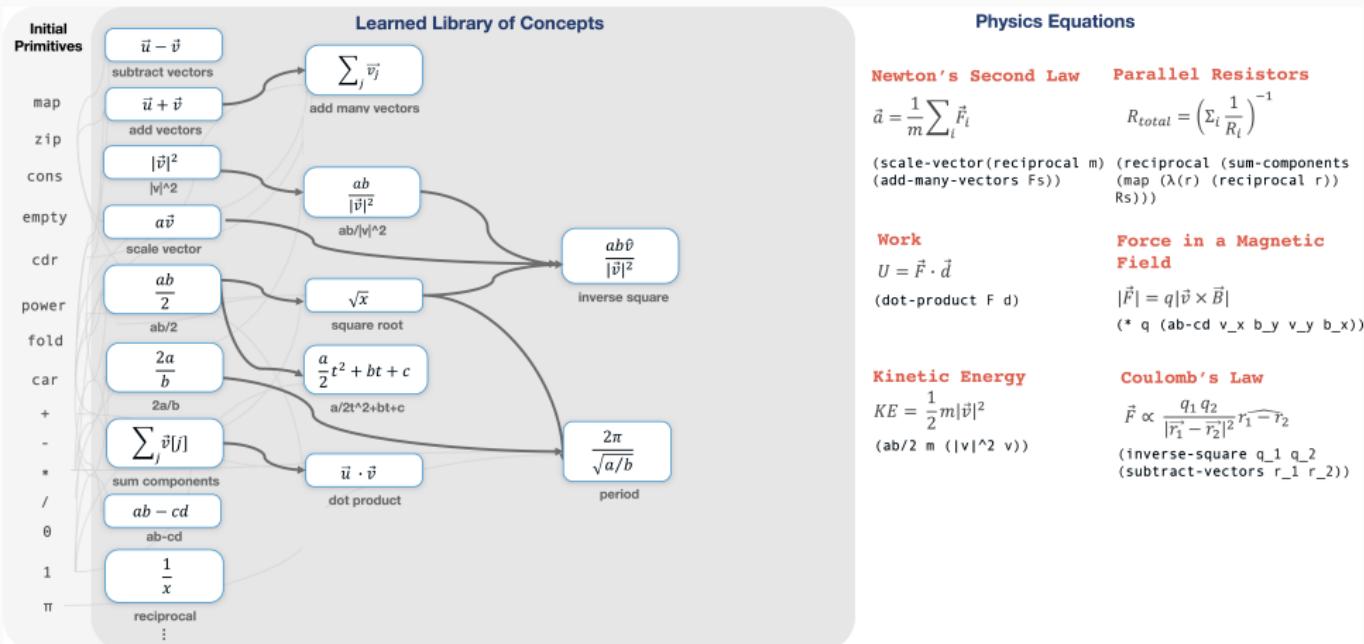
Growing languages for vector algebra and physics



Growing languages for vector algebra and physics



Growing languages for vector algebra and physics



Growing languages for vector algebra and physics

Initial
Primitives

$\vec{u} - \vec{v}$
subtract vectors

map

$\vec{u} + \vec{v}$
add vectors

zip

cons

empty

cdr

power

fold

car

+

-

*

/

0

1

π

Learned Library of Concepts

$\sum_j \vec{v}_j$

add many vectors

$|\vec{v}|^2$

$|\vec{v}|^2$

$a\vec{v}$

scale vector

$\frac{ab}{2}$

$ab/2$

$\frac{2a}{b}$

$2a/b$

$\sum_j \vec{v}[j]$

sum components

$ab - cd$

$ab - cd$

$\frac{1}{x}$

reciprocal

:

\sqrt{x}

square root

$\frac{a}{2}t^2 + bt + c$

$a/2t^2 + bt + c$

$\vec{u} \cdot \vec{v}$

dot product

$\frac{ab\hat{v}}{|\vec{v}|^2}$

inverse square

$\frac{2\pi}{\sqrt{a/b}}$

period

Physics Equations

Newton's Second Law

$$\vec{a} = \frac{1}{m} \sum_l \vec{F}_l$$

$$R_{total} = \left(\sum_i \frac{1}{R_i} \right)^{-1}$$

Parallel Resistors

Work

$$U = \vec{F} \cdot \vec{d}$$

(dot-product F d)

Force in a Magnetic Field

$$|\vec{F}| = q|\vec{v} \times \vec{B}|$$

(* q (ab-cd v_x b_y v_y b_x))

Kinetic Energy

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

(ab/2 m (|v|^2 v))

Coulomb's Law

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \widehat{\vec{r}_1 - \vec{r}_2}$$

(inverse-square q_1 q_2
(subtract-vectors r_1 r_2))

(lambda (x y z u) (map (lambda (v) (* (/ (* (power (/ (* x x) (fold (zip z u (lambda (w a) (- w a)))) theta (lambda (b c) (+ (* b b) c)))) (/ (* 1 1) (+ 1 1))) y) (fold (zip z u (lambda (d e) (- d e))) theta (lambda (f g) (+ (* f f) g)))) v)) (zip z u (lambda (h i) (- h i)))))

Solution to Coulomb's Law if expressed in initial primitives

Growing a language for recursive programming

Initial Primitives

Y
combinator
cons
car
cdr
nil
if
nil?
+
-
0
1
=

Recursive Programming Algorithms

Stutter

[] → []
[] → []

Take every other

[] → []
[] → []

List lengths

[, []] → [3 1]
[[], [], []] → [2 0 1]

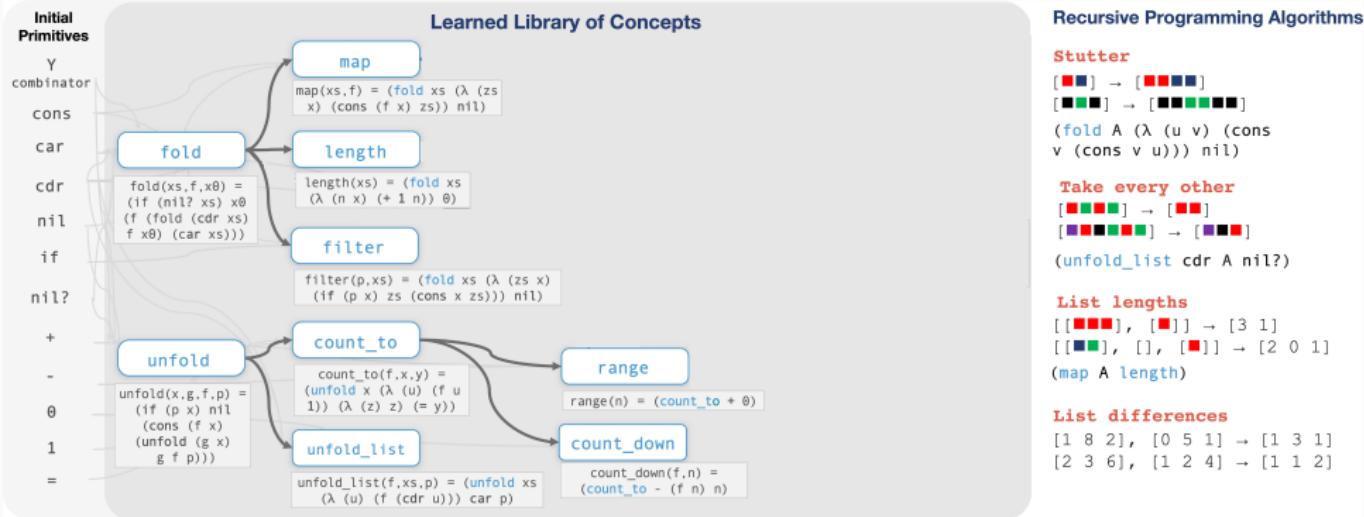
List differences

[1 8 2], [0 5 1] → [1 3 1]
[2 3 6], [1 2 4] → [1 1 2]

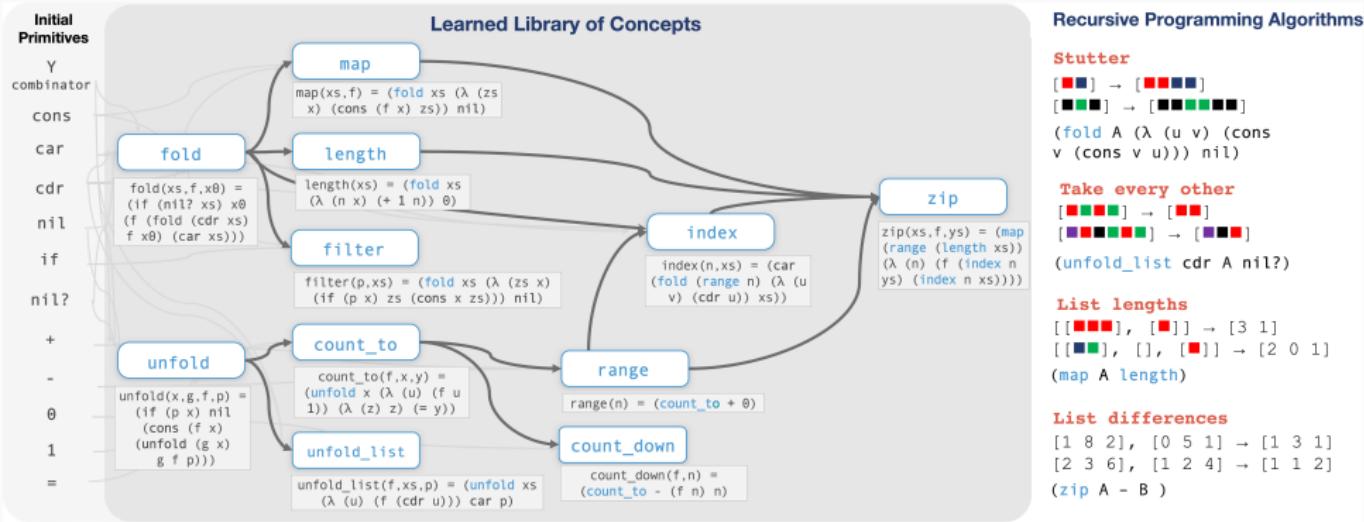
Growing a language for recursive programming



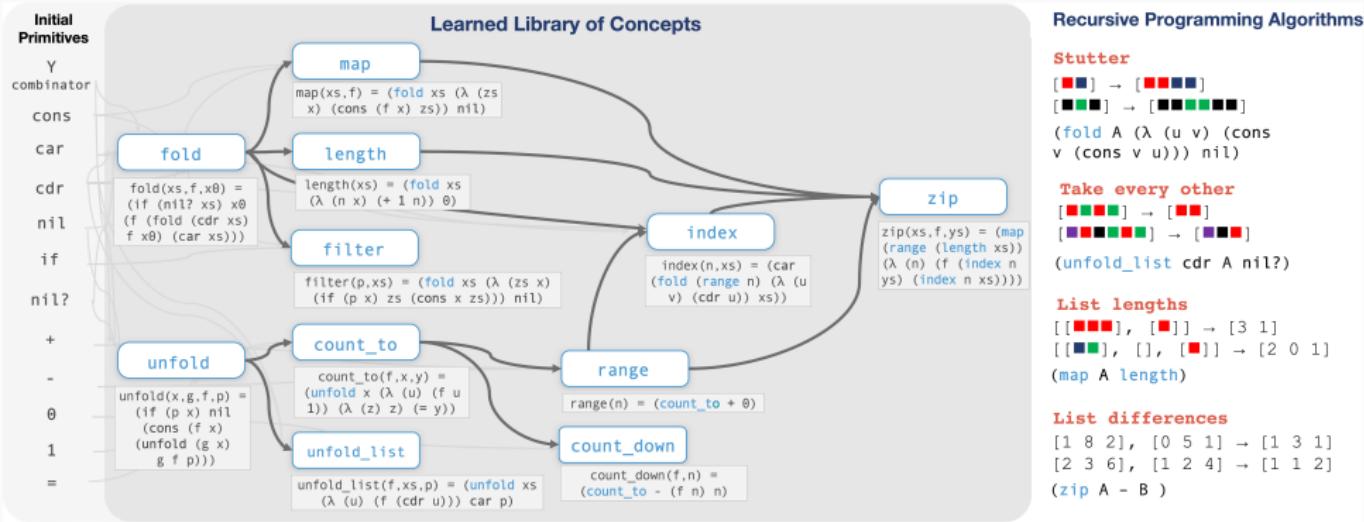
Growing a language for recursive programming



Growing a language for recursive programming

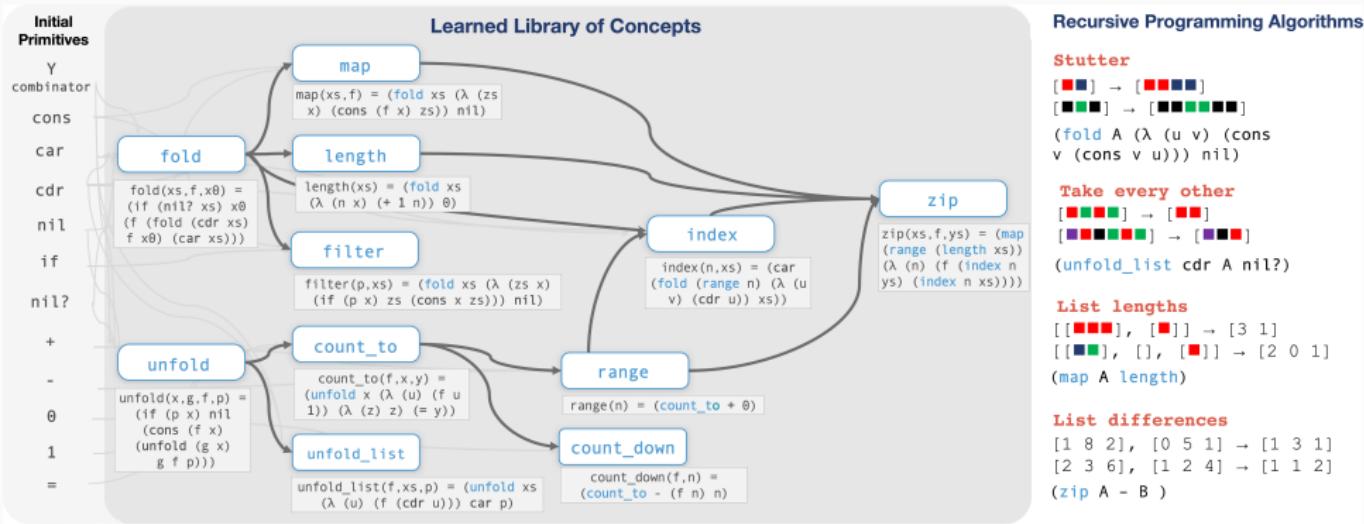


Growing a language for recursive programming

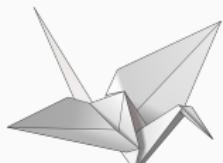


Origami Programming: Jeremy Gibbons, 2003

Growing a language for recursive programming



1 year of compute. 5 days on 64 CPUs.



Origami Programming: Jeremy Gibbons, 2003

Lessons

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

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

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

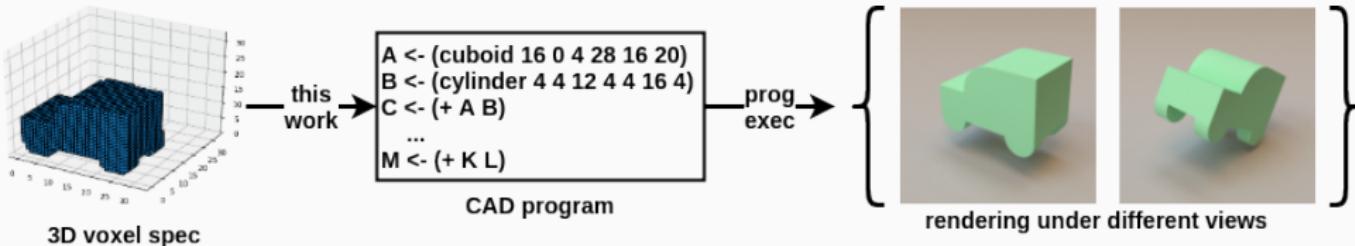


Lucas Morales



thank
you

3D program induction



Challenge: combinatorial search!

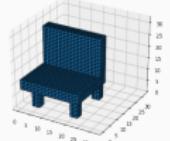
Branching factor: > 1.3 million per line of code, ≈ 20 lines of code
search space size: $(1.3 \text{ million})^{20} \approx 10^{122}$ programs

Ellis*, Nye*, Pu*, Sosa*, Tenenbaum, Solar-Lezama. NeurIPS 2019.

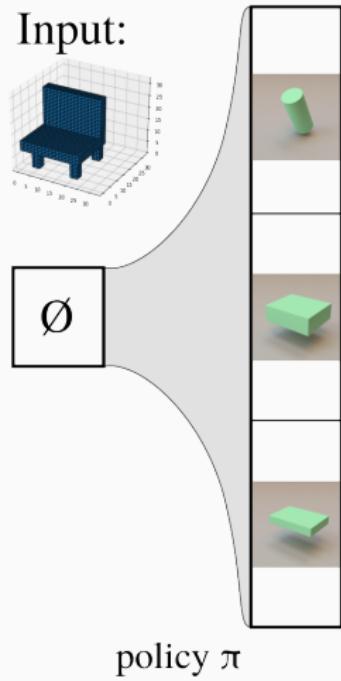
*equal contribution

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]

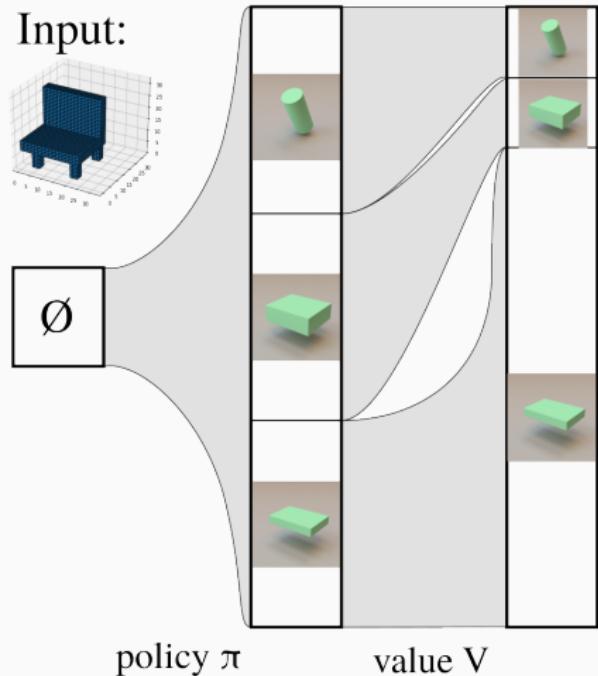
Input:



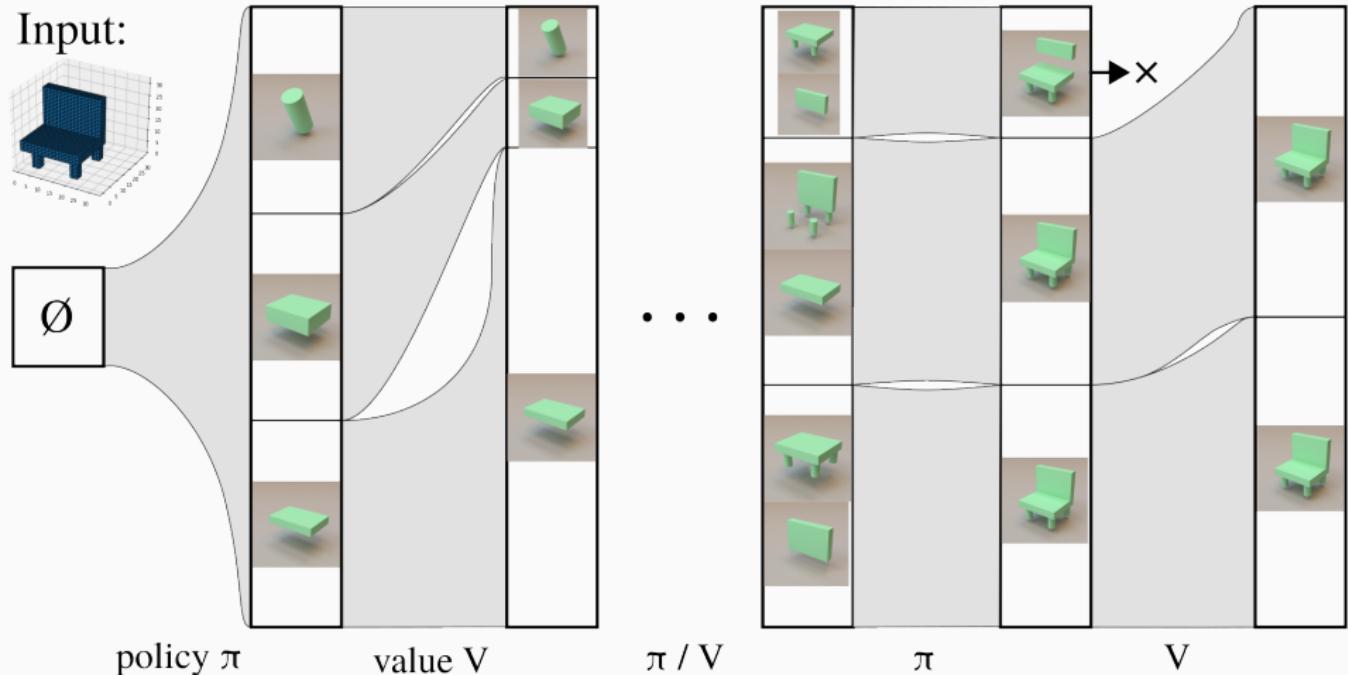
Solution: stochastic **tree search** + learn **policy** that writes code
+ learn **value** function that assesses execution of program so far;
analogous to **AlphaGo** [Silver et al. 2016]



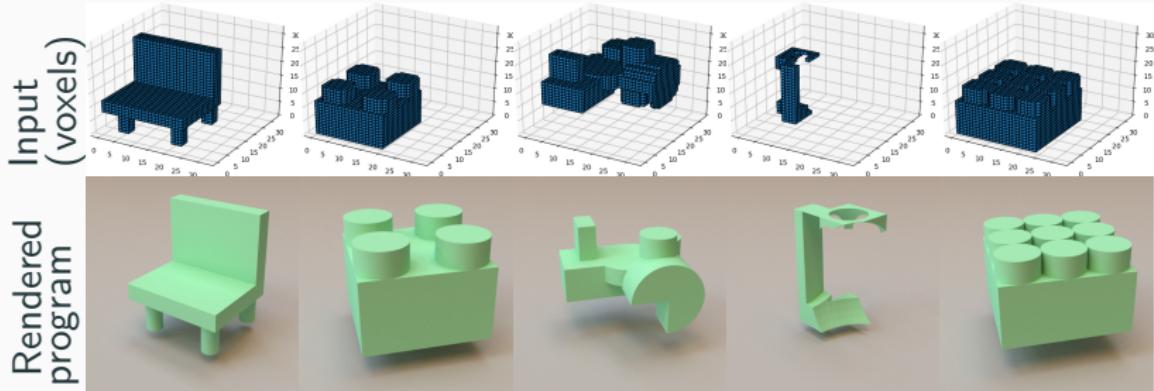
Solution: stochastic **tree search** + learn **policy** that writes code
+ learn **value** function that assesses execution of program so far;
analogous to **AlphaGo** [Silver et al. 2016]



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]



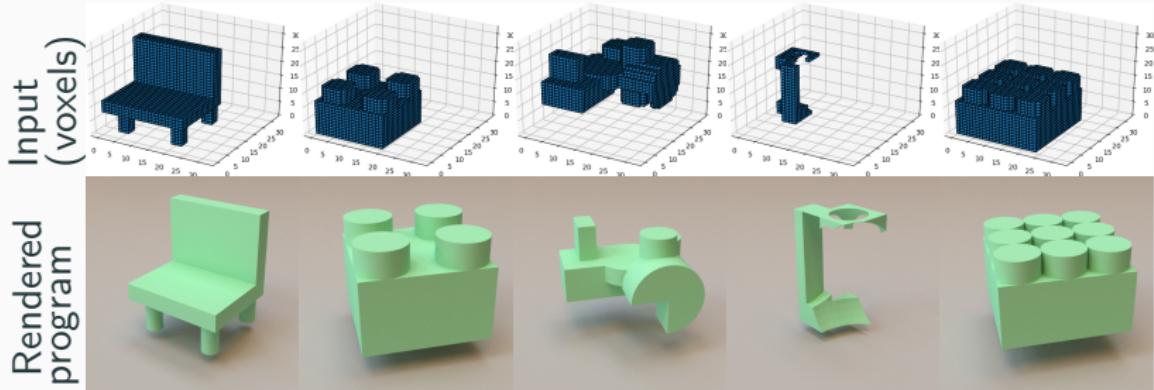
3D program induction



Ellis*, Nye*, Pu*, Sosa*, Tenenbaum, Solar-Lezama. NeurIPS 2019.

*equal contribution

3D program induction



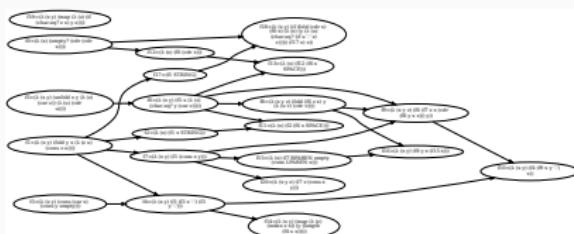
same architecture learns to synthesize text editing programs
(FlashFill, Gulwani 2012)

Ellis*, Nye*, Pu*, Sosa*, Tenenbaum, Solar-Lezama. NeurIPS 2019.

*equal contribution

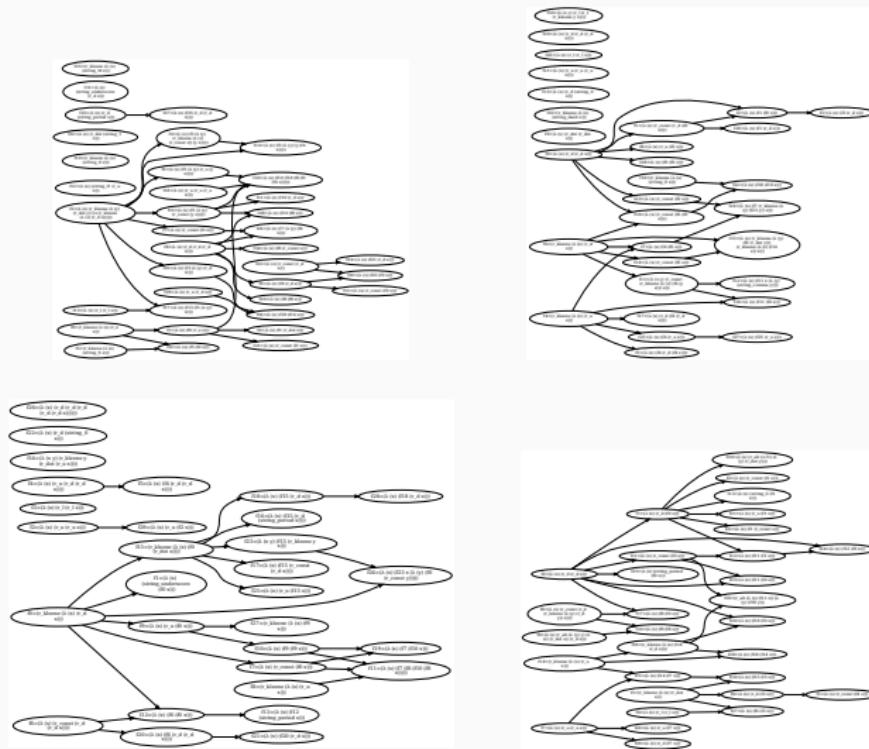
Library structure: Text Editing

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

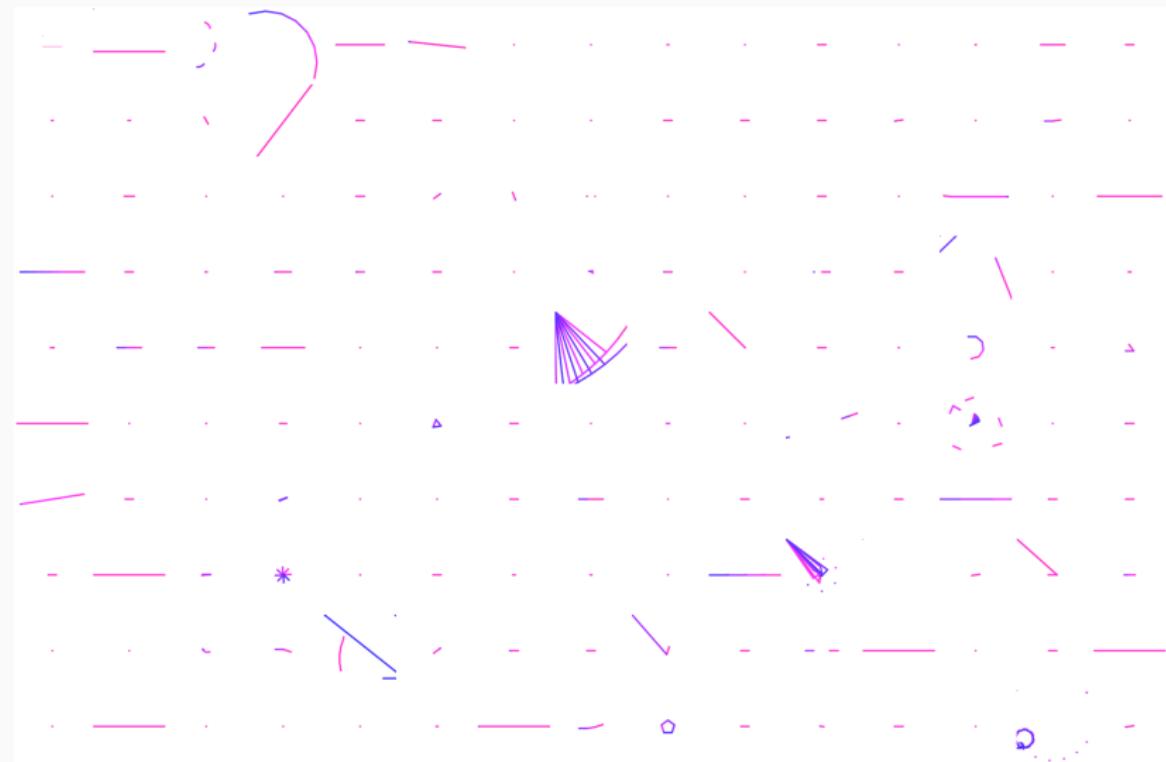


Library structure: Generating Text

Libraries for probabilistic generative models over text:
data from crawling web for CSV files



150 random dreams before learning



150 random dreams after learning

