

DreamCoder:

# **Bootstrapping Inductive Program Synthesis with Wake-Sleep Library Learning**

---

Kevin Ellis (speaker), Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer,  
Lucas Morales, Luke Hewitt, Luc Cary,  
Armando Solar-Lezama, Joshua B. Tenenbaum

May 28, 2021

PLDI 2021

# Inductive program synthesis

## FlashFill (Gulwani 2012)

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

<i>Input</i> $v_1$	<i>Output</i>
<i>Company\Code\index.html</i>	<i>Company\Code\</i>
<i>Company\Docs\Spec\specs.doc</i>	<i>Company\Docs\Spec\</i>

*String Program:*

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

# Inductive program synthesis

## 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

# Inductive program synthesis

## FlashFill (Gulwani 2012)

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

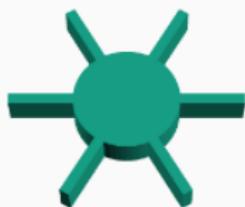
Input $v_1$	Output
Company\Code\index.html	Company\Code\
Company\Docs\Spec\specs.doc	Company\Docs\Spec\

String Program:

```
SubStr(v1, CPos(0), Pos(SlashTok, ε, -1))
```

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

## Szalinski (Nandi 2020)



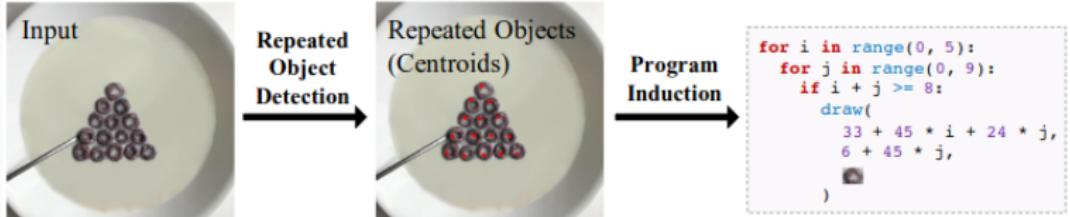
(a) CAD model of ship's wheel

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

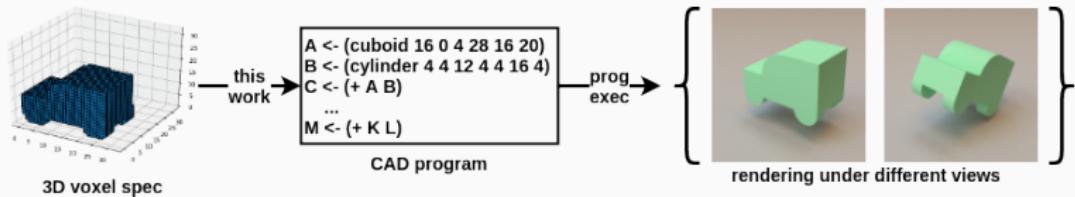
(b) Caddy program

op ::= + | - | × | /      num ::= R | <var> | <num> <op> <num>  
vec2 ::= [(num), (num)]      vec3 ::= [(num), (num), (num)]  
affine ::= Translate | Rotate | Scale | TranslateSpherical  
binop ::= Union | Difference | Intersection  
cad ::= (Cuboid <vec3>) | (Sphere <num>)  
| (Cylinder <vec2>) | (HexPrism <vec2>) | ...  
| ((affine) <vec3>) (cad)  
| ((binop) <cad> <cad>)  
| (Fold <binop> <cad-list>)  
cad-list ::= (List <cad>+)  
| (Concat <cad-list>+)  
| (Tabulate (<var> Z\*)+ <cad>)  
| (Map2 <affine> <vec3-list> <cad-list>)  
vec3-list ::= (List <vec3>+)

# Visual programs



Mao\*, Zhang\*, et al 2019



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



**for**  $i = 1..3$   
    **for**  $j = 1..1$   
        **draw**( $i*2$ ,  $j*1$ , )  
        \*\*\*

partial image  $x_{part}$

synthesized program  $P_{part}$

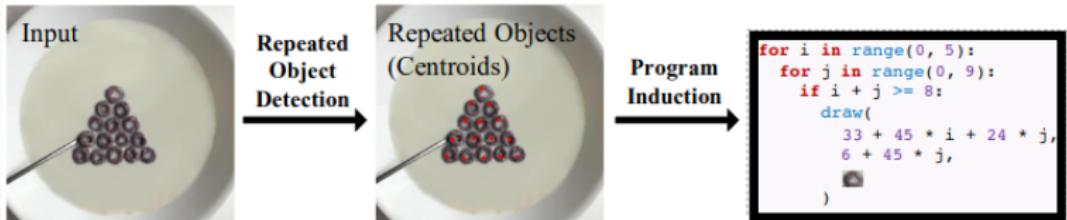


Draw("Top", "Circle", position, geometry)  
  
**for**( $i < 2$ , "translation", a)  
    **for**( $j < 2$ , "translation", b)  
        Draw("Leg", "Cub", position +  $i*a + j*b$ ,  
  
**for**( $i < 2$ , "translation", c)  
        Draw("Layer", "Rec", position +  $i*c$ , geomet

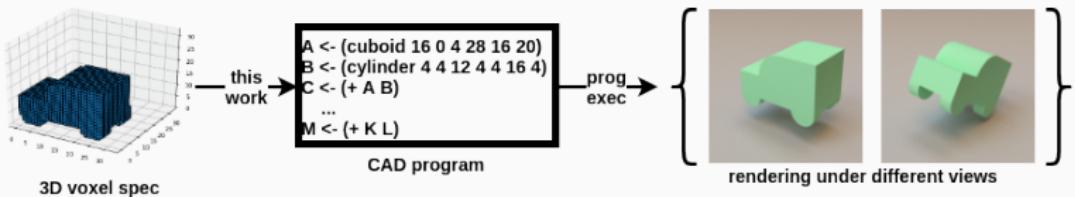
Young et al 2019

Tian et al 2019

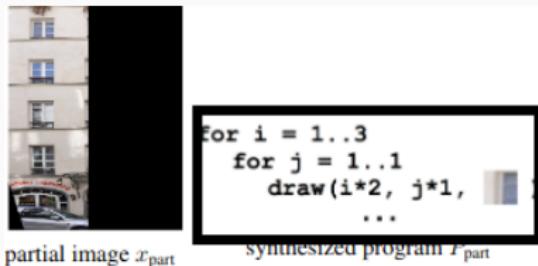
# Where does this language come from?



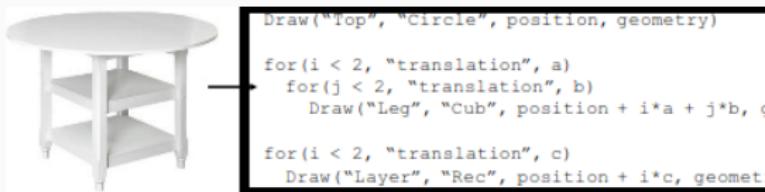
Mao\*, Zhang\*, et al 2019



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

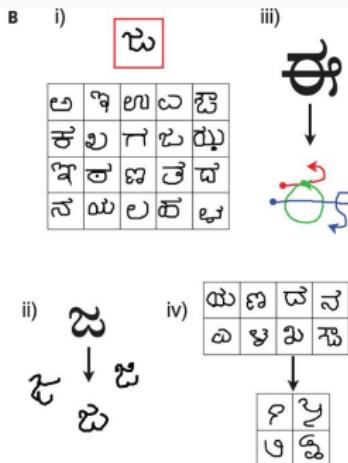


Young et al 2019



Tian et al 2019

# Human-level Program induction



한국어로 된 예제입니다. 각각의 단계에서 어떤 규칙이나 패턴을 찾고 적용하는지 보여주는 예시입니다.

DreamCoder and learning to learn

- learning a library
- learning to search
- synergy between library+learned search

## Learning to write code

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

- Library of components
- Search strategy (synthesis algorithm)

# 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

:

:

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]

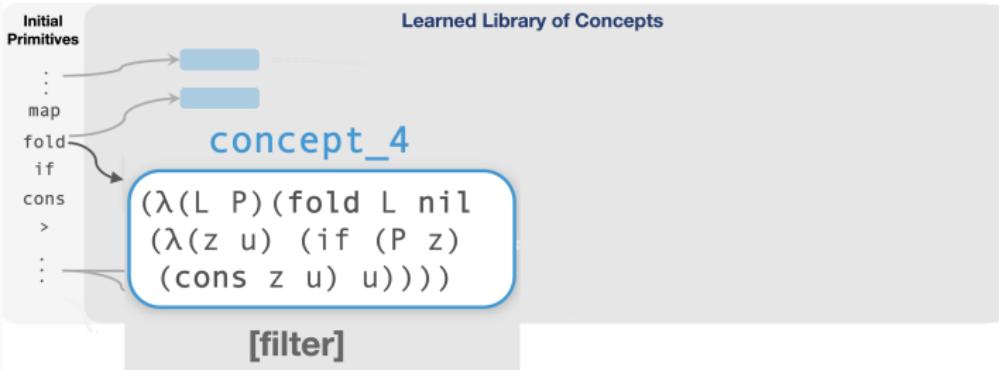
[6 2 2 3 8 5] → [2 2 3 5 6 8]

...

# Library learning



# 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]  
...

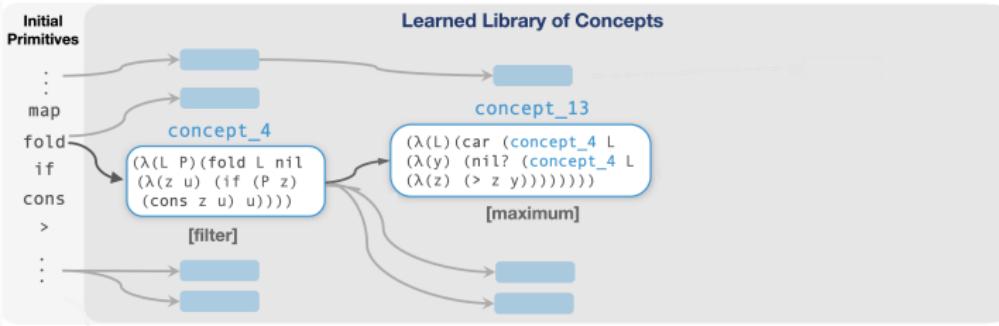
# Library learning



## 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]  
...

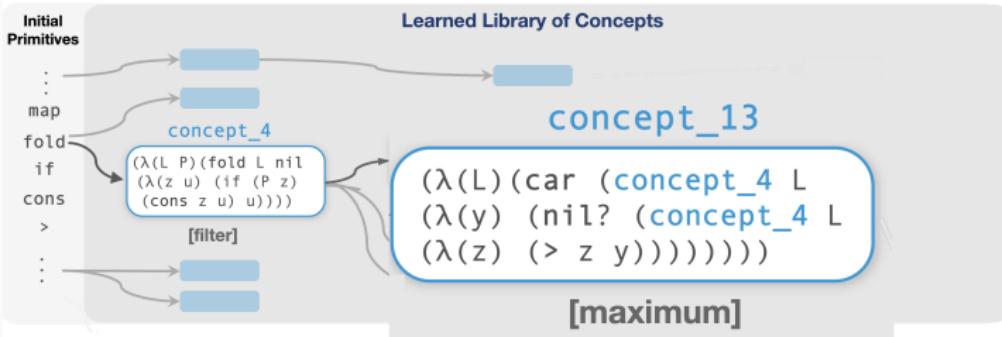
# Library learning



## 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]  
...

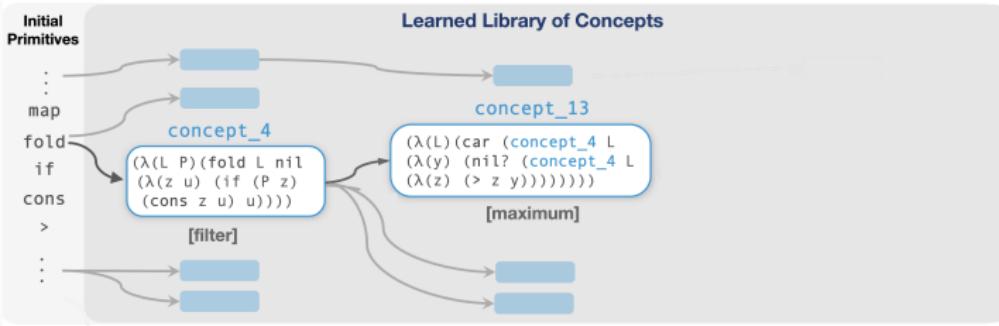
# Library learning



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]  
...

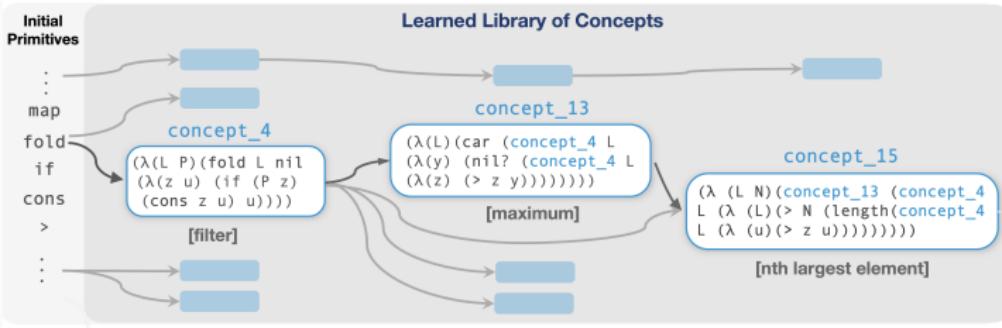
# Library learning



## 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]  
...

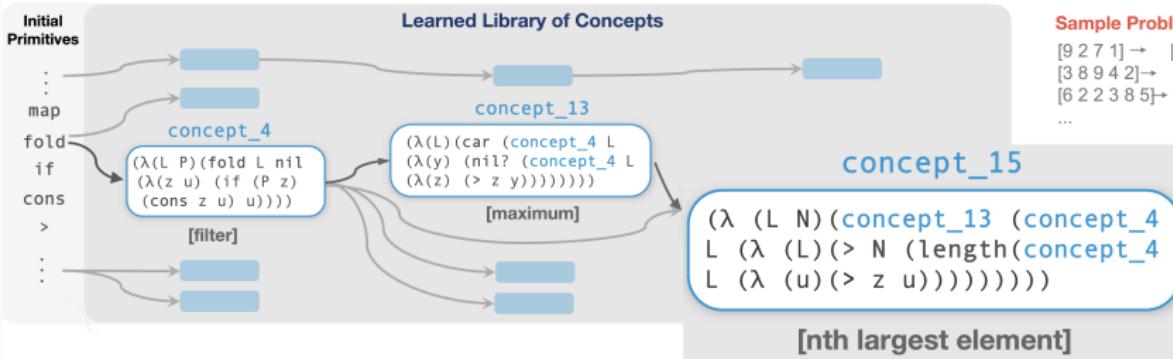
# Library learning



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]  
...

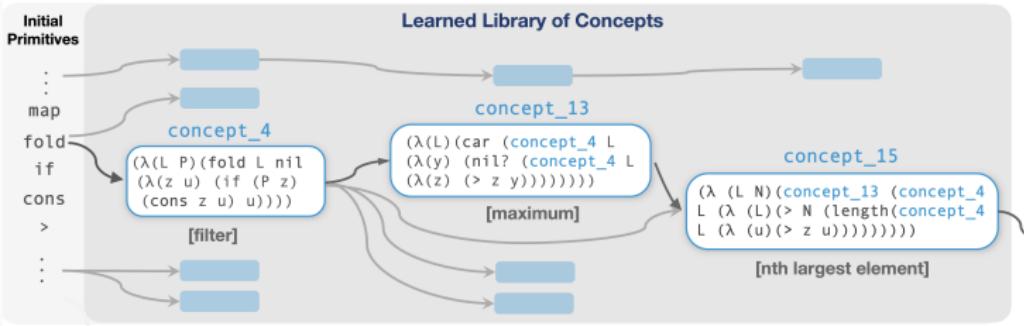
# Library learning



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



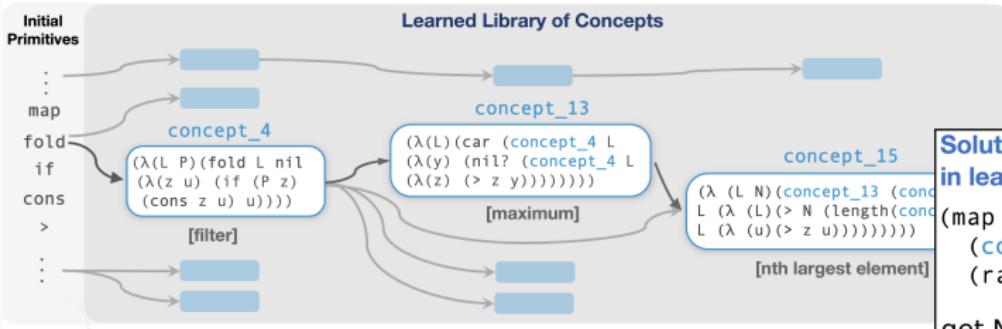
## Sample Problem: sort list

[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]  
...

## Solution to sort list discovered in learned language:

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

# Library learning



Sample Problem: sort list

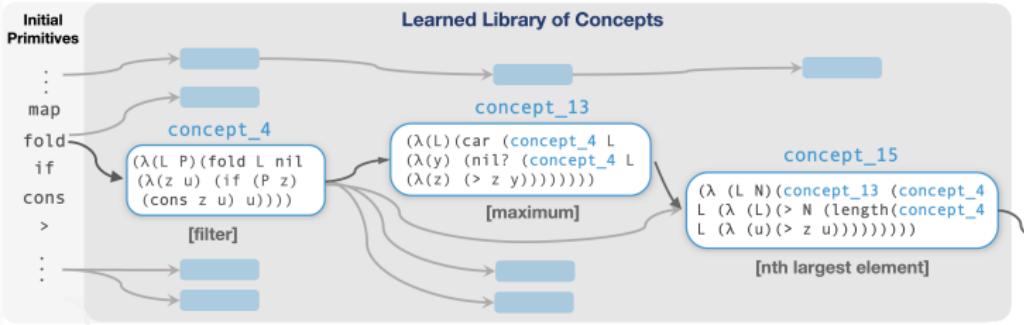
[9 2 7 1] → [1 2 7 9]  
[3 8 9 4 2] → [2 3 4 8 9]  
[6 2 2 3 8 5] → [2 2 3 5 6 8]  
...

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, ...

# 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:

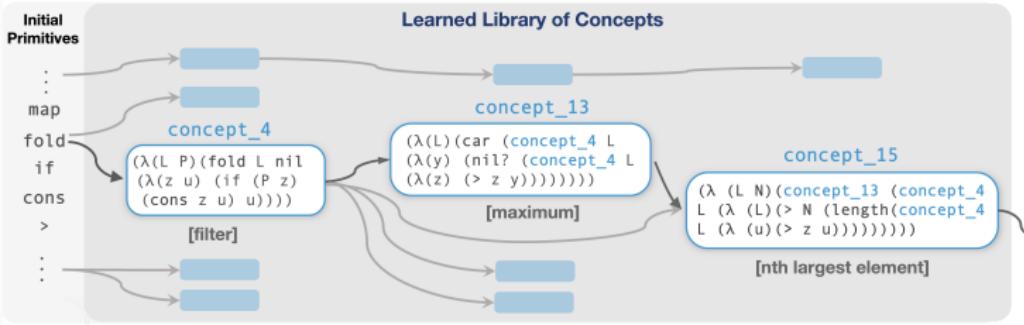
```
(map (λ (n)
  (concept_15 L (+ 1 n)))
  (range (length L)))
```

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



## 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:

```
(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:

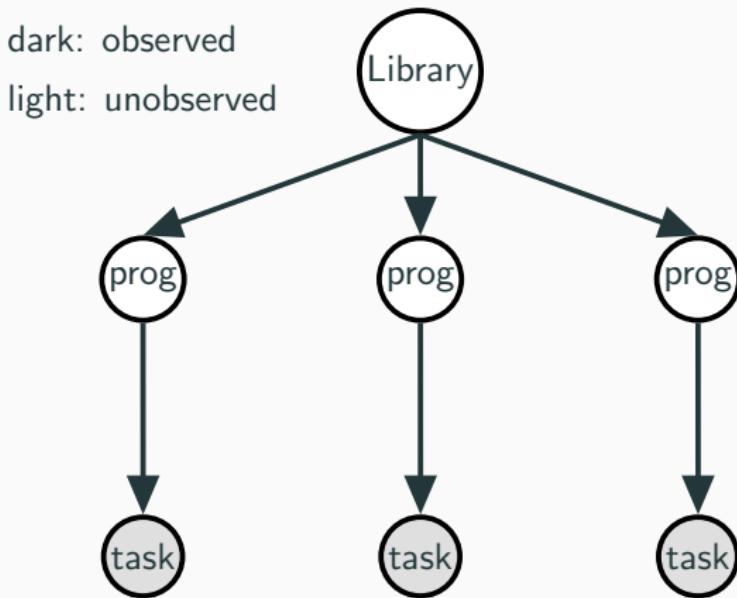
```
(lambda (x) (map (lambda (y) (car (fold (fold x nil (lambda (z u) (if (gt? (+ y 1) (length
(fold x nil (lambda (v w) (if (gt? z v) (cons v w) w)))))) (cons z u) u))) nil (lambda (a b) (if
(nil? (fold (fold x nil (lambda (c d) (if (gt? (+ y 1) (length (fold x nil (lambda (e f) (if
(gt? c e) (cons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

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

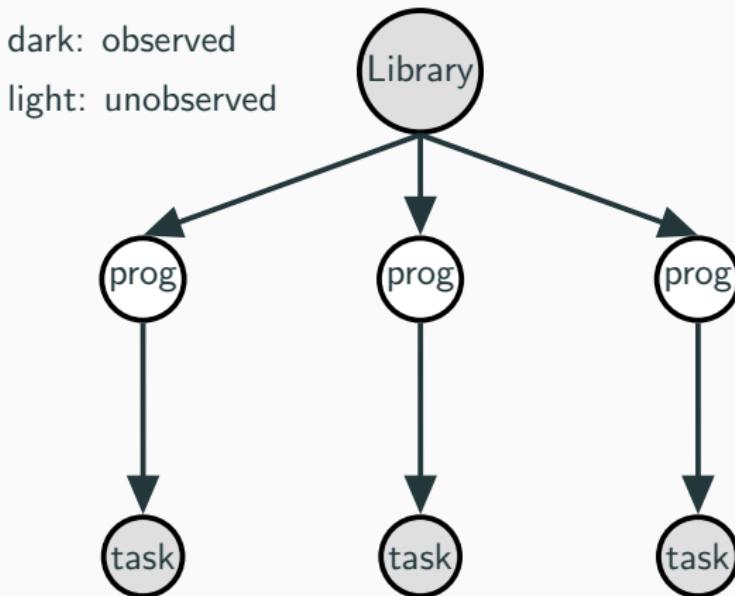
cf. Helmholtz machine, wake/sleep neural network training algorithms

# Library learning as Bayesian inference



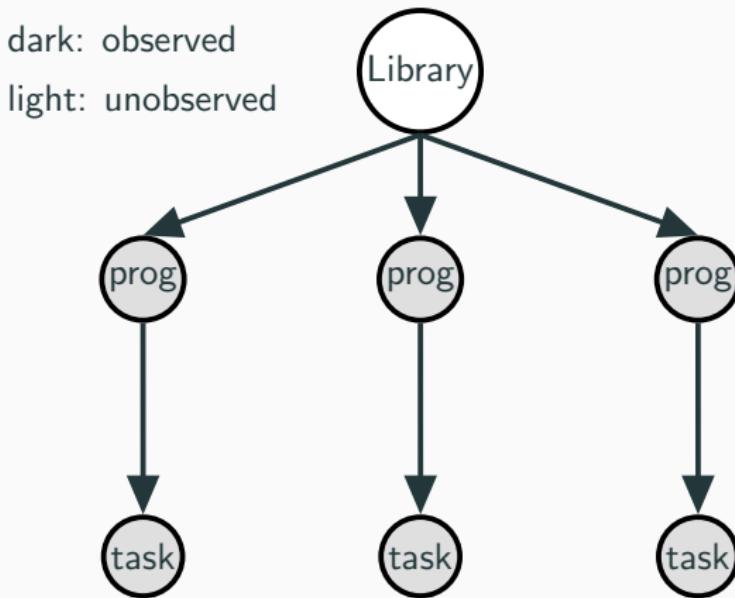
[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

# Library learning as Bayesian inference



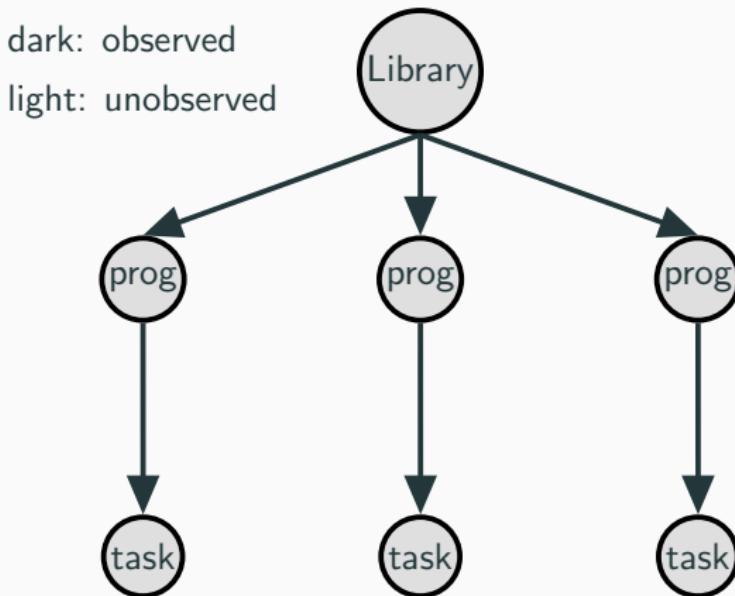
[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

# Library learning as Bayesian inference



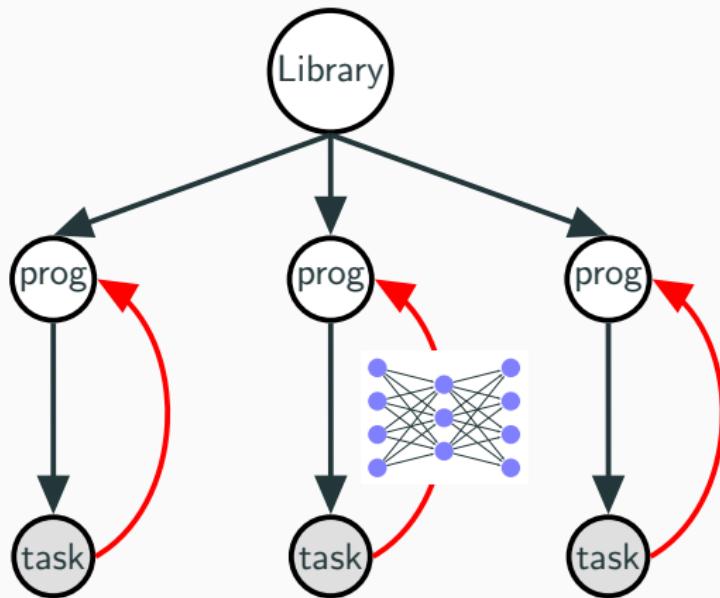
[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

# Library learning as Bayesian inference

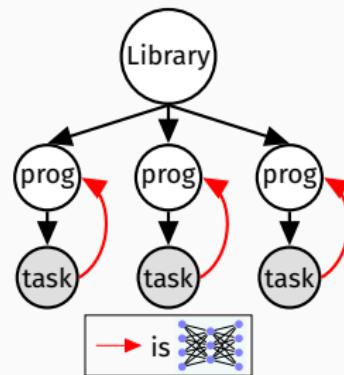


[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

# 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])

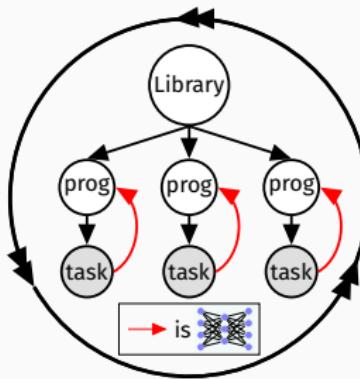


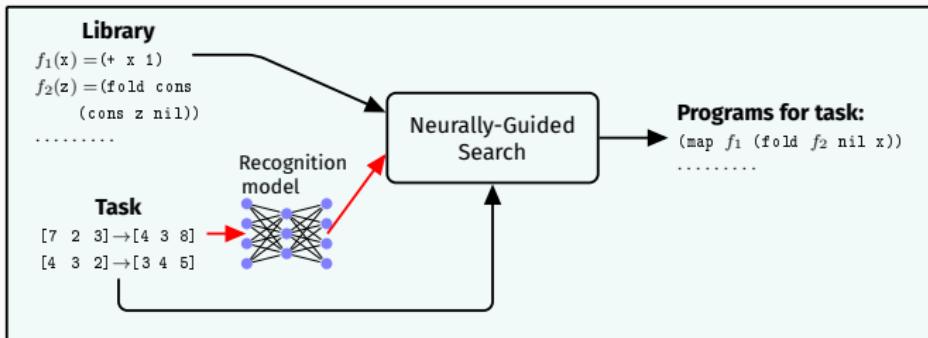
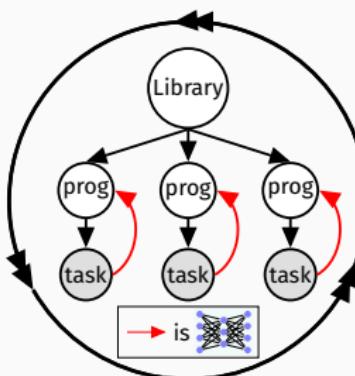
WAKE

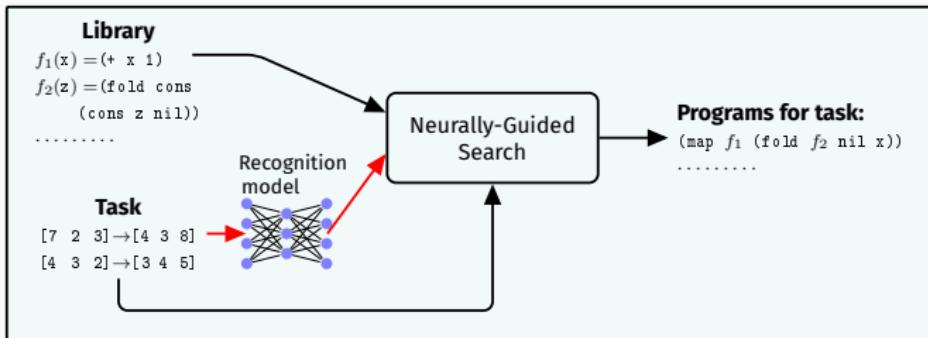
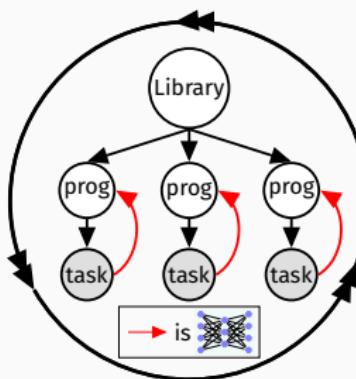
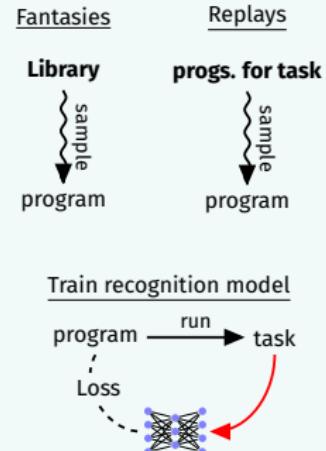


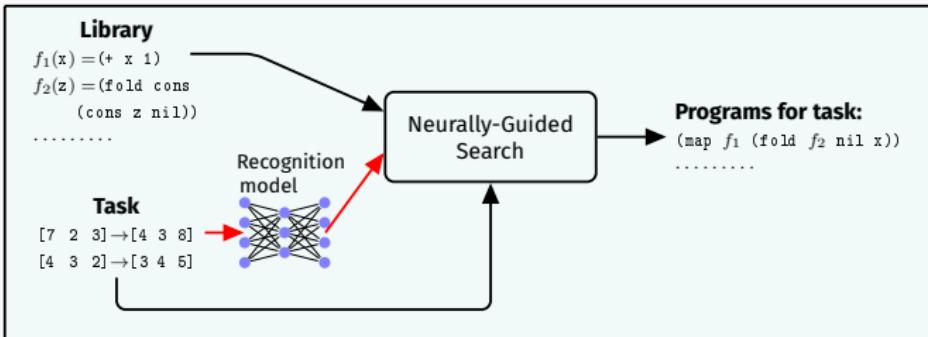
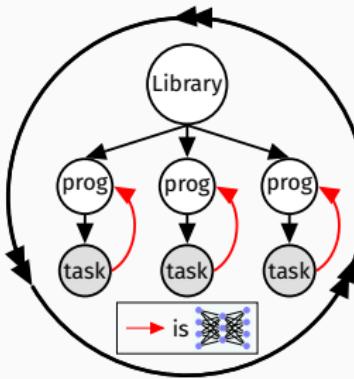
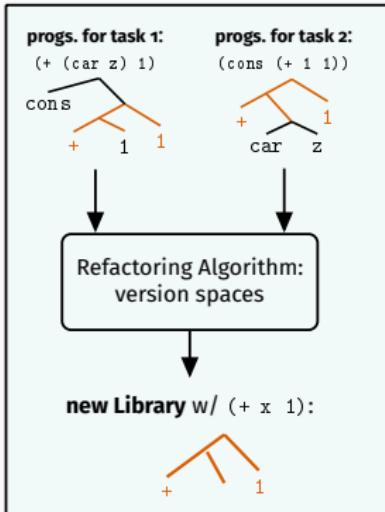
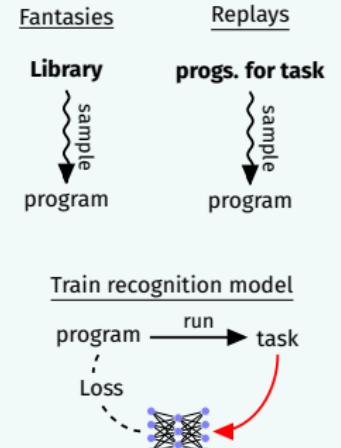
SLEEP: ABSTRACTION

SLEEP: DREAMING

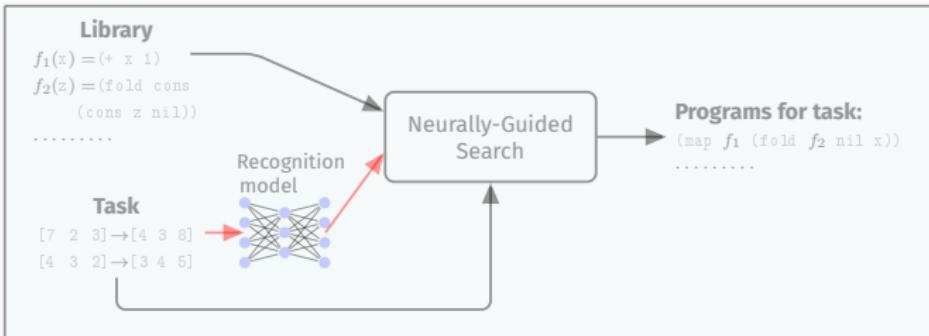


**WAKE****SLEEP: ABSTRACTION****SLEEP: DREAMING**

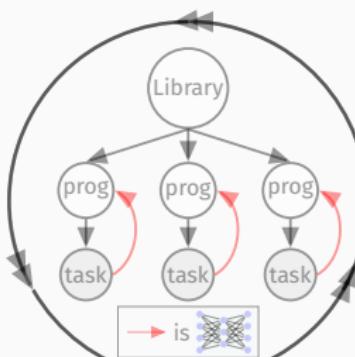
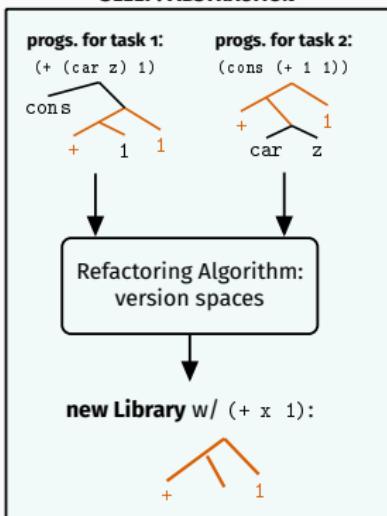
**WAKE****SLEEP: ABSTRACTION****SLEEP: DREAMING**

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

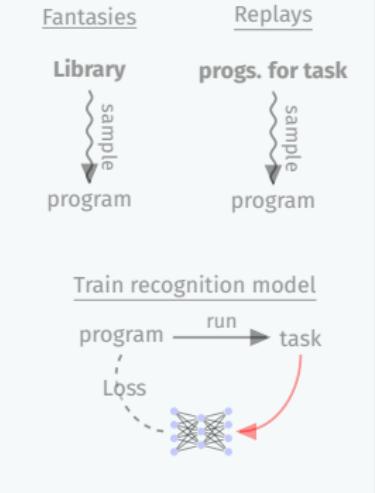
## WAKE



## SLEEP: ABSTRACTION



## SLEEP: DREAMING



DreamCoder and learning to learn  
learning a library  
learning to search  
synergy between library+learned search

# 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)))
```

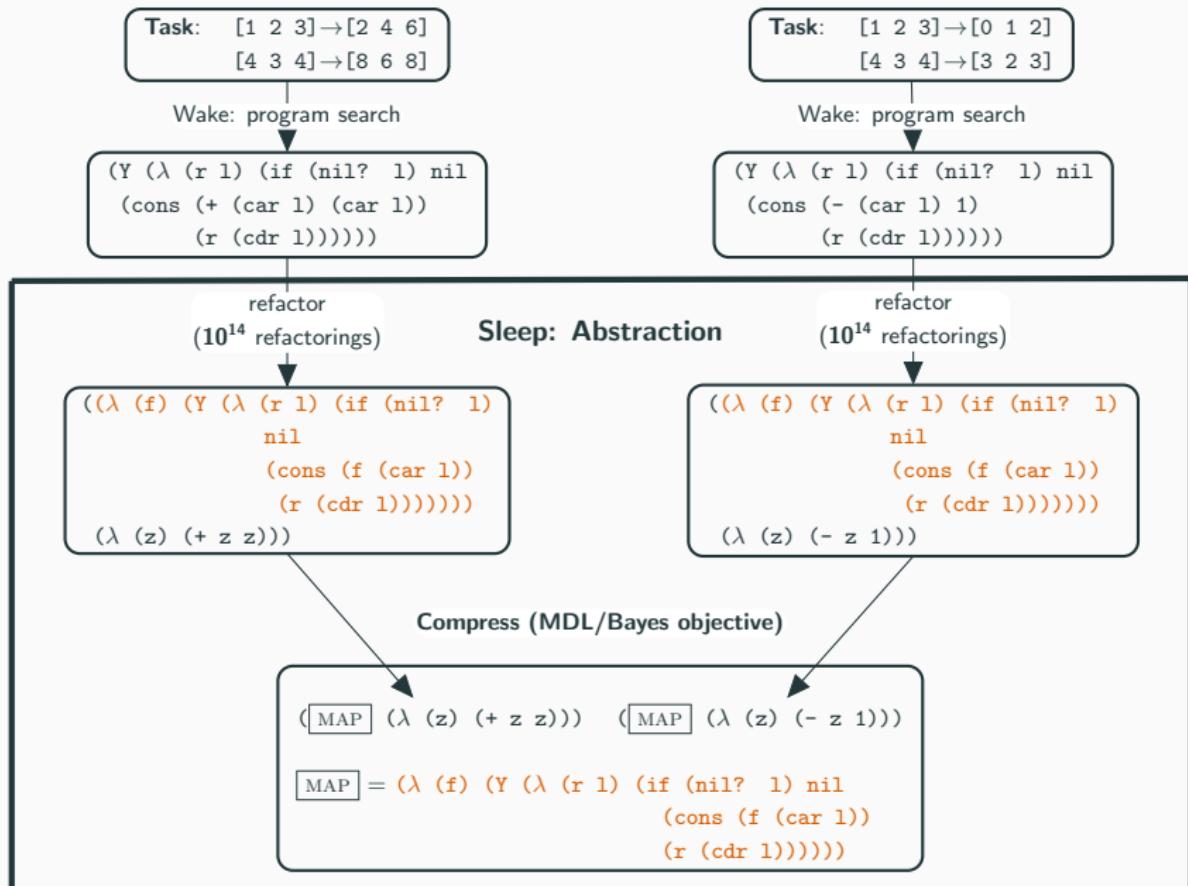
refactor

$(10^{14}$  refactorings)

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

## Sleep: Abstraction

# 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

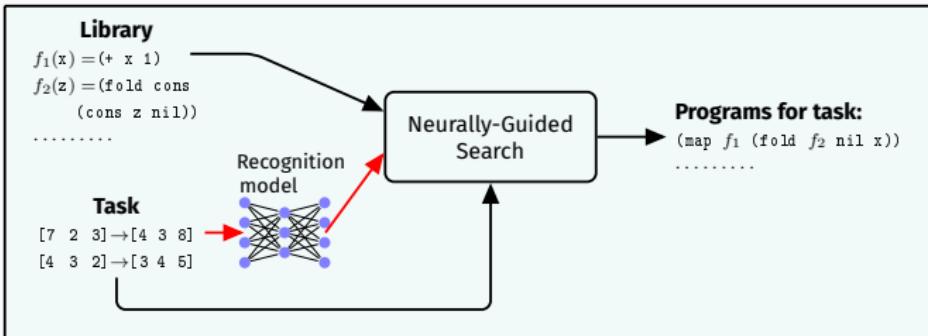
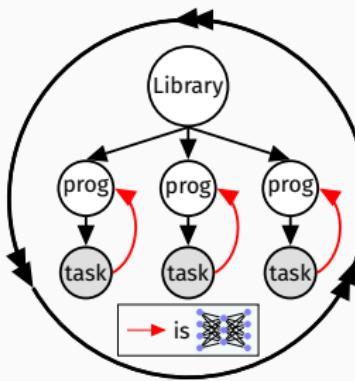
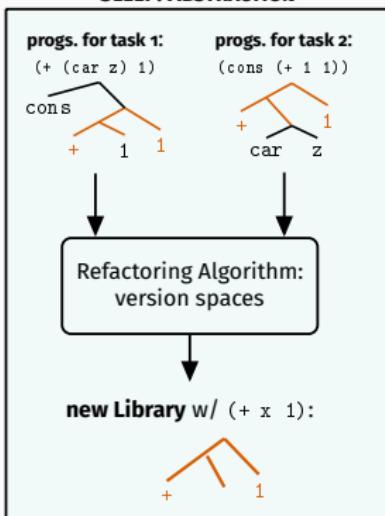
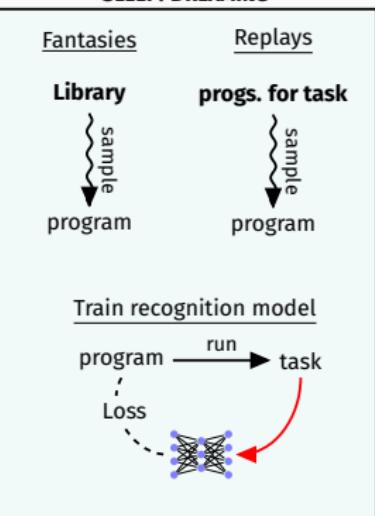
c.f. [Tate et al 2009], [Gulwani 2012]

$\cdot 1?)$   
 $) \cdot 1))$   
 $) \cdot 1)))$

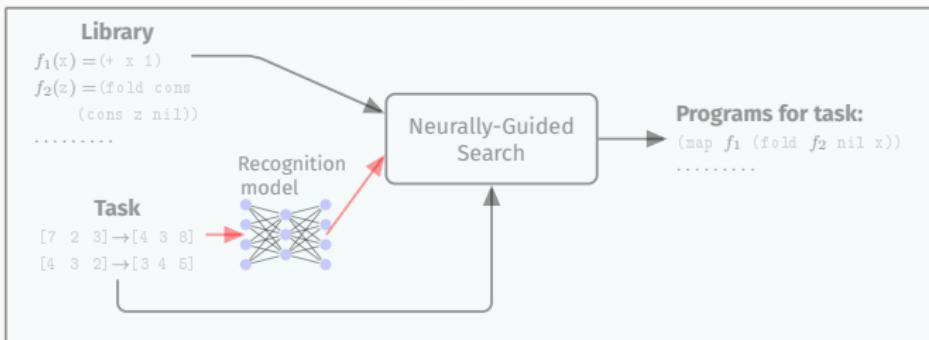
Compress (MDL/Bayes objective)

```
(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)))))))
```

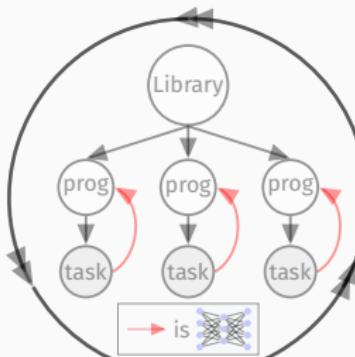
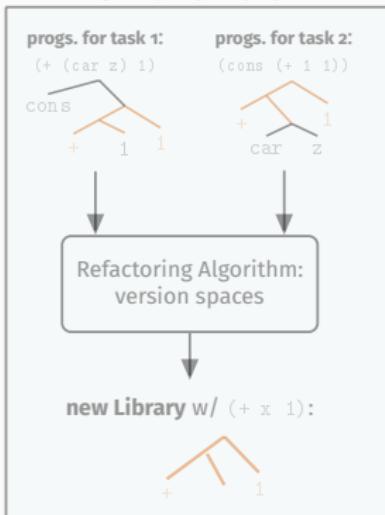
DreamCoder and learning to learn  
learning a library  
**learning to search**  
synergy between library+learned search

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

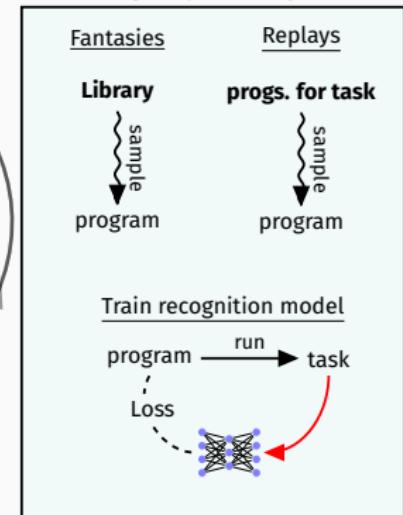
## WAKE



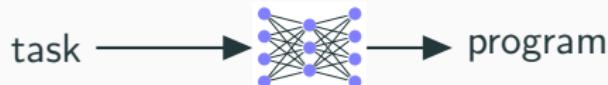
## SLEEP: ABSTRACTION



## SLEEP: DREAMING



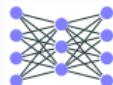
# Neural recognition model guides search



# Neural recognition model guides search



# Neural recognition model guides search

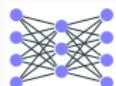


is a...

recurrent network (Devlin et al 2017)

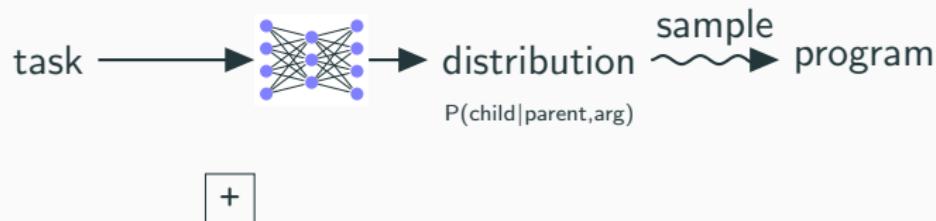
unigram model (Menon et al 2013; Balog et al 2016)

# Neural recognition model guides search

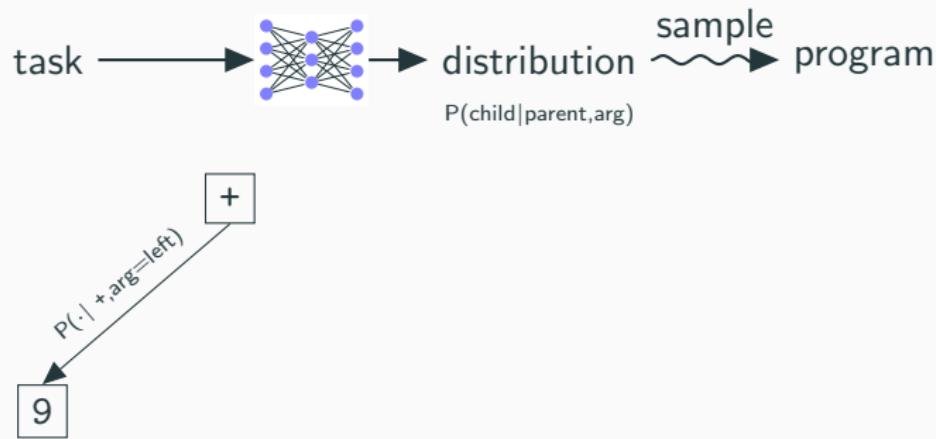


is a “**bigram**” model over syntax trees

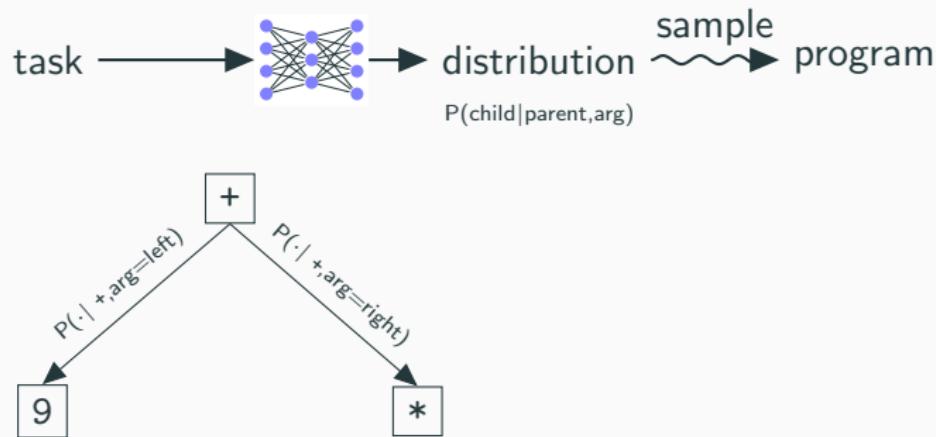
# Neural recognition model guides search



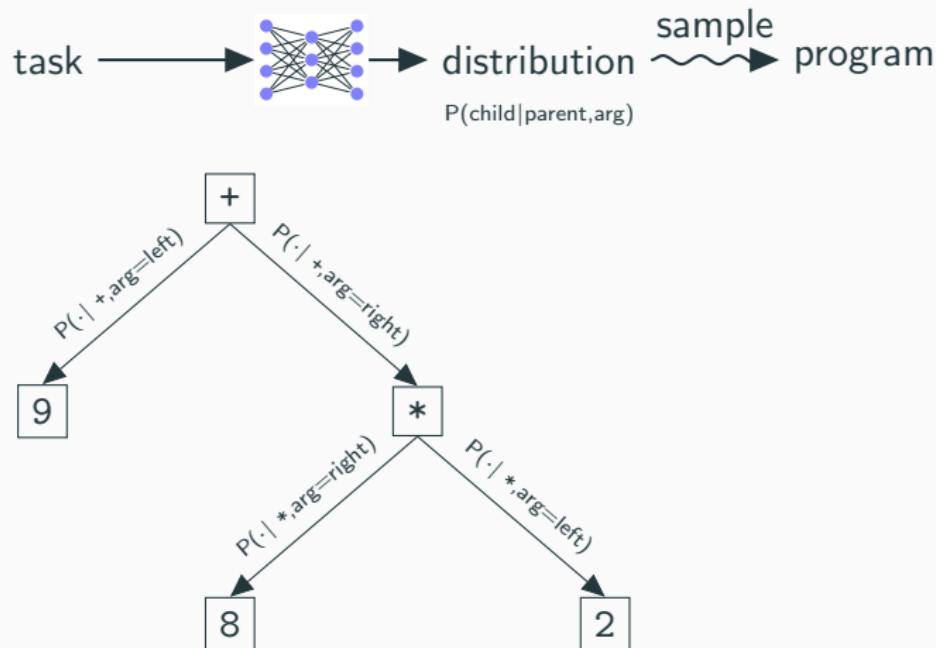
# Neural recognition model guides search



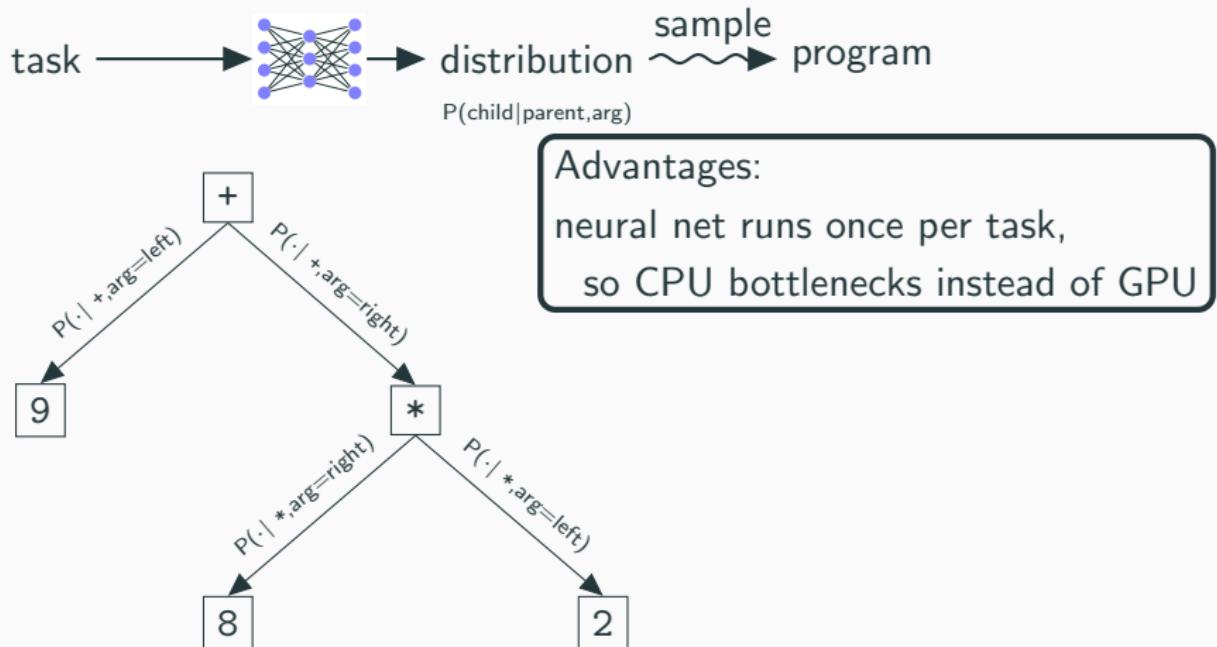
# Neural recognition model guides search



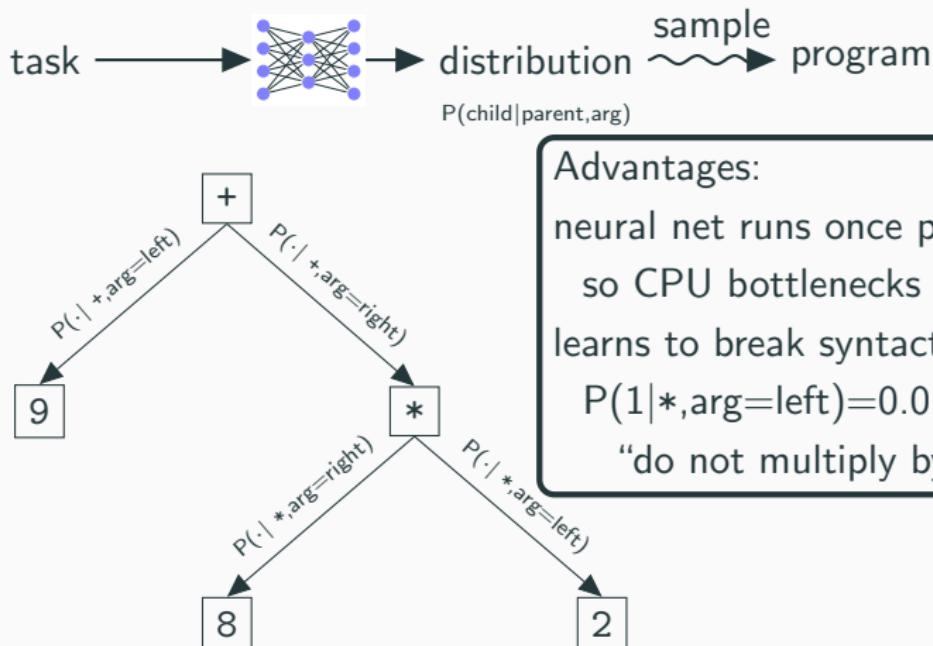
# Neural recognition model guides search



# Neural recognition model guides search



# Neural recognition model guides search



DreamCoder and learning to learn  
learning a library  
learning to search  
synergy between library+learned search

# 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

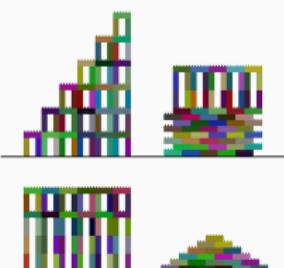
\$100.25

\$4.50

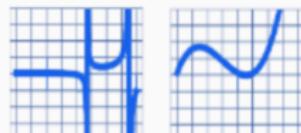
## LOGO Graphics



## 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

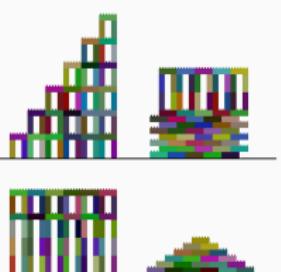
\$100.25

\$4.50

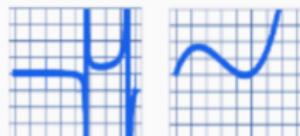
## LOGO Graphics



## Block Towers



## Symbolic Regression



$$y = f(x)$$

## Recursive Programming

### Filter Red

[■■■■■■■■■■■■] → [■■■■■■■■■■■■]

[■■■■■■■■■■■■■■■■] → [■■■■■■■■■■■■■■■■]

[■■■■■■■■■■■■■■■■■■] → [■■■■■■■■■■■■■■■■■■]

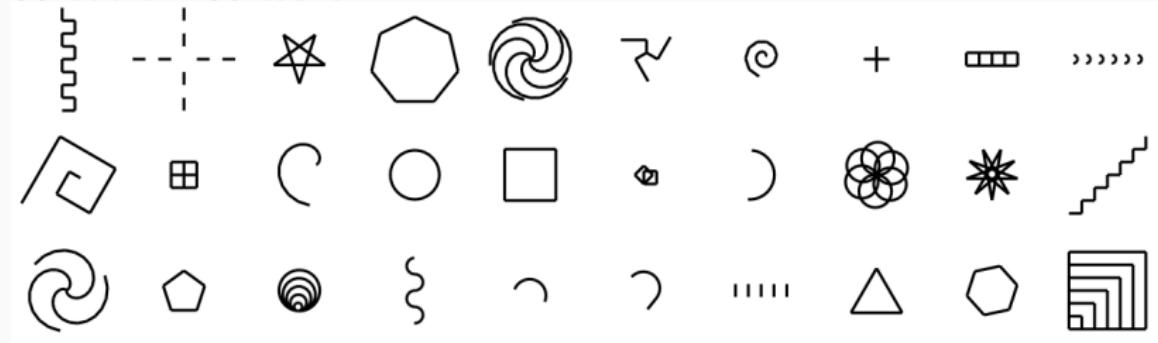
## Physical Laws

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

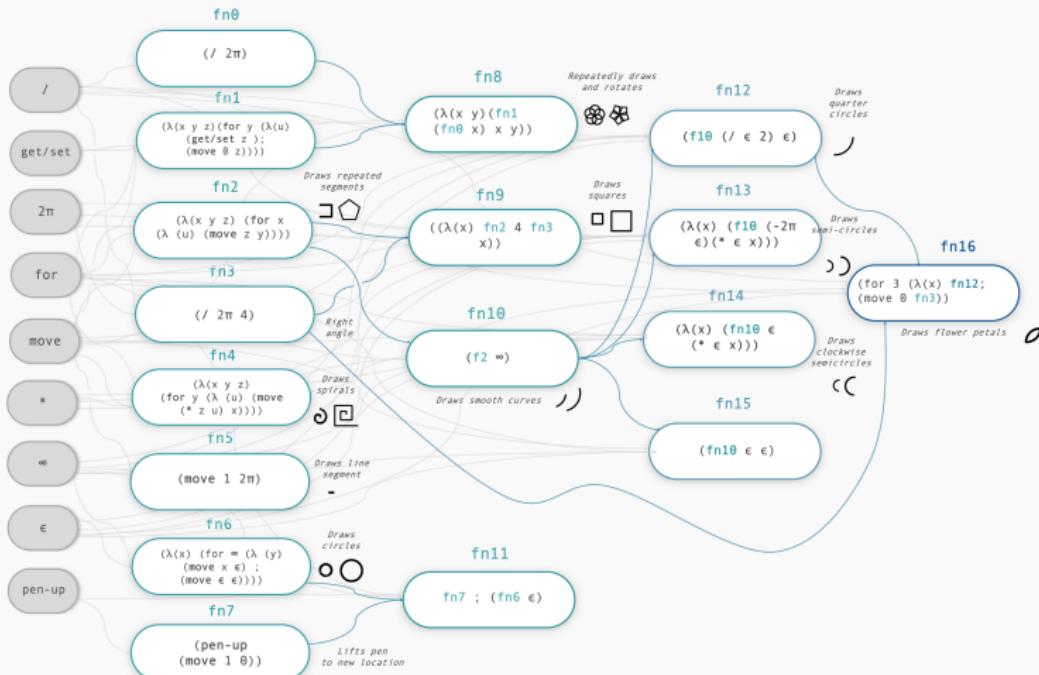
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}|^2} \hat{r}$$

# LOGO Turtle Graphics

30 out of 160 tasks



# LOGO Turtle Graphics – learning an interpretable library



**(fn8 5 (fn4 (\* \epsilon 2) \infty \epsilon))**



**(for 7 (\lambda (x) (fn9 x)))**

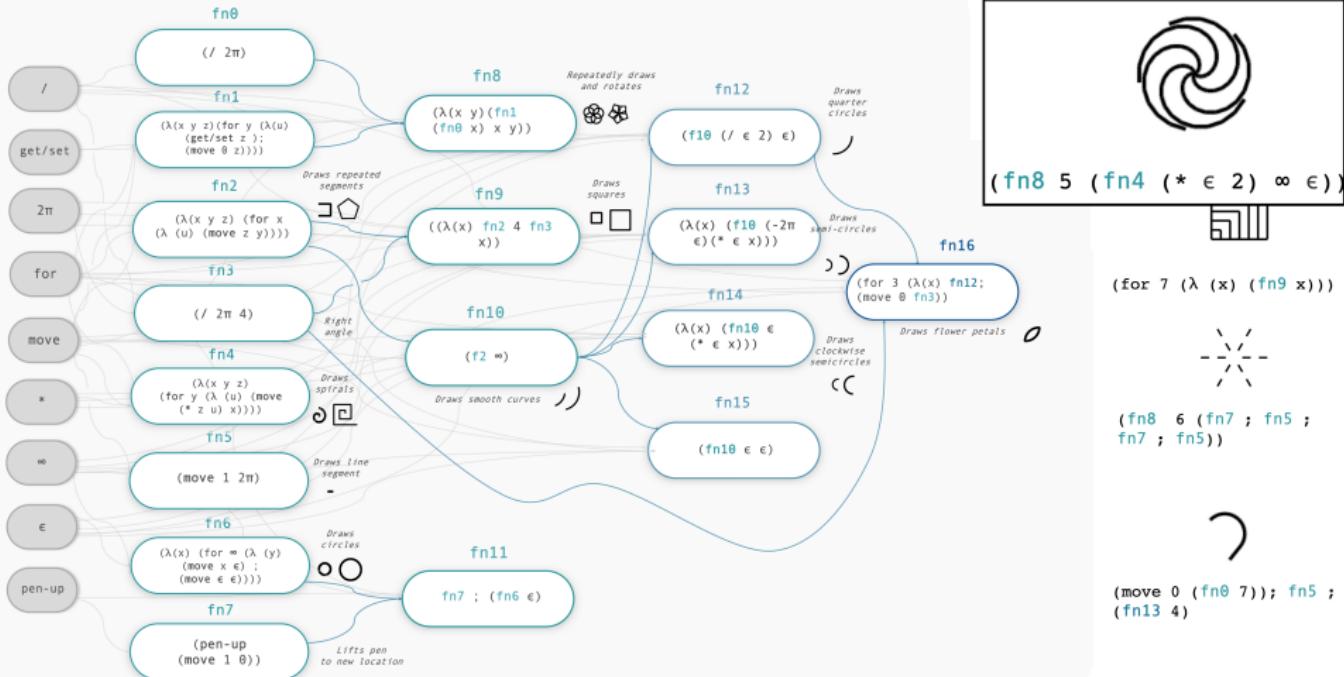


**(fn8 6 (fn7 ; fn5 ; fn7 ; fn5))**

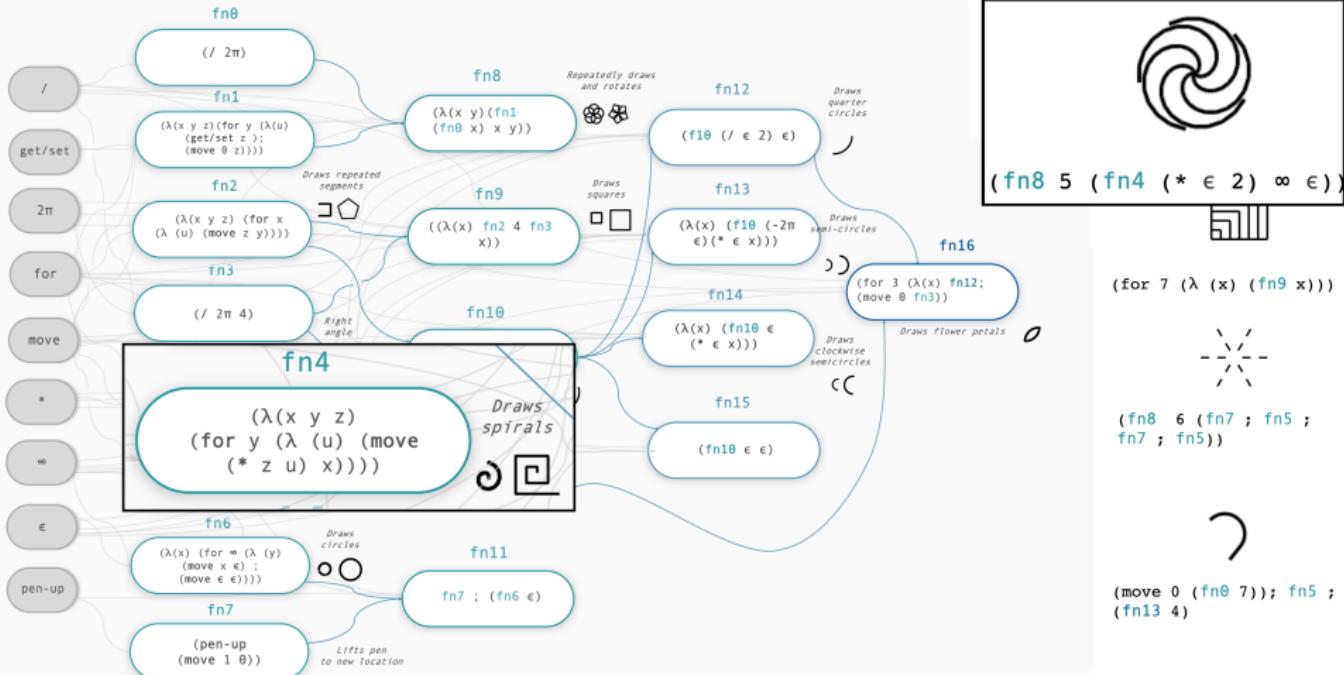


**(move 0 (fn0 7)); fn5 ; (fn13 4)**

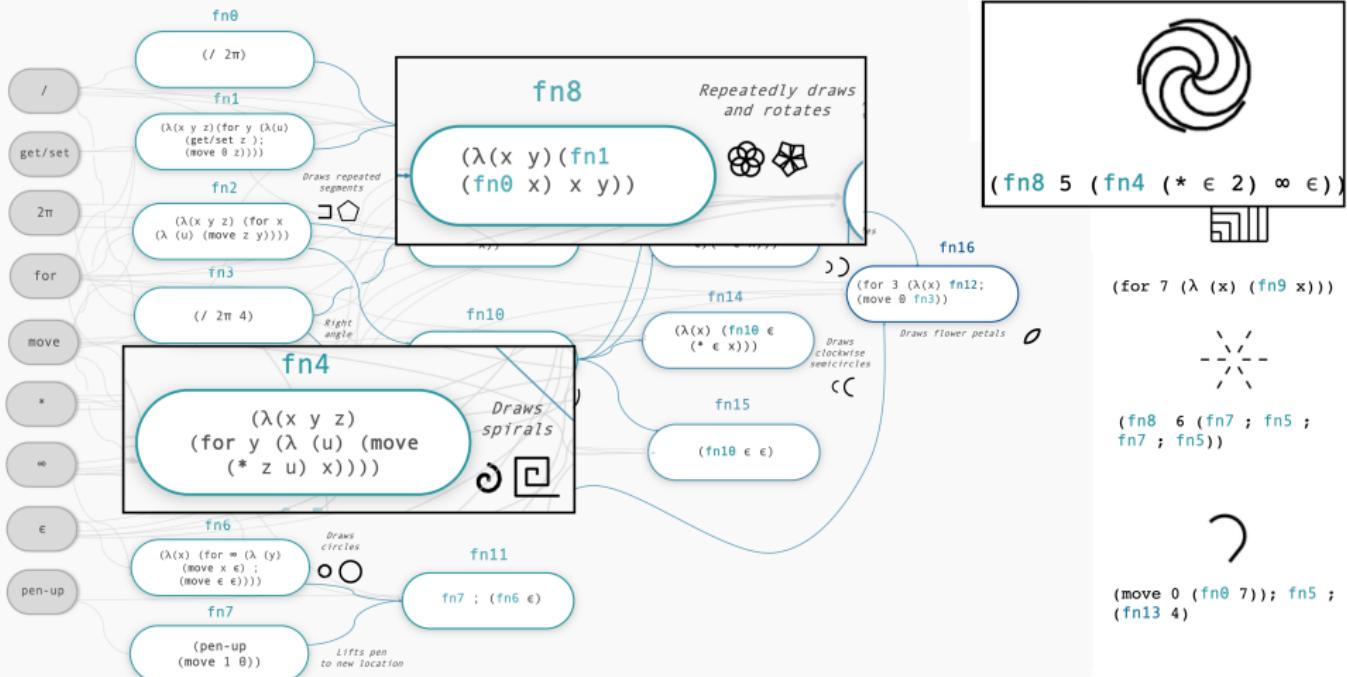
# LOGO Turtle Graphics – learning an interpretable library



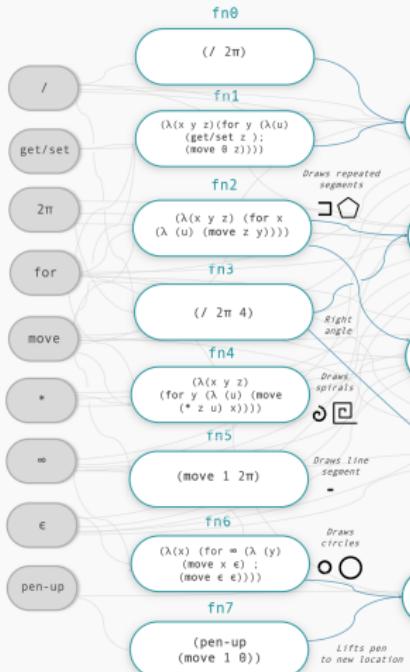
# LOGO Turtle Graphics – learning an interpretable library



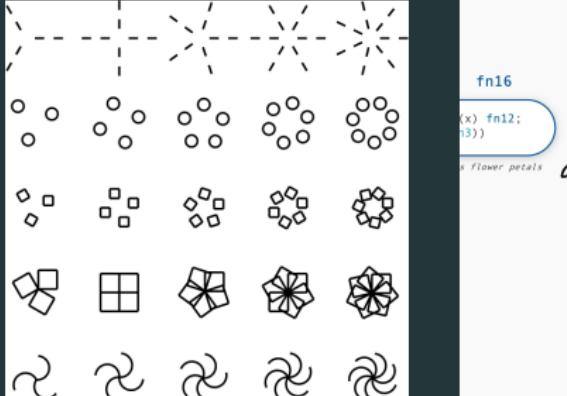
# LOGO Turtle Graphics – learning an interpretable library



# LOGO Turtle Graphics – learning an interpretable library



radial symmetry( $n$ , body)



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



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

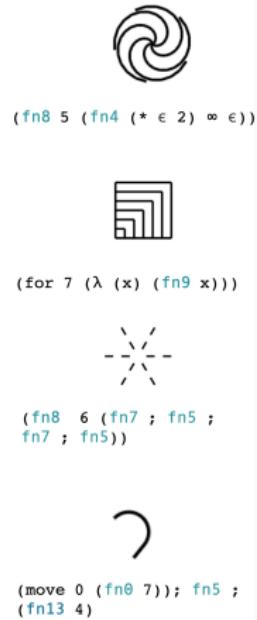
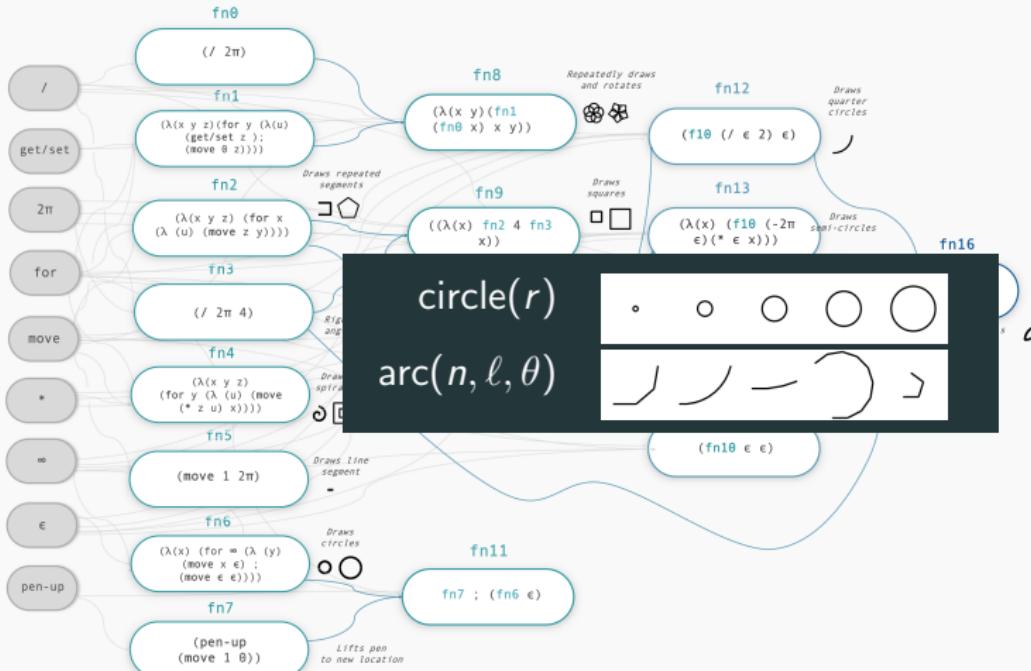


$(\text{fn8 } 6 \ (\text{fn7}; \ \text{fn5}); \ \text{fn7}; \ \text{fn5}))$

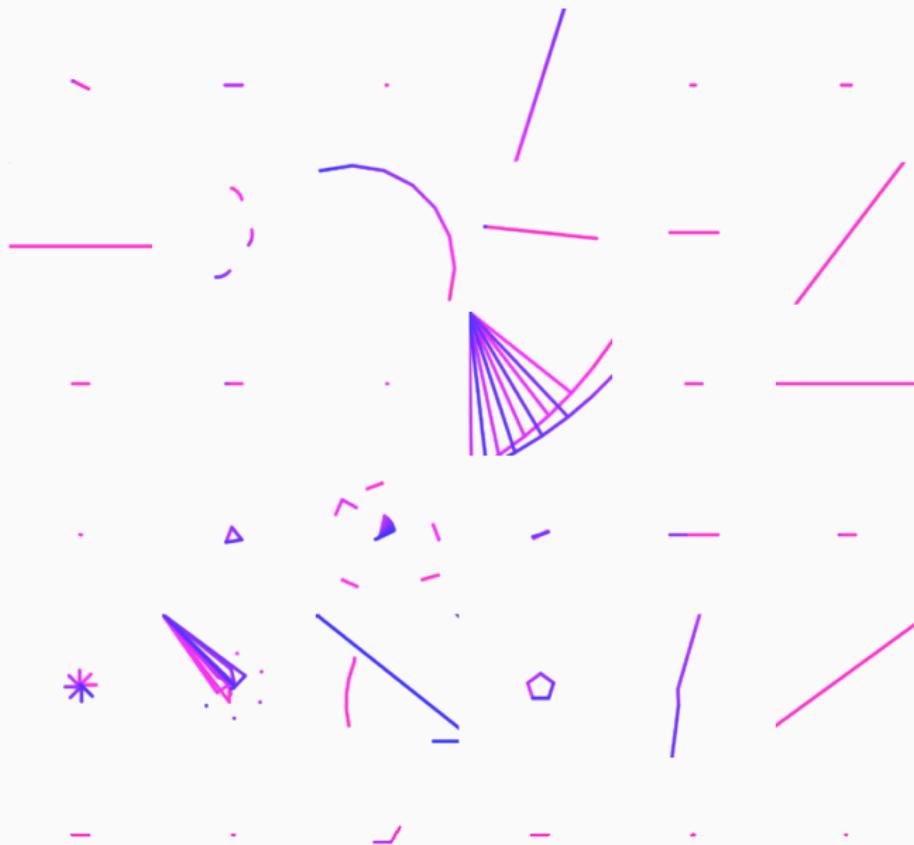


$(\text{move } 0 \ (\text{fn0 } 7)); \ \text{fn5}; \ (\text{fn13 } 4))$

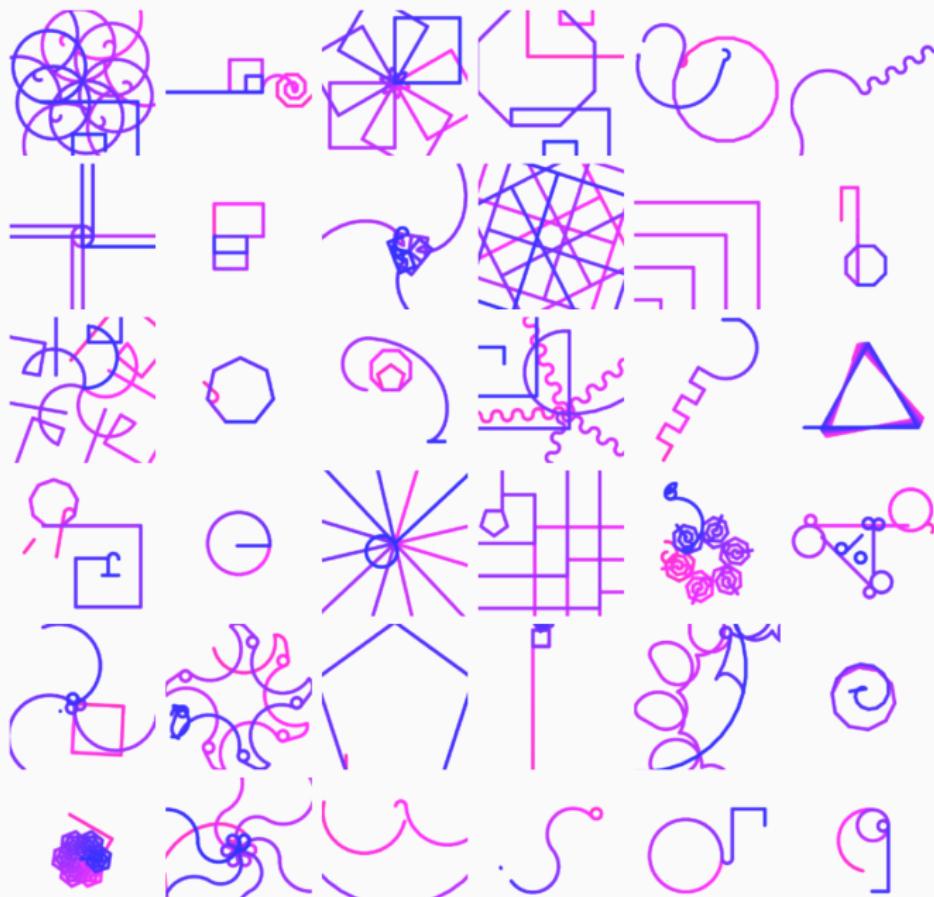
# LOGO Turtle Graphics – learning an interpretable library



# What does DreamCoder dream of? (before learning)

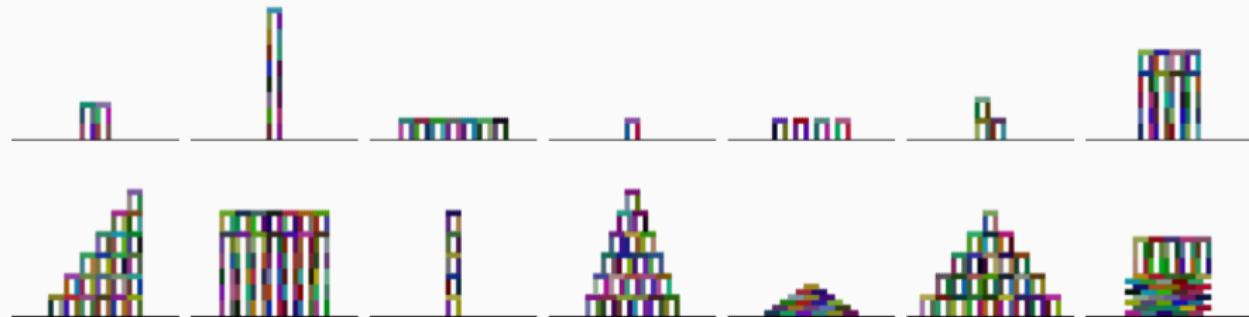


# What does DreamCoder dream of? (after learning)



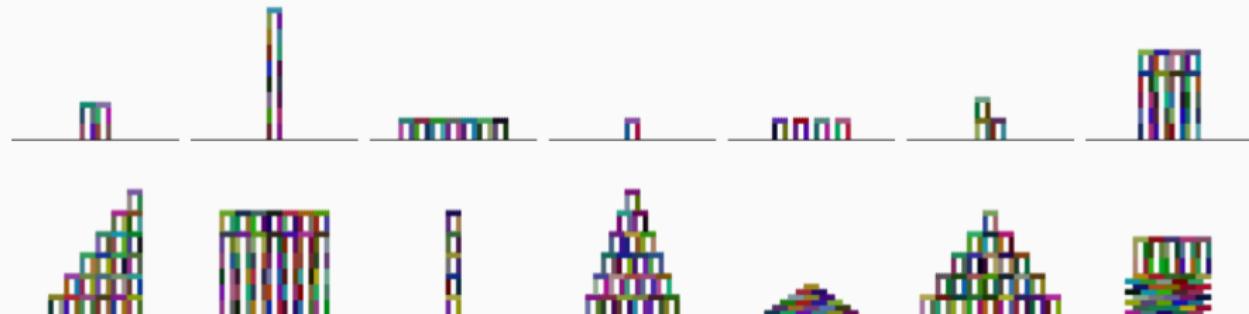
# Planning to build towers

example tasks (112 total)

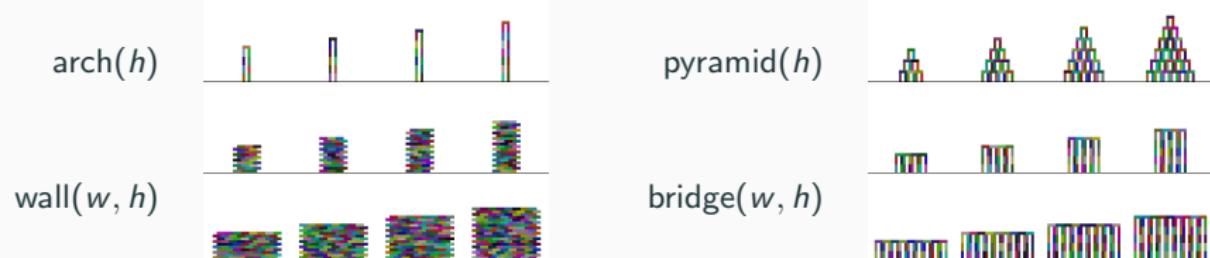


# Planning to build towers

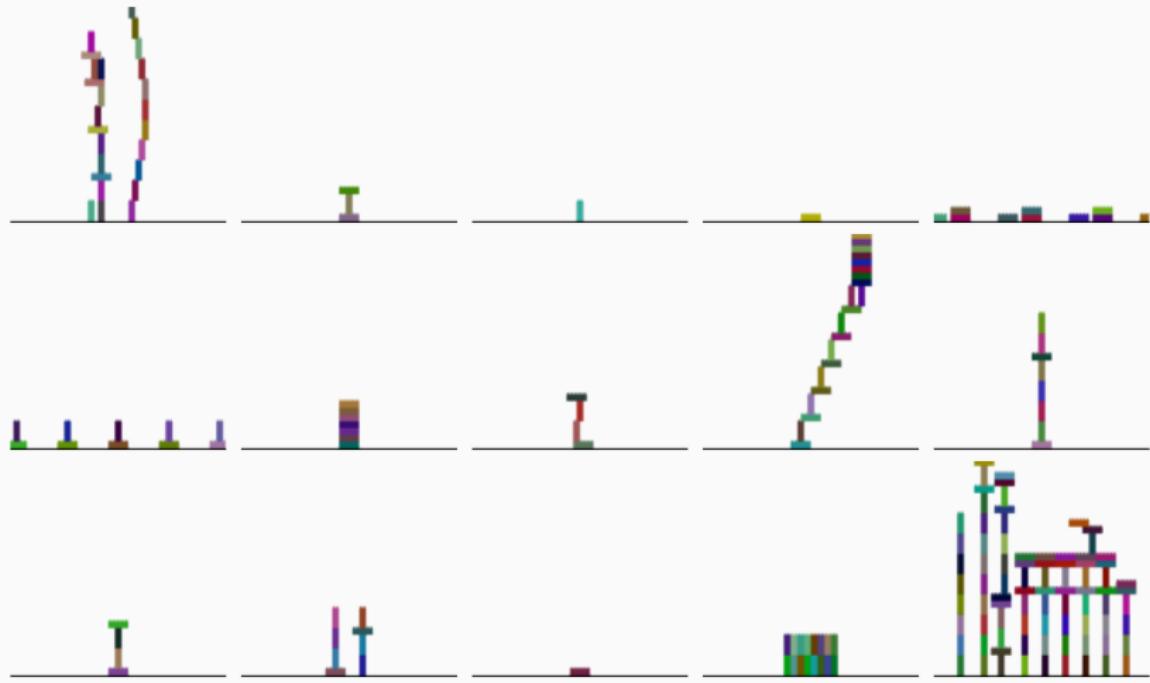
example tasks (112 total)



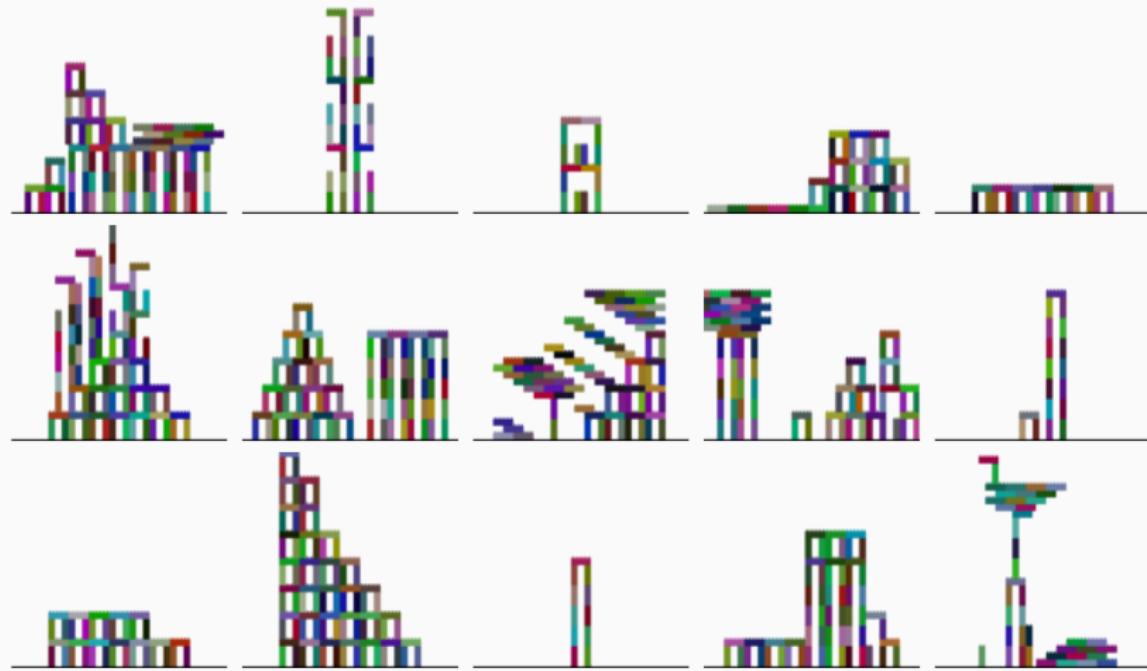
learned library routines ( $\approx 20$  total)



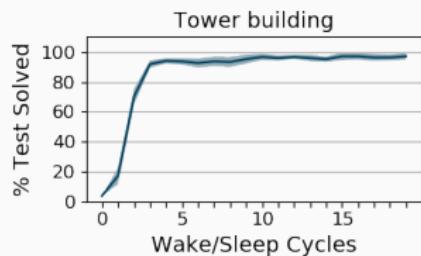
## Dreams before learning



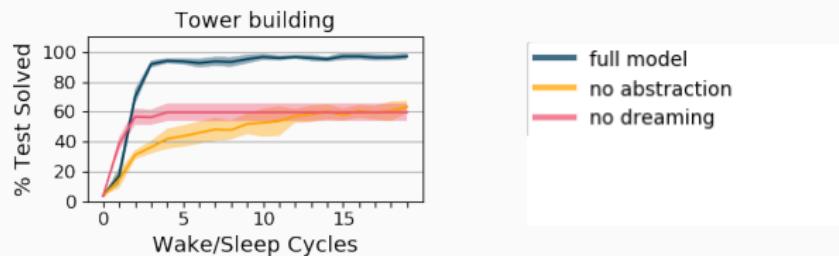
## Dreams after learning



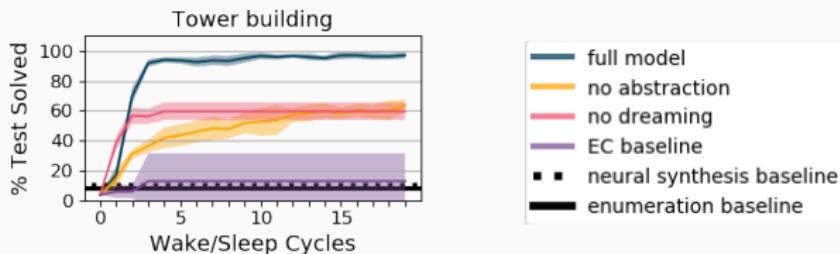
# Learning dynamics



# Learning dynamics



# Learning dynamics

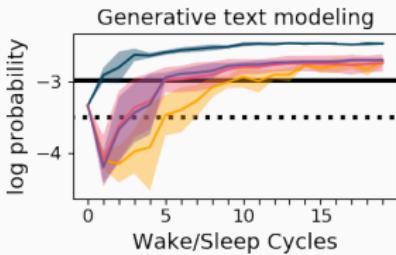
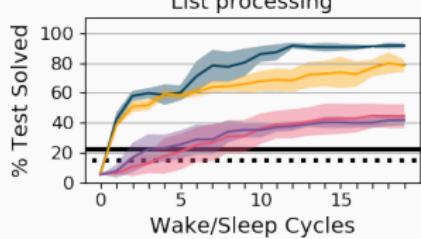
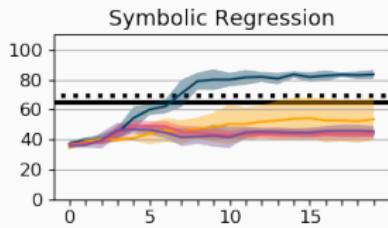
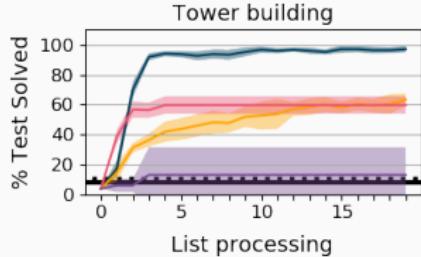
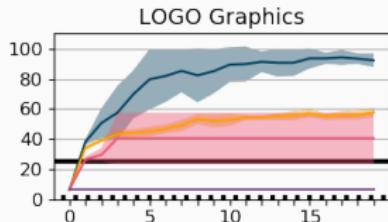
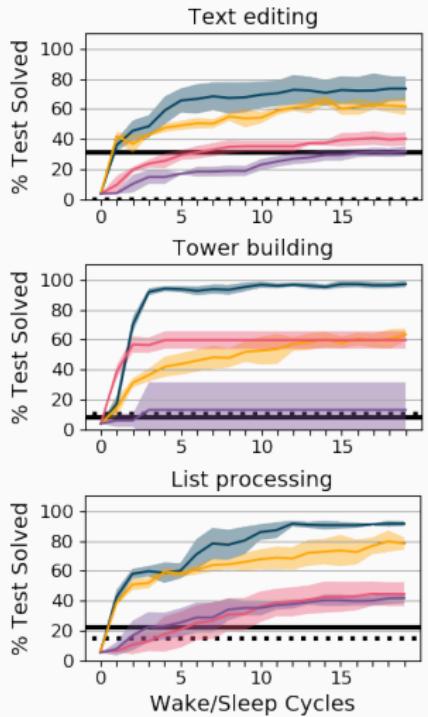


baselines: Exploration-Compression, EC [Dechter et al. 2013]

neural program synthesis, RobustFill [Devlin et al. 2017]

24 hours of brute-force enumeration

# Learning dynamics



- full model
- no abstraction
- no dreaming
- EC baseline
- neural synthesis baseline
- enumeration baseline

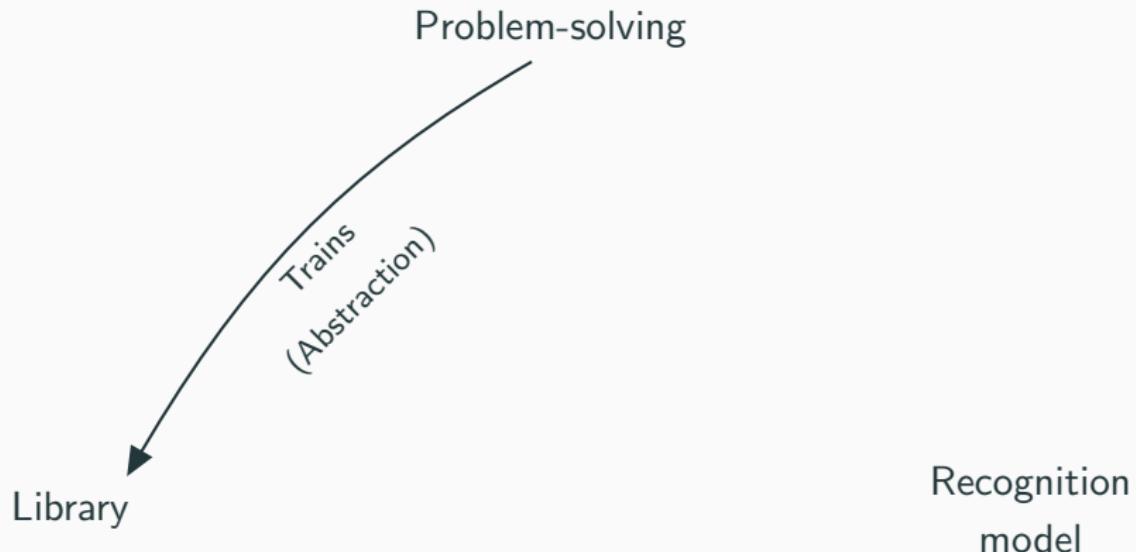
# Synergy between recognition model and library learning

Problem-solving

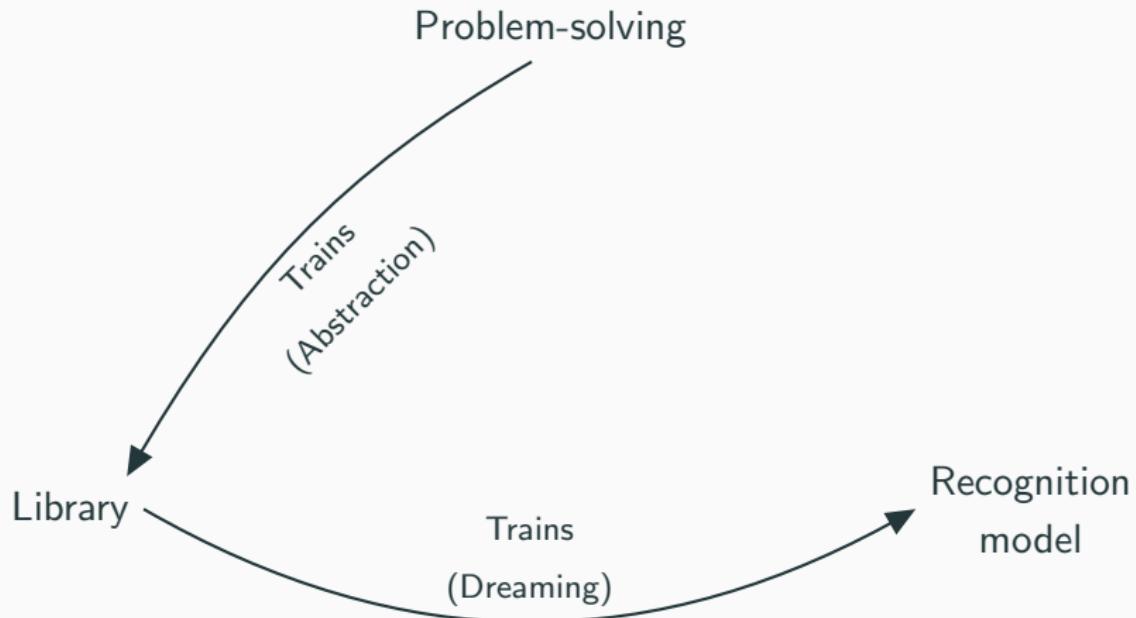
Library

Recognition  
model

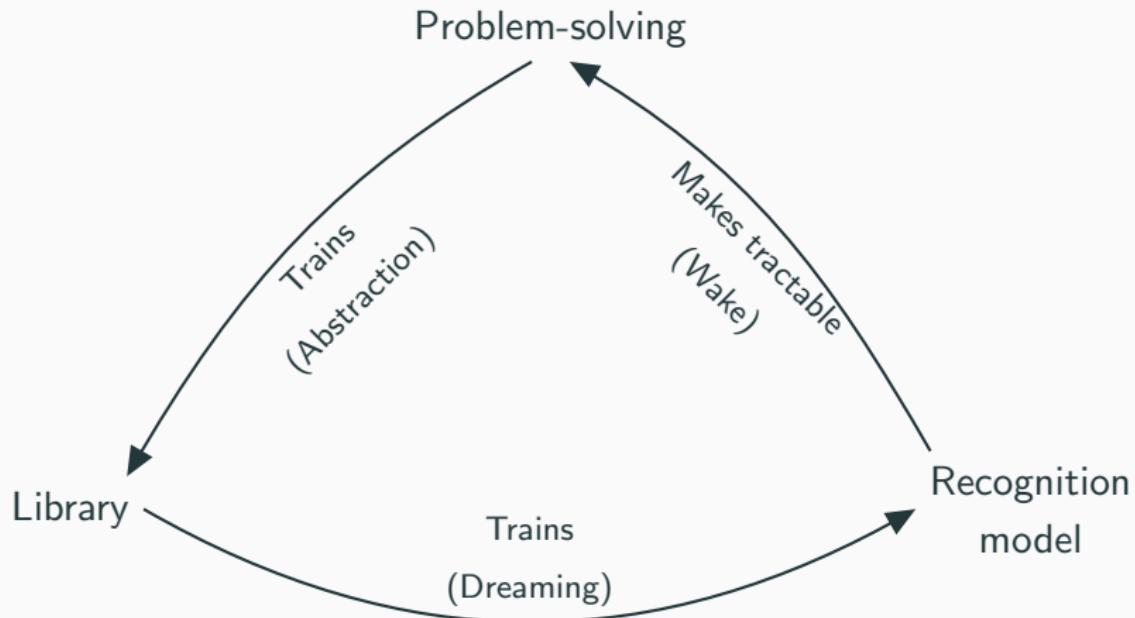
# Synergy between recognition model and library learning



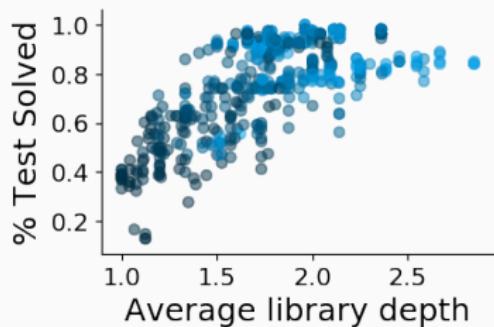
# Synergy between recognition model and library learning



# Synergy between recognition model and library learning



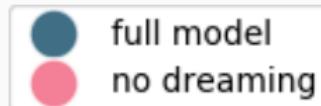
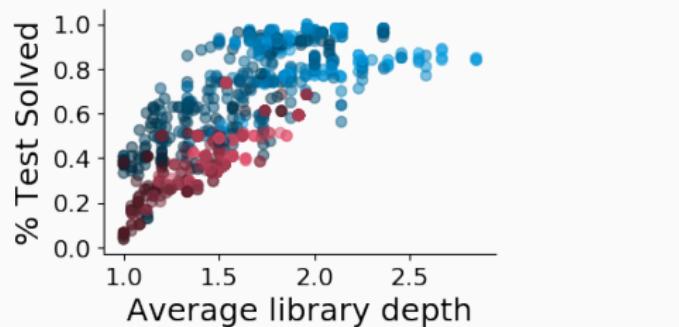
# Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Brighter: Later in learning

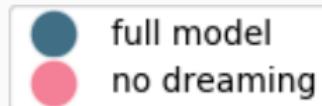
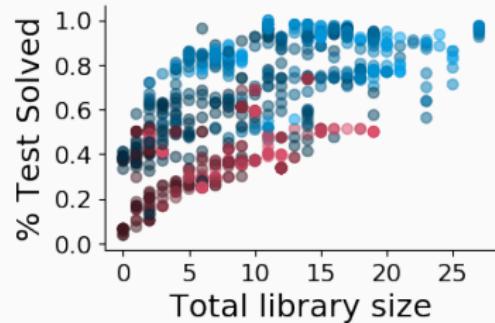
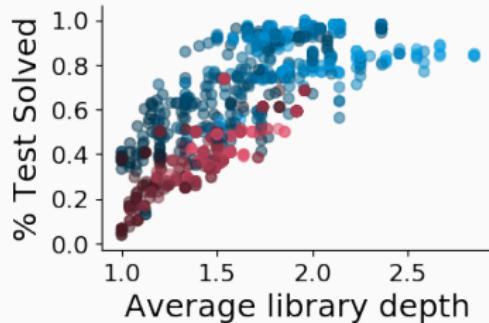
# Evidence for dreaming bootstrapping better libraries



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}$
$KE = \frac{1}{2}mv^2$	$\vec{r} = r \times F$	$\text{PE}_{\text{gravity}} = -G \frac{m_1 m_2}{r}$
$\Delta PE = mg\Delta y$	$L = I\omega$	$p = \frac{m}{V}$
$\Delta E = W = Fd\cos\theta$	$\Delta L = \tau \Delta t$	$KE = \frac{1}{2}I\omega^2$

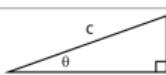
## Electricity

$ \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

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

## 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  
 $\alpha$  = angular acceleration  
 $\lambda$  = wavelength  
 $\mu$  = coefficient of friction

# Growing languages for vector algebra and physics

## Initial Primitives

map  
zip  
cons  
empty  
cdr  
power  
fold  
car  
+  
-  
\*  
/  
θ  
1  
π -

## 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

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

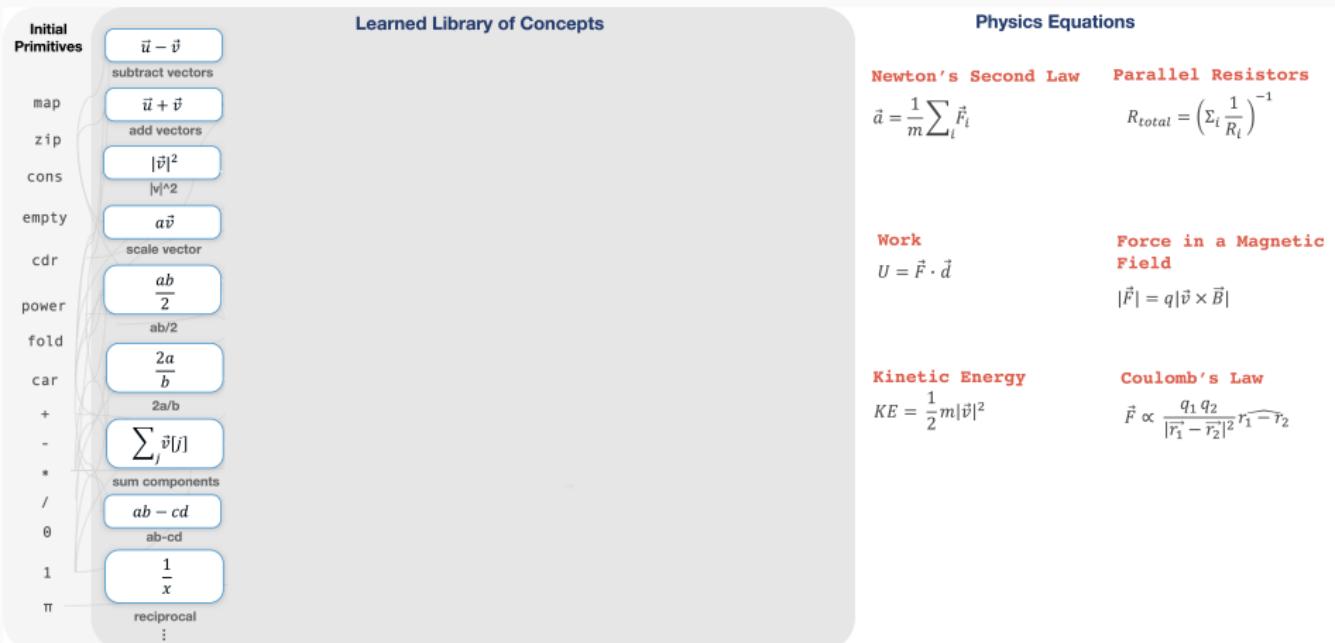
### Kinetic Energy

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

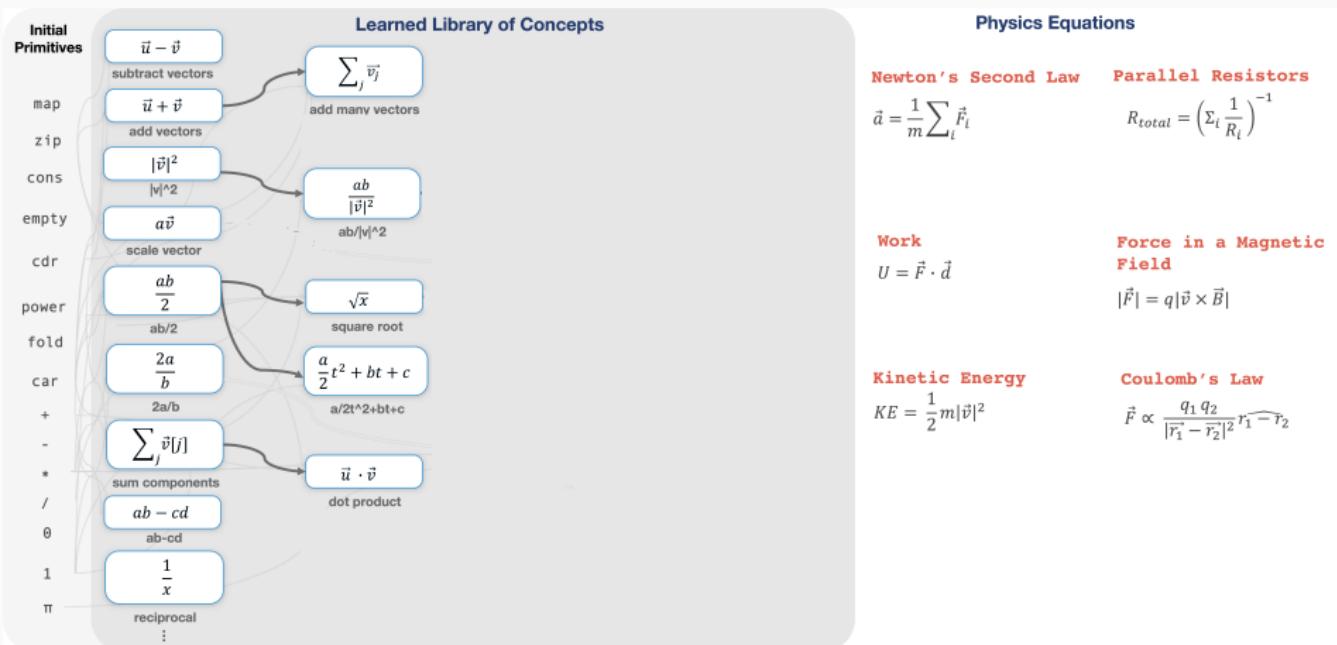
### Coulomb's Law

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

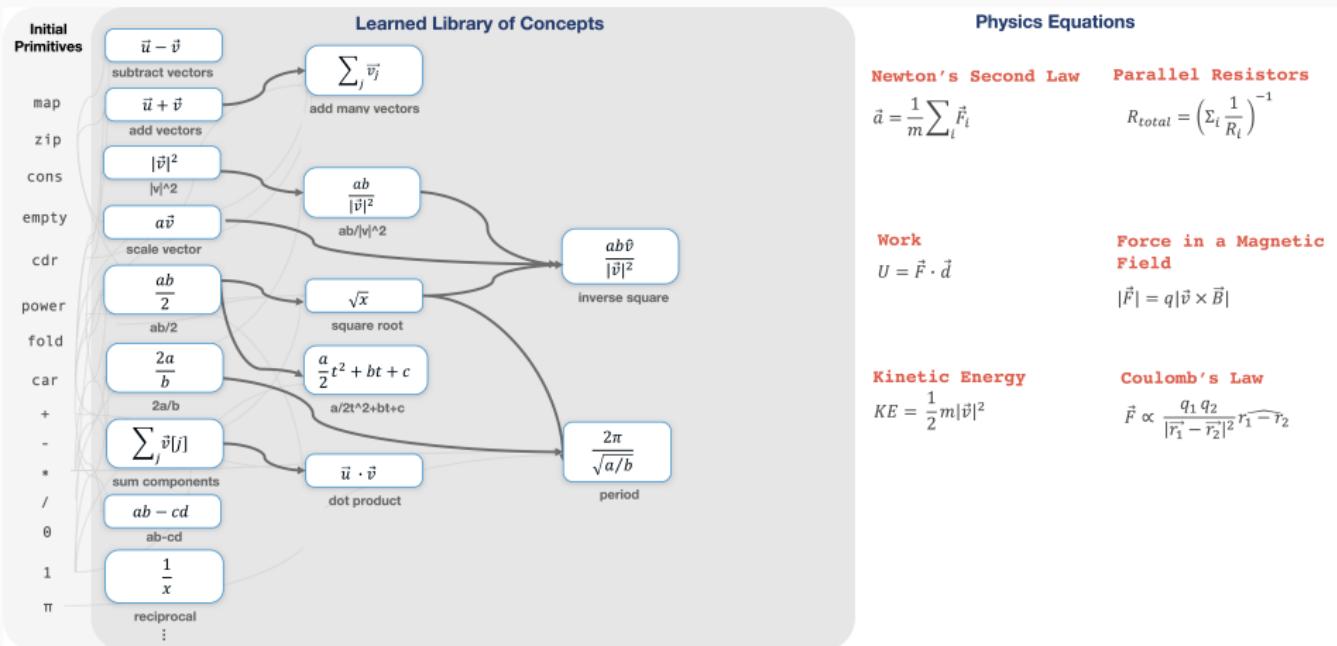
# Growing languages for vector algebra and physics



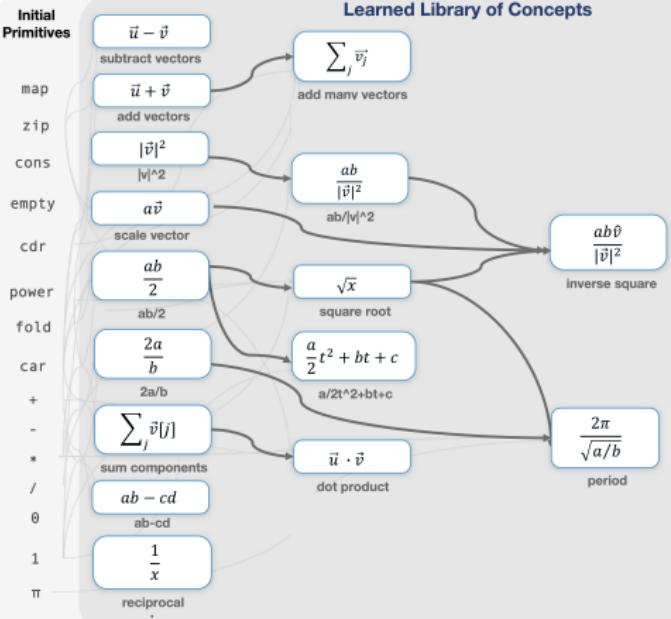
# Growing languages for vector algebra and physics



# Growing languages for vector algebra and physics

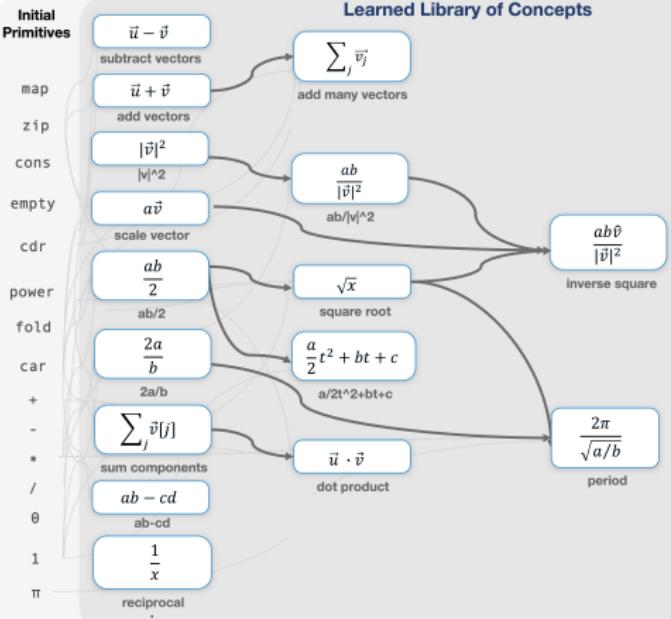


# Growing languages for vector algebra and physics



Physics Equations	
Newton's Second Law	Parallel Resistors
$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$	$R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$
(scale-vector(reciprocal m) (reciprocal (sum-components (add-many-vectors Fs))) (map (lambda(r) (reciprocal r)) Rs)))	(scale-vector(reciprocal m) (reciprocal (sum-components (add-many-vectors Fs))) (map (lambda(r) (reciprocal r)) Rs)))
Work	Force in a Magnetic Field
$U = \vec{F} \cdot \vec{d}$	$ \vec{F}  = q  \vec{v} \times \vec{B} $
(dot-product F d)	(* q (ab-cd v_x b_y v_y b_x))
Kinetic Energy	Coulomb's Law
$KE = \frac{1}{2} m  \vec{v} ^2$	$\vec{F} \propto \frac{q_1 q_2}{ \vec{r}_1 - \vec{r}_2 ^2} \hat{r}_1 - \hat{r}_2$
(ab/2 m (( v ^2 v))	(inverse-square q_1 q_2 (subtract-vectors r_1 r_2))

# Growing languages for vector algebra and physics



## Physics Equations

### Newton's Second Law

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

(scale-vector(reciprocal m) (reciprocal (sum-components (add-many-vectors Fs)))  
(map (lambda(r) (reciprocal r)) Rs)))

### Work

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

(dot-product F d)

### Parallel Resistors

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

(scale-vector(reciprocal m) (reciprocal (sum-components (add-many-vectors Fs)))  
(map (lambda(r) (reciprocal r)) Rs)))

### 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

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

(inverse-square q\_1 q\_2  
(subtract-vectors r\_1 r\_2))

```
(λ (x y z u) (map (λ (v) (* (/ (* (power (/ (* x) (fold (zip z u (λ (w a) (- w a)))) θ (λ (b c) (+ (* b b) c)))) (/ (* 1 1) (+ 1 1)))) y) (fold (zip z u (λ (d e) (- d e))) θ (λ (f g) (+ (* f f) g)))) v)) (zip z u (λ (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

## Initial Primitives

$\lambda$

combinator

cons

car

cdr

nil

if

nil?

+

-

0

1

=

## Learned Library of Concepts

**fold**

```
fold(xs,f,x0) =  
(if (nil? xs) x0  
  (f (fold (cdr xs)  
           f x0) (car xs)))
```

**unfold**

```
unfold(x,g,f,p) =  
(if (p x) nil  
  (cons (f x)  
        (unfold (g x)  
               g f p)))
```

## Recursive Programming Algorithms

### Stutter

```
[■■] → [■■■■]  
[■■■■] → [■■■■■■]  
(fold A (λ (u v) (cons v (cons v u))) nil)
```

### Take every other

```
[■■■■■■] → [■■]  
[■■■■■■■■] → [■■■■]
```

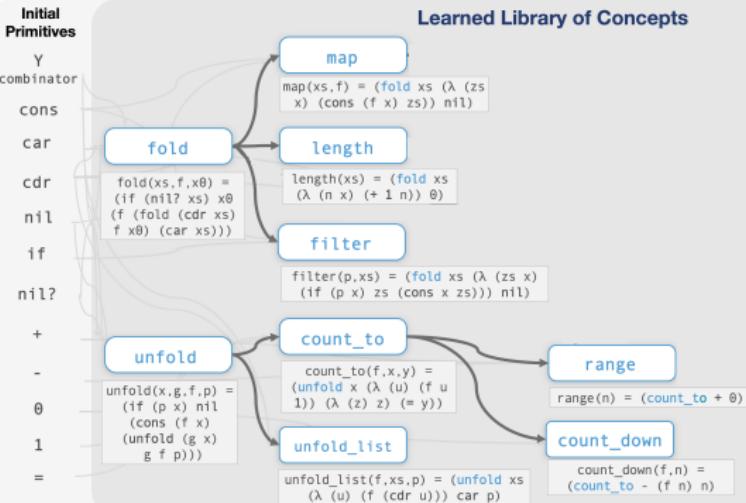
### List lengths

```
[[■■■], [■]] → [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



**Recursive Programming Algorithms**

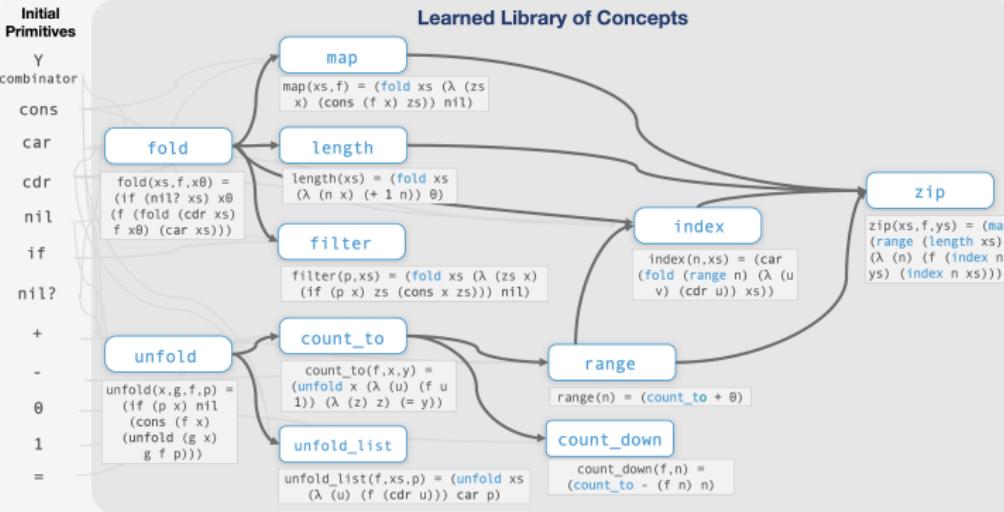
**Stutter**  
 $[\blacksquare\blacksquare] \rightarrow [\blacksquare\blacksquare\blacksquare\blacksquare]$   
 $[\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare]$   
 $(\text{fold } A \ (\lambda (u v) (\text{cons } v (\text{cons } v u))) \ \text{nil})$

**Take every other**  
 $[\blacksquare\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare\blacksquare]$   
 $[\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare\blacksquare\blacksquare\blacksquare]$   
 $(\text{unfold\_list } \text{cdr } A \ \text{nil?})$

**List lengths**  
 $[[\blacksquare\blacksquare\blacksquare], [\blacksquare]] \rightarrow [3 \ 1]$   
 $[[\blacksquare\blacksquare], [], [\blacksquare]] \rightarrow [2 \ 0 \ 1]$   
 $(\text{map } A \ \text{length})$

**List differences**  
 $[1 \ 8 \ 2], [0 \ 5 \ 1] \rightarrow [1 \ 3 \ 1]$   
 $[2 \ 3 \ 6], [1 \ 2 \ 4] \rightarrow [1 \ 1 \ 2]$

# Growing a language for recursive programming



**Recursive Programming Algorithms**

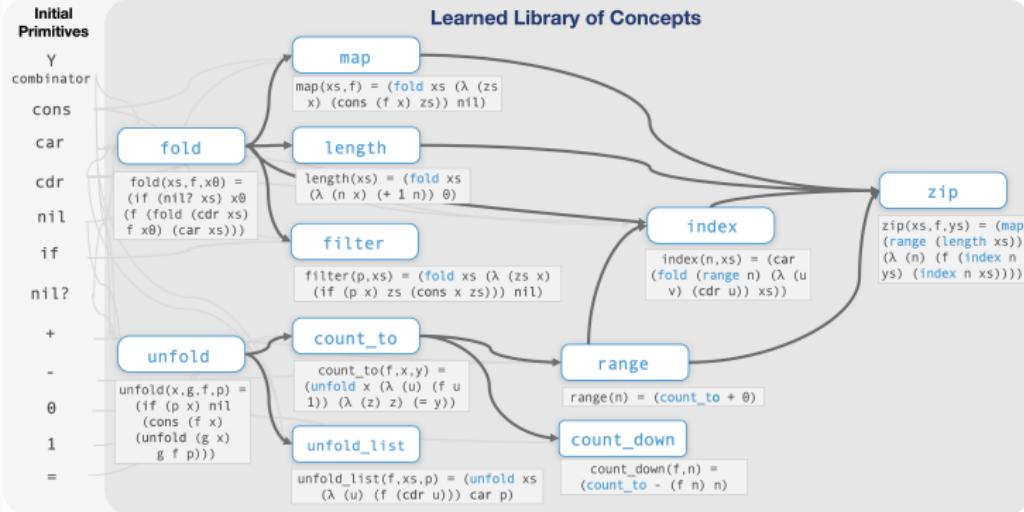
**Stutter**  
 $[\text{■■■}] \rightarrow [\text{■■■■■}]$   
 $[\text{■■■■}] \rightarrow [\text{■■■■■■}]$   
 $(\text{fold } A \text{ } (\lambda \text{ } (u \text{ } v) \text{ } (\text{cons } v \text{ } (\text{cons } v \text{ } u))) \text{ } \text{nil})$

**Take every other**  
 $[\text{■■■■■■■■}] \rightarrow [\text{■■}]$   
 $[\text{■■■■■■■■■■}] \rightarrow [\text{■■■■}]$   
 $(\text{unfold\_list } \text{cdr } A \text{ } \text{nil?})$

**List lengths**  
 $[[\text{■■■■}], [\text{■}]] \rightarrow [3 \text{ } 1]$   
 $[[\text{■■■}], [], [\text{■}]] \rightarrow [2 \text{ } 0 \text{ } 1]$   
 $(\text{map } A \text{ } \text{length})$

**List differences**  
 $[1 \text{ } 8 \text{ } 2], [0 \text{ } 5 \text{ } 1] \rightarrow [1 \text{ } 3 \text{ } 1]$   
 $[2 \text{ } 3 \text{ } 6], [1 \text{ } 2 \text{ } 4] \rightarrow [1 \text{ } 1 \text{ } 2]$   
 $(\text{zip } A - B)$

# Growing a language for recursive programming



## Recursive Programming Algorithms

### Stutter

$[\text{■■■}] \rightarrow [\text{■■■■■■}]$   
 $[\text{■■■■}] \rightarrow [\text{■■■■■■■■}]$   
 $(\text{fold } \text{A} \text{ } (\lambda \text{ } (\text{u } \text{v})) \text{ } (\text{cons } \text{v} \text{ } (\text{cons } \text{v } \text{u}))) \text{ } \text{nil})$

### Take every other

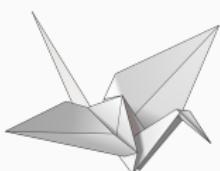
$[\text{■■■■■■■■}] \rightarrow [\text{■■}]$   
 $[\text{■■■■■■■■■■}] \rightarrow [\text{■■■■}]$   
 $(\text{unfold\_list } \text{cdr } \text{A} \text{ } \text{nil?})$

### List lengths

$[[\text{■■■■}], [\text{■}]] \rightarrow [3 \text{ } 1]$   
 $[[\text{■■■}], [], [\text{■}]] \rightarrow [2 \text{ } 0 \text{ } 1]$   
 $(\text{map } \text{A} \text{ } \text{length})$

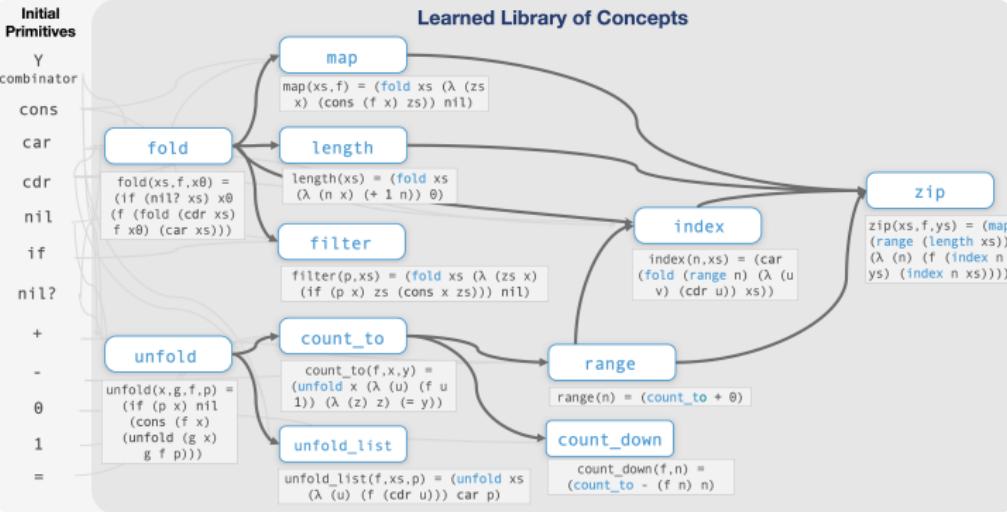
### List differences

$[1 \text{ } 8 \text{ } 2], [0 \text{ } 5 \text{ } 1] \rightarrow [1 \text{ } 3 \text{ } 1]$   
 $[2 \text{ } 3 \text{ } 6], [1 \text{ } 2 \text{ } 4] \rightarrow [1 \text{ } 1 \text{ } 2]$   
 $(\text{zip } \text{A} \text{ } \text{B})$



Origami Programming: Jeremy Gibbons, 2003

# Growing a language for recursive programming



## Recursive Programming Algorithms

### Stutter

$[\text{■■■}] \rightarrow [\text{■■■■■■}]$   
 $[\text{■■■■}] \rightarrow [\text{■■■■■■■■}]$   
 $(\text{fold } A (\lambda (\text{u } v)) (\text{cons } v (\text{cons } v \text{ u}))) \text{ nil}$

### Take every other

$[\text{■■■■■■■■}] \rightarrow [\text{■■}]$   
 $[\text{■■■■■■■■■■}] \rightarrow [\text{■■■■}]$   
 $(\text{unfold\_list } \text{cdr } A \text{ nil?})$

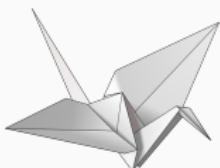
### List lengths

$[[\text{■■■}], [\text{■}]] \rightarrow [3 \ 1]$   
 $[[\text{■■■■}], [], [\text{■}]] \rightarrow [2 \ 0 \ 1]$   
 $(\text{map } A \text{ length})$

### List differences

$[1 \ 8 \ 2], [0 \ 5 \ 1] \rightarrow [1 \ 3 \ 1]$   
 $[2 \ 3 \ 6], [1 \ 2 \ 4] \rightarrow [1 \ 1 \ 2]$   
 $(\text{zip } A \text{ - } B)$

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

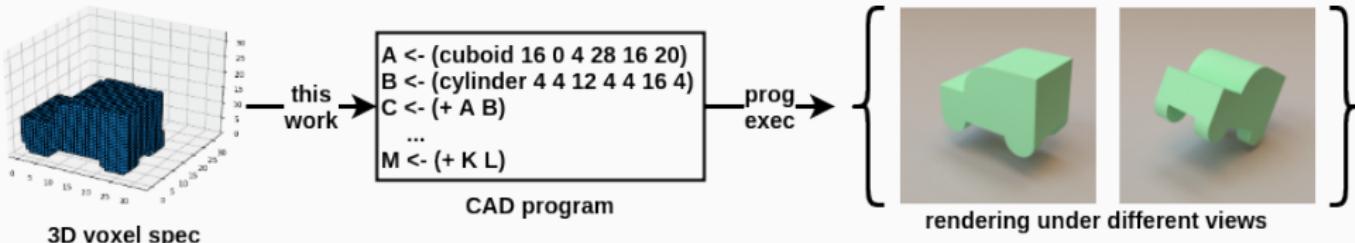
Library learning interacts synergistically with neural synthesis:  
bootstrapping, more than sum of parts

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.

# 3D program induction

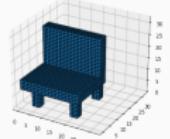


Challenge: combinatorial search!

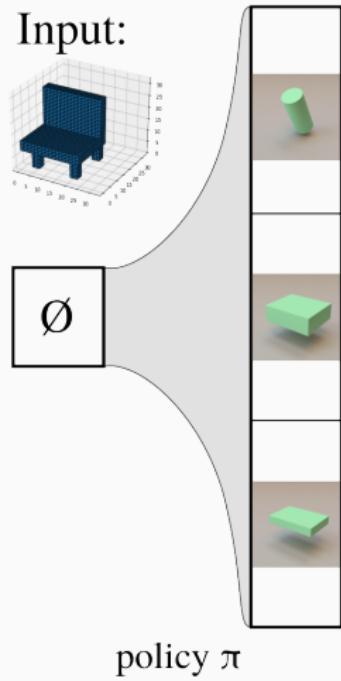
Branching factor:  $> 1.3$  million per line of code,  $\approx 20$  lines of code  
search space size:  $(1.3 \text{ million})^{20} \approx 10^{122}$  programs

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]

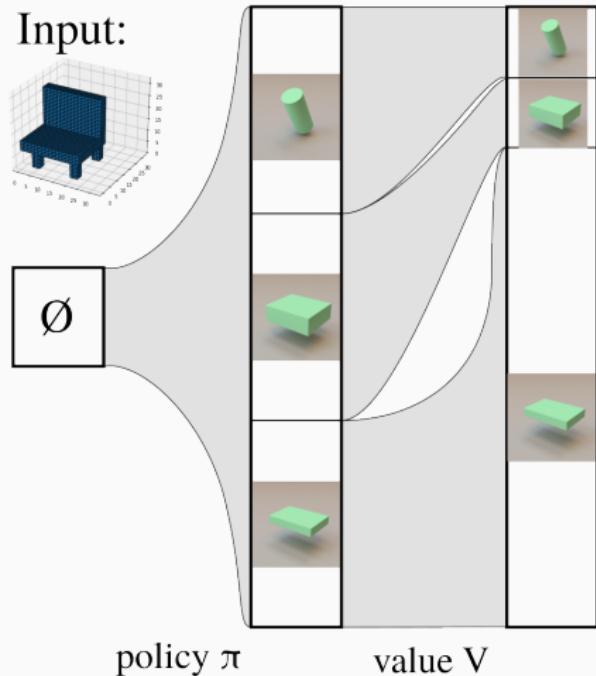
Input:



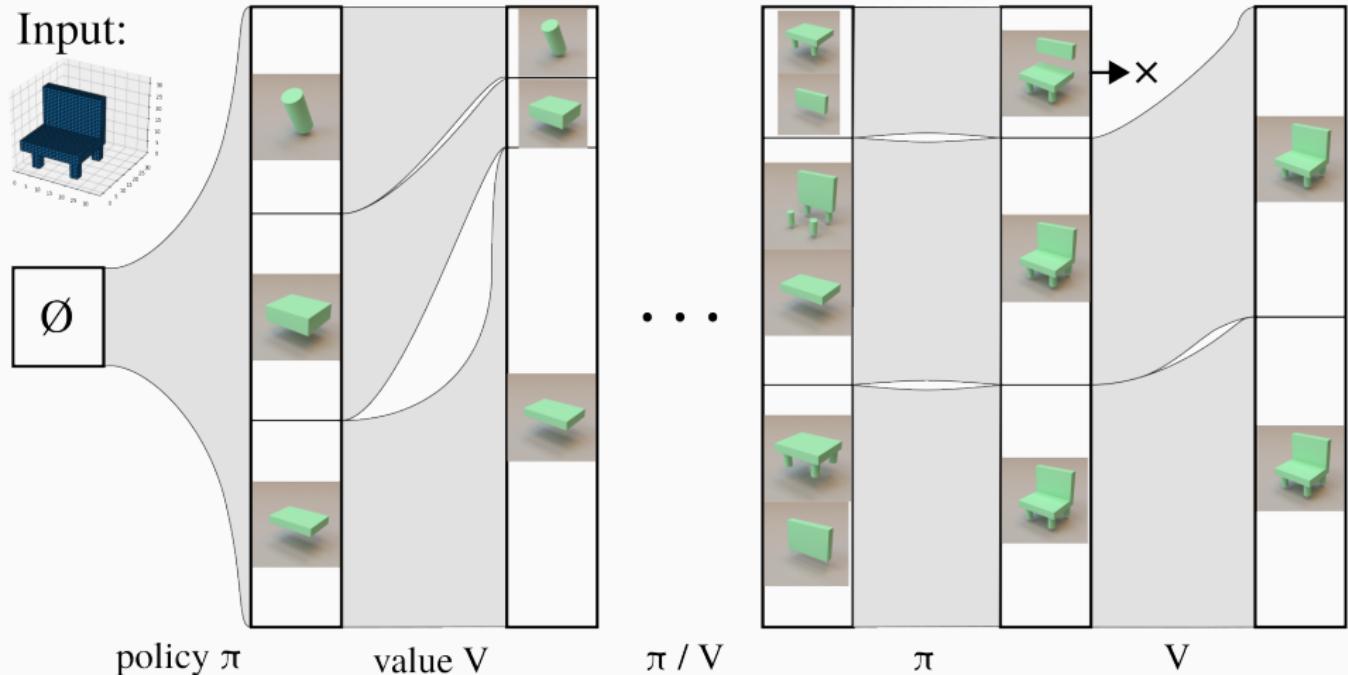
Solution: stochastic **tree search** + learn **policy** that writes code  
+ learn **value** function that assesses execution of program so far;  
analogous to **AlphaGo** [Silver et al. 2016]



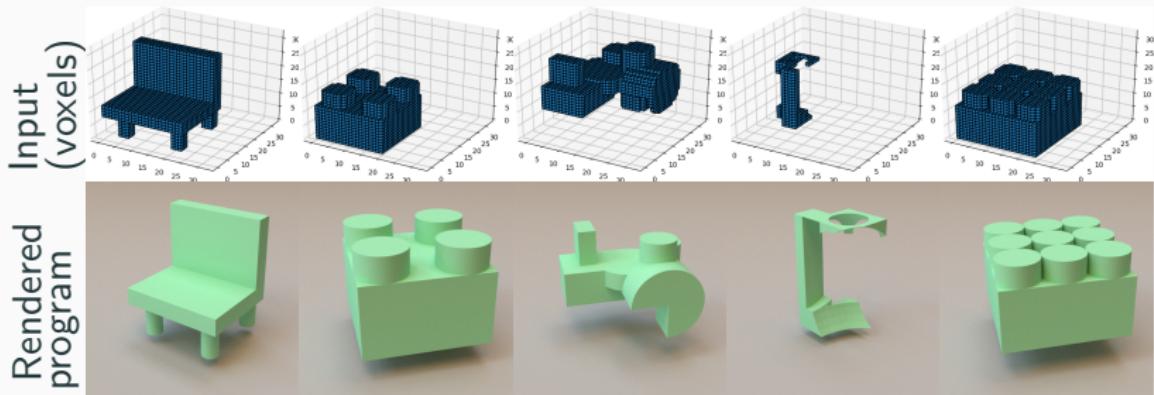
Solution: stochastic **tree search** + learn **policy** that writes code  
+ learn **value** function that assesses execution of program so far;  
analogous to **AlphaGo** [Silver et al. 2016]



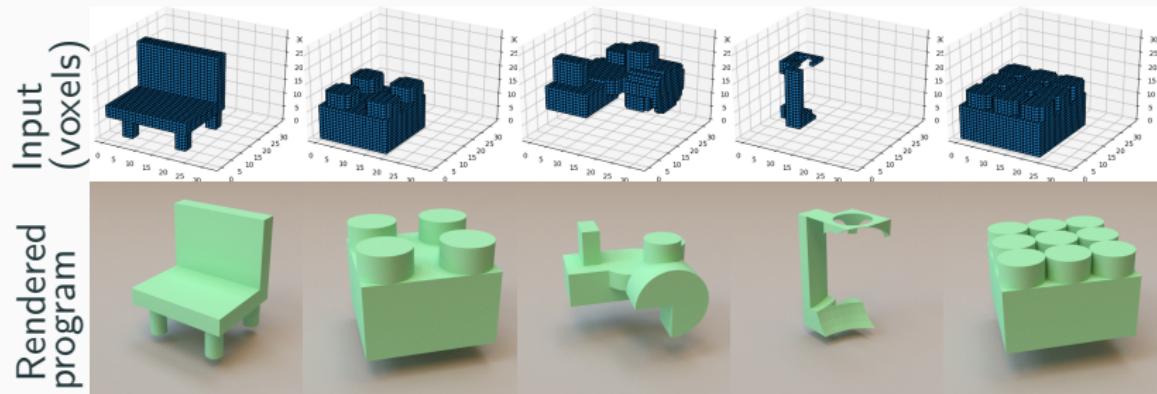
Solution: stochastic **tree search** + learn **policy** that writes code  
+ learn **value** function that assesses execution of program so far;  
analogous to **AlphaGo** [Silver et al. 2016]



# 3D program induction



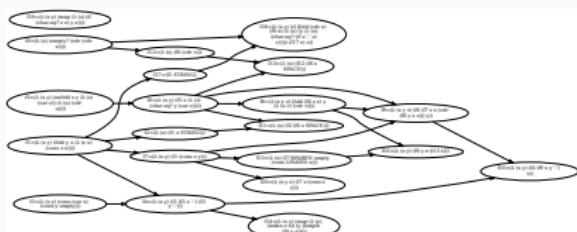
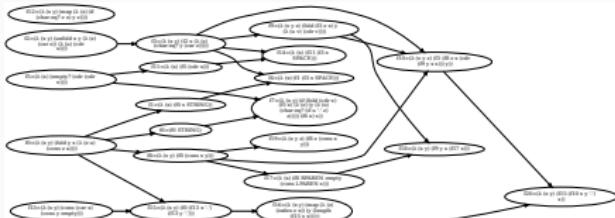
# 3D program induction



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

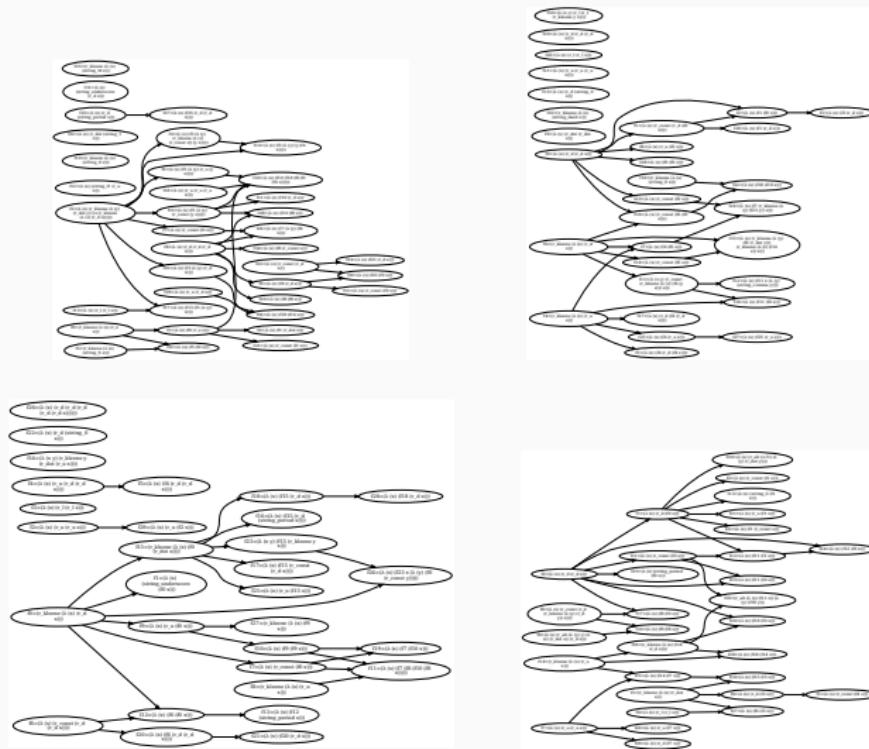
# Library structure: Text Editing

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

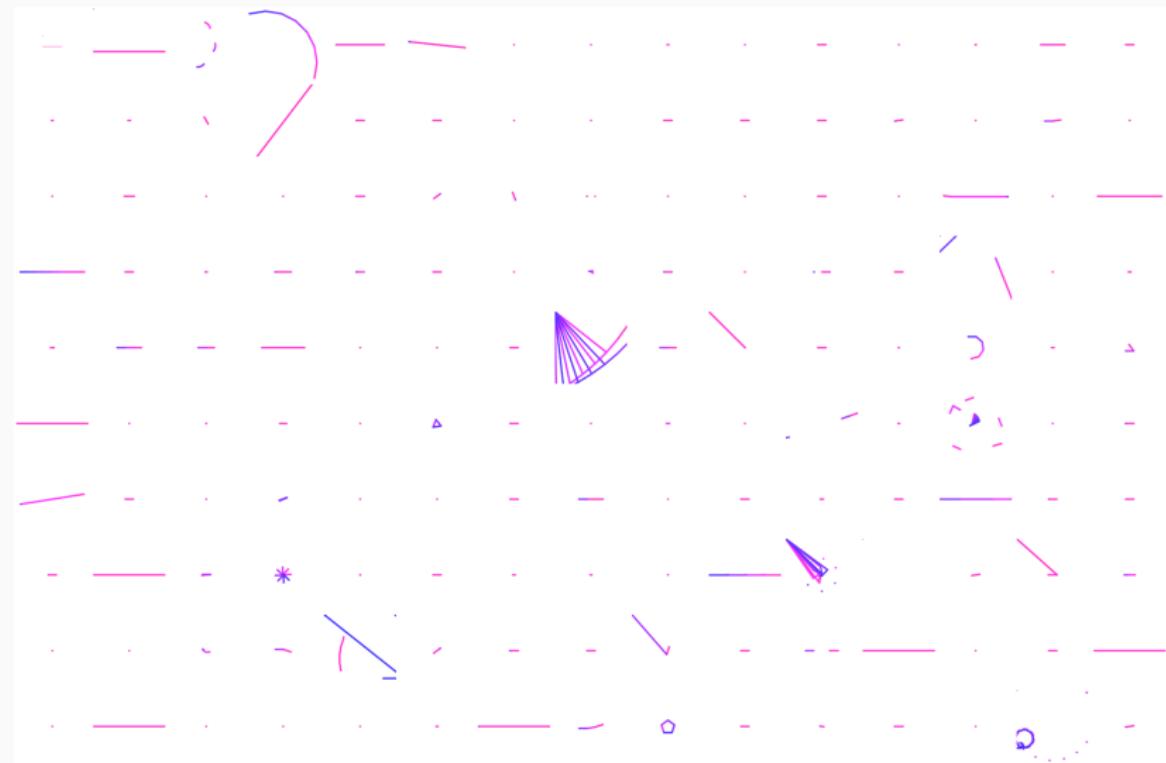


# Library structure: Generating Text

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



# 150 random dreams before learning



# 150 random dreams after learning

