

# **Program Induction: Bridging AI and program synthesis**

---

Kevin Ellis

2020

MIT

# What computational problems are solved by intelligence?

an endless range of problems

language



using new devices



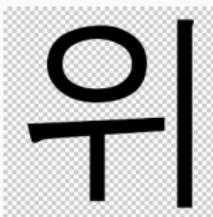
engineering



science



writing new characters



design



coding

```
(MEMBER  
(LAMBDA (X L)  
(COND ((NULL L) NIL)  
      ((EQ X (FIRST L)) T)  
      (T (MEMBER X (REST L)))))))
```

Allen, Anatomy of Lisp, 1975



play



# What computational problems are solved by intelligence?

an endless range of problems

language



science



design



- learning from modest data/experience
- communicating & representing knowledge in understandable formats
- bootstrapping & learning-to-learn
- creatively composing knowledge to produce new concepts and artifacts  
(& compositionality)



Allen, Anatomy of Lisp, 1975

engineering



play



## What computational substrates can contribute to this picture?

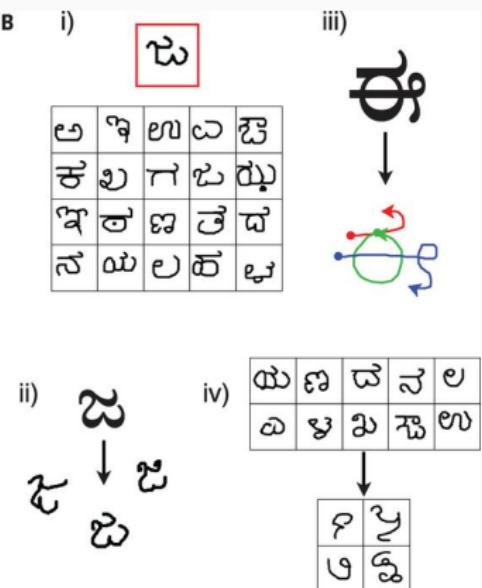
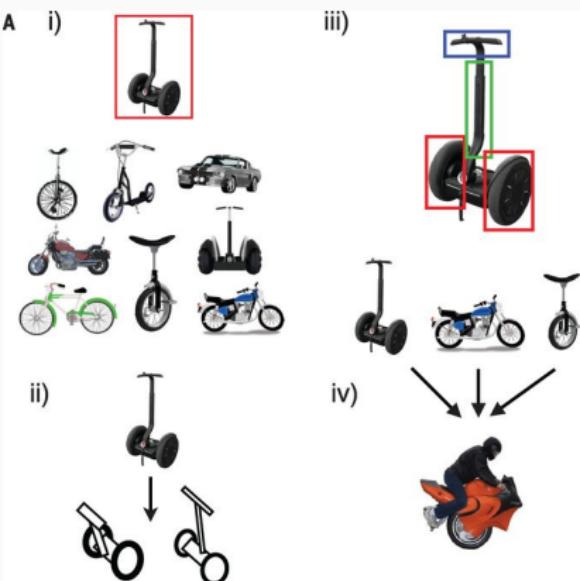
Three AI traditions: Symbolic, Probabilistic, Neural

# What computational substrates can contribute to this picture?

Three AI traditions: Symbolic, Probabilistic, Neural

**Program induction:** roots in the works of Solomonoff, Fodor, McCarthy.

Combines and complements these three traditions.



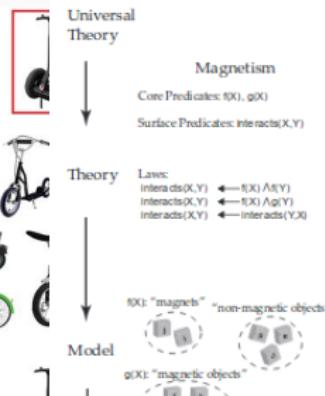
# What computational substrates can contribute to this picture?

Three AI traditions: Symbolic, Probabilistic, Neural

**Program induction:** roots in the works of Solomonoff, Fodor, McCarthy.

Combines and complements these three traditions.

A i)



ii)



Probabilistic Horn Clause Grammar

**Magnetism**

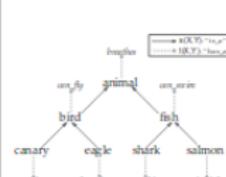
Core Predicates:  $I(X)$ ,  $g(X)$   
Surface Predicates:  $\text{Interacts}(X,Y)$

**Taxonomy**

Core Predicates:  $s(X,Y)$ ,  $t(X,Y)$   
Surface Predicates:  $\text{has\_a}(X,Y)$ ,  $\text{is\_a}(X,Y)$

**Kinship**

Core Predicates:  $u(X)$ ,  $v(X,Y)$ ,  $w(X,Y)$   
Surface Predicates:  $\text{female}(X)$ ,  $\text{child}(X,Y)$ ,  $\text{parent}(X,Y)$ ,  
 $\text{spouse}(X,Y)$ ,  $\text{father}(X,Y)$ , ...

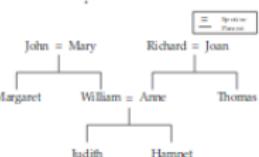


"a shark is a fish"  
"a bird can fly"  
"a canary can fly"  
"a salmon can breathe"

Ellipses indicate more facts follow.

**Kinship**

Core Predicates:  $u(X)$ ,  $v(X,Y)$ ,  $w(X,Y)$   
Surface Predicates:  $\text{female}(X)$ ,  $\text{spouse}(X,Y)$ ,  
 $\text{spouse}(Y,X)$ ,  $\text{V(X,Y)}$ ,  
 $\text{child}(X,Y)$ ,  $\text{child}(Y,X)$ ,  $\text{parent}(X,Y)$ ,  
 $\text{parent}(Y,X)$ ,  $\text{spouse}(Z,Y)$ ,  
 $\text{father}(X,Y)$ ,  $\text{mother}(Y,X)$ , ...



"John is William's father"  
"John is Judith's grandfather"  
"Judith is Hamnet's sister"  
"Margaret is Judith's aunt"

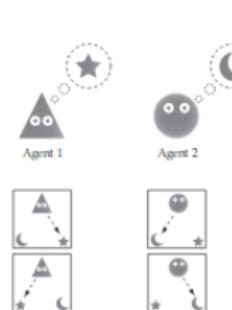
Ellipses indicate more facts follow.

**Psychology**

Core Predicates:  $d(X,Y)$   
Surface Predicates:  $\text{goes\_to}(X,Y,S)$   
Background:  $\text{location}(X,Y,S)$

Laws:  
 $\text{goes\_to}(X,Y,S) \leftarrow d(X,Z) \wedge \text{location}(Z,Y,S)$

Ellipses indicate more laws follow.



# What computational substrates can contribute to this picture?

Three AI traditions: Symbolic, Probabilistic, Neural

**Program induction:** roots in the works of Solomonoff, Fodor, McCarthy.

Combines and complements these three traditions.

A i)



Universal Theory

Magnetism  
Core Predicates:  $\text{m}(X)$ ,  $\text{p}(X)$

Probabilistic Horn Clause Grammar

Taxonomy  
Core Predicates:  $\text{t}(X, Y)$ ,  $\text{b}(X, Y)$

Kinship

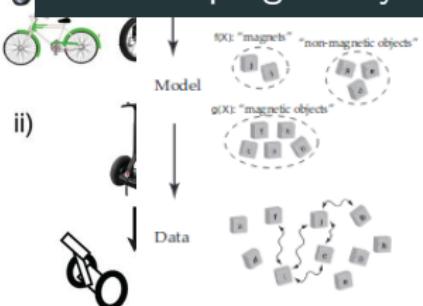
Core Predicates:  $\text{u}(X)$ ,  $\text{v}(X, Y)$ ,  $\text{w}(X, Y)$

Psychology

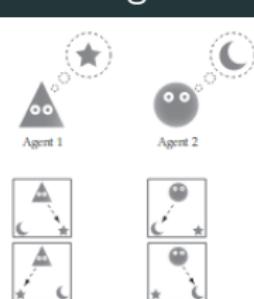
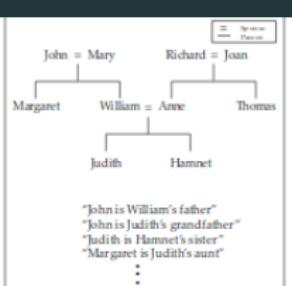
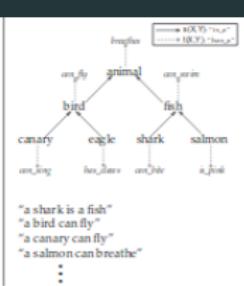
Core Predicates:  $\text{d}(X, Y)$

my work:

how learning can make program synthesis more scalable & practical  
how program synthesis can contribute to machine intelligence

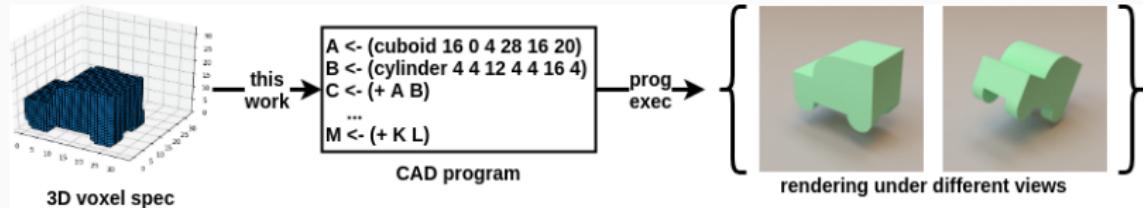
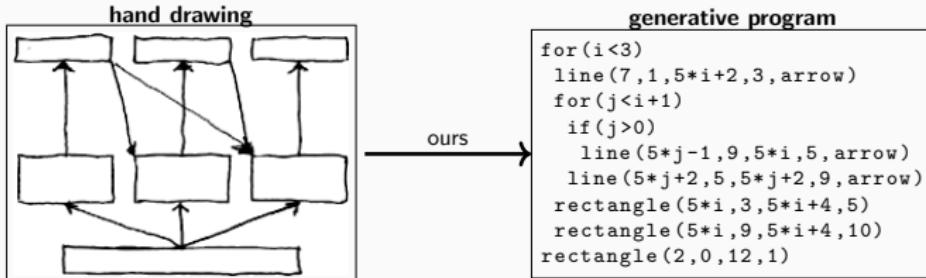


ii)



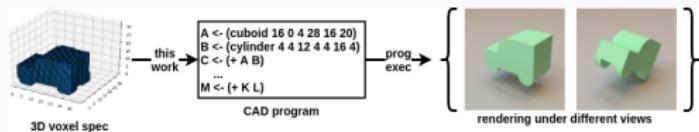
# Perception, Learning-to-Learn, Synthesizing models

Theme #1: high-level scene understanding, pixels→programs



# Perception, Learning-to-Learn, Synthesizing models

Theme #1: high-level scene understanding, pixels→programs



Theme #2: Learning to synthesize programs

List Processing

*Sum List*

$[1 \ 2 \ 3] \rightarrow 6$

$[4 \ 6 \ 8 \ 1] \rightarrow 17$

*Double*

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

$[6 \ 5 \ 1] \rightarrow [12 \ 10 \ 2]$

*Check Evens*

$[0 \ 2 \ 3] \rightarrow [T \ T \ F]$

$[2 \ 4 \ 9 \ 6] \rightarrow [T \ T \ F \ T]$

Text Editing

*Abbreviate*

Allen Newell → A.N.

Herb Simon → H.S.

*Drop Last Characters*

jabberwocky → jabberw

copycat → cop

Regexes

*Phone Numbers*

(555) 867-5309

(650) 555-2368

LOGO Graphics

*Currency*

\$100.25

\$4.50

*Dates*

Y1775/0704

Y2000/0101

Block Towers



Symbolic Regression



Recursive Programming

*Filter*

$[\text{red red}] \rightarrow [\text{red}]$

$[\text{red red black}] \rightarrow [\text{black}]$

$[\text{red black red}] \rightarrow [\text{red}]$

*Index List*

$0, [\text{red red red}] \rightarrow \text{red}$

$1, [\text{red red black}] \rightarrow \text{black}$

$1, [\text{red black red}] \rightarrow \text{red}$

Physics

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

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

*Length*

$[\text{red red red}] \rightarrow 4$

$[\text{red red black}] \rightarrow 6$

$[\text{red black red}] \rightarrow 3$

*Every Other*

$[\text{red red}] \rightarrow [\text{red}]$

$[\text{red red black}] \rightarrow [\text{black}]$

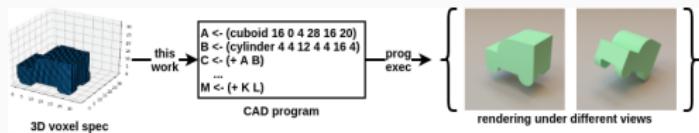
$[\text{red black red}] \rightarrow [\text{red}]$

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

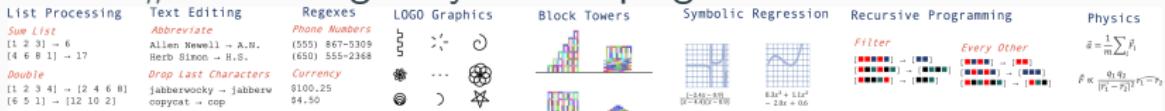
$$R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$$

# Perception, Learning-to-Learn, Synthesizing models

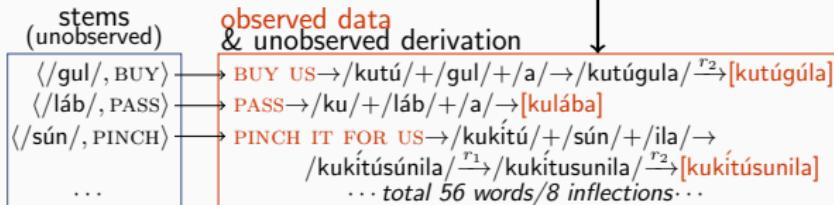
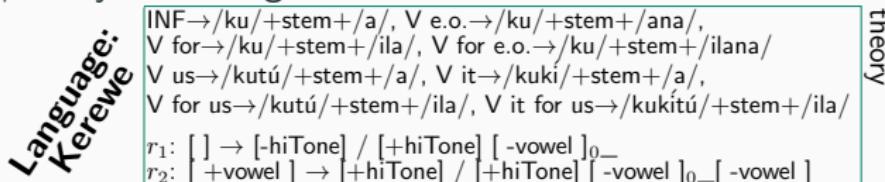
Theme #1: high-level scene understanding, pixels→programs



Theme #2: Learning to synthesize programs



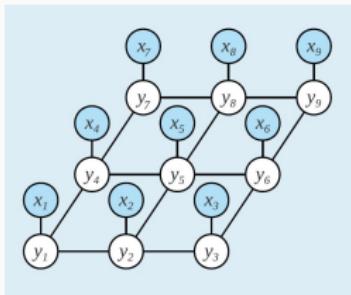
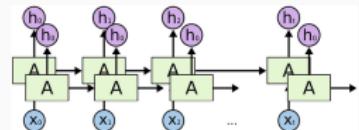
Theme #3: Synthesizing human-understandable models



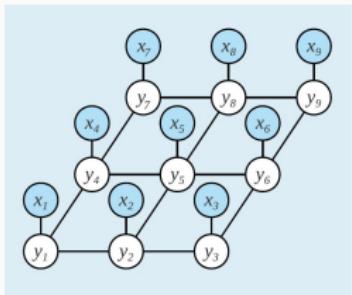
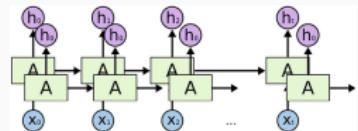
# High-level, abstract visual abilities



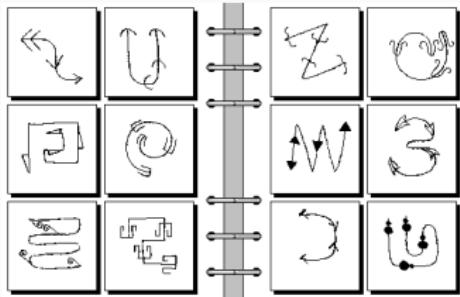
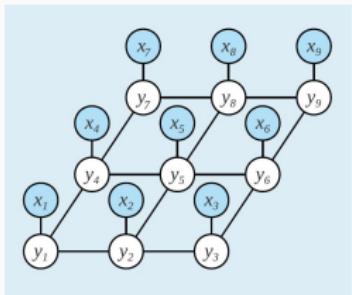
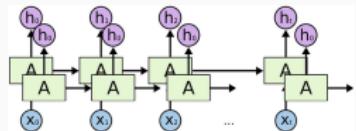
# High-level, abstract visual abilities



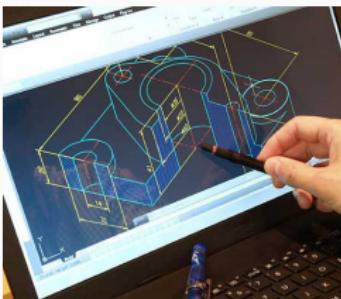
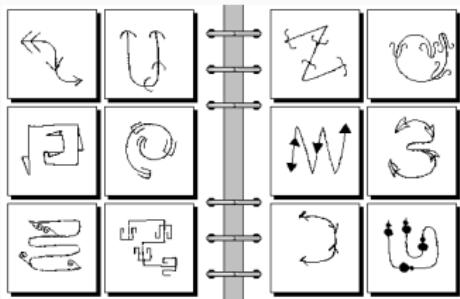
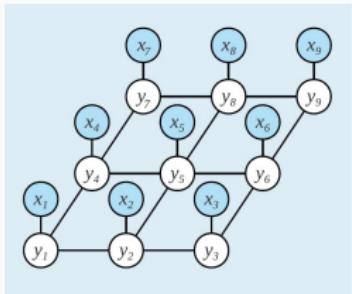
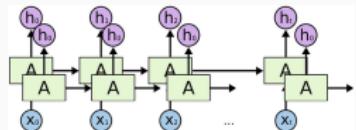
# High-level, abstract visual abilities



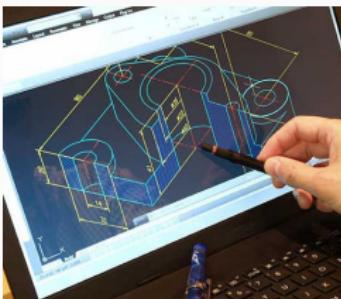
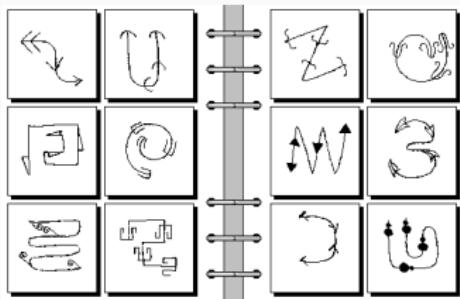
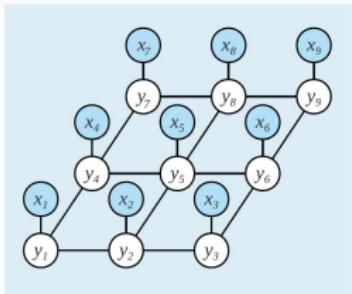
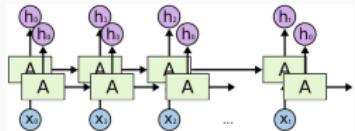
# High-level, abstract visual abilities



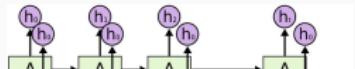
# High-level, abstract visual abilities



# High-level, abstract visual abilities



# High-level, abstract visual abilities

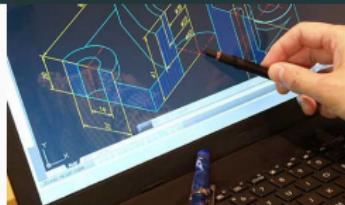
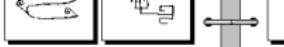


why?

impute missing objects, extrapolate percepts,  
learn visual concepts ('arch', 'spiral', HMM),  
assist graphic design, assist 3D modeling

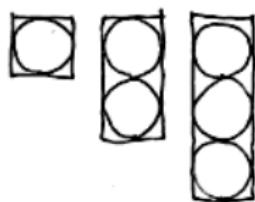
how?

Bayesian inference of graphics program conditioned on image,  
+program synthesis  
+learning



# Learning to infer graphics programs from hand-drawn images

model infers program from drawing



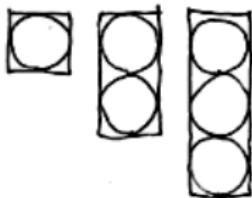
→

```
for (0 <= i < 3)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```

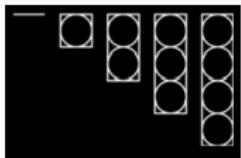
# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

**zero-shot generalization / extrapolation**



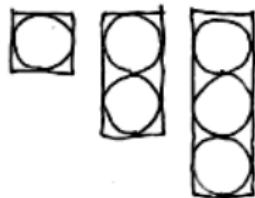
```
for (0 <= i < 3 + 1)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```



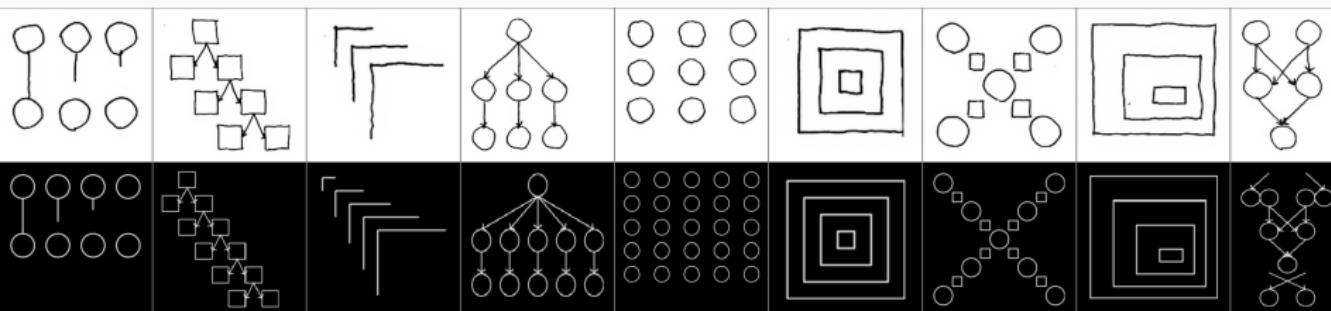
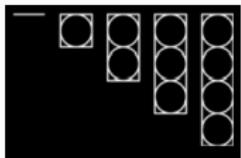
# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

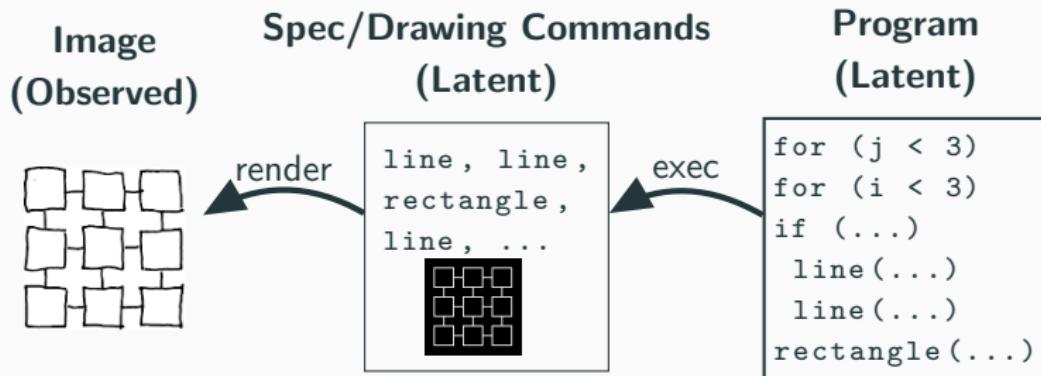
**zero-shot generalization / extrapolation**



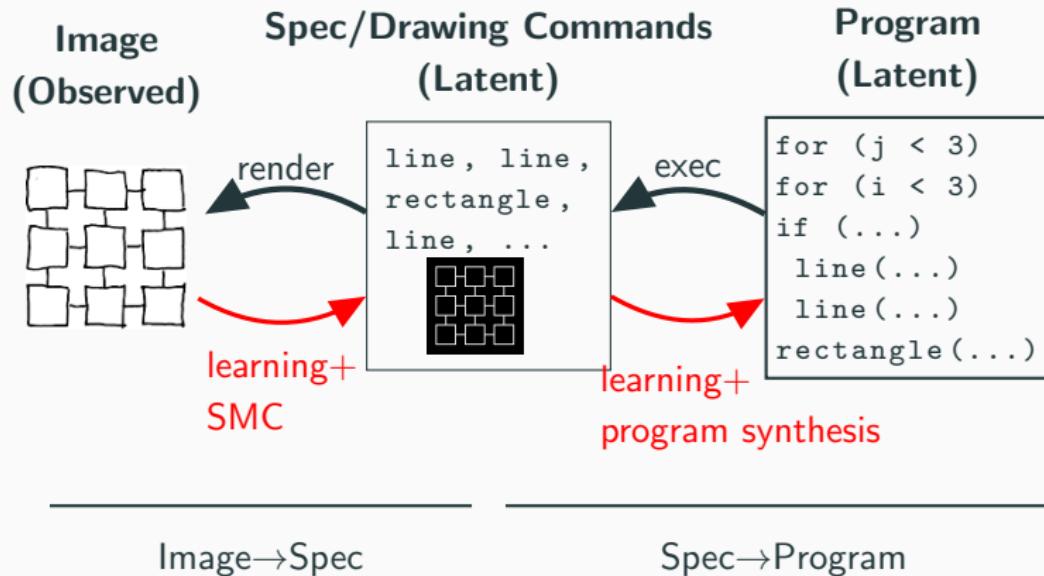
```
for (0 <= i < 3 + 1)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```



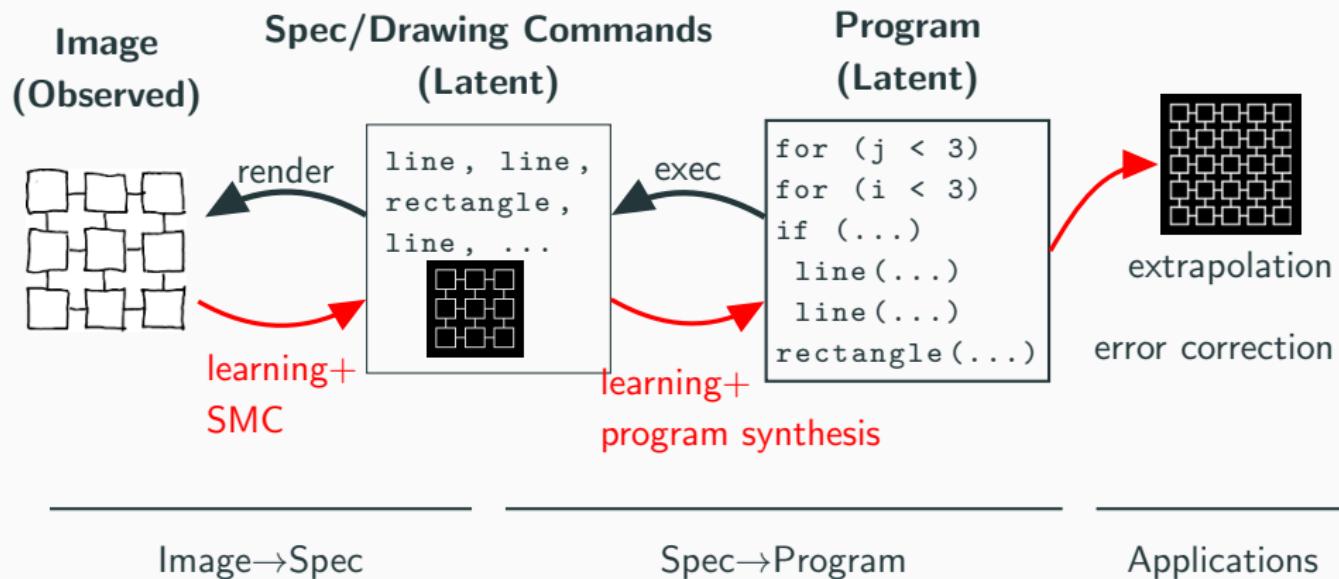
# How to infer graphics programs from hand-drawn images



# How to infer graphics programs from hand-drawn images



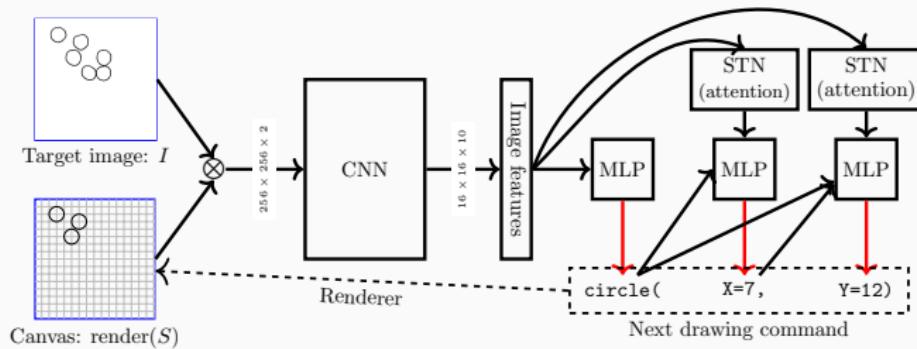
# How to infer graphics programs from hand-drawn images



# Parsing images into specs ( $\text{\LaTeX}$ TikZ commands)

Neurally Guided Procedural Modeling (Ritchie et al 2016) +

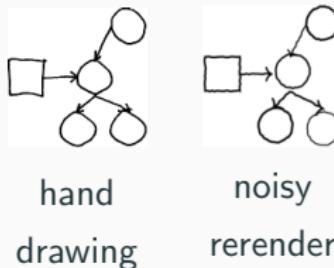
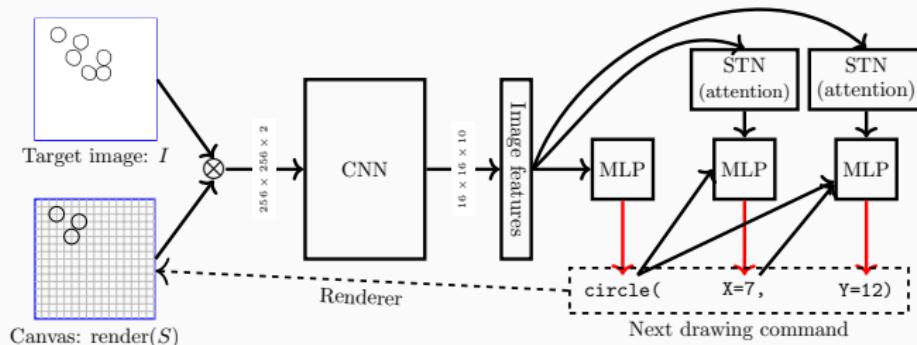
Attend, Infer, Repeat (Eslami et al 2016)



# Parsing images into specs ( $\text{\LaTeX}$ TikZ commands)

Neurally Guided Procedural Modeling (Ritchie et al 2016) +

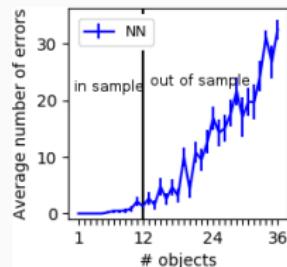
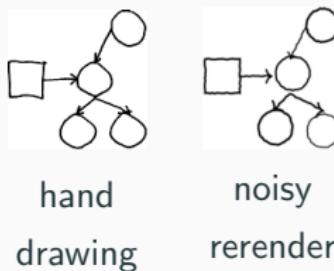
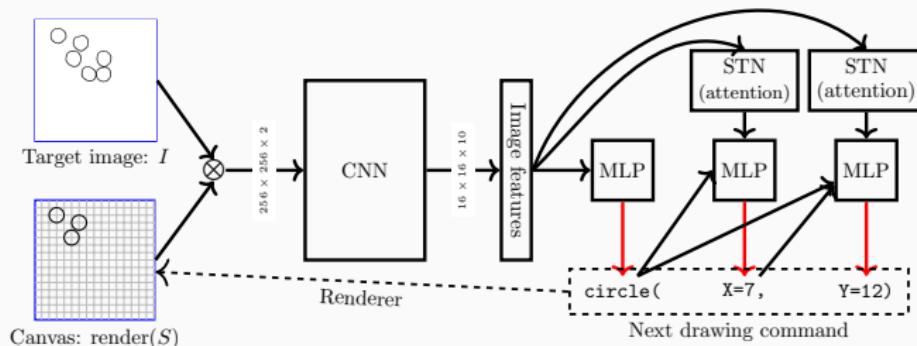
Attend, Infer, Repeat (Eslami et al 2016)



# Parsing images into specs ( $\text{\LaTeX}$ TikZ commands)

Neurally Guided Procedural Modeling (Ritchie et al 2016) +

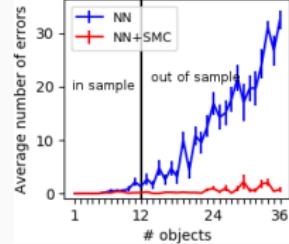
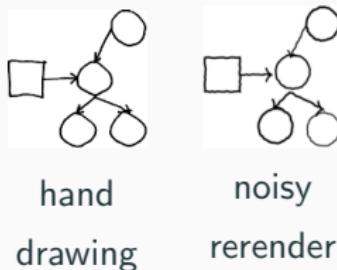
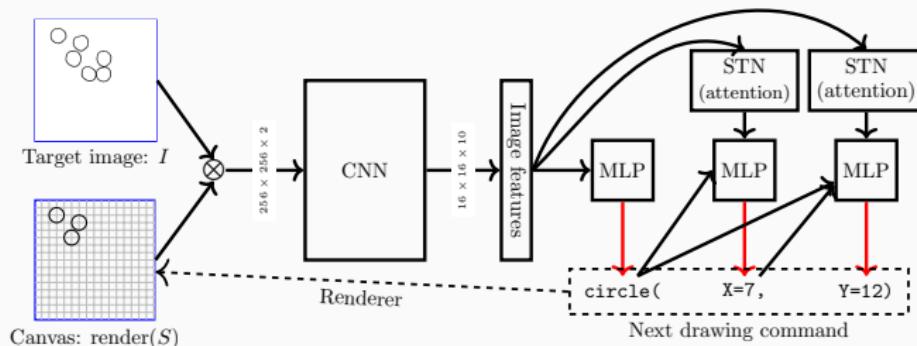
Attend, Infer, Repeat (Eslami et al 2016)



# Parsing images into specs ( $\text{\LaTeX}$ TikZ commands)

Neurally Guided Procedural Modeling (Ritchie et al 2016) +

Attend, Infer, Repeat (Eslami et al 2016)



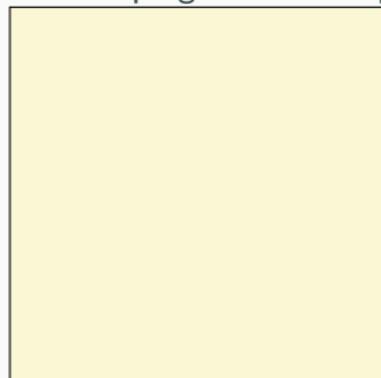
## Learning to quickly synthesize programs

Learn search policy  $\pi(\text{program subspace}|\text{spec})$

Think of the subspace as an “ansatz”

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space



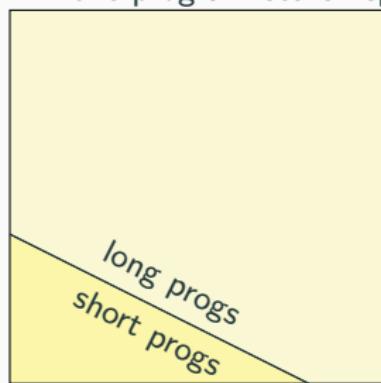
# Learning to quickly synthesize programs

Learn search policy  $\pi(\text{program subspace}|\text{spec})$

Think of the subspace as an “ansatz”

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space



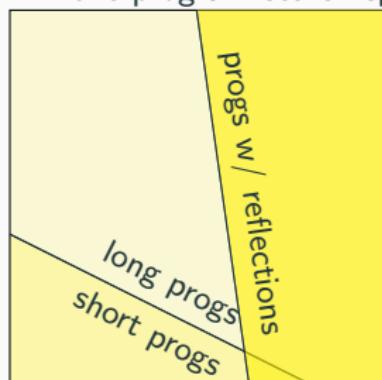
# Learning to quickly synthesize programs

Learn search policy  $\pi(\text{program subspace}|\text{spec})$

Think of the subspace as an “ansatz”

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space



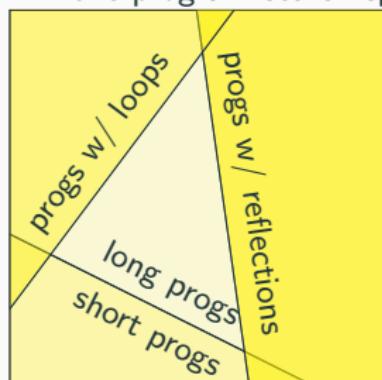
# Learning to quickly synthesize programs

Learn search policy  $\pi(\text{program subspace}|\text{spec})$

Think of the subspace as an “ansatz”

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space



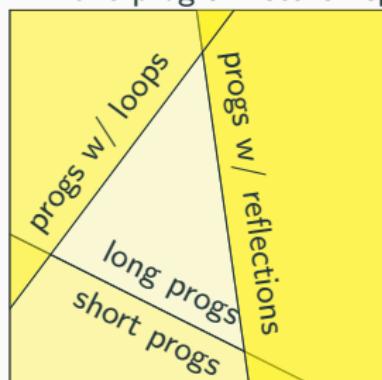
# Learning to quickly synthesize programs

Learn search policy  $\pi(\text{program subspace}|\text{spec})$

Think of the subspace as an “ansatz”

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space



$$\pi(\text{short, no loop/reflect}|S) = \square$$

$$\pi(\text{long, loops}|S) = \square$$

$$\pi(\text{long, no loop/reflect}|S) = \square$$

$$\pi(\text{long, reflects}|S) = \square$$

etc.

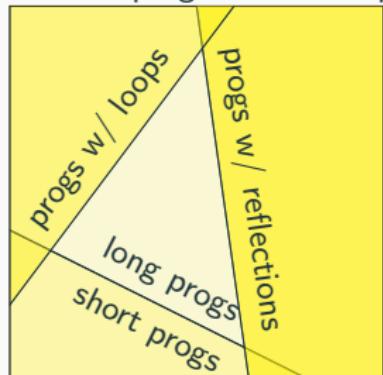
# Learning to quickly synthesize programs

Learn search policy  $\pi(\text{program subspace}|\text{spec})$

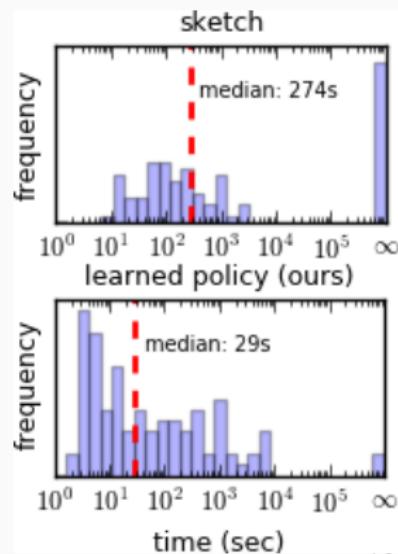
Think of the subspace as an “ansatz”

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space

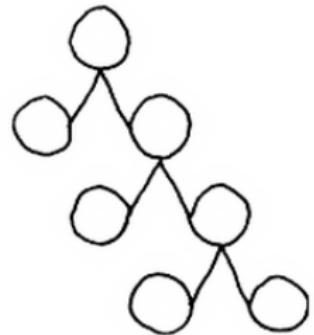


$$\begin{aligned}\pi(\text{short, no loop/reflect}|S) &= \square \\ \pi(\text{long, loops}|S) &= \square \\ \pi(\text{long, no loop/reflect}|S) &= \square \\ \pi(\text{long, reflects}|S) &= \square \\ \text{etc.} &\end{aligned}$$

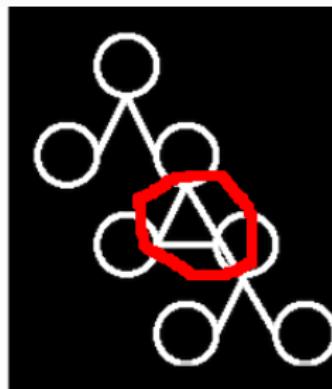


# Top-down influences on perception

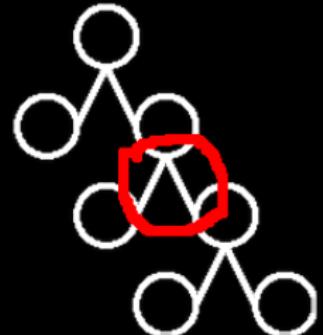
drawing



bottom-up neural net

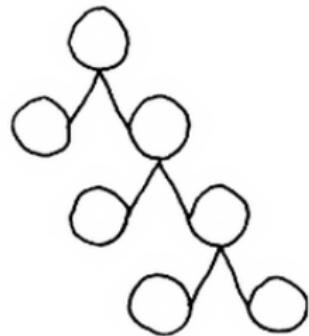


w/ top-down program bias

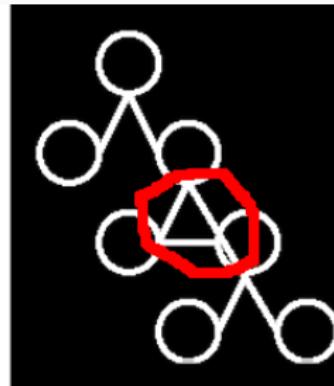


## Top-down influences on perception

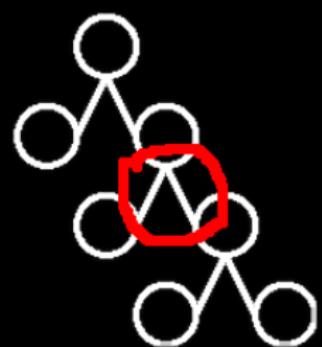
drawing



bottom-up neural net



w/ top-down program bias

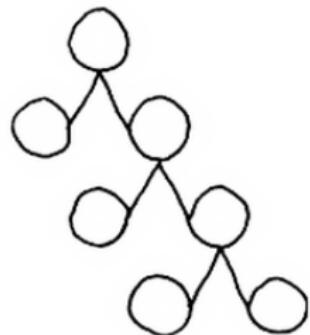


$$\hat{S}(I) = \arg \max_{S \in \mathcal{F}(I)} \mathbb{P}(I | \text{render}(S)) \times \mathbb{P}_\beta[\text{program}(S)]$$

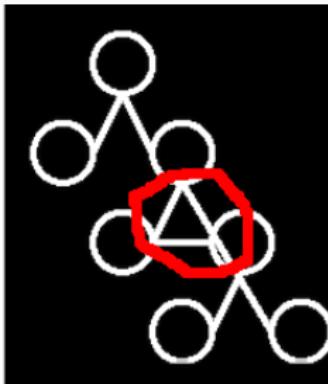
$$\beta^* = \arg \max_\beta \mathbb{E} \left[ \log \frac{\mathbb{P}_\beta[\text{program}(S)] \times \mathbb{P}(I | \text{render}(S))}{\sum_{S' \in \mathcal{F}(I)} \mathbb{P}_\beta[\text{program}(S')] \times \mathbb{P}(I | \text{render}(S'))} \right]$$

# Top-down influences on perception

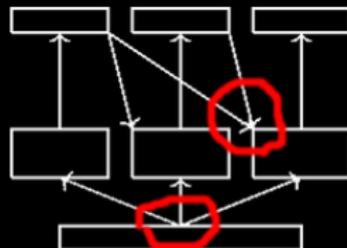
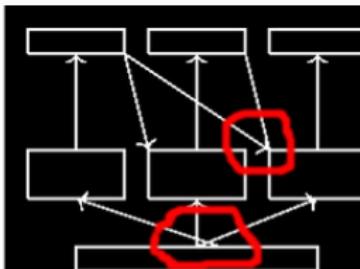
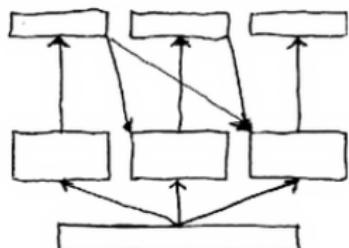
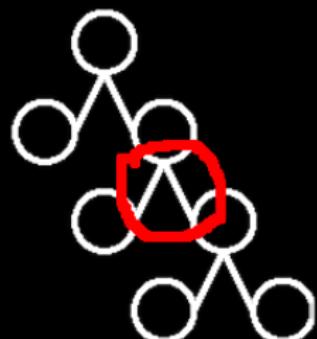
drawing



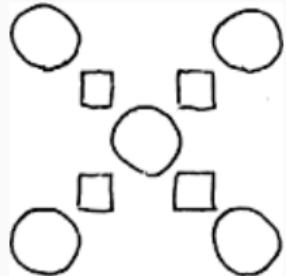
bottom-up neural net

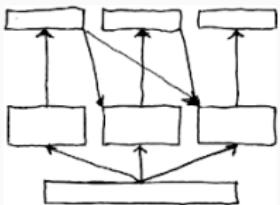


w/ top-down program bias



# Example programs

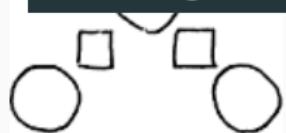
Drawing	Spec	Program
	<pre>Line(2,15, 4,15) Line(4,9, 4,13) Line(3,11, 3,14) Line(2,13, 2,15) Line(3,14, 6,14) Line(4,13, 8,13)</pre>	<pre>for(i&lt;3)     line(i,-1*i+6,           2*i+2,-1*i+6)     line(i,-2*i+4,i,-1*i+6)</pre>
	<pre>Circle(2,8) Rectangle(6,9, 7,10) Circle(8,8) Rectangle(6,12, 7,13) Rectangle(3,9, 4,10) ... etc. ....; 9 lines</pre>	<pre>reflect(y=8) for(i&lt;3)     if(i&gt;0)         rectangle(3*i-1,2,3*i,3)     circle(3*i+1,3*i+1)</pre>

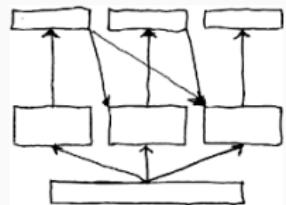
	<pre>Line(3,10,3,14,arrow) Rectangle(11,8,15,10) Rectangle(11,14,15,15) Line(13,10,13,14,arrow) ... etc. ....; 16 lines</pre>	<pre>for(i&lt;3)     line(7,1,5*i+2,3,arrow)     for(j&lt;i+1)         if(j&gt;0)             line(5*j-1,9,5*i,5,arrow)             line(5*j+2,5,5*j+2,9,arrow)         rectangle(5*i,3,5*i+4,5)         rectangle(5*i,9,5*i+4,10)     rectangle(2,0,12,1)</pre>
--	---	--

# Example programs

Drawing	Spec	Program
	<pre>Line(2,15, 4,15) Line(4,9, 4,13) Line(3,11, 3,14) Line(2,13, 2,15) Line(3,14, 6,14) Line(4,13, 8,13)</pre>	<pre>for(i&lt;3)     line(i,-1*i+6,           2*i+2,-1*i+6)     line(i,-2*i+4,i,-1*i+6)</pre>

Learning played a role...  
but much of this system is specific to 2-D graphics

	<pre>rectangle(8,12, 7,18) Rectangle(3,9, 4,10) ... etc. ....; 9 lines</pre>	<pre>rectangle(3*i-1,2,3*i,3) circle(3*i+1,3*i+1)</pre>
---	--	---

	<pre>Line(3,10,3,14,arrow) Rectangle(11,8,15,10) Rectangle(11,14,15,15) Line(13,10,13,14,arrow) ... etc. ....; 16 lines</pre>	<pre>for(i&lt;3)     line(7,1,5*i+2,3,arrow)     for(j&lt;i+1)         if(j&gt;0)             line(5*j-1,9,5*i,5,arrow)             line(5*j+2,5,5*j+2,9,arrow)         rectangle(5*i,3,5*i+4,5)         rectangle(5*i,9,5*i+4,10)         rectangle(2,0,12,1)</pre>
--	---	--

## Growing domain-specific knowledge

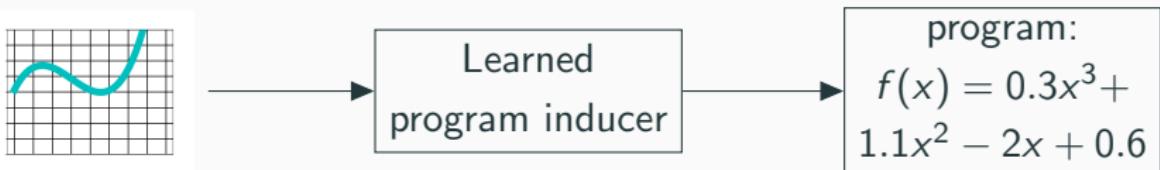
Goal: acquire domain-specific knowledge needed to induce a class of programs

- Library of concepts (declarative knowledge; domain specific language; generative model over programs)
- Inference strategy (procedural knowledge; synthesis algorithm)

# Growing domain-specific knowledge

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

- Library of concepts (declarative knowledge; domain specific language; generative model over programs)
- Inference strategy (procedural knowledge; synthesis algorithm)



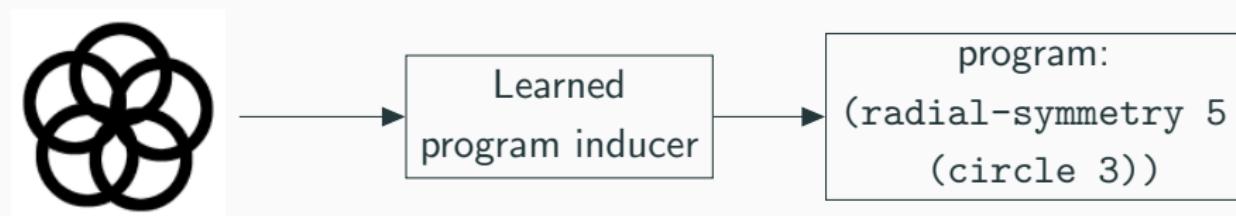
Concepts:  $x^3$ ,  $\alpha x + \beta$ , etc

Inference strategy: neurosymbolic search for programs

# Growing domain-specific knowledge

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

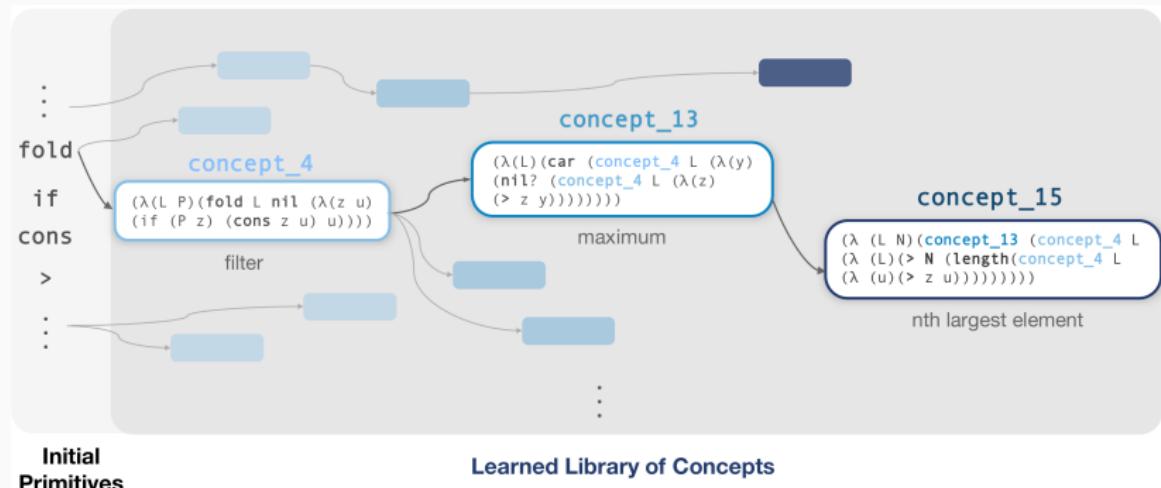
- Library of concepts (declarative knowledge; domain specific language; generative model over programs)
- Inference strategy (procedural knowledge; synthesis algorithm)



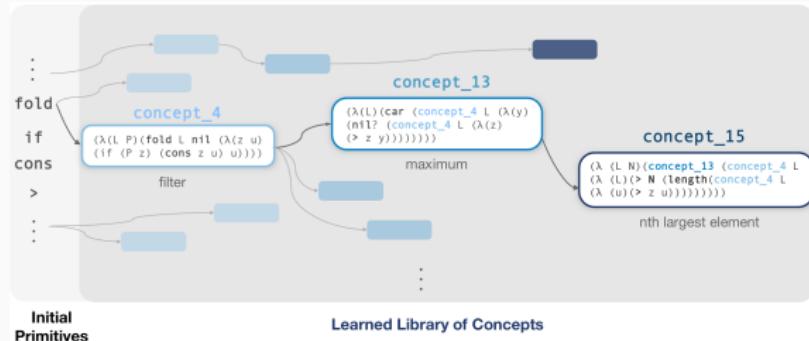
Concepts: circle, radial-symmetry, etc

Inference strategy: neurosymbolic search for programs

# Library learning



## Library learning



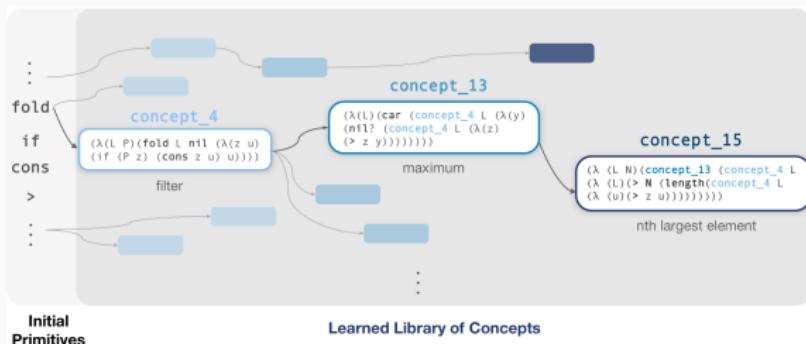
## Problem: sort list

### Solution:

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

get nth largest element where  $n = 1, 2, 3 \dots \text{length of list}$

## Library learning



## Problem: sort list

### Solution:

(map (λ (n) (concept\_15 L (+ 1 n))) (range (length L))))  
get nth largest element where n = 1, 2, 3 ...length of list

**Solution in initial primitives:**

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

# DreamCoder

- **Wake:** Solve problems by writing programs
- **Sleep:** Improve DSL and neural recognition model:
  - **Abstraction sleep:** Improve library
  - **Dream sleep:** Improve neural inference model
- Combines ideas from Wake-Sleep & Exploration-Compression



# DreamCoder

- **Wake:** Solve problems by writing programs
- **Sleep:** Improve DSL and neural recognition model:
  - **Abstraction sleep:** Improve library
  - **Dream sleep:** Improve neural inference model
- Combines ideas from Wake-Sleep & Exploration-Compression

## List Processing

*Sum List*  
 $[1 \ 2 \ 3] \rightarrow 6$   
 $[4 \ 6 \ 8 \ 1] \rightarrow 17$

## Double

$[1 \ 2 \ 3 \ 4] \rightarrow [2 \ 4 \ 6 \ 8]$   
 $[6 \ 5 \ 1] \rightarrow [12 \ 10 \ 2]$

## Check Evens

$[0 \ 2 \ 3] \rightarrow [T \ T \ F]$   
 $[2 \ 4 \ 9 \ 6] \rightarrow [T \ T \ F \ T]$

## Text Editing

*Abbreviate*  
Allen Newell → A.N.  
Herb Simon → H.S.

*Drop Last Characters*  
jabberwocky → jabberw  
copycat → cop

## Extract

see spot(run) → run  
a (bee) see → bee

## Regexes

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

## Currency

\$100.25  
\$4.50

## Dates

Y1775/07/04  
Y2000/01/01

## LOGO Graphics



Physics

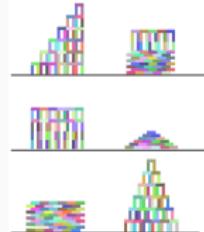
$$KE = \frac{1}{2}m|\vec{v}|^2$$

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

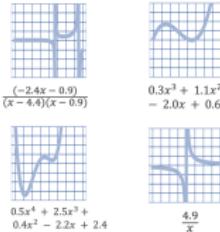
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{r}_1 - \hat{r}_2$$

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

## Block Towers



## Symbolic Regression



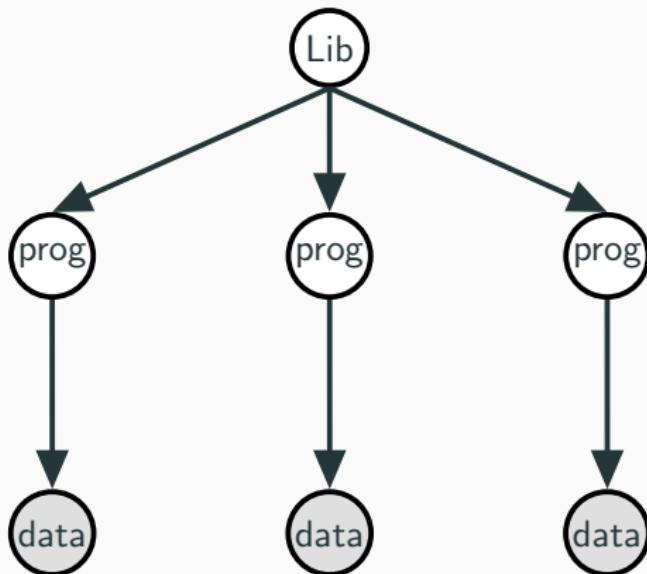
## Recursive Programming

Filter	Index List
$[\text{■■■■■}] \rightarrow [\text{■■}]$	$0, [\text{■■■■■}] \rightarrow \text{■}$
$[\text{■■■■■■■■}] \rightarrow [\text{■■■■■■}]$	$1, [\text{■■■■■■■■}] \rightarrow \text{■}$
$[\text{■■■■■■■■■}] \rightarrow [\text{■■■■■■■}]$	$1, [\text{■■■■■■■■■}] \rightarrow \text{■}$

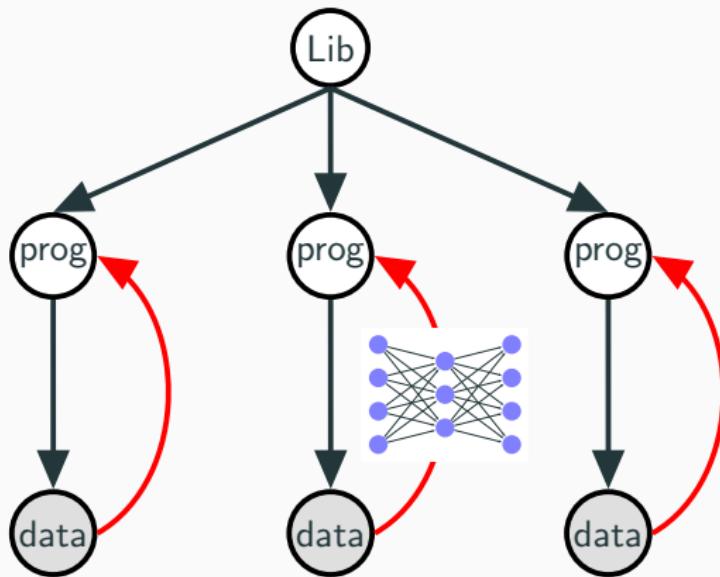
Length	Every Other
$[\text{■■■■■}] \rightarrow 4$	$[\text{■■■■■}] \rightarrow [\text{■■}]$
$[\text{■■■■■■■■}] \rightarrow 6$	$[\text{■■■■■■■■}] \rightarrow [\text{■■■■}]$
$[\text{■■■■■■■■■}] \rightarrow 3$	$[\text{■■■■■■■■■}] \rightarrow [\text{■■■}]$

## Library learning as Bayesian inference

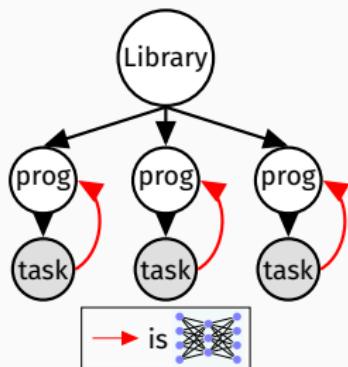


[Dechter et al., 2013] [Liang et al, 2010]; [Lake et al, 2015]

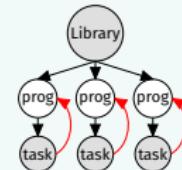
# Library learning as amortized Bayesian inference



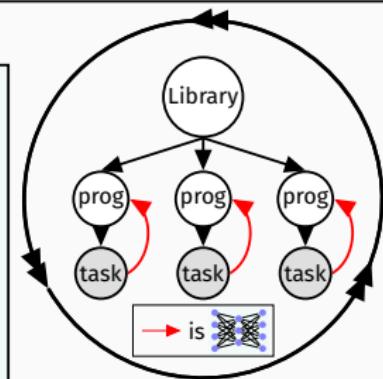
amortized inference +  
better program representation (Lisp) +  
library learning via program analysis +  
new neural inference network for program synthesis



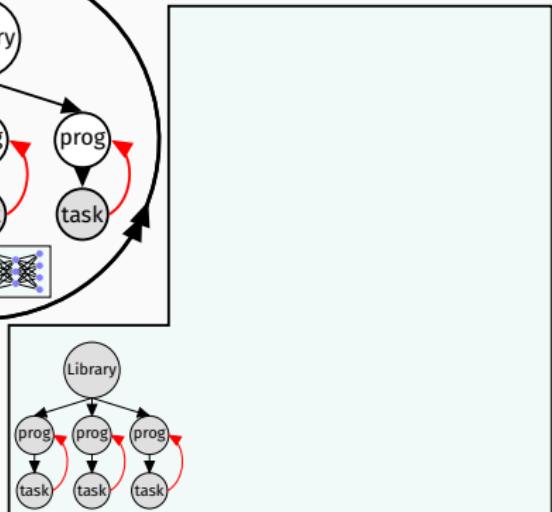
## WAKE

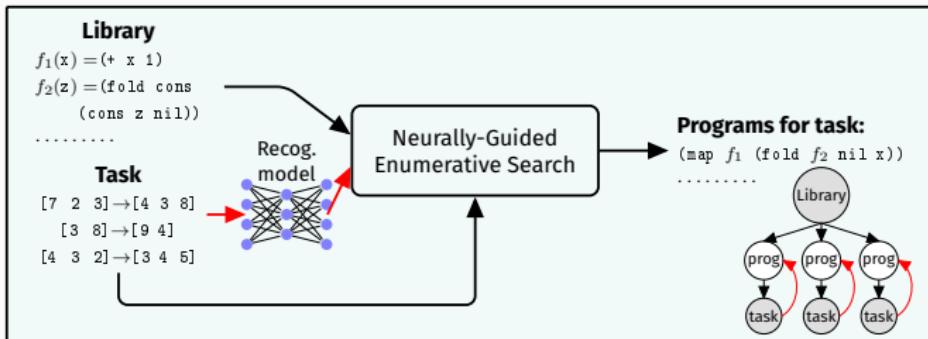
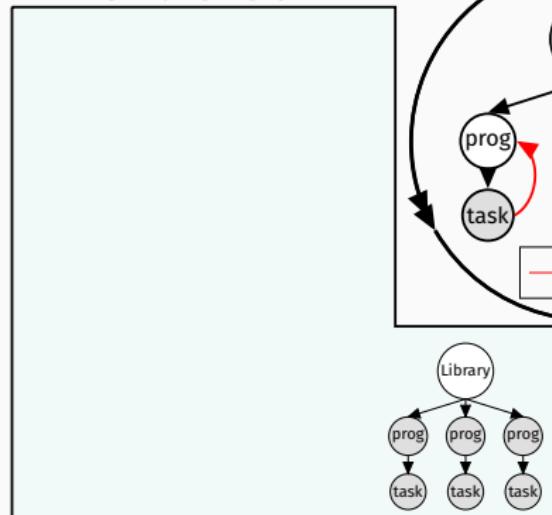
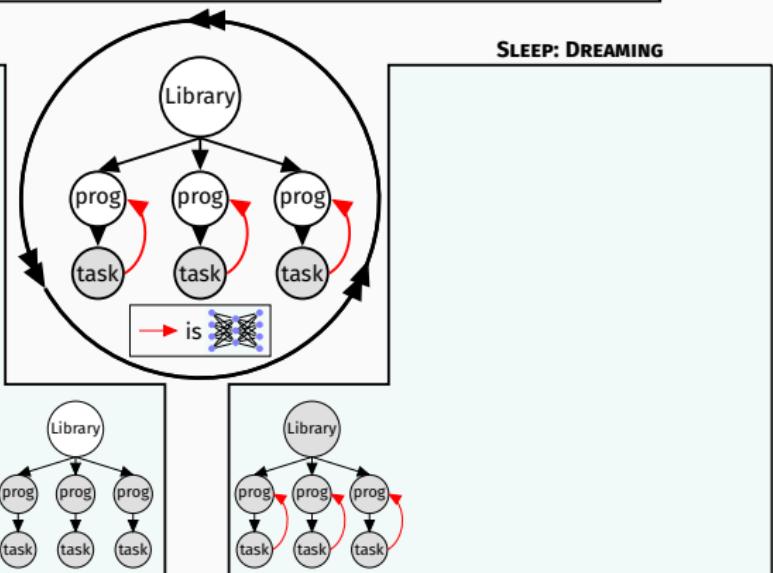


## SLEEP: ABSTRACTION

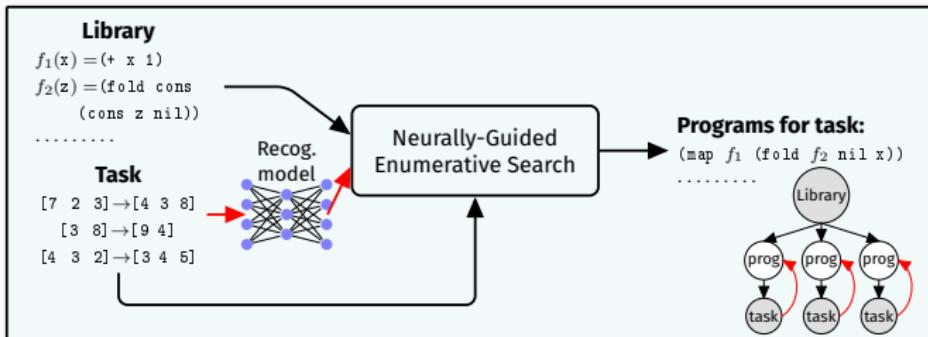


## SLEEP: DREAMING

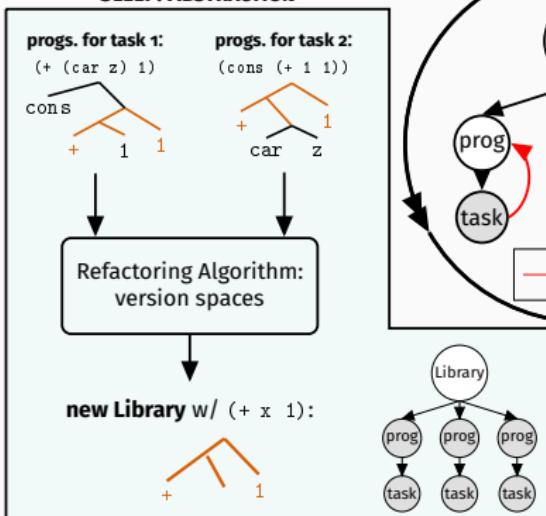


**WAKE****SLEEP: ABSTRACTION****SLEEP: DREAMING**

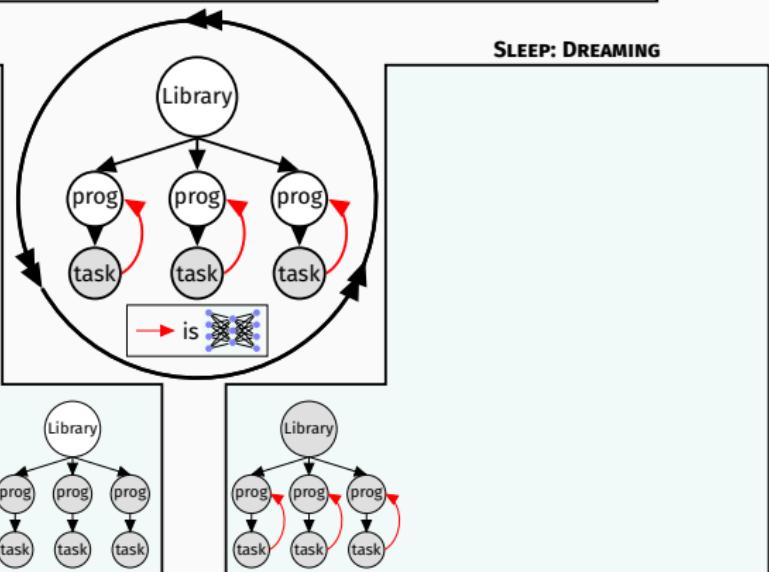
# WAKE



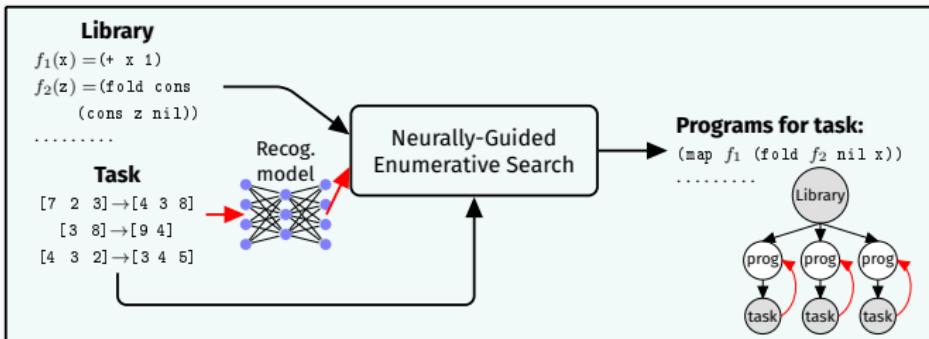
# SLEEP: ABSTRACTION



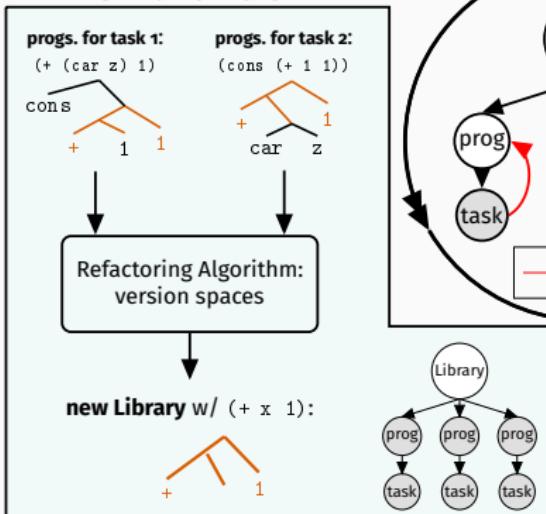
# SLEEP: DREAMING



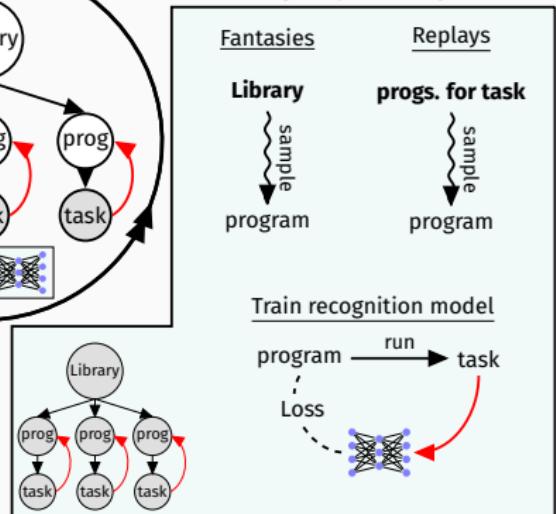
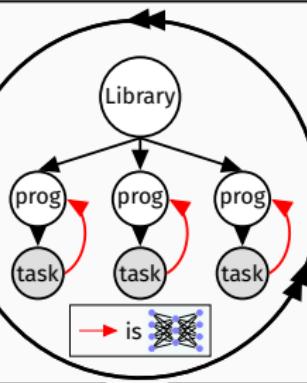
# WAKE



# SLEEP: ABSTRACTION



# SLEEP: DREAMING



# Abstraction Sleep: Growing the library via refactoring

**Task:**  $(1\ 2\ 3) \rightarrow (2\ 4\ 6)$   
 $(4\ 3\ 2) \rightarrow (8\ 6\ 8)$

Wake: program search

```
(Y (\lambda (r l) (if (nil? l) nil  
                      (cons (+ (car l) (car l))  
                            (r (cdr l)))))))
```

**Task:**  $(1\ 2\ 3) \rightarrow (0\ 1\ 2)$   
 $(4\ 3\ 2) \rightarrow (3\ 2\ 3)$

Wake: program search

```
(Y (\lambda (r l) (if (nil? l) nil  
                      (cons (- (car l) 1)  
                            (r (cdr l)))))))
```

# Abstraction Sleep: Growing the library via refactoring

Task:  $(1\ 2\ 3) \rightarrow (2\ 4\ 6)$   
 $(4\ 3\ 2) \rightarrow (8\ 6\ 8)$

Wake: program search

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

Task:  $(1\ 2\ 3) \rightarrow (0\ 1\ 2)$   
 $(4\ 3\ 2) \rightarrow (3\ 2\ 3)$

Wake: program search

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

refactor  
 $(10^{14}$  refactorings)

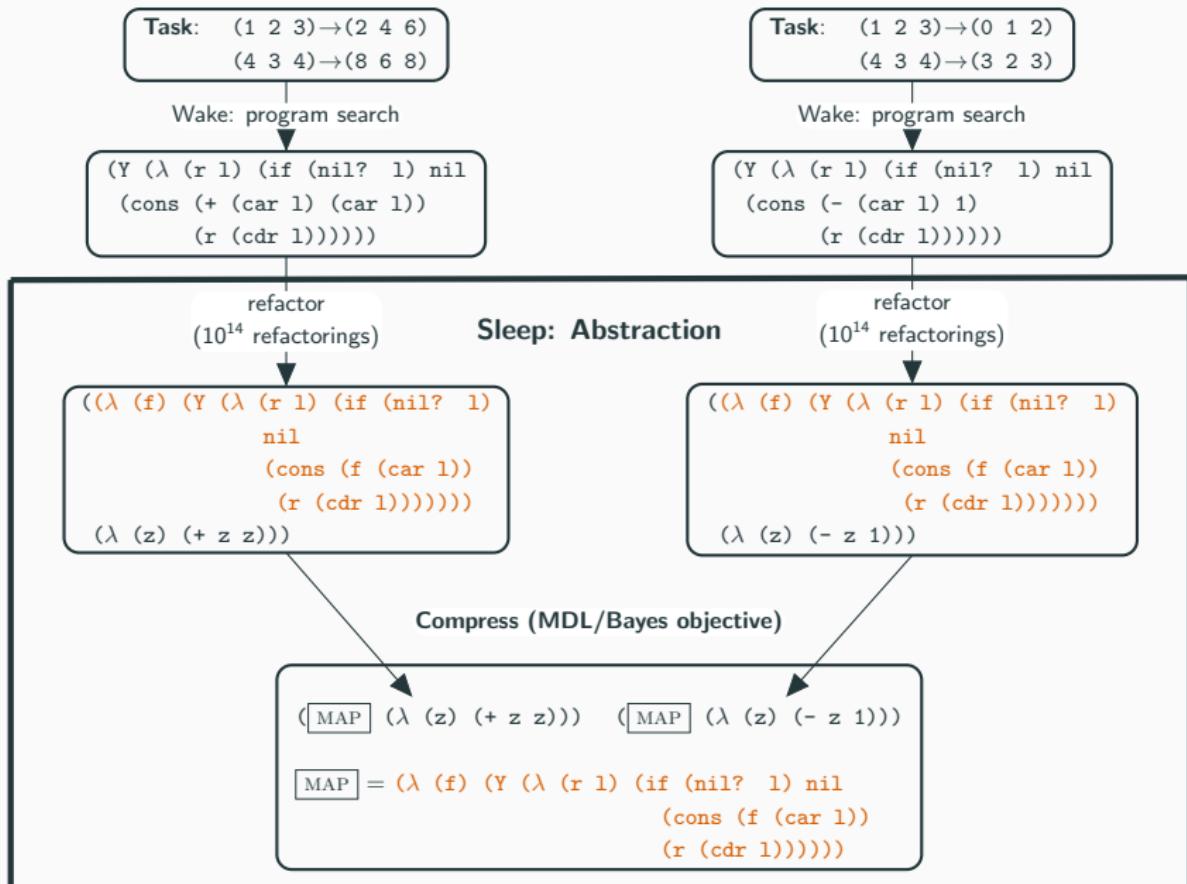
## Sleep: Abstraction

refactor  
 $(10^{14}$  refactorings)

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

```
((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



# Version space algebra for refactoring

$\text{Expr} \rightarrow \text{var}$

- |  $\lambda\text{var}.\text{Expr}$
- |  $(\text{Expr } \text{Expr})$
- | primitive

# Version space algebra for refactoring

$\text{Expr} \rightarrow \text{var}$

|  $\lambda\text{var}.\text{Expr}$   
| ( $\text{Expr } \text{Expr}$ )  
| primitive

$\text{VS} \rightarrow \text{var}$

|  $\lambda\text{var}.\text{VS}$   
| ( $\text{VS } \text{VS}$ )  
| primitive  
|  $\text{VS} \uplus \text{VS}$       *nondeterministic choice*  
|  $\Lambda$                   *choose any expression*  
|  $\emptyset$                   *choose no expression*

# Version space algebra for refactoring

$\text{Expr} \rightarrow \text{var}$

|  $\lambda \text{var}. \text{Expr}$   
| ( $\text{Expr} \ \text{Expr}$ )  
| primitive

$\text{VS} \rightarrow \text{var}$

|  $\lambda \text{var}. \text{VS}$   
| ( $\text{VS} \ \text{VS}$ )  
| primitive  
|  $\text{VS} \uplus \text{VS}$      *nondeterministic choice*  
|  $\Lambda$                 *choose any expression*  
|  $\emptyset$                 *choose no expression*

what version spaces mean:

$$[\![\text{var}]\!] = \{\text{var}\}$$

$$[\![v_1 \uplus v_2]\!] = \{e : v \in \{v_1, v_2\}, e \in [\![v]\!]\}$$

$$[\![\lambda x. v]\!] = \{\lambda x. e : e \in [\![v]\!]\}$$

$$[\![(v_1 v_2)]\!] = \{(e_1 e_2) : e_1 \in [\![v_1]\!], e_2 \in [\![v_2]\!]\}$$

$$[\![\emptyset]\!] = \emptyset$$

$$[\![\Lambda]\!] = \Lambda$$

## Using version spaces

$VS \rightarrow var \mid \lambda var.VS \mid (VS\ VS) \mid primitive$

|  $VS \cup VS$       *nondeterministic choice*

|  $\Lambda$             *choose any expression*

|  $\emptyset$             *choose no expression*

## Using version spaces

$VS \rightarrow var \mid \lambda var.VS \mid (VS \: VS) \mid primitive$   
|  $VS \sqcup VS$       *nondeterministic choice*  
|  $\Lambda$                 *choose any expression*  
|  $\emptyset$                 *choose no expression*

exploit the fact that  $e \in \llbracket v \rrbracket$  can be efficiently computed:

$$\text{REFACTOR}(v|\text{Lib}) = \begin{cases} e, & \text{if } e \in \text{Lib and } e \in \llbracket v \rrbracket \\ \text{REFACTOR}'(v|\text{Lib}), & \text{otherwise.} \end{cases}$$

$\text{REFACTOR}'(v|\text{Lib}) = v$ , if  $v$  is a primitive or variable

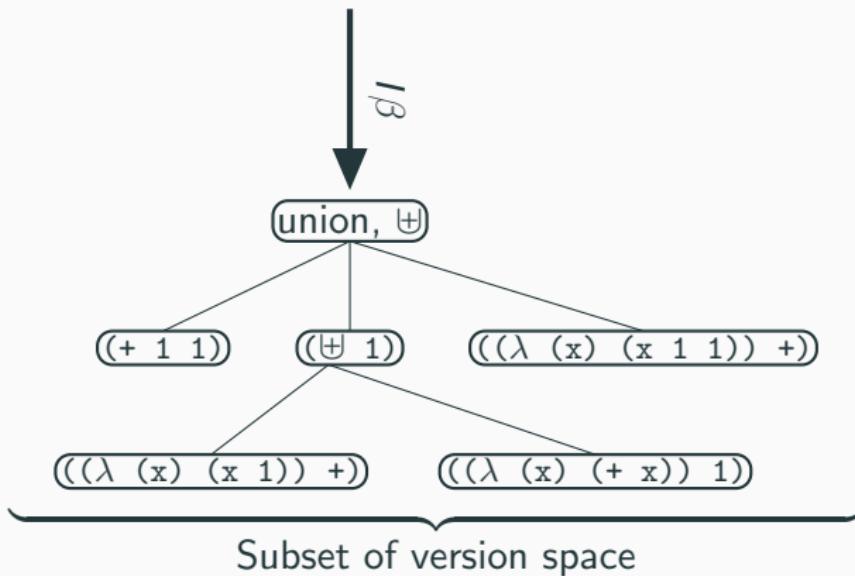
$\text{REFACTOR}'(\lambda x.b|\text{Lib}) = \lambda x.\text{REFACTOR}(b|\text{Lib})$

$\text{REFACTOR}'(v_1 v_2|\text{Lib}) = (\text{REFACTOR}(v_1|\text{Lib}) \text{ REFACTOR}(v_2|\text{Lib}))$

## Using version spaces

$VS \rightarrow var \mid \lambda var.VS \mid (VS \vee VS) \mid primitive$   
|  $VS \oplus VS$       *nondeterministic choice*  
|  $\Lambda$                 *choose any expression*  
|  $\emptyset$                *choose no expression*

Program:  $(+ 1 1)$



## Using version spaces

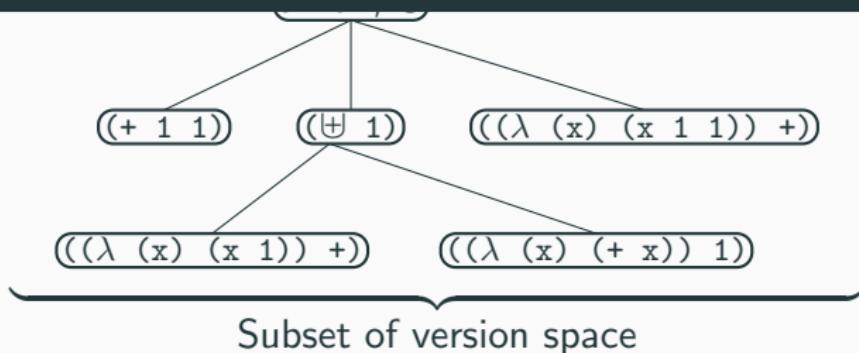
$VS \rightarrow var \mid \lambda var.VS \mid (VS\ VS) \mid primitive$   
|  $VS \sqcup VS$       *nondeterministic choice*  
|  $\Lambda$                 *choose any expression*  
|  $\emptyset$                 *choose no expression*

**completeness:**  $I\beta$  gets all the refactorings

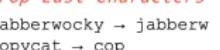
let  $v_2 = I\beta(v_1)$  and  $e_1 \in \llbracket v_1 \rrbracket$ . for any  $e_2 \rightarrow e_1$  then  $e_2 \in \llbracket v_2 \rrbracket$

**consistency:**  $I\beta$  only gets valid refactorings

let  $v_2 = I\beta(v_1)$  and  $e_2 \in \llbracket v_2 \rrbracket$ . then there is a  $e_1 \in \llbracket v_1 \rrbracket$  where  $e_2 \rightarrow e_1$



# DreamCoder Domains

List Processing	Text Editing	Regexes	LOGO Graphics
<b>Sum List</b> [1 2 3] → 6 [4 6 8 1] → 17	<b>Abbreviate</b> Allen Newell → A.N. Herb Simon → H.S.	<b>Phone Numbers</b> (555) 867-5309 (650) 555-2368	↶ ↷ ↸ ↹
<b>Double</b> [1 2 3 4] → [2 4 6 8] [6 5 1] → [12 10 2]	<b>Drop Last Characters</b> jabberwocky → jabberw copycat → cop	<b>Currency</b> \$100.25 \$4.50	✿ ⋯ ⋮ ⋆
<b>Check Evens</b> [0 2 3] → [T T F] [2 4 9 6] → [T T F T]	<b>Extract</b> see spot(run) → run a (bee) see → bee	<b>Dates</b> Y1775/0704 Y2000/0101	⊗ ⊖ ⊚ ⊛
<b>Block Towers</b> 	<b>Symbolic Regression</b>  $(-2.4x - 0.9) / (x - 4.4)(x - 0.9)$ $0.3x^3 + 1.1x^2 - 2.0x + 0.6$	<b>Recursive Programming</b> <b>Filter</b> [■■■■■] → [■■] [■■■■■■] → [■■■■] [■■■■■■■] → [■■■■■] <b>Length</b> [■■■■■] → 4 [■■■■■■] → 6 [■■■■] → 3	<b>Physics</b> $KE = \frac{1}{2} m  \vec{v} ^2$ $\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$ $\vec{F} \propto \frac{q_1 q_2}{ \vec{r}'_1 - \vec{r}'_2 ^2} \hat{\vec{r}} = \hat{\vec{r}}$ $R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$

# DreamCoder Domains

## List Processing

### Sum List

[1 2 3] → 6

[4 6 8 1] → 17

### Double

[1 2 3 4] → [2 4 6 8]

[6 5 1] → [12 10 2]

### Check Evens

[0 2 3] → [T T F]

[2 4 9 6] → [T T F T]

## Text Editing

### Abbreviate

Allen Newell → A.N.

Herb Simon → H.S.

### Drop Last Characters

jabberwocky → jabberw

copycat → cop

### Extract

see spot(run) → run

a (bee) see → bee

## Regexes

### Phone Numbers

(555) 867-5309

(650) 555-2368

### Currency

\$100.25

\$4.50

### Dates

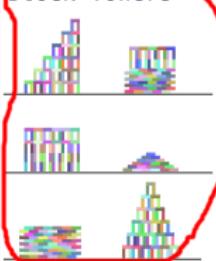
Y1775/0704

Y2000/0101

## LOGO Graphics



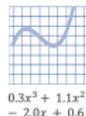
## Block Towers



## Symbolic Regression



$$\frac{(-2.4x - 0.9)}{(x - 4.4)(x - 0.3)}$$
$$0.5x^4 + 2.5x^3 + 0.4x^2 - 2.2x + 2.4$$



$$0.3x^3 + 1.1x^2 - 2.0x + 0.6$$
$$\frac{4.9}{x}$$

## Recursive Programming

### Filter

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

### Length

[■■■■] → 4  
[■■■■■] → 6  
[■■■] → 3

### Index List

0, [■■■■■■] → ■  
1, [■■■■■■] → ■■  
1, [■■■■■■] → ■■■

### Every Other

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

## Physics

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

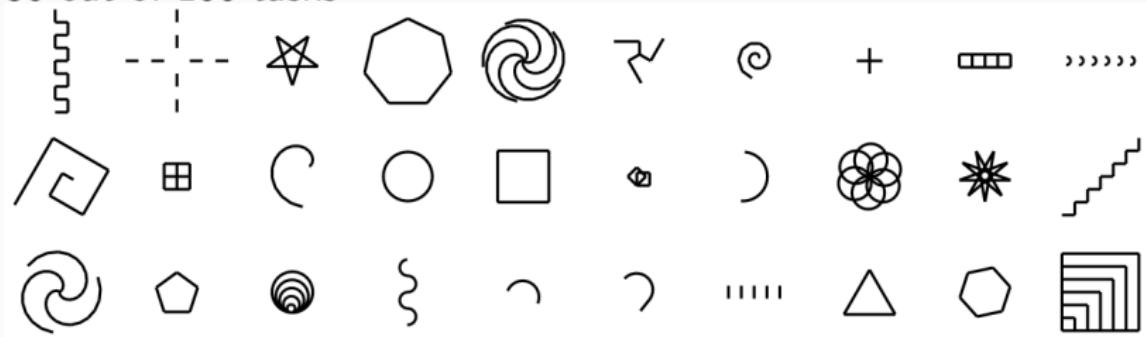
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

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

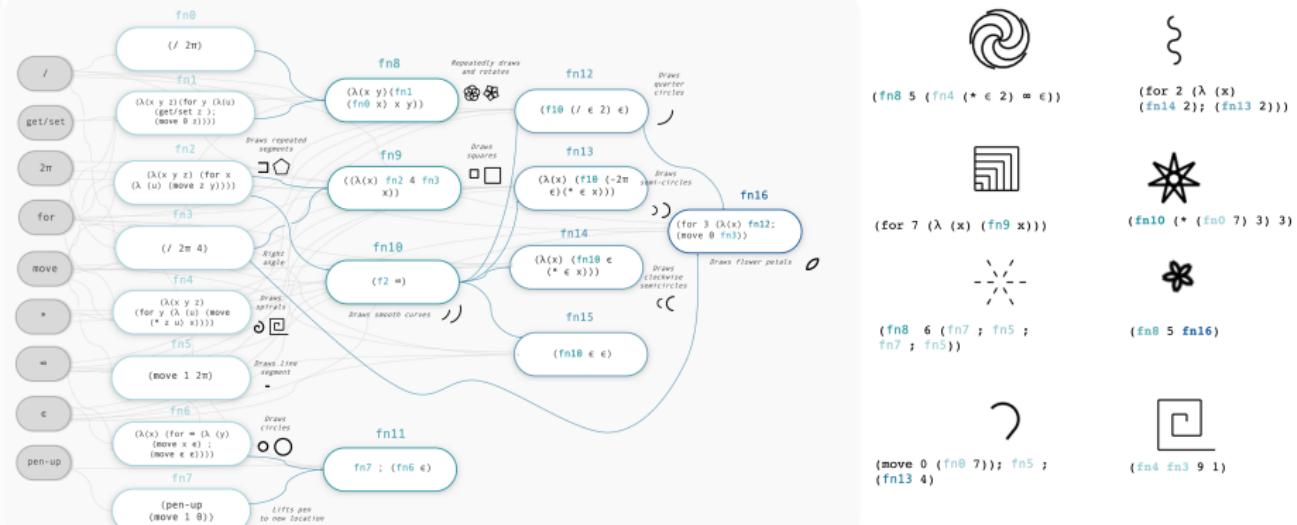
$$V_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$$

# LOGO Graphics

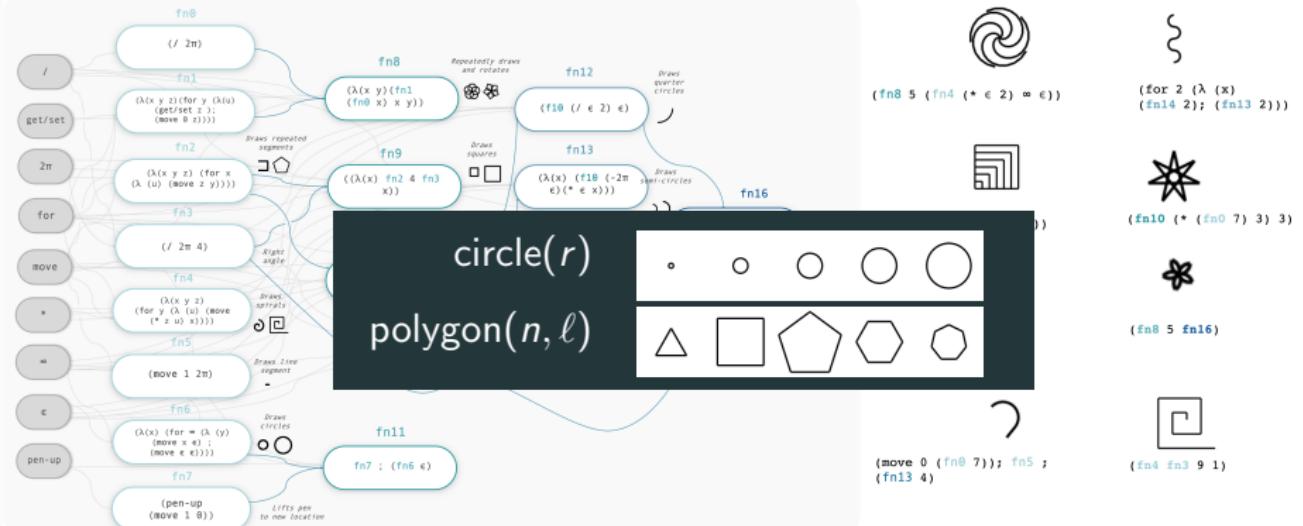
30 out of 160 tasks



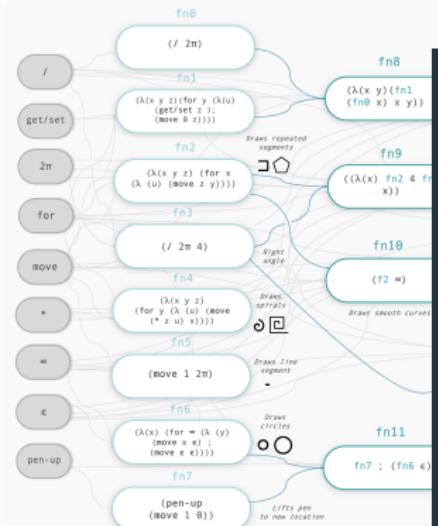
# LOGO Graphics – learning interpretable library of concepts



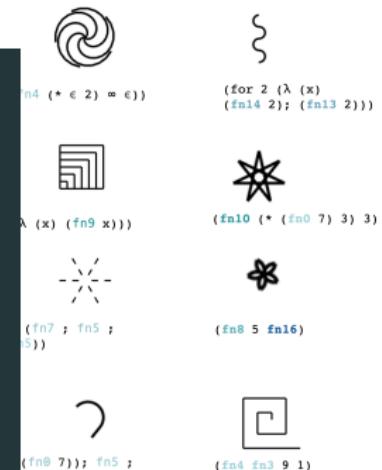
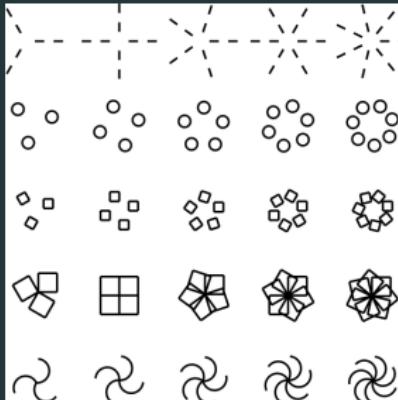
# LOGO Graphics – learning interpretable library of concepts



# LOGO Graphics – learning interpretable library of concepts



radial symmetry( $n$ , body)

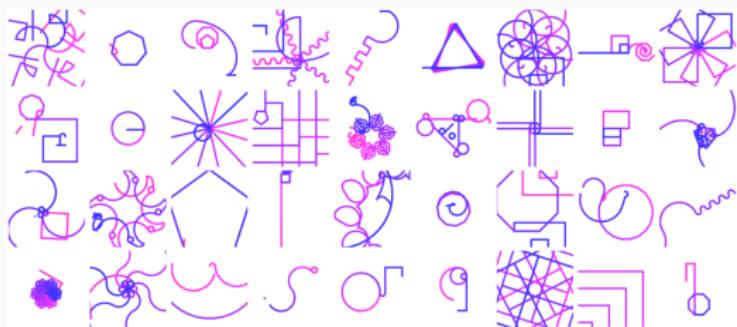


# what does DreamCoder dream of?

before learning

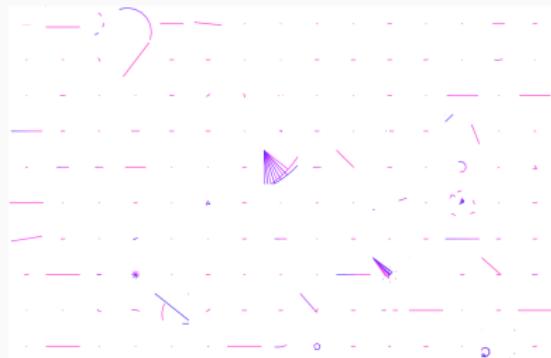


after learning

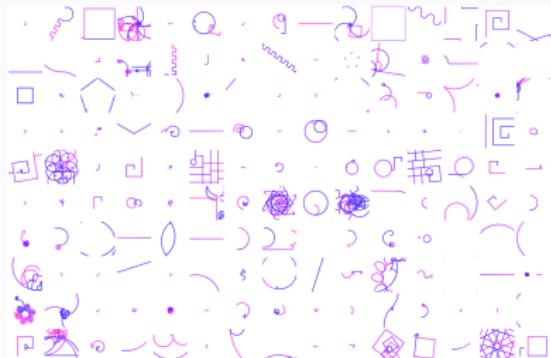


# what does DreamCoder dream of?

before learning

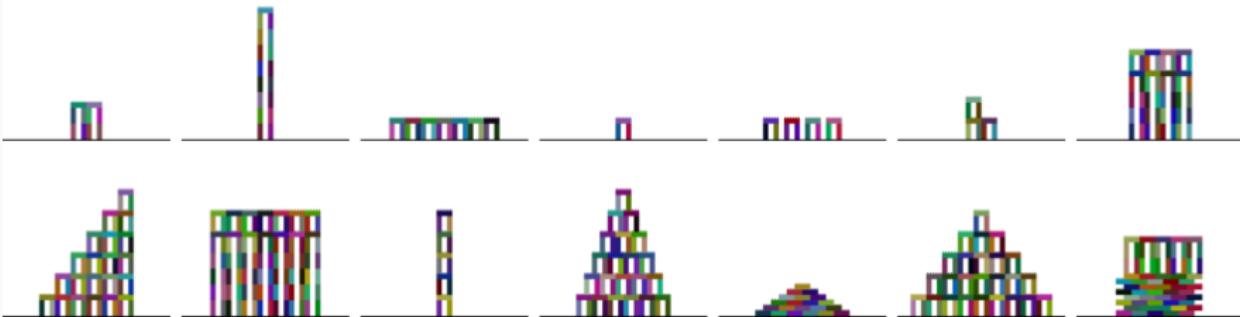


after learning



# Planning to build towers

example tasks (112 total)

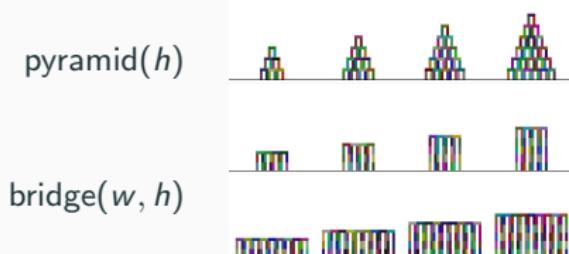
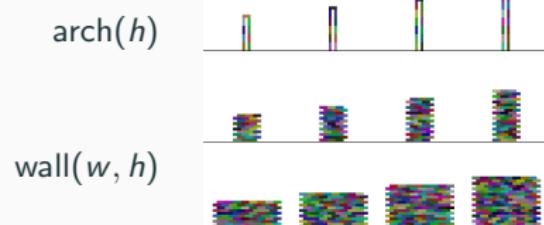


# Planning to build towers

example tasks (112 total)

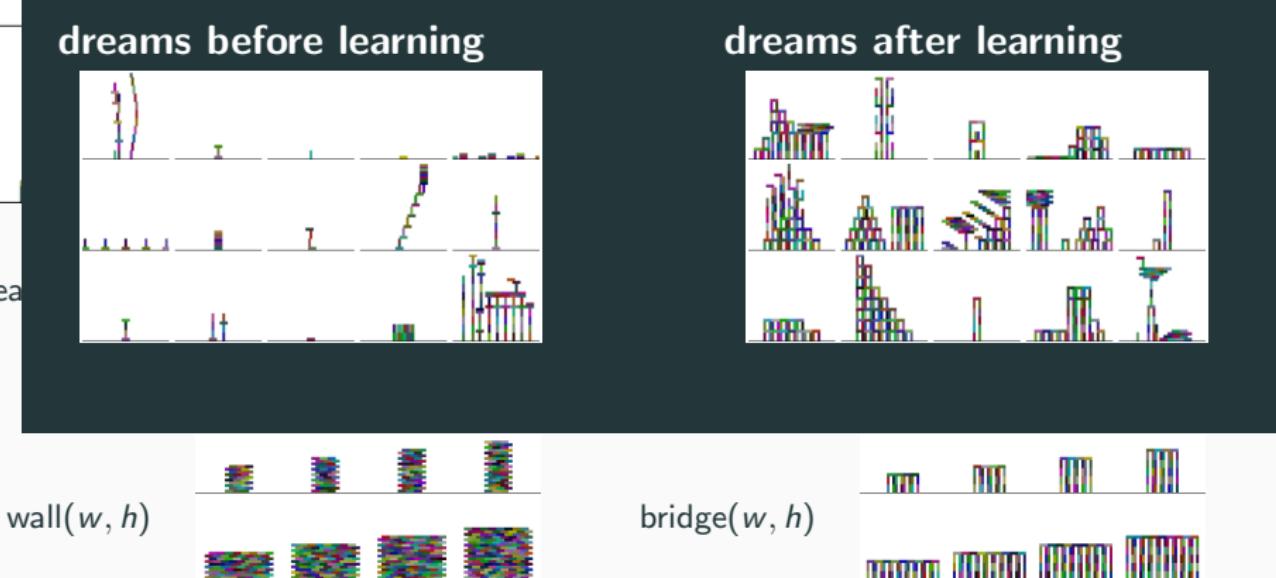


learned library routines ( $\approx 20$  total)

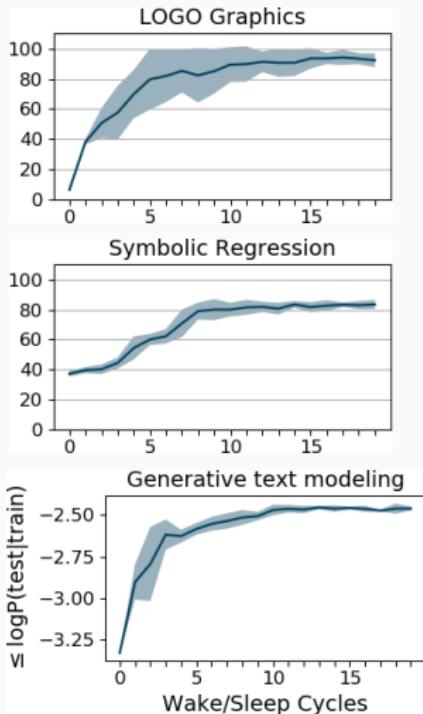
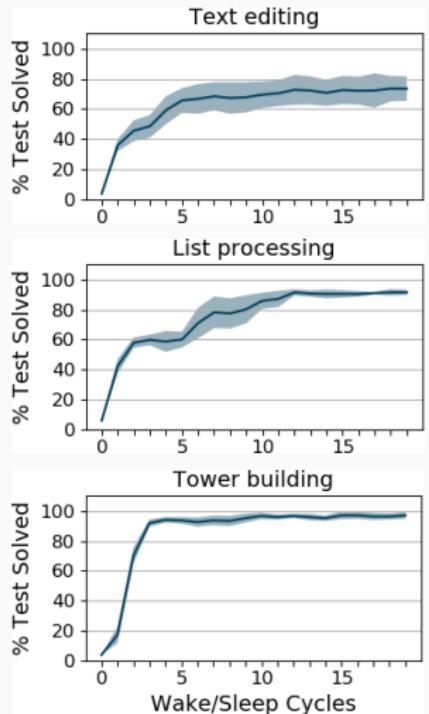


# Planning to build towers

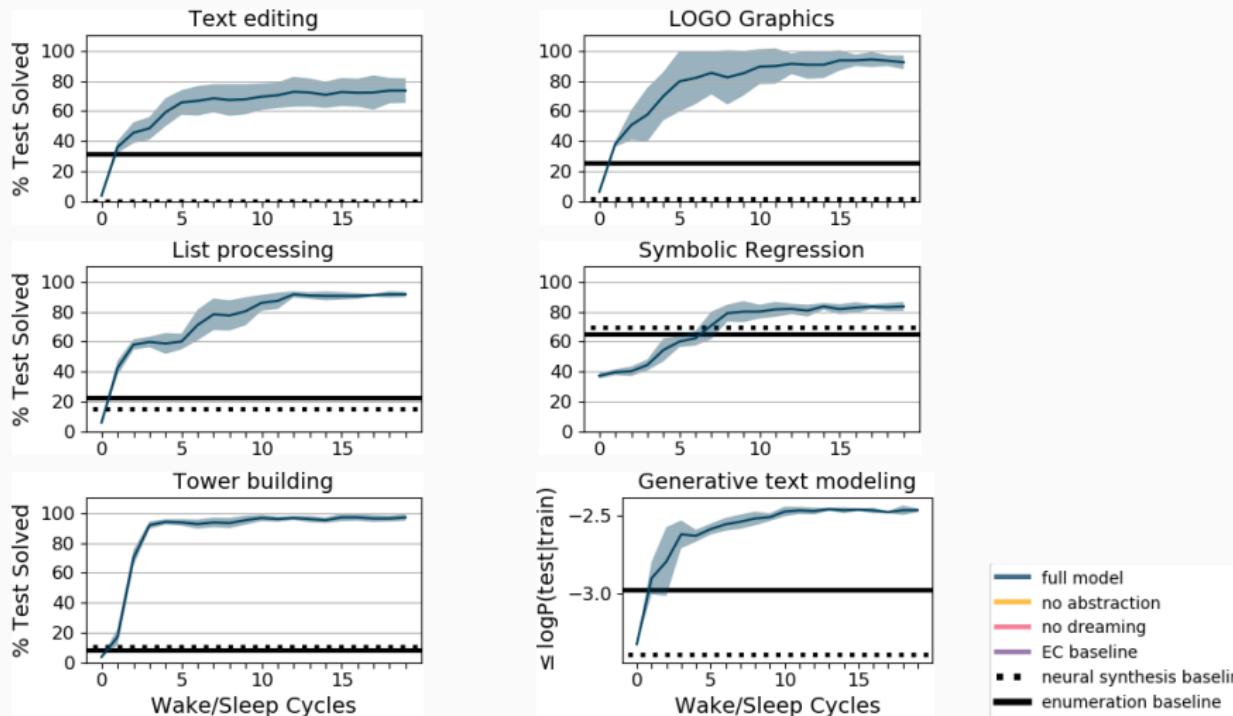
example tasks (112 total)



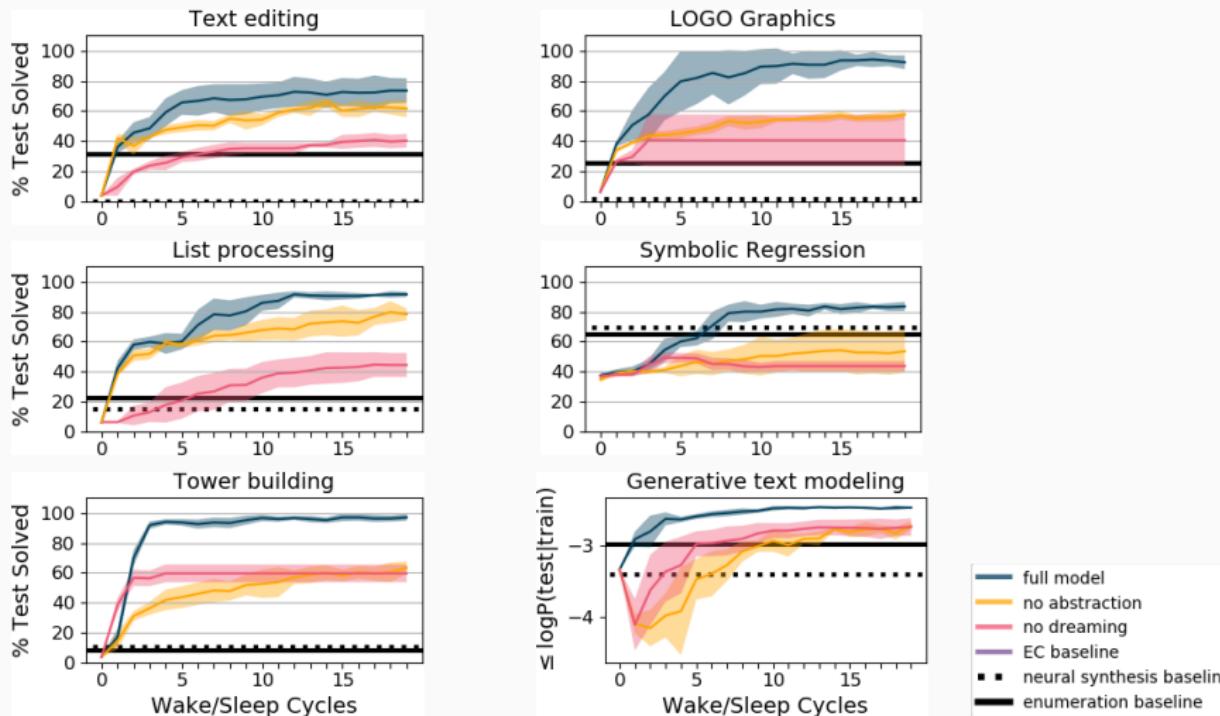
# synergy between dreaming and library learning



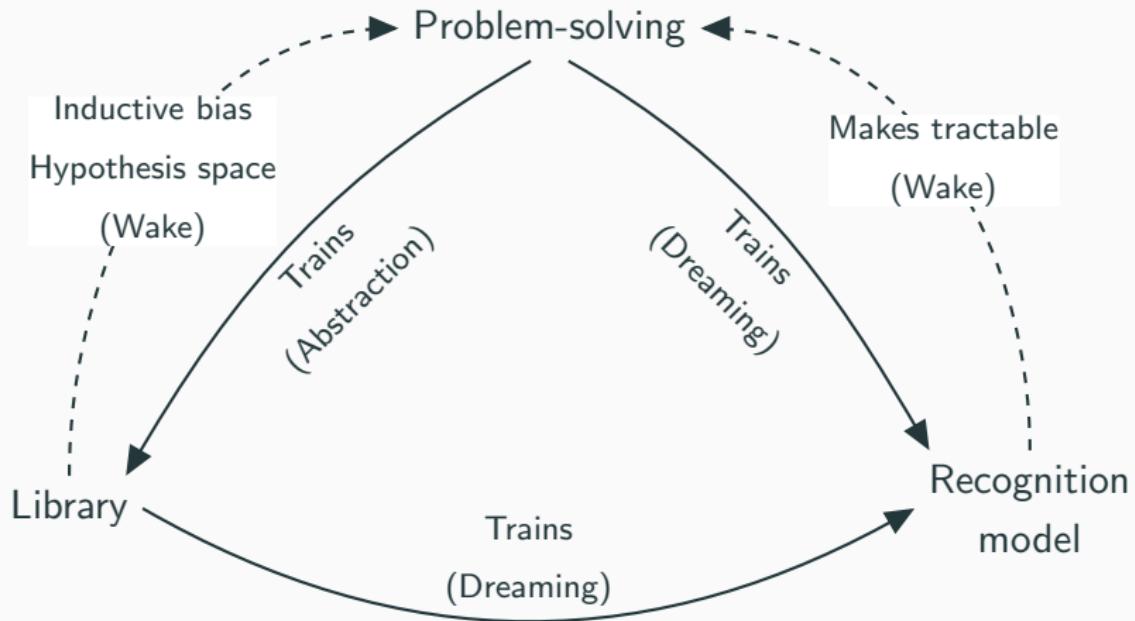
# synergy between dreaming and library learning



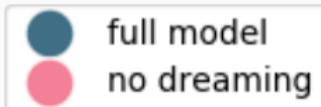
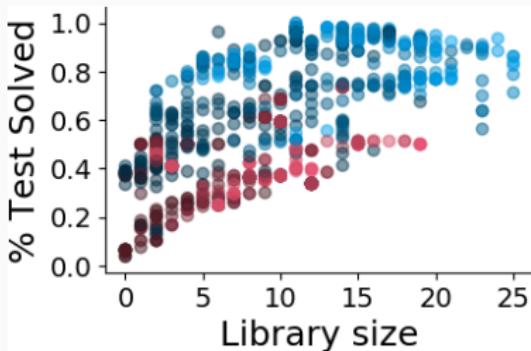
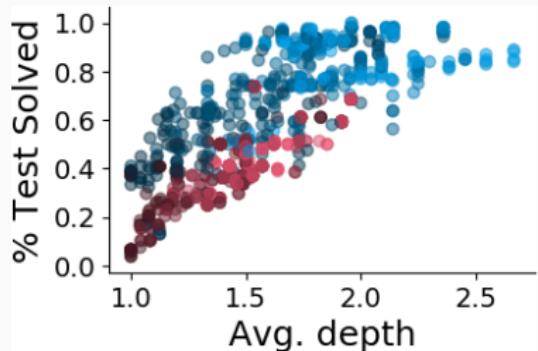
# synergy between dreaming and library learning



# synergy between dreaming and library learning



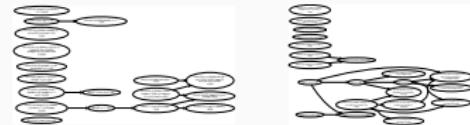
# Evidence for dreaming bootstrapping better libraries



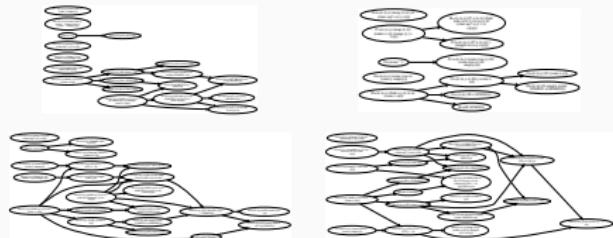
Dark→Light: Early in learning→Later in learning

# Variability in learned library

List processing



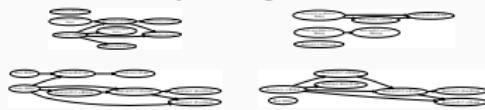
Text editing



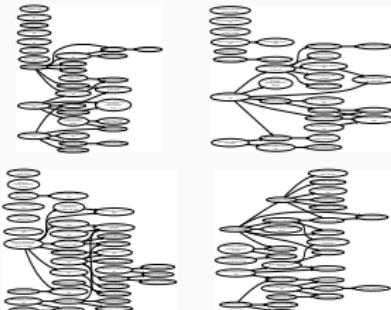
Tower building



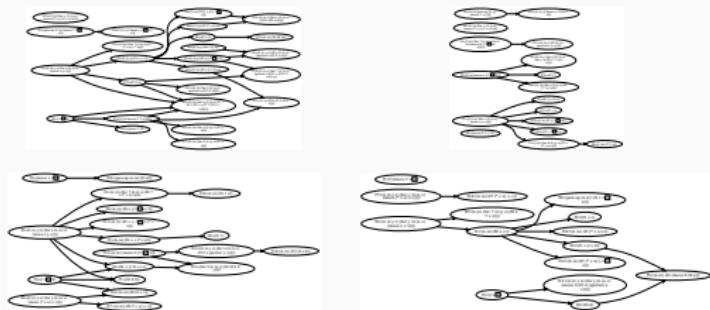
Symbolic regression



Generative regex



LOGO graphics



## From learning libraries to learning languages

these experiments study how DreamCoder grows from a “beginner” state to “expert”:

- “beginner:” basic domain-specific procedures, only easiest problems have short solutions
- “expert:” learned library allowing hardest problems to have short meaningful solutions

## From learning libraries to learning languages

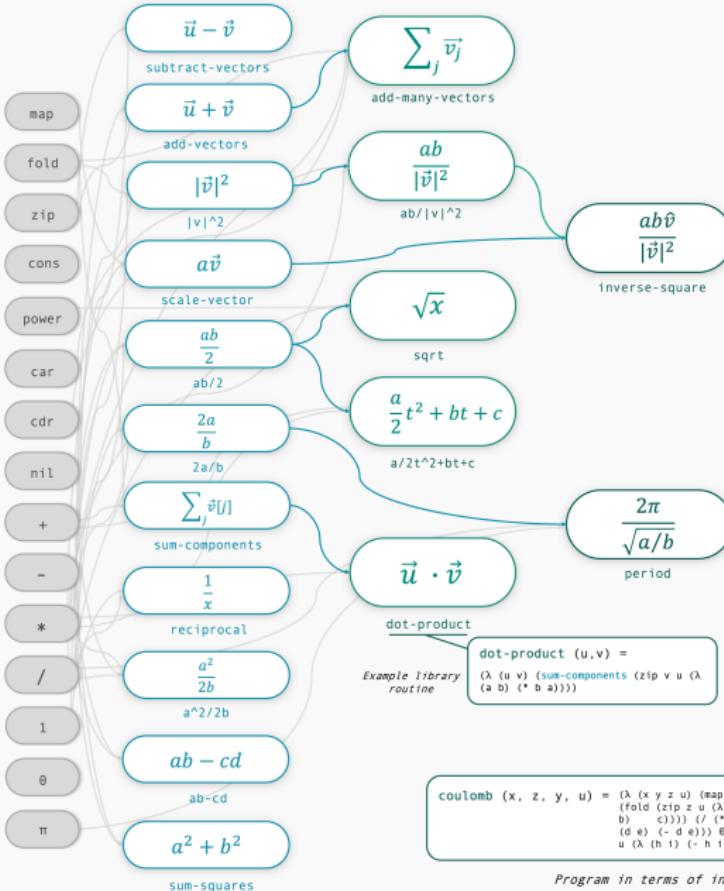
these experiments study how DreamCoder grows from a “beginner” state to “expert”:

- “beginner:” basic domain-specific procedures, only easiest problems have short solutions
- “expert:” learned library allowing hardest problems to have short meaningful solutions

go beyond: start with generic arithmetic & control flow, learn fundamentals of domain

- Physics from recursive higher-order functions
- Recursive higher-order language from 1959 proto-Lisp & Y-combinator

# Rediscovering vector algebra



Electric Field to Charge Flux

$$\vec{E} = \rho \vec{J}$$

(scale-vector p J)

Newton's Second Law

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

(scale-vector (reciprocal m) (add-many-vectors Fs))

Kinetic Energy

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

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

Parallel Resistors

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

(reciprocal (sum-components (map (lambda (r) (reciprocal r)) Rs)))

Energy in a Capacitor

$$U = \frac{1}{2} CV^2$$

(a^2/2b v (reciprocal c))

Ballistic Motion

$$x(t) = \frac{a}{2} t^2 + v_0 t + x_0$$

(a/2 t^2+bt+c c b a t)

Force in a Magnetic Field

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

(\* q (ab-cd v\_x b\_y v\_y b\_x))

Work

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

(dot-product F d)

Coulomb's Law

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

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

Center of Mass

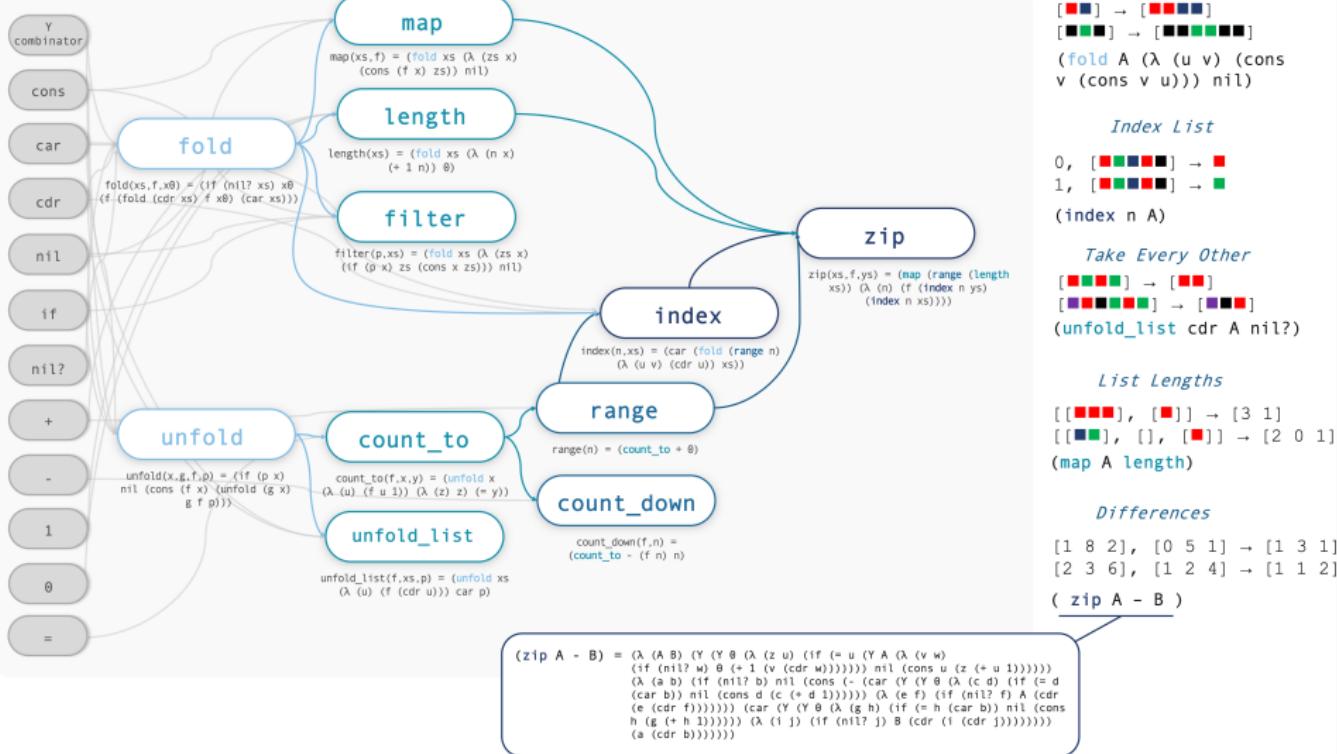
$$X_{CM} = \frac{\sum_l m_l x_l}{\sum_l m_l}$$

(/ (dot-product x m) (sum-components m))

```
coulomb (x, z, y, u) = 
  (lambda (x y z u) (map (lambda (v) (* (/ (* (power (/ (* x x)
    (fold (zip z u (lambda (w a) (- w a)))) 0) (lambda (b c) (+ (* b
      b) (- c c)))) (/ (* 1 1) (+ 1 1)))) y) (fold (zip z u (lambda
        (d e) (- d e)))) 0 (lambda (f g) (+ (* f f) g)))) v) (zip z
        u (lambda (h i) (- h i)))))
```

Program in terms of initial library

# Rediscovering origami programming



## From learning libraries to using interpreters

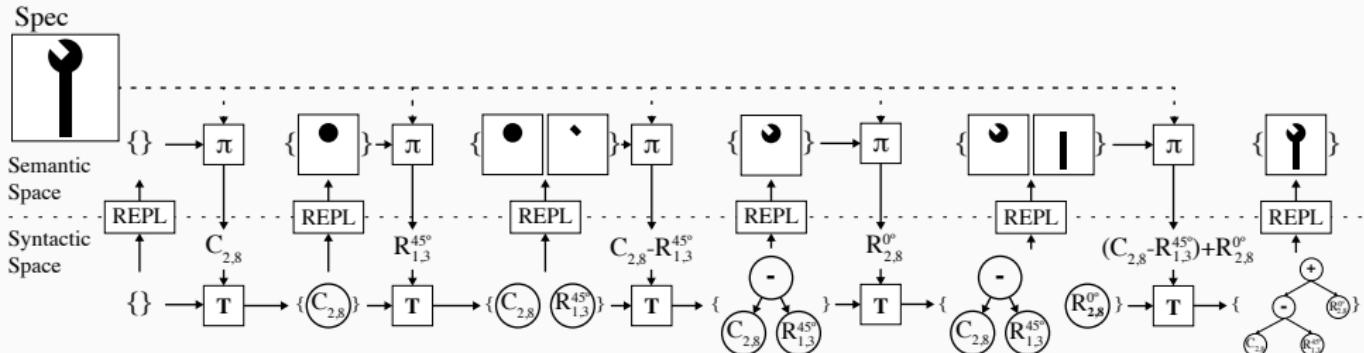
DreamCoder: library building as inspiration

Software engineers: build libraries, use interpreters, version control, debuggers, ...

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

\*equal contribution

# Synthesis with a REPL



REPL: Bridges syntax and semantics

$\pi$ : policy, writes code conditioned on REPL state

$T$ : Markov Decision Process transition function

Spec: Image to draw

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

\*equal contribution

# Interleaving policy+value+REPL in stochastic tree search

c.f. AlphaGo, TD-Gammon

Spec:



$g_0$

$\pi$

$g_1$

$v$

$g_{1,v}$

$\pi / v$

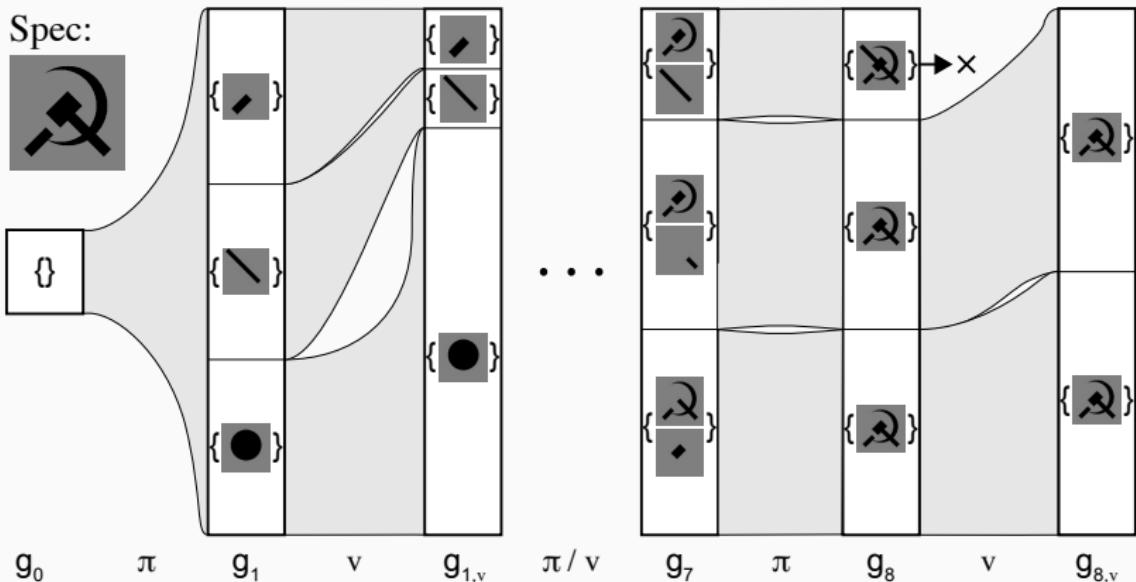
$g_7$

$\pi$

$g_8$

$v$

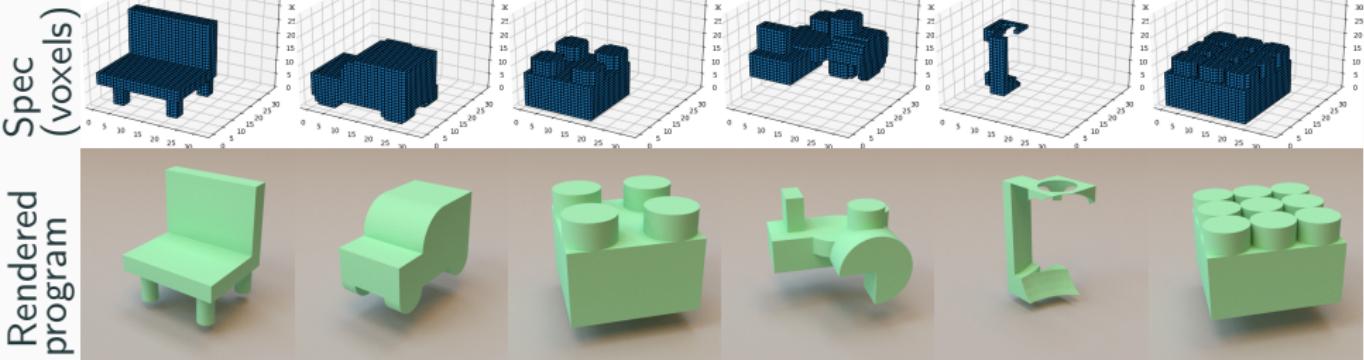
$g_{8,v}$



# Scaling to long programs

Branching factor: > 1.3 million per line of code

Successfully synthesizes >20-line programs in seconds (>100 tokens)



# Scaling to long programs

Branching factor: > 400 actions

Successfully synthesizes 40-action programs

Spec:
6/12/2003 → date: 12 mo: 6 year: 2003
3/16/1997 → date: 16 mo: 3 year: 1997
Held out test instance:
12/8/2019 → date: 8 mo: 12 year: 2019
Results:
<b>Ours</b> → <b>date: 8 mo: 12 year: 2019</b>
RobustFill → date:12/8/2019