

# **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



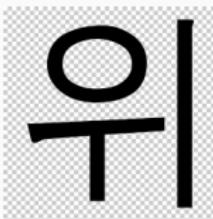
engineering



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 frameworks can contribute to this picture?

## Three AI traditions

### Symbolic



```
In[34]:= Solve[{(hw - hw^2) == z}, h]
```

```
Out[34]= {}
```



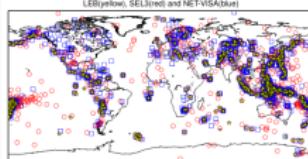
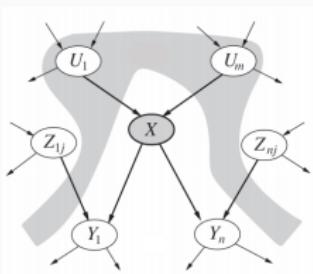
Input interpretation:

```
solve h w - h w^2 = Z for h
```

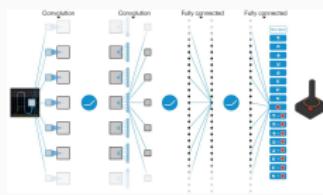
Result:

$$h = \frac{Z}{w - w^2} \text{ and } w^2 = w$$

### Probabilistic



### Neural



# What computational frameworks can contribute to this picture?

## Three AI traditions

### Symbolic

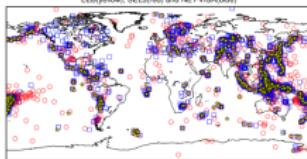
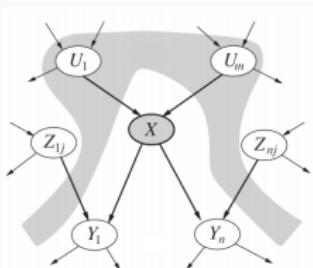


```
In[34]:= Solve[{(hw - hw^2) == z}, h]
```

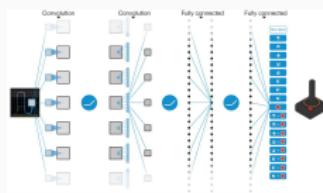
```
Out[34]= {}
```



### Probabilistic



### Neural

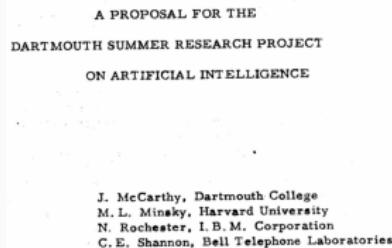


**Program induction:** agents that learn by writing their own code

strong generalization

# Why didn't this old idea work?

Program induction goes back to 1956 Dartmouth Workshop that founded the field of AI



## Why will it work this time?

better toolkits:

- **probabilistic** methods for uncertainty and learning-to-learn
- **neural** methods for guiding combinatorial search
- **symbolic** methods, from the **programming languages** community
  - maturing **program synthesis** techniques
  - type systems, program analysis, constraint solving, ...

# Why will it work this time?

better toolkits:

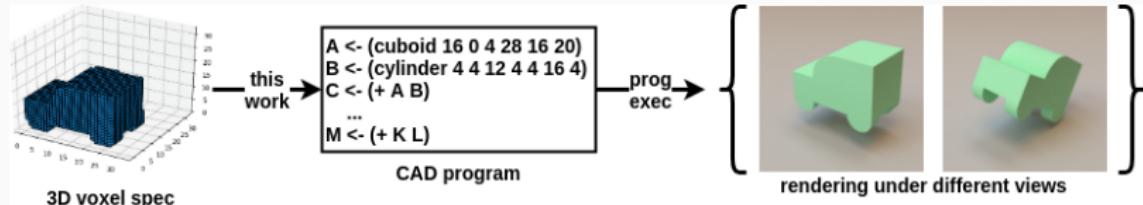
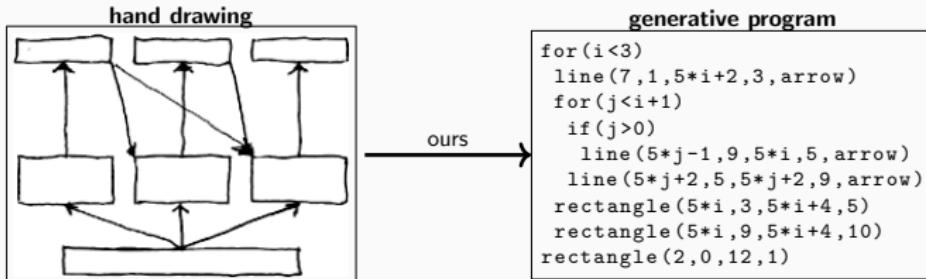
- **probabilistic** methods for uncertainty and learning-to-learn
- **neural** methods for guiding combinatorial search
- **symbolic** methods, from the **programming languages** community
  - maturing **program synthesis** techniques
  - type systems, program analysis, constraint solving, ...

better problems:

- inverse perception, analysis-by-synthesis
- learning models
- learning to learn
- natural language→code
- programming by examples, computer-aided-programming

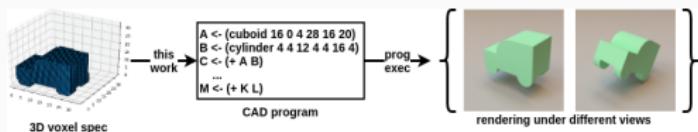
# Perception, Synthesizing models, Learning-to-Learn

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

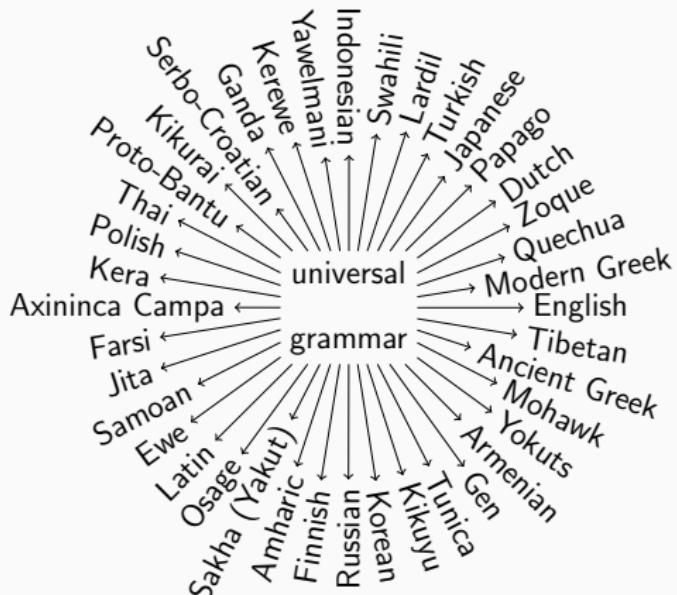


# Perception, Synthesizing models, Learning-to-Learn

Theme #1: high-level scene understanding, pixels $\rightarrow$ programs

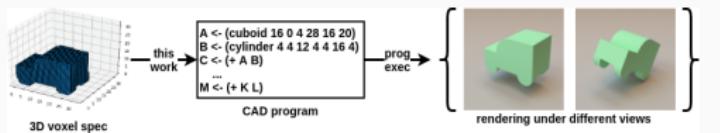


Theme #2: Synthesizing human-understandable models



# Perception, Synthesizing models, Learning-to-Learn

Theme #1: high-level scene understanding, pixels $\rightarrow$ programs



Theme #2: Synthesizing interpretable models

Theme #3: 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  $\rightarrow$  A.N.  
Herb Simon  $\rightarrow$  H.S.

##### Drop Last Characters

jabberwocky  $\rightarrow$  jabberw  
copycat  $\rightarrow$  cop

##### Extract

see spot(run)  $\rightarrow$  run  
a (bee) see  $\rightarrow$  bee

#### Regexes

##### Phone Numbers

(555) 867-5309  
(650) 555-2368

##### Currency

\$100.25  
\$4.50

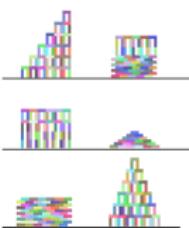
##### Dates

Y1775/07/04  
Y2000/01/01

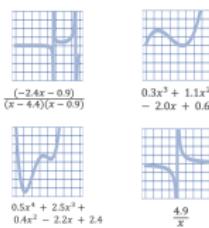
#### LOGO Graphics



#### Block Towers



#### Symbolic Regression



#### Recursive Programming

##### Filter

[■■■■■]  $\rightarrow$  [■■■]  
[■■■■■■]  $\rightarrow$  [■■■■]  
[■■■■■■■]  $\rightarrow$  [■■■■■]

##### Length

[■■■■■]  $\rightarrow$  4  
[■■■■■■]  $\rightarrow$  6  
[■■■■]  $\rightarrow$  3

#### Physics

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

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

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

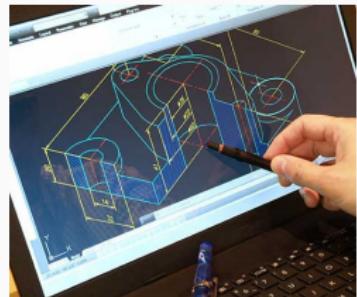
$$R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$$

# High-level, abstract visual abilities

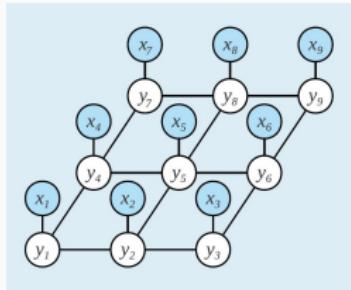
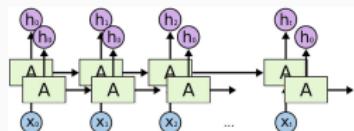
...in art



...in engineering



...in AI



# High-level, abstract visual abilities

...in art



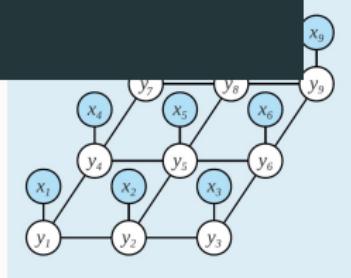
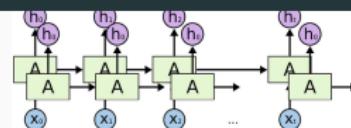
why?

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

how?

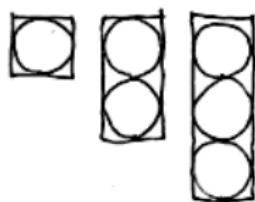
Bayesian inference of graphics program conditioned on image,  
+program synthesis  
+learning

...in AI



# Learning to infer graphics programs from hand-drawn images

model infers program from drawing



→

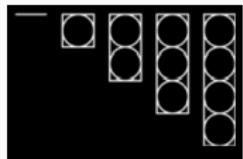
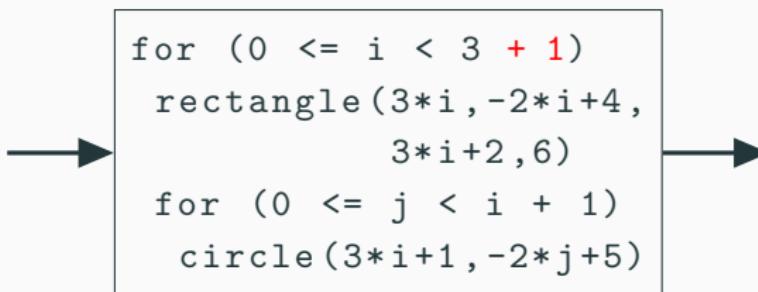
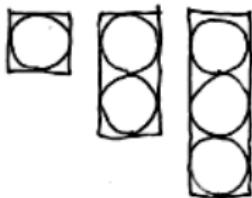
```
for (0 <= i < 3)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```

Ellis, Ritchie, Solar-Lezama, Tenenbaum. NeurIPS 2018.

# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

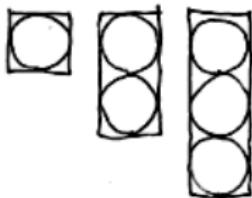
**zero-shot generalization / extrapolation**



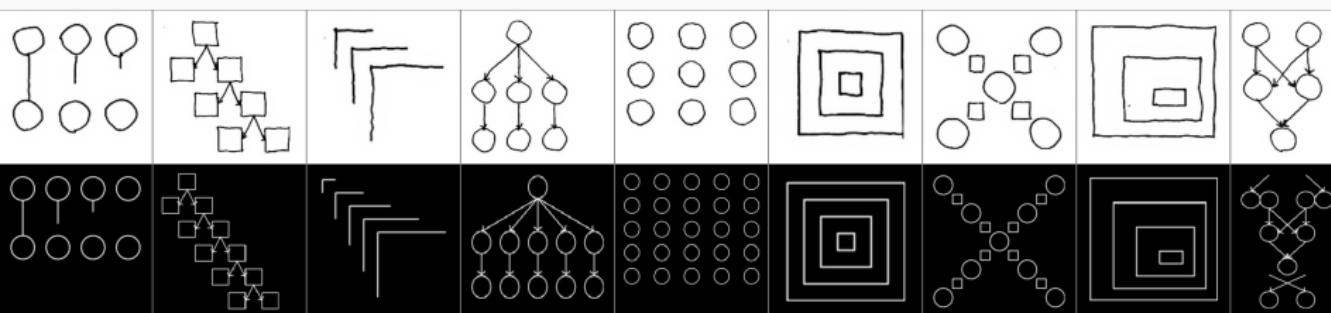
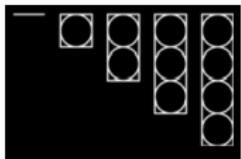
# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

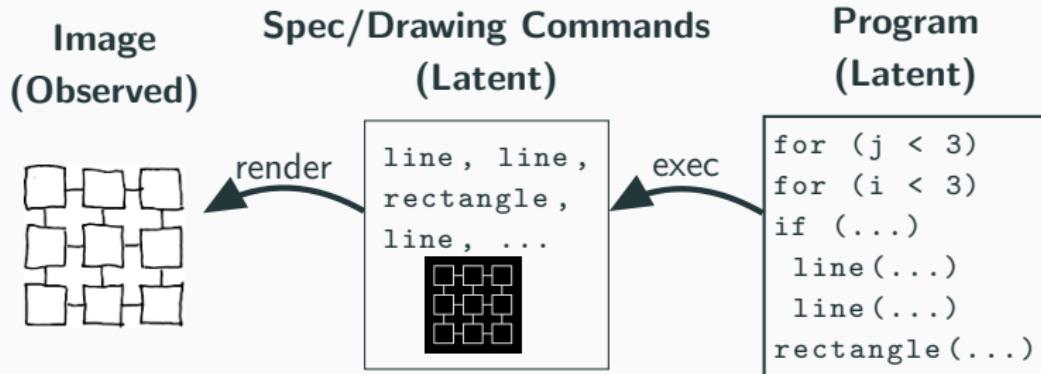
**zero-shot generalization / extrapolation**



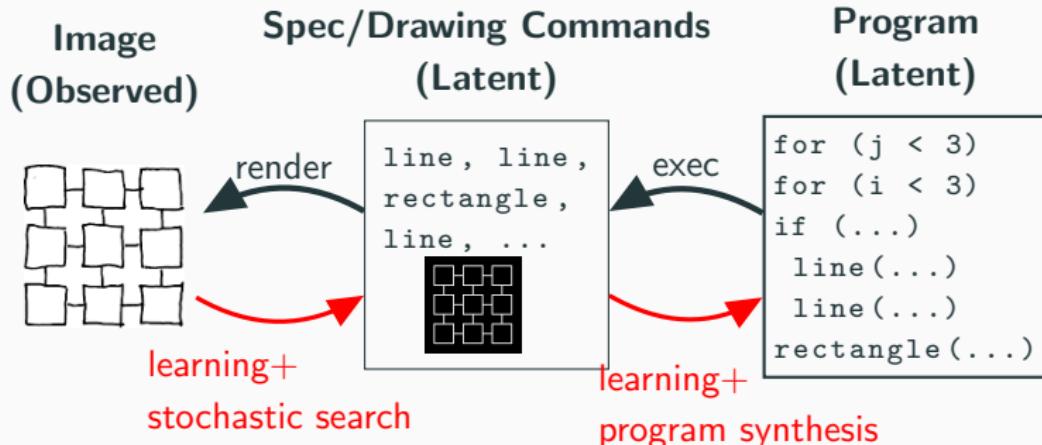
```
for (0 <= i < 3 + 1)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```



# How to infer graphics programs from hand-drawn images



# How to infer graphics programs from hand-drawn images

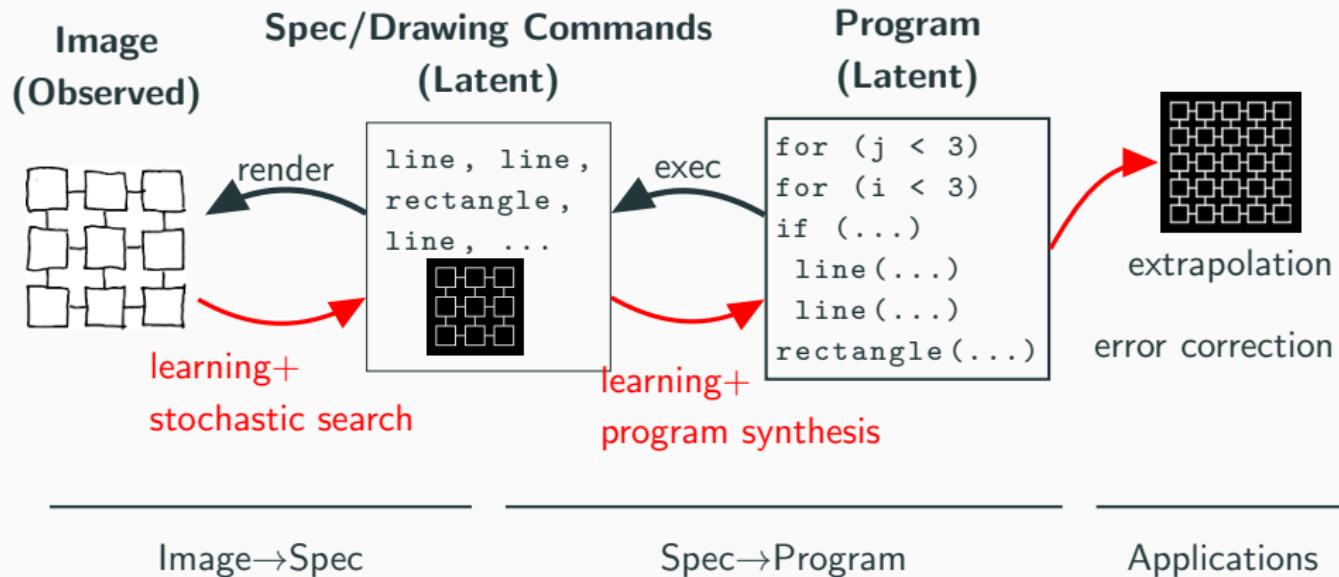


---

Image→Spec

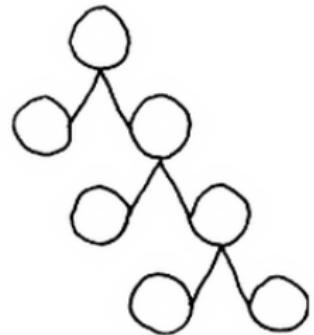
Spec→Program

# How to infer graphics programs from hand-drawn images

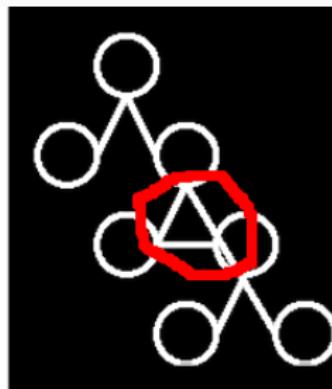


# Top-down influences on perception

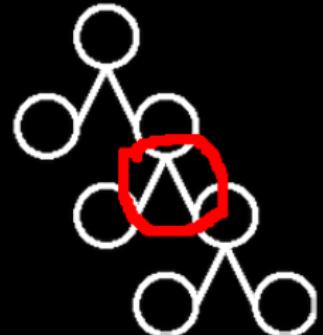
drawing



bottom-up neural net

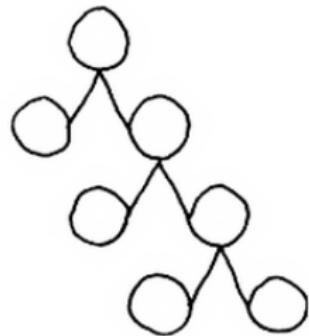


w/ top-down program bias

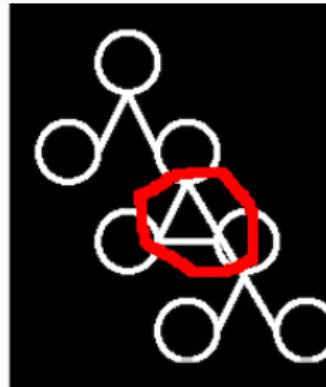


# Top-down influences on perception

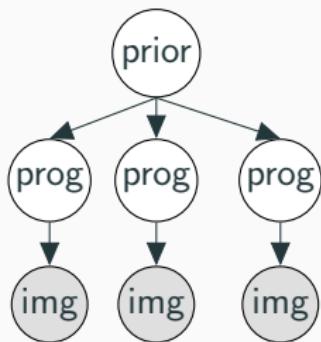
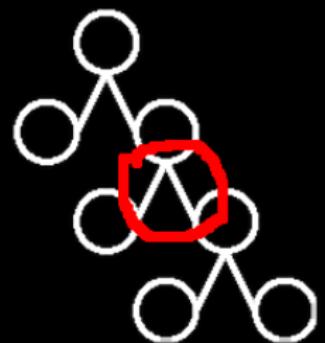
drawing



bottom-up neural net



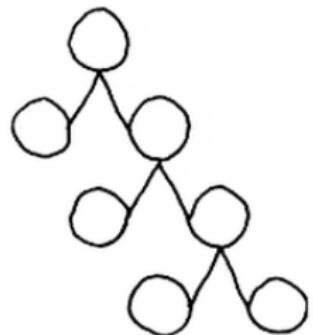
w/ top-down program bias



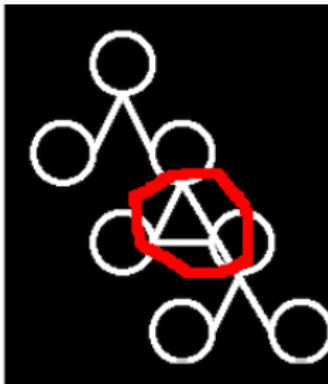
predicted program =  
 $\arg \max_{\text{progs}} \mathbb{P} [\text{img}|\text{prog}] \mathbb{P} [\text{prog}|\text{prior}]$

# Top-down influences on perception

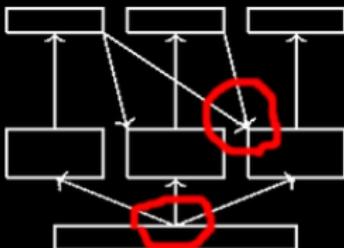
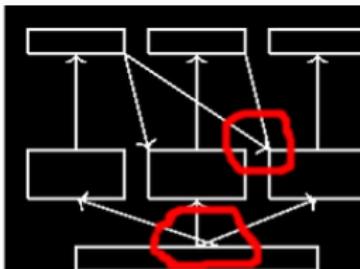
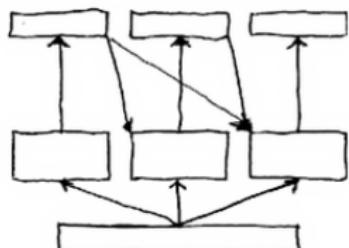
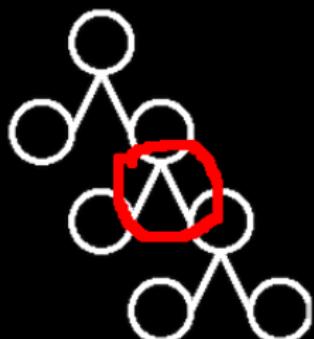
drawing



bottom-up neural net



w/ top-down program bias



# Example programs

Drawing	Program
	<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>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>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>

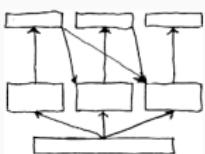
# Example programs

Drawing	Program
	<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

Goal: a general algorithm for learning to synthesize programs

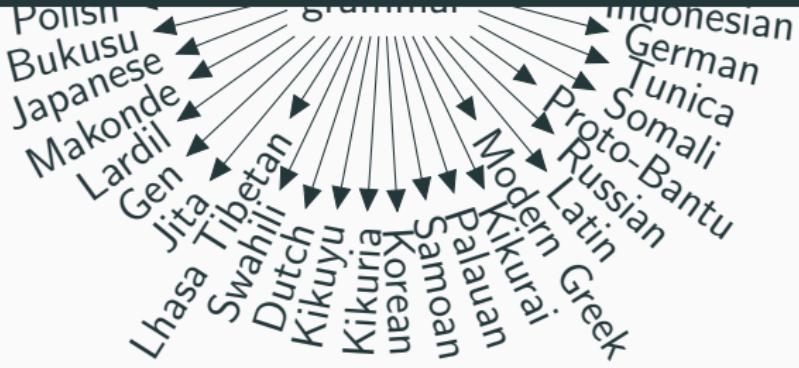
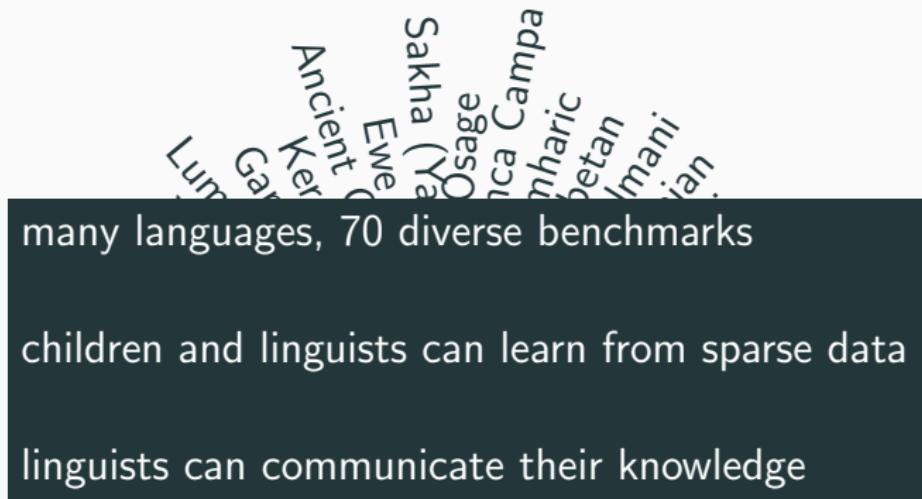
	<pre>for(i&lt;3)     line(7,1,5*i+2,3,arrow)     for(j&lt;i+1)         if(j&gt;0)             line(5*j-1,9,5*i,5,arrow)             line(5*j+2,5,5*j+2,9,arrow)         rectangle(5*i,3,5*i+4,5)         rectangle(5*i,9,5*i+4,10)         rectangle(2,0,12,1)</pre>
---	--



# Synthesizing human-understandable models of language



# Synthesizing human-understandable models of language



# Few-shot language learning experiment

Farsi:

	singular	plural
“lip”	læb	læban
“gazelle”	ahu	ahuan
“mother”	valede	???

# Few-shot language learning experiment

Farsi:

	singular	plural
“lip”	læb	læban
“gazelle”	ahu	ahuan
“mother”	valede	valedean

# Few-shot language learning experiment

Mandarin:

	adjective	adverb
“slow”	man	manmandə
“fast”	kuai	kuaikuaidə
“small”	xiao	???

# Few-shot language learning experiment

Mandarin:

	adjective	adverb
“slow”	man	manmandə
“fast”	kuai	kuaikuaidə
“small”	xiao	xiaoxiaodə

# Few-shot language learning experiment

Mandarin:

	adjective	adverb
“slow”	man	manmandə
“fast”	kuai	kuaikuaidə
“small”	xiao	xiaoxiaodə

stem+stem+də

# Few-shot language learning experiment

Serbo-Croatian:

	masculine	feminine
“rich”	bogat	bogata
“mild”	blag	blaga
“green”	zelen	???

# Few-shot language learning experiment

Serbo-Croatian:

	masculine	feminine
“rich”	bogat	bogata
“mild”	blag	blaga
“green”	zelen	zelena

# Few-shot language learning experiment

Serbo-Croatian:

	mASCULINE	fEMININE
“rich”	bogat	bogata
“mild”	blag	blaga
“green”	zelen	zelena

*add “a” to stem to make feminine*

# Few-shot language learning experiment

Serbo-Croatian:

	mASCULINE	fEMININE
“rich”	bogat	bogata
“mild”	blag	blaga
“green”	zelen	zelena
“clear”	???	yasna

*add “a” to stem to make feminine*

# Few-shot language learning experiment

Serbo-Croatian:

	masculine	feminine	stem (unobserved)
“rich”	bogat	bogata	bogat
“mild”	blag	blaga	blag
“green”	zelen	zelena	zelen
“clear”	<b>yasan</b>	yasna	yasn

*add “a” to stem to make feminine*

*insert “a” between two word-final consonants*

$\emptyset \rightarrow a / C_C\#$

# Diverse Linguistic Phenomena

## Serbo-Croatian

grammar

MASC → stem, FEM → stem+a,  
NEUT → stem+o, PL → stem+i

$r_1: [+vowel] \rightarrow [+stress] / [+stress] [-vowel]_0$   
 $r_2: [+vowel] \rightarrow [-stress] / [-vowel]_0 [+stress]$   
**rules 1 & 2 shift stress to final vowel**

$r_3: \emptyset \rightarrow a / [-vowel]_- [-vowel] \#$   
**rule 3 inserts “a” between word-final consonants**

$r_4: [-sonorant] \rightarrow [-voice] / [-voice]$   
**rule 4 changes voicing of sound**

$r_5: l \rightarrow o / -\#$   
**rule 5 changes word-final “l” to “o”**

stems  
(unobserved)

observed data  
& unobserved derivation

- ⟨/sítn/, TINY⟩
- ⟨/blízk/, CLOSE⟩
- ⟨/túp/, BLUNT⟩
- ⟨/míl/, DEAR⟩

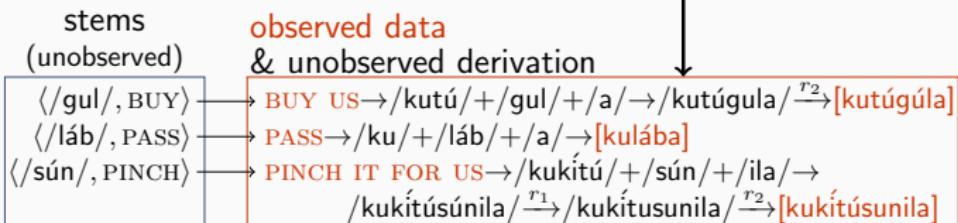
- TINY(MASC) → /sítn/  $\xrightarrow{r_3}$  [sítan]
- CLOSE(PL) → /blízk/+ /i/ → /blízki/  $\xrightarrow{r_1}$  /blízki/  $\xrightarrow{r_2}$  /blizkí/  $\xrightarrow{r_4}$  [bliskí]
- BLUNT(FEM) → /túp/+ /a/ → /túpa/  $\xrightarrow{r_1}$  /túpa/  $\xrightarrow{r_2}$  [tupá]
- DEAR(MASC) → /míl/  $\xrightarrow{r_5}$  [mío]

# Languages with tones

## Kerewe: Tanzanian Bantu

grammar

INF → /ku/+stem+/a/, V e.o. → /ku/+stem+/ana/,  
V for → /ku/+stem+/ila/, V for e.o. → /ku/+stem+/ilana/  
V us → /kutú/+stem+/a/, V it → /kuki/+stem+/a/,  
V for us → /kutú/+stem+/ila/, V it for us → /kukítú/+stem+/ila/  
 $r_1: [ ] \rightarrow [-\text{hiTone}] / [+ \text{hiTone}] [ -\text{vowel}]_0$  \_\_\_\_\_  
neutralize tone when right of tone  
 $r_2: [ +\text{vowel}] \rightarrow [+ \text{hiTone}] / [+ \text{hiTone}] [ -\text{vowel}]_0 [ -\text{vowel}]$  \_\_\_\_\_  
spread tone rightward except to last vowel



# Vowel “harmony”

## Turkic Sakha (Yakut)

grammar

NOUN→stem, PL→stem+/lar/,  
ASSOCIATIVE→stem+/iin/

$r_1: l \rightarrow d / [-\text{lateral} \ -\text{tense}]$   
“l” becomes “d” next to “r”, “t”, but not “l”

$r_2: C \rightarrow [-\text{voice}] / [-\text{voice}]$   
do not voice next to voiceless

$r_3: V \rightarrow [+ \text{rounded}] / [+ \text{rounded}] [- \text{low}]_0$   
 $r_4: [+ \text{continuant} \ -\text{high}] \rightarrow [- \text{rounded}] / u C_0$   
“harmonize” round vowels like “u”, “o”

$r_5: V \rightarrow [- \text{back} \ -\text{low}] / [- \text{back} \ +\text{vowel}] []_0$   
“harmonize” vowels to be not at back of mouth

$r_6: [-\text{sonorant} \ +\text{voice}] \rightarrow [+ \text{nasal}] / [+ \text{nasal}]$   
“nasalize” consonant next to a nasal, like “n”, “m”

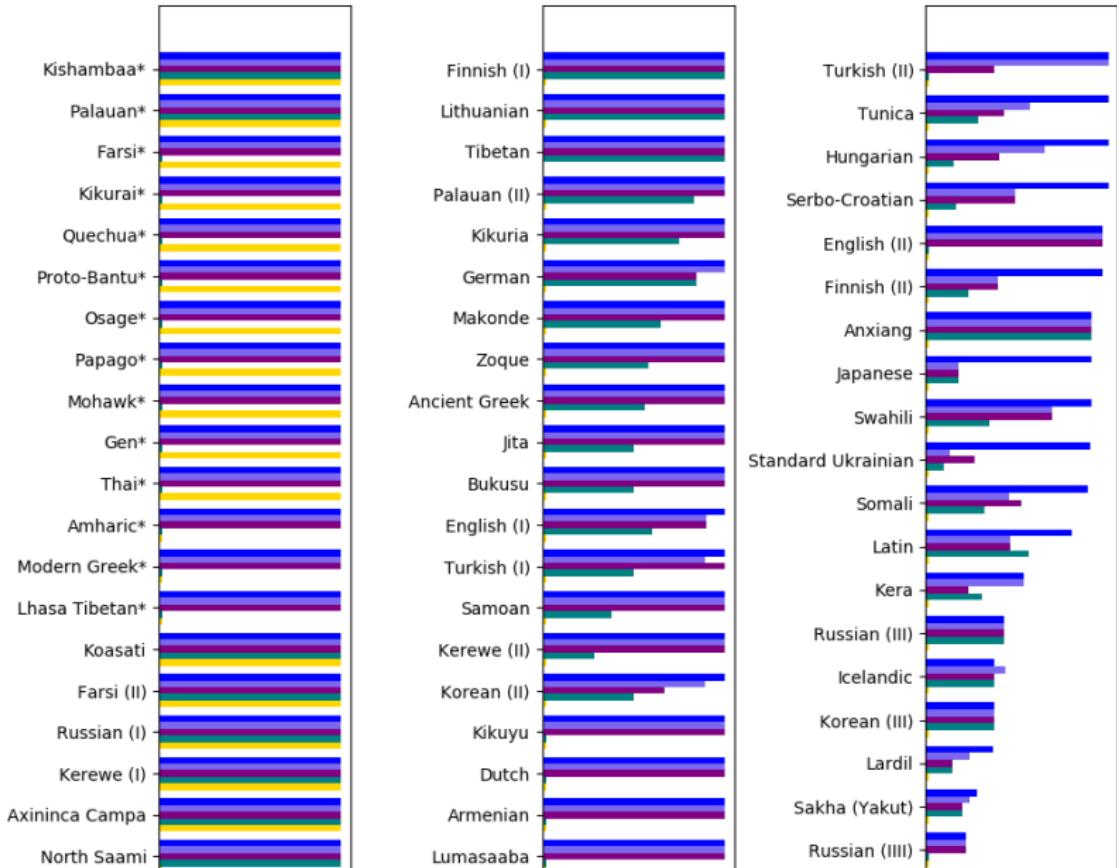
stems  
(unobserved)

observed data  
& unobserved derivation

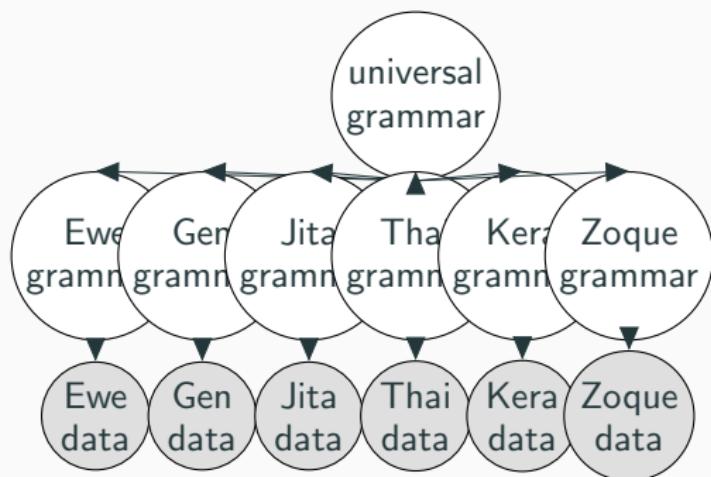
- ⟨/is̥kaap/, CABINET⟩
- ⟨/orɔn/, BED⟩
- ⟨/bie/, MARE⟩
- ⟨/örus/, RIVER⟩

CABINETS → /is̥kaap/+/lar/ → /is̥kaaplar/ $\xrightarrow{r_1}$ /is̥kaapdar/ $\xrightarrow{r_2}$ [is̥kaaptar]  
BEDS → /orɔn/+/lar/ → /orɔnlar/ $\xrightarrow{r_1}$ /orondar/ $\xrightarrow{r_3}$ /orondor/ $\xrightarrow{r_6}$ [oronnor]  
MARES → /bie/+/lar/ → /bielar/ $\xrightarrow{r_5}$ [bieler]  
RIVER (ASSOC) → /örus/+/iin/ → /örusliin/ $\xrightarrow{r_1}$ /örusdiin/ $\xrightarrow{r_2}$   
/örustiin/ $\xrightarrow{r_3}$ /örustuu/ $\xrightarrow{r_5}$ [örüstüün]

# 57 languages drawn from 70 datasets



# Distilling higher-level knowledge



# Distilling higher-level knowledge

Discovered  
universal grammar  
schema

w/o learned  
universal grammar

w/ learned  
universal grammar

## consonant/vowel distinction

sounds $\leftarrow$  [-vowel]

sounds $\leftarrow$  [+vowel]

*a set of sounds is commonly  
all consonants,  
or all vowels*

Tibetan:     [-nasal]  $\rightarrow \emptyset / \#_$

[-vowel]  $\rightarrow \emptyset / \# \_ [-vowel]$

## Growing domain-specific knowledge

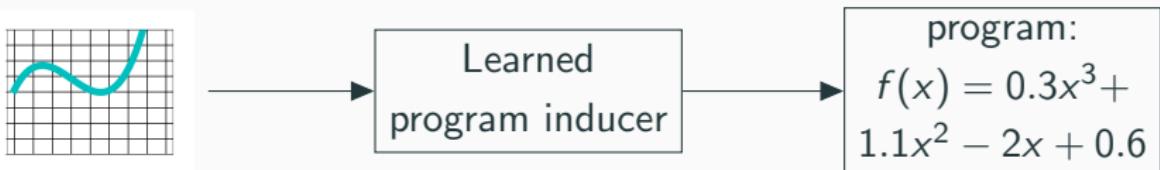
Goal: acquire domain-specific knowledge needed to induce a class of programs

- 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)



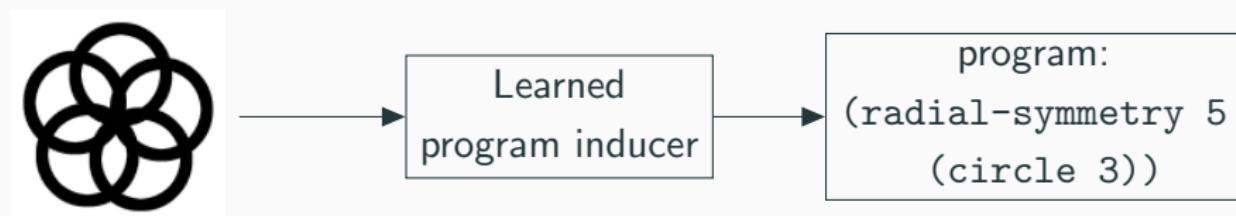
Concepts:  $x^3, \alpha x + \beta$ , etc

Inference strategy: neurosymbolic search for programs

# Growing domain-specific knowledge

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

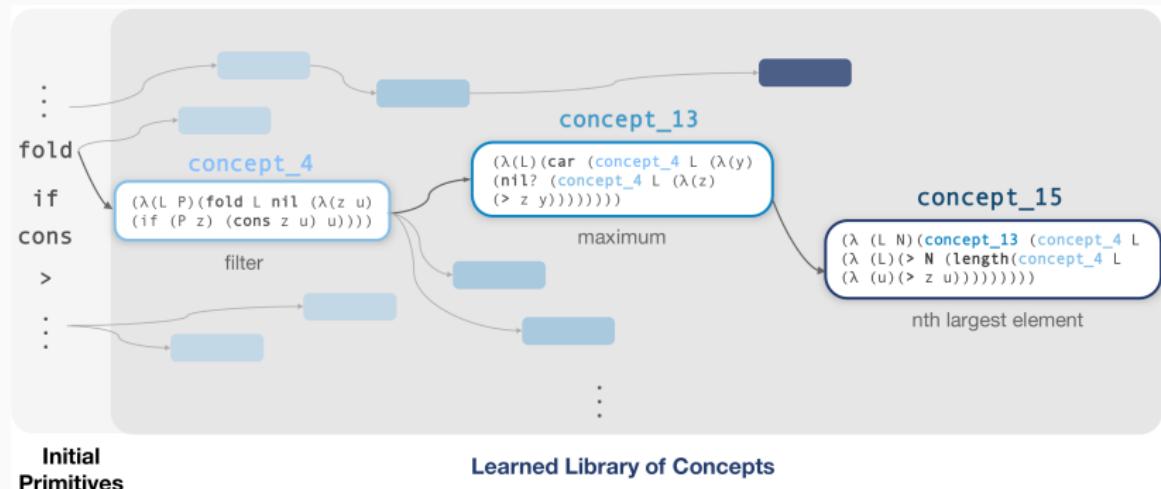
- Library of concepts (declarative knowledge; domain specific language; generative model over programs)
- Inference strategy (procedural knowledge; synthesis algorithm)



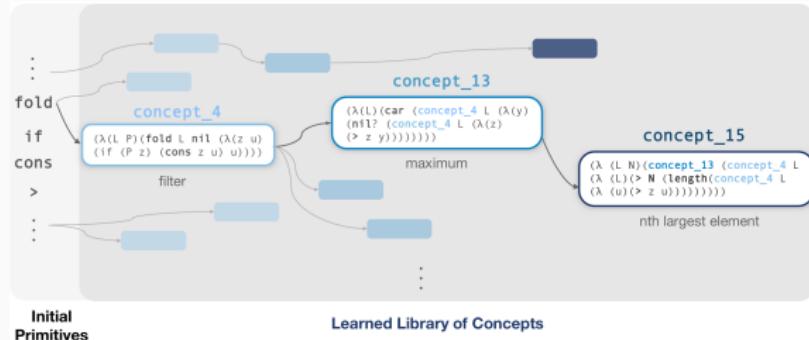
Concepts: circle, radial-symmetry, etc

Inference strategy: neurosymbolic search for programs

# Library learning



# Library learning



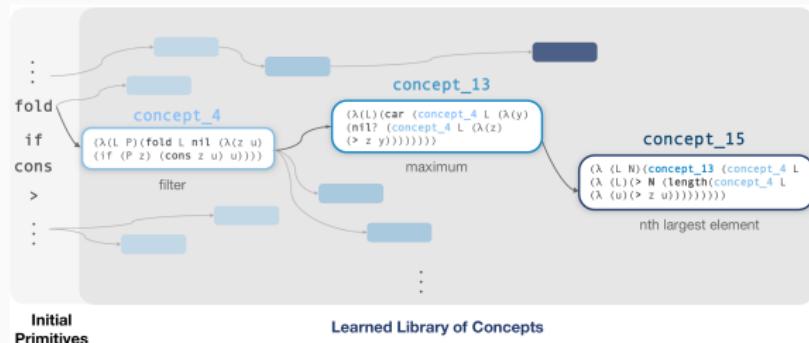
**Problem:** sort list

**Solution:**

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

get nth largest element      where n = 1, 2, 3 ....length of list

# Library learning



**Problem:** sort list

**Solution:**

`(map (\lambda (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:**

```
(\lambda(x) (map (\lambda(y) (car (fold x nil (\lambda(z u) (if (gt? (+ y 1) (length (fold x nil (\lambda(v w) (if (gt? z v) (cons v 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))))
```

**Discovered Problem Solutions**

# 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

Y1775/0704  
Y2000/0101

## LOGO Graphics



Physics

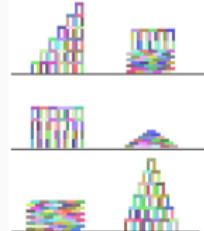
$$KE = \frac{1}{2}m|\vec{v}|^2$$

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

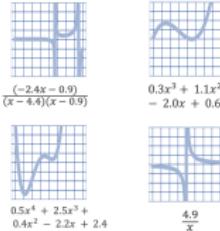
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{\vec{r}}_1 \cdot \hat{\vec{r}}_2$$

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

## Block Towers



## Symbolic Regression



## Recursive Programming

### Filter

$[■■■■■] \rightarrow [■■■]$   
 $[■■■■■■■■] \rightarrow [■■■■■■]$   
 $[■■■■■■■] \rightarrow [■■■■■]$

### Length

$[■■■■■] \rightarrow 4$   
 $[■■■■■■■■] \rightarrow 6$   
 $[■■■■■■■] \rightarrow 3$

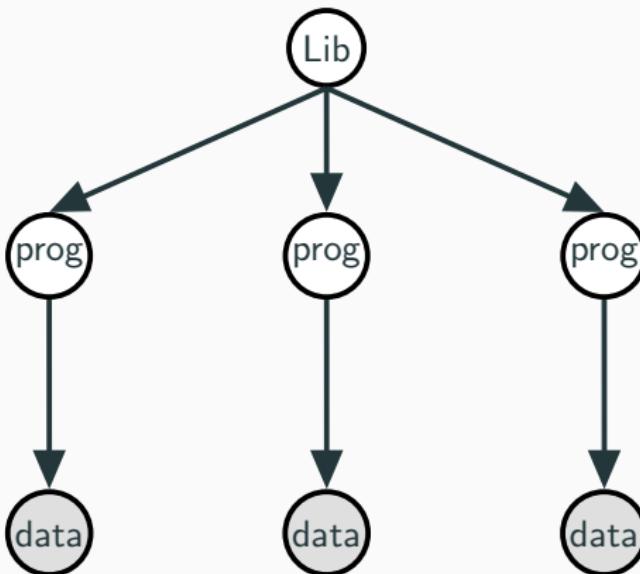
### Index List

$0, [■■■■■■■■] \rightarrow ■$   
 $1, [■■■■■■■■] \rightarrow ■■$   
 $1, [■■■■■■■■] \rightarrow ■■■$

### Every Other

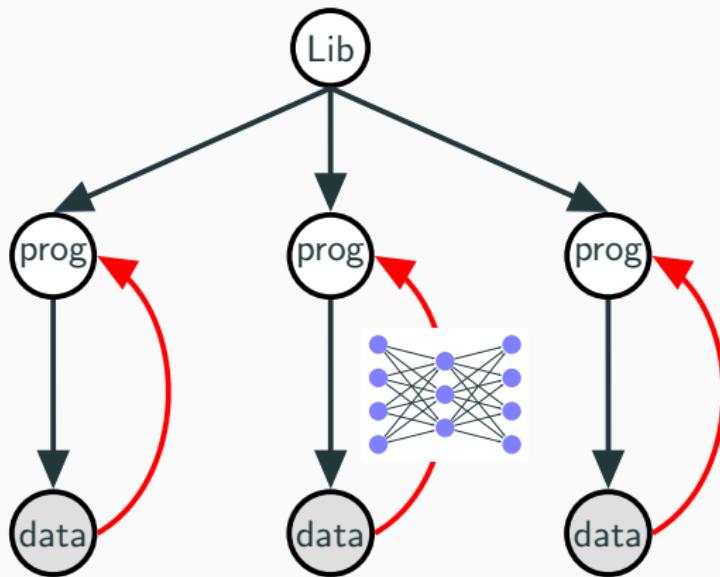
$[■■■■■■■■] \rightarrow [■■■■]$   
 $[■■■■■■■■] \rightarrow [■■■■■■]$   
 $[■■■■■■■■] \rightarrow [■■■■■■■]$

## Library learning as Bayesian inference

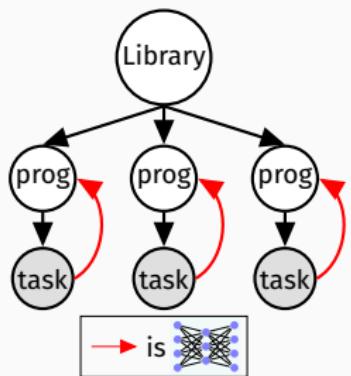


[Dechter et al., 2013] [Liang et al, 2010]; [Lake et al, 2015]

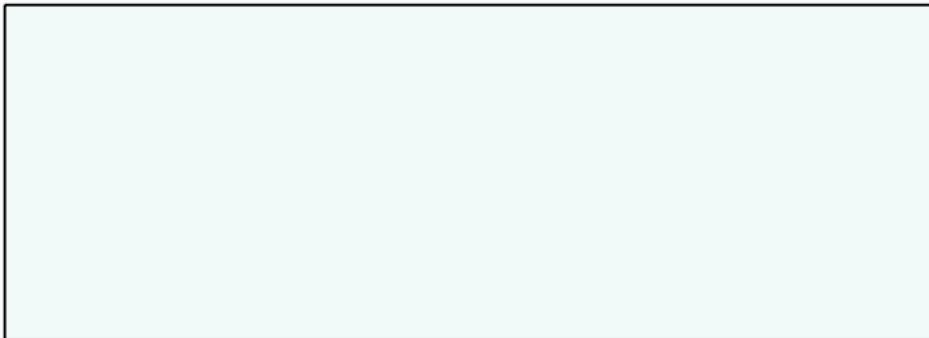
# Library learning as amortized Bayesian inference



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

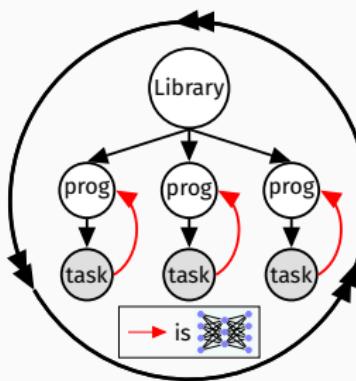


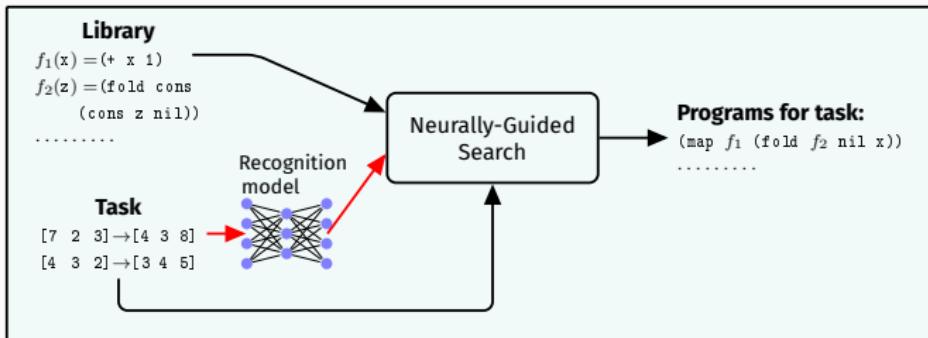
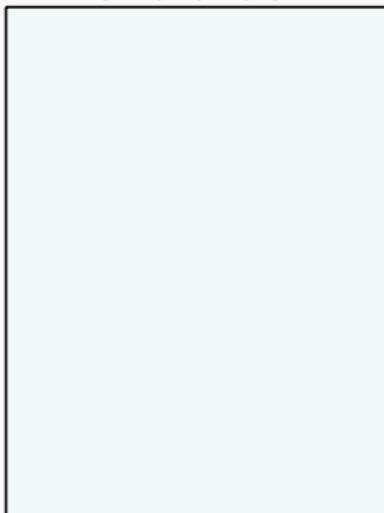
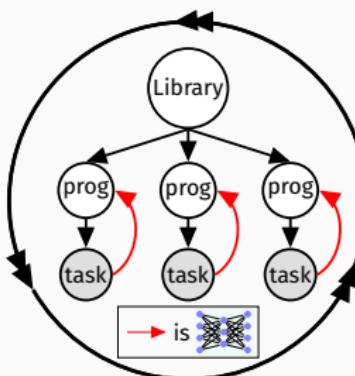
WAKE

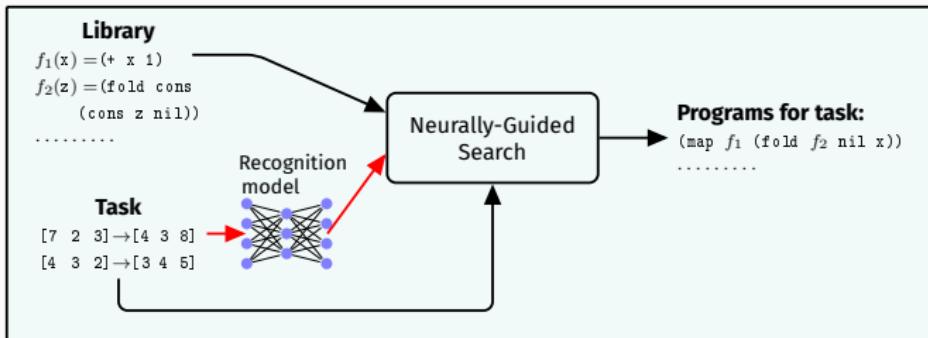
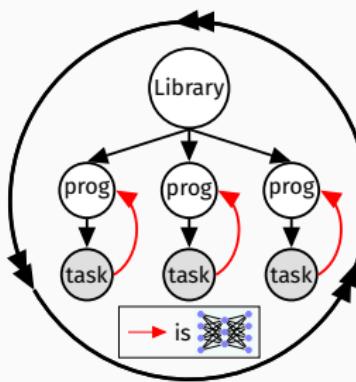
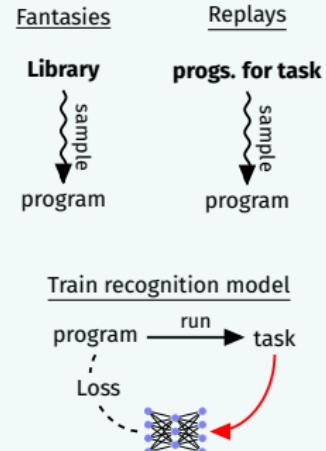


SLEEP: ABSTRACTION

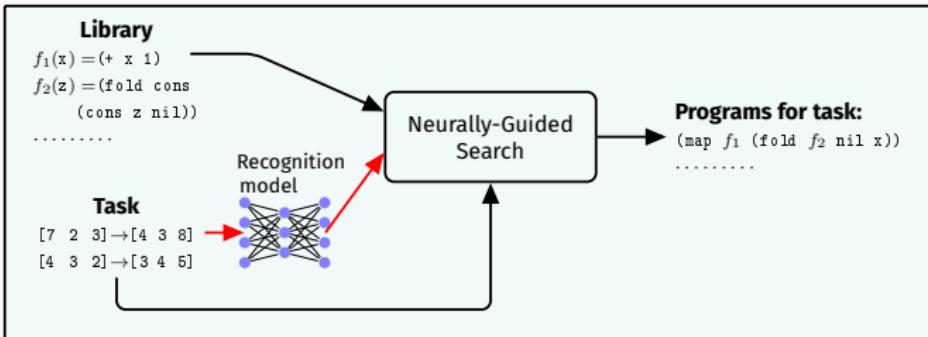
SLEEP: DREAMING



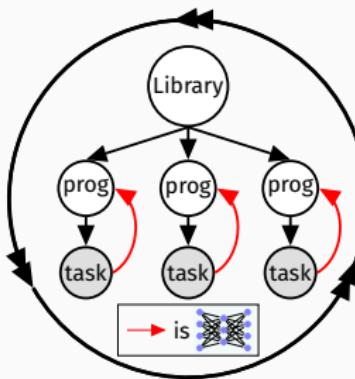
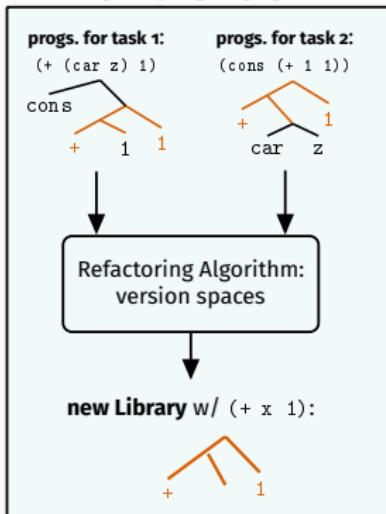
**WAKE****SLEEP: ABSTRACTION****SLEEP: DREAMING**

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

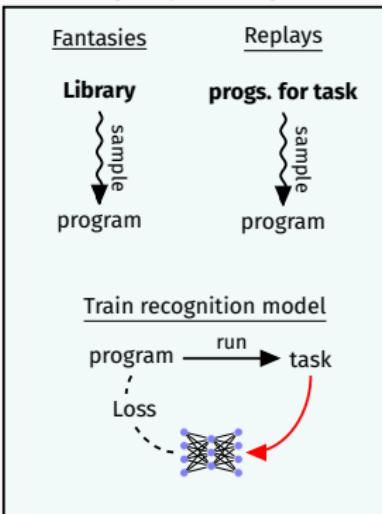
## WAKE



## SLEEP: ABSTRACTION



## SLEEP: DREAMING



## version space

$$5 + 5$$

## version space

$5 + 5$

(+ 5 5)

## version space

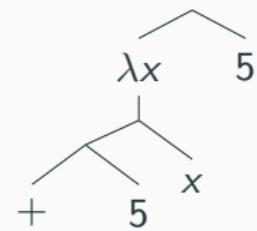
$5 + 5$



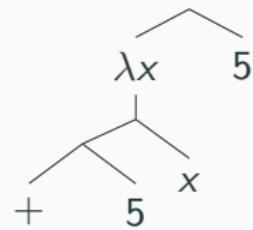
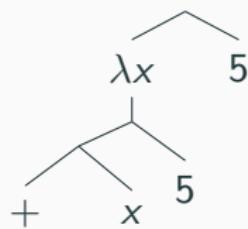
## version space



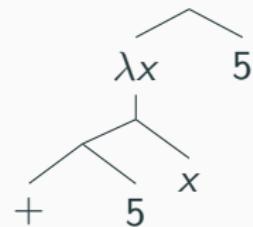
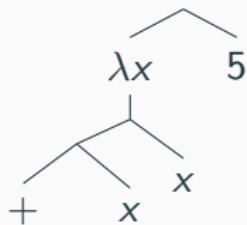
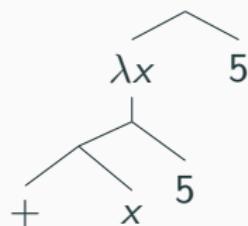
## version space



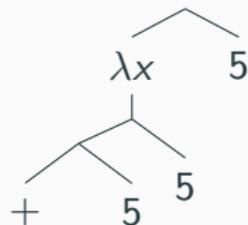
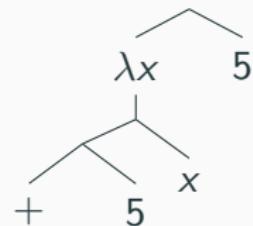
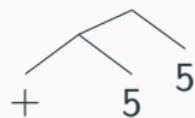
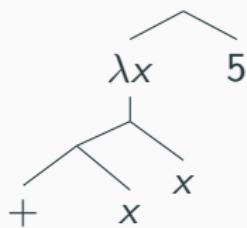
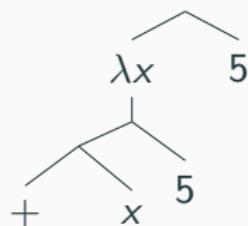
## version space



## version space

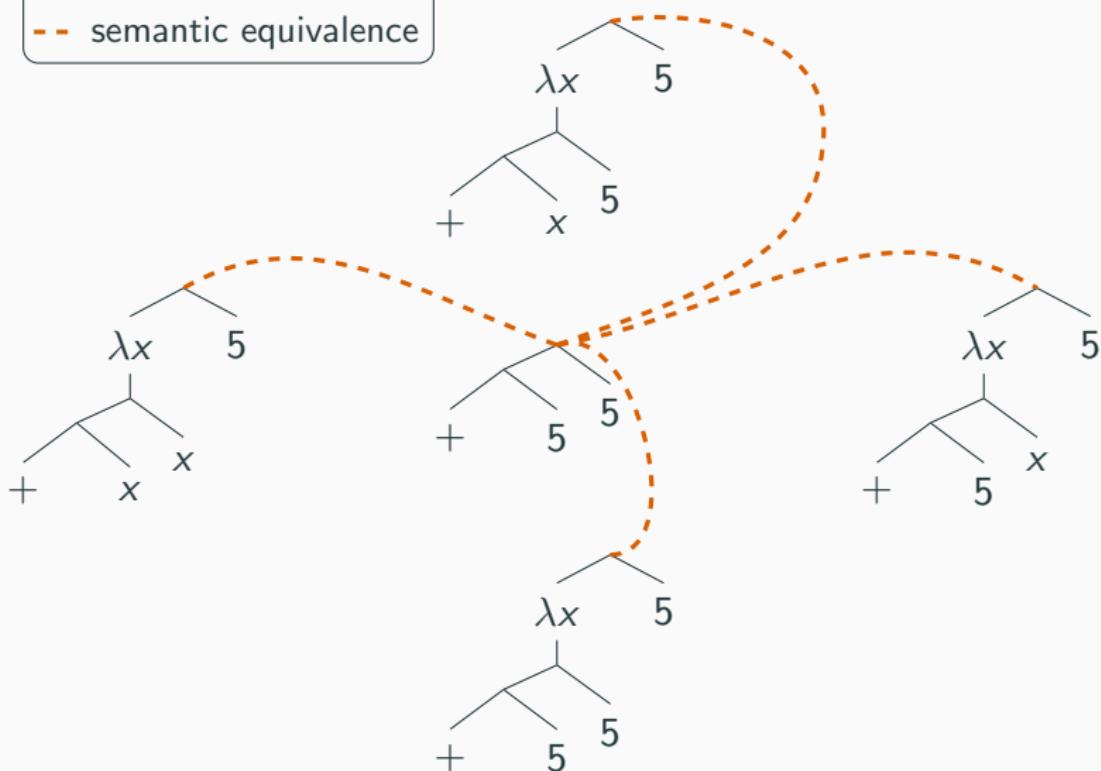


## version space



# version space

legend  
--- semantic equivalence

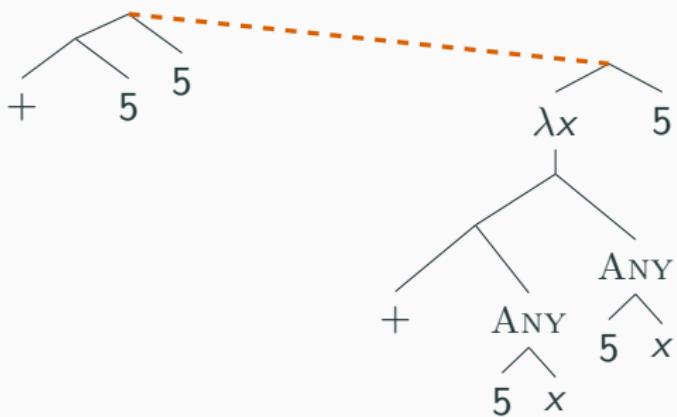


# version space

legend

--- semantic equivalence

ANY nondeterministic choice

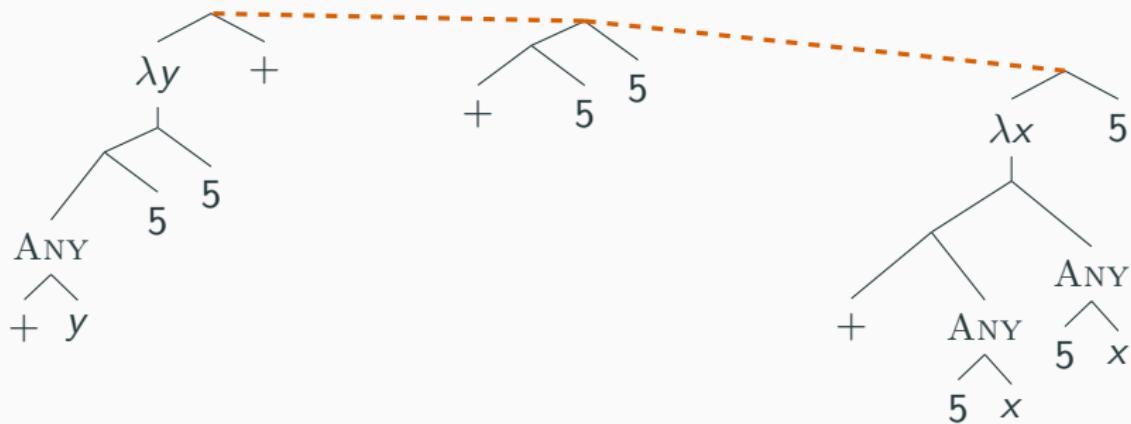


# version space

legend

semantic equivalence

ANY nondeterministic choice

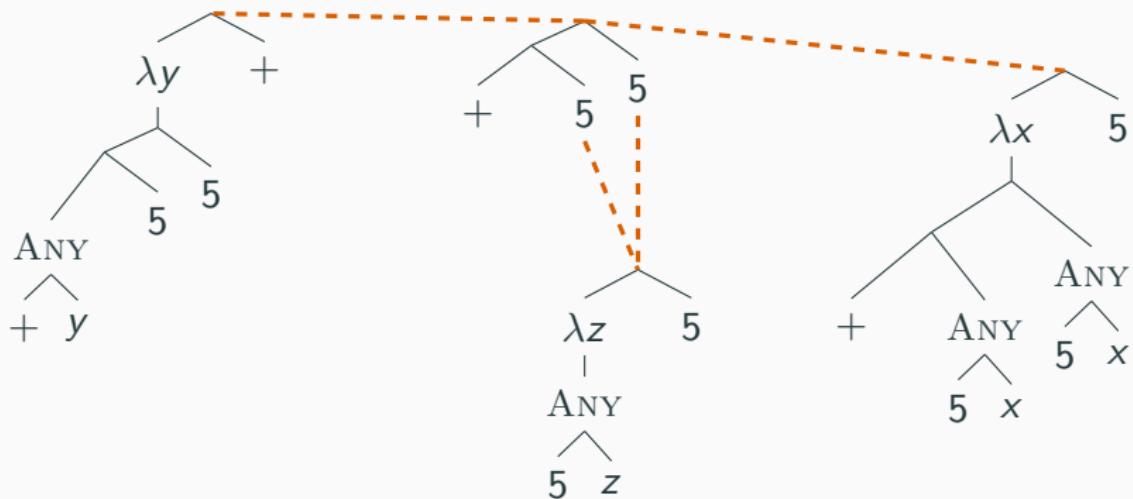


# version space

legend

dashed orange line semantic equivalence

ANY nondeterministic choice



# 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)

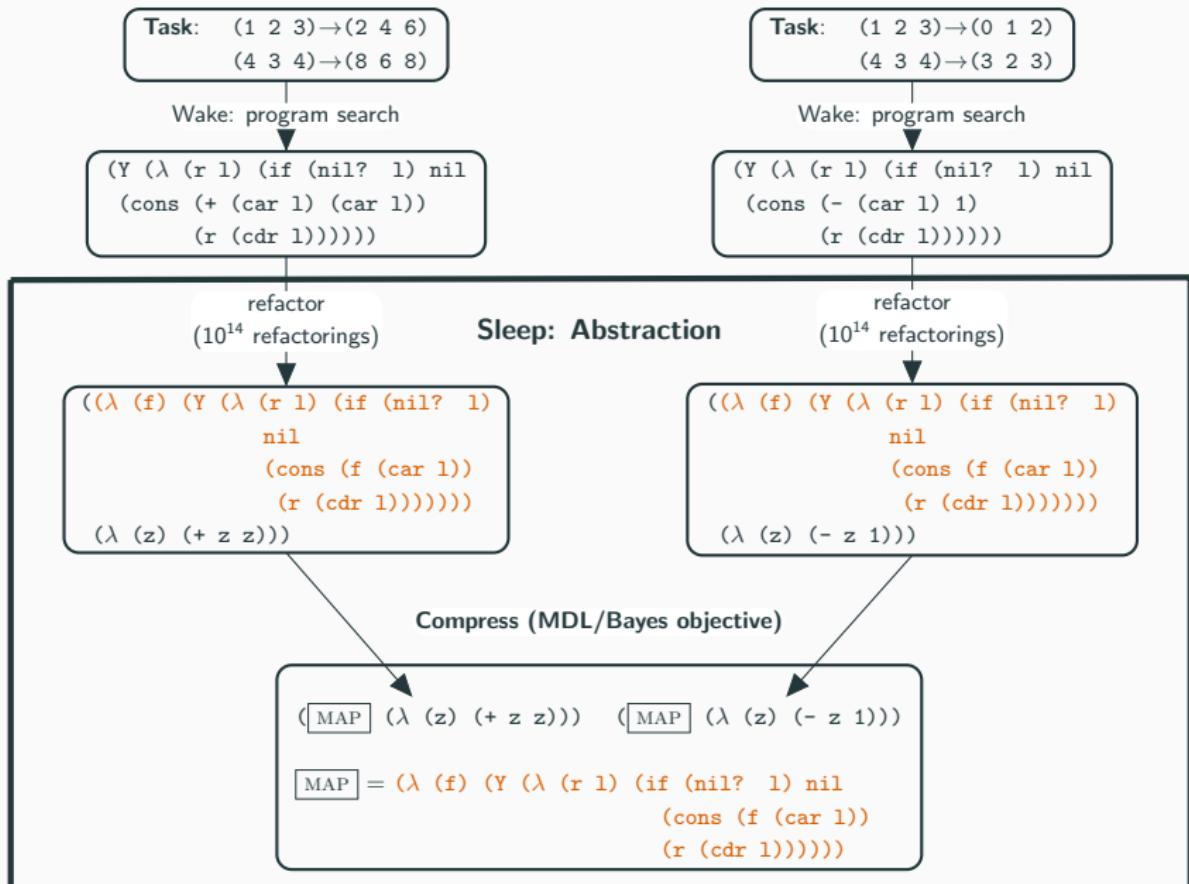
## Sleep: Abstraction

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

$$\begin{aligned} \text{VS} \rightarrow & \text{ var } | \lambda \text{var}. \text{VS} | (\text{VS } \text{VS}) | \text{ primitive} \\ & | \text{VS} \sqcup \text{VS} \quad \textit{nondeterministic choice} \\ & | \Lambda \quad \textit{choose any expression} \\ & | \emptyset \quad \textit{choose no expression} \end{aligned}$$

exploit the fact that  $e \in [\![v]\!]$  can be efficiently computed:

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

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

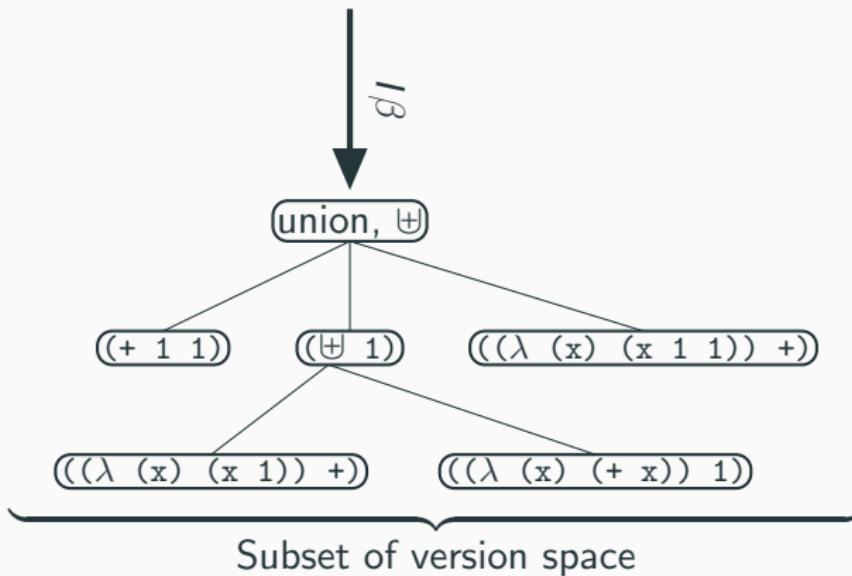
$$\text{REFACTOR}'(\lambda x. b|\text{Lib}) = \lambda x. \text{REFACTOR}(b|\text{Lib})$$

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

## 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*

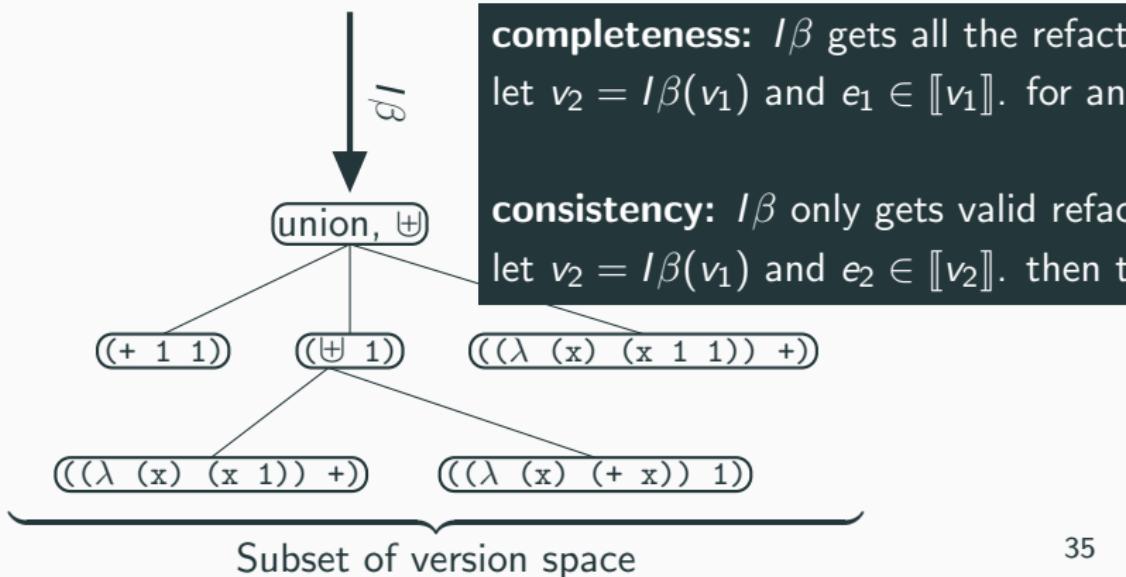
Program:  $(+ 1 1)$



# Using version spaces

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

Program:  $(+ 1 1)$



# DreamCoder Domains

## 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

### *Currency*

\$100.25

\$4.50

### *Dates*

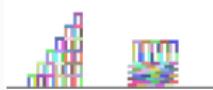
Y1775/0704

Y2000/0101

## LOGO Graphics



## Block Towers



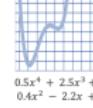
## Symbolic Regression



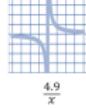
$$\frac{(-2.4x - 0.9)}{(x - 4.4)(x - 0.9)}$$



$$0.3x^3 + 1.1x^2 - 2.0x + 0.6$$



$$0.5x^4 + 2.5x^3 + 0.4x^2 - 2.2x + 2.4$$



$$\frac{4.9}{x}$$

## Recursive Programming

### *Filter*

[■■■■■] → [■■■]

[■■■■■■■■] → [■■■■■■]

[■■■■■■] → [■■■■]

### *Length*

[■■■■■] → 4

[■■■■■■■■] → 6

[■■■■] → 3

### *Index List*

0, [■■■■■■■■] → ■

1, [■■■■■■■■] → ■■

1, [■■■■■■■■] → ■■■

### *Every Other*

[■■■■■■] → [■■■]

[■■■■■■■■] → [■■■■■■]

[■■■■■■■■] → [■■■■■■]

## Physics

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

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

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

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

# 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

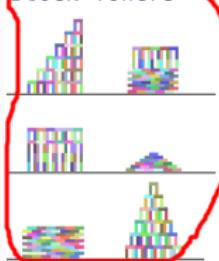
Y1775/0704

Y2000/0101

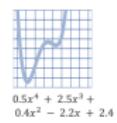
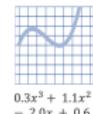
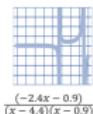
## LOGO Graphics



## Block Towers



## Symbolic Regression



## Recursive Programming

### Filter

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

### Length

[■■■■] → 4  
[■■■■■] → 6  
[■■■] → 3

### Index List

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

### Every Other

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

## Physics

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

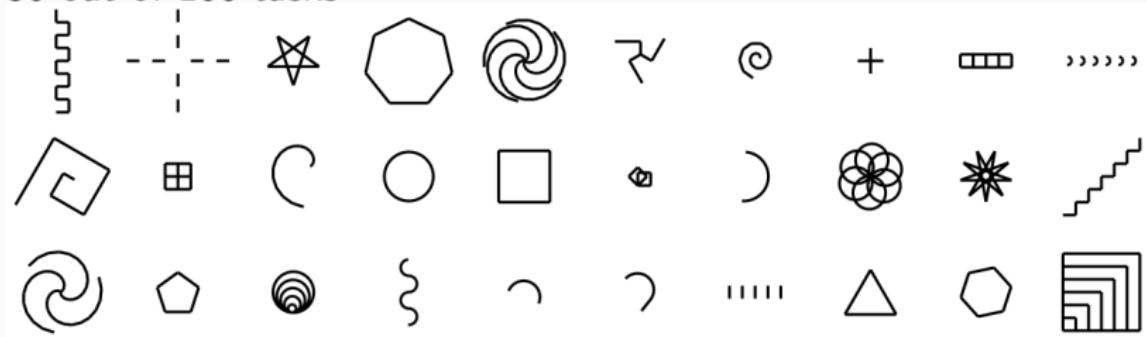
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

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

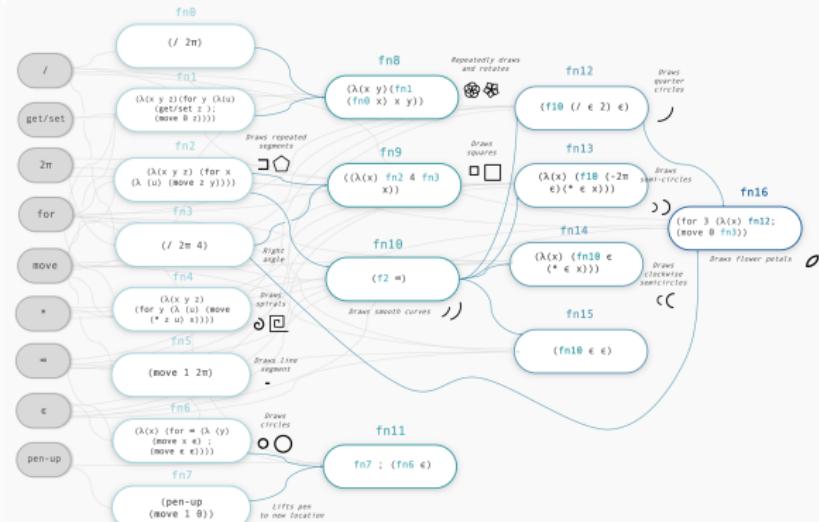
$$V_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$$

# LOGO Graphics

30 out of 160 tasks



# LOGO Graphics – learning interpretable library of concepts



Repetitively drawn spiral:  $(\text{fn8} \cdot 5 \cdot (\text{fn4} \cdot (* \cdot 2) \cdot \epsilon))$

Spiral shape:  $(\text{for } 2 \cdot (\lambda(x) (\text{fn14} \cdot 2); (\text{fn13} \cdot 2)))$

Square spiral:  $(\text{fn10} \cdot (* \cdot (\text{fn0} \cdot 7) \cdot 3) \cdot 3)$

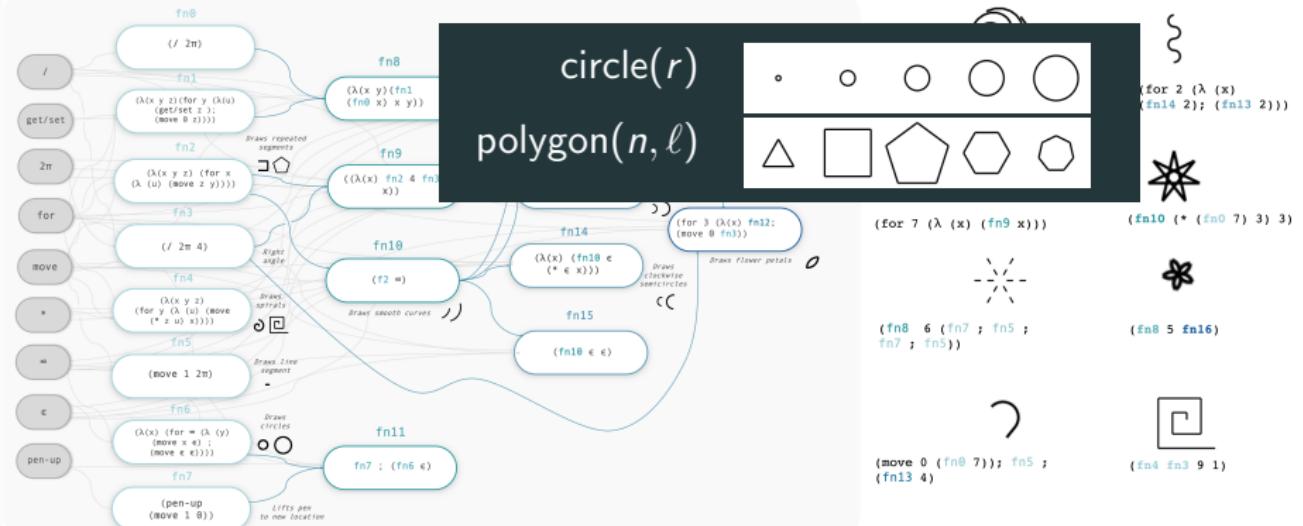
Star shape:  $(\text{fn8} \cdot 6 \cdot (\text{fn7} \cdot \text{fn5} \cdot \text{fn7} \cdot \text{fn5}))$

Flower shape:  $(\text{fn8} \cdot 5 \cdot \text{fn16})$

Question mark shape:  $(\text{move } 0 \cdot (\text{fn0} \cdot 7)); \text{fn5} \cdot (\text{fn13} \cdot 4)$

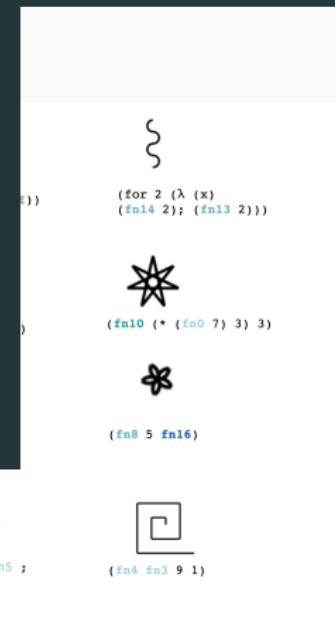
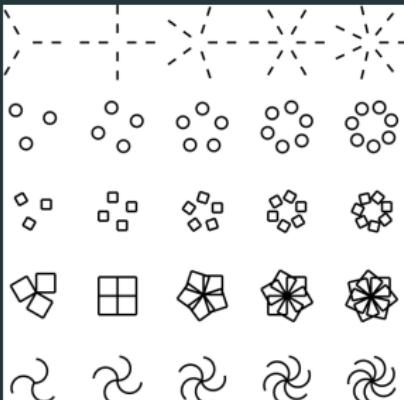
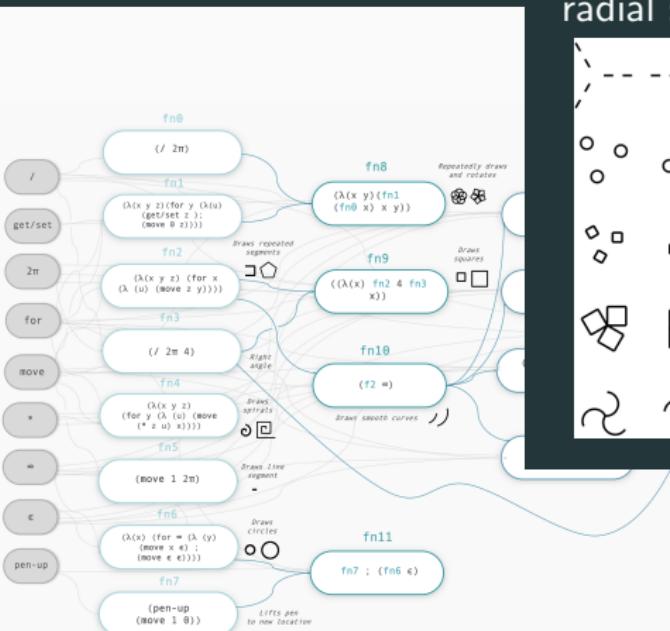
Box shape:  $(\text{fn4} \cdot \text{fn3} \cdot 9 \cdot 1)$

# LOGO Graphics – learning interpretable library of concepts



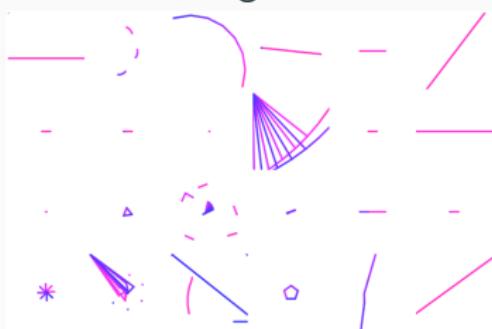
# LOGO Graphics – learning interpretable library of concepts

radial symmetry( $n$ , body)

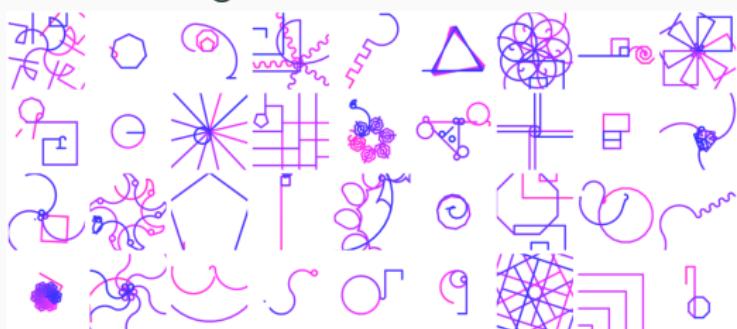


# what does DreamCoder dream of?

before learning

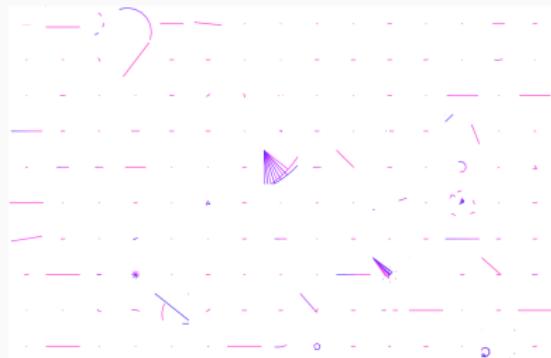


after learning

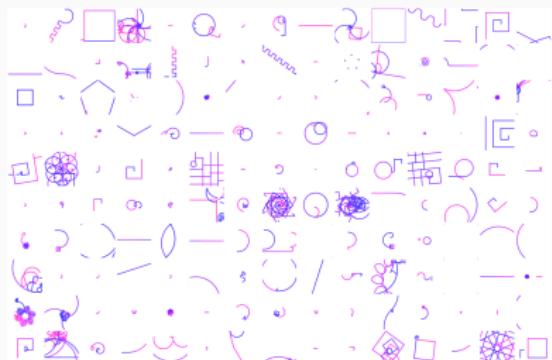


# what does DreamCoder dream of?

before learning

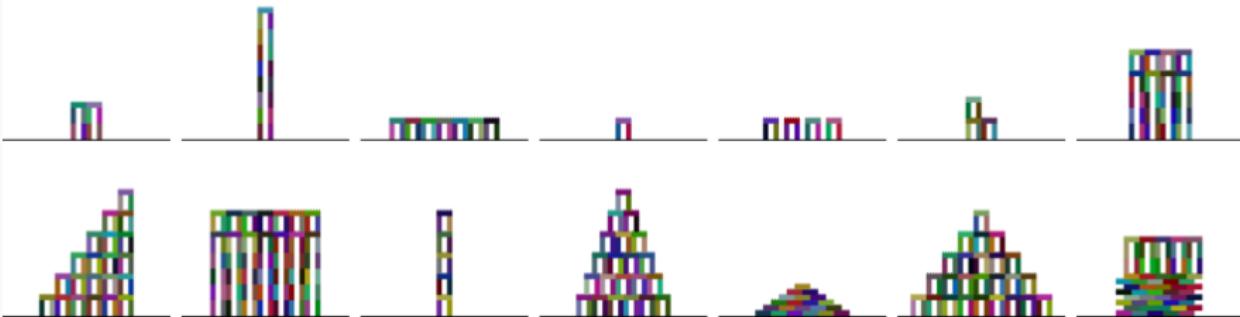


after learning



# Planning to build towers

example tasks (112 total)



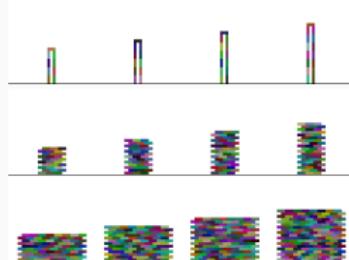
# Planning to build towers

example tasks (112 total)

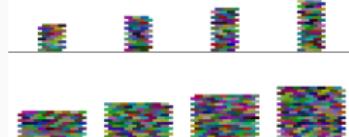


learned library routines ( $\approx 20$  total)

$\text{arch}(h)$



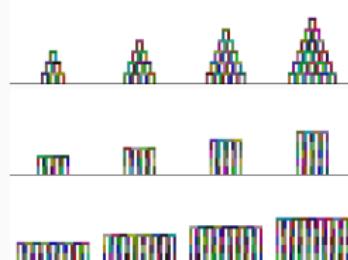
$\text{wall}(w, h)$



$\text{pyramid}(h)$



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



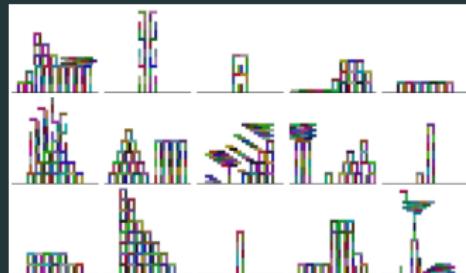
# Planning to build towers

example tasks (112 total)

**dreams before learning**



**dreams after learning**



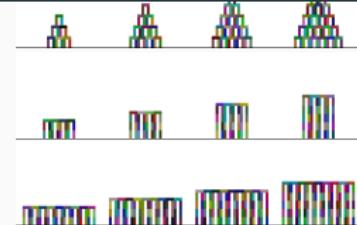
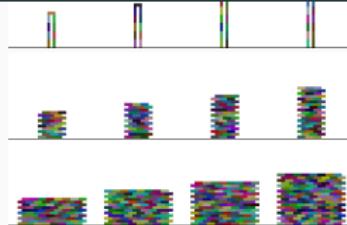
learned lib

arch( $h$ )

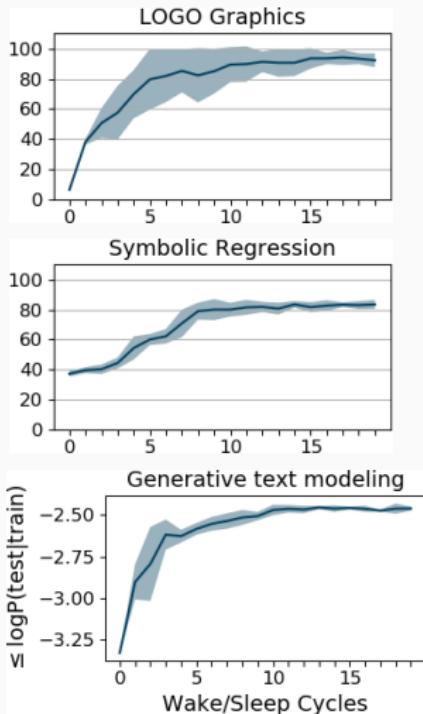
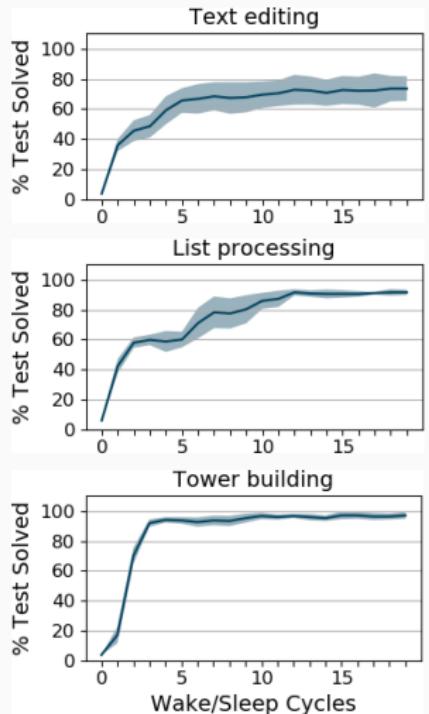
pyramid( $h$ )

wall( $w, h$ )

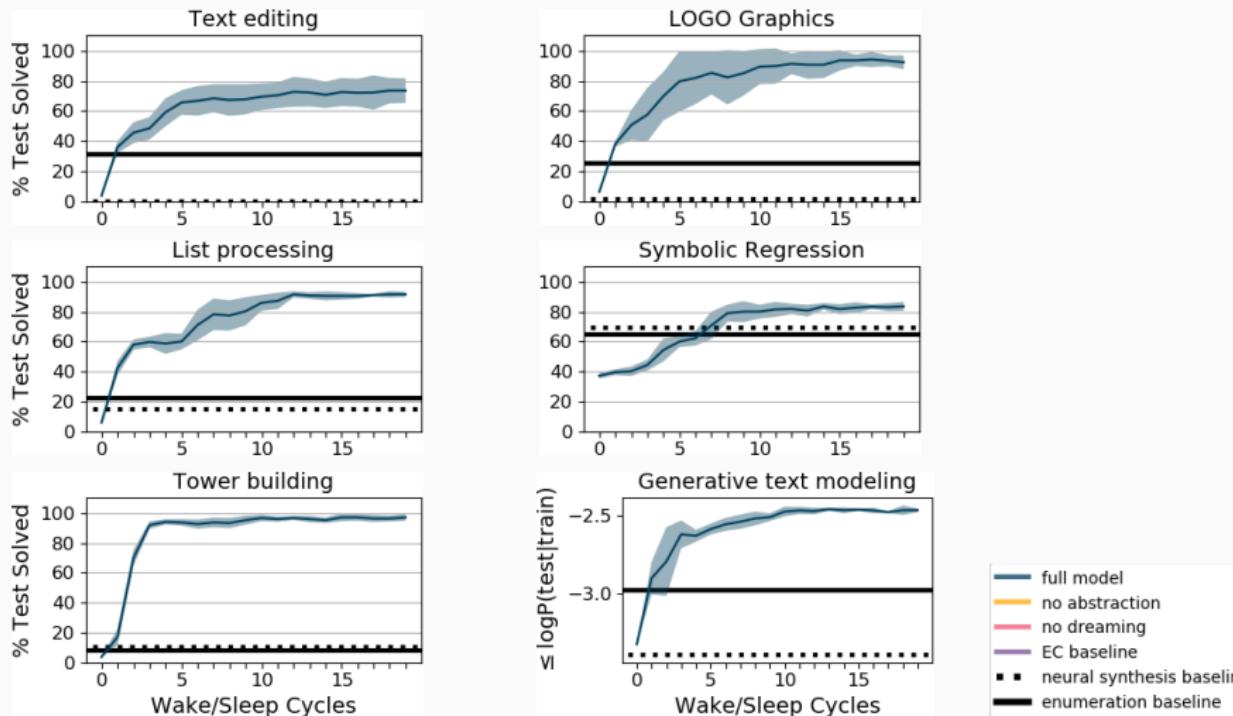
bridge( $w, h$ )



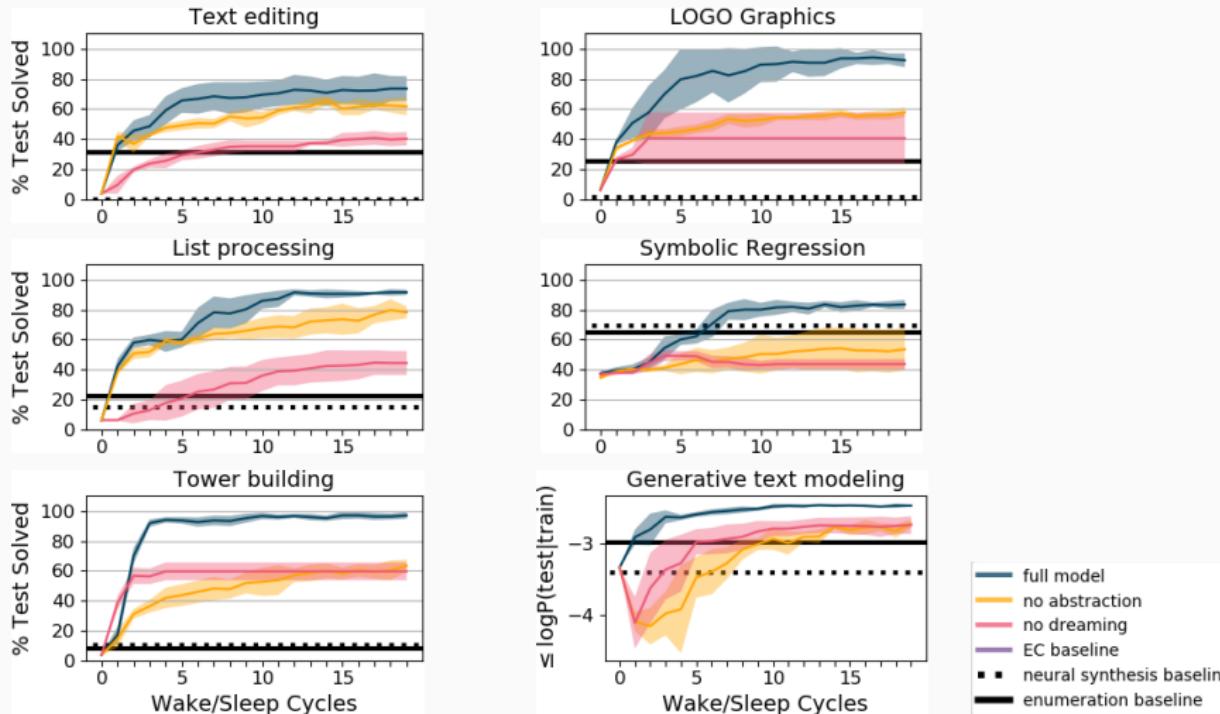
# synergy between dreaming and library learning



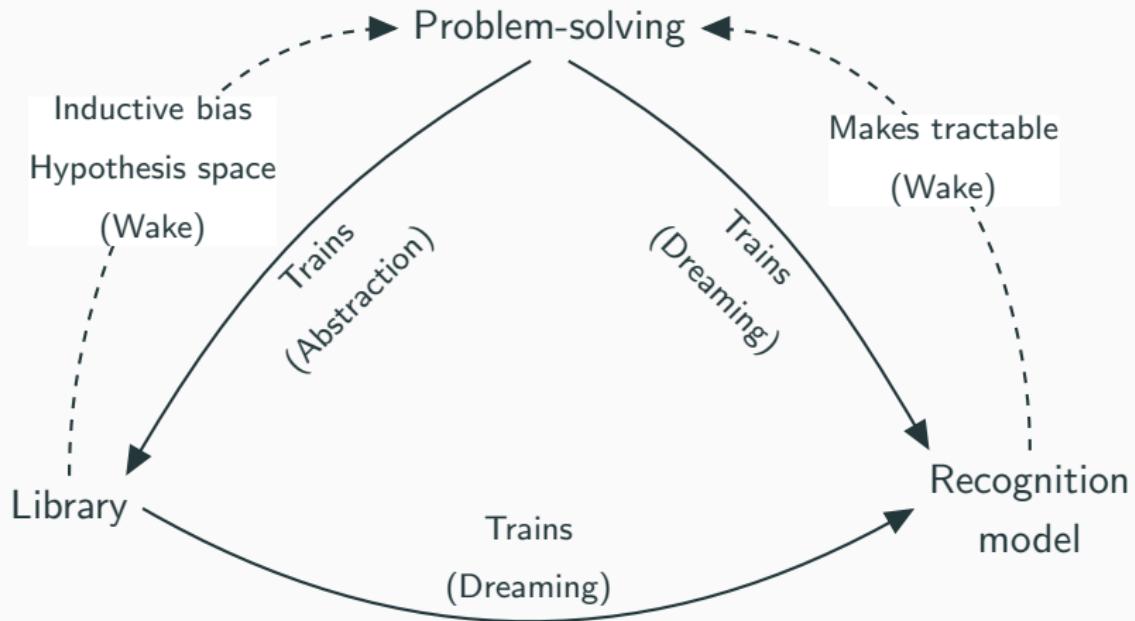
# synergy between dreaming and library learning



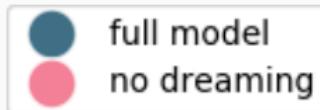
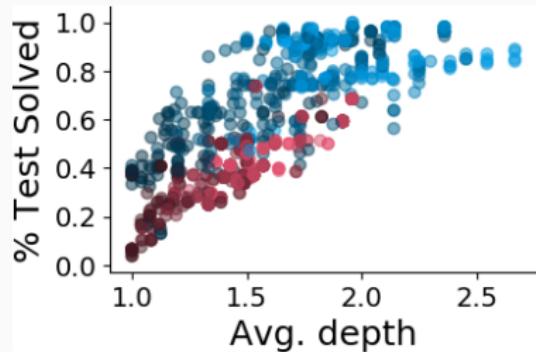
# synergy between dreaming and library learning



# synergy between dreaming and library learning



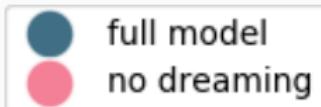
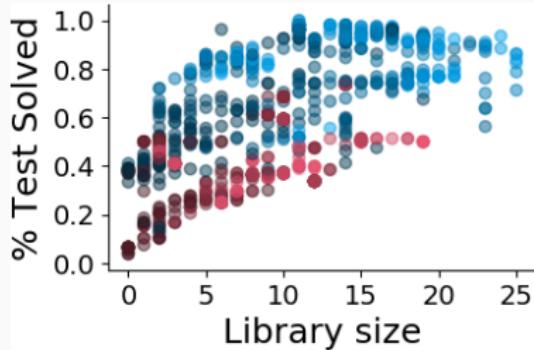
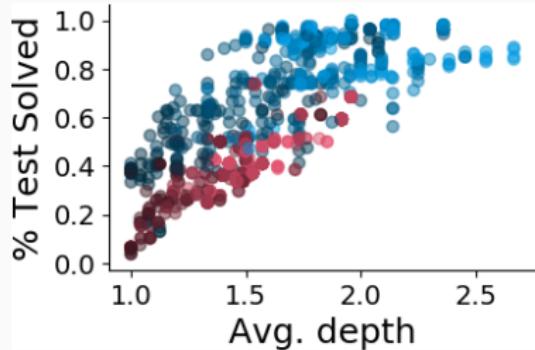
# Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Lighter: Later in learning

# Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

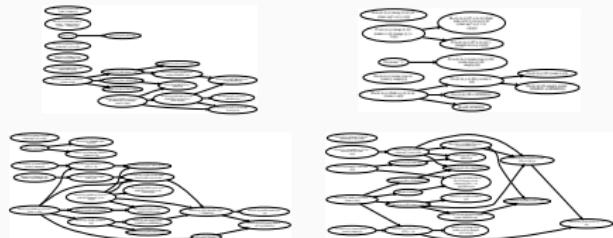
Lighter: Later in learning

# Variability in learned library

List processing



Text editing



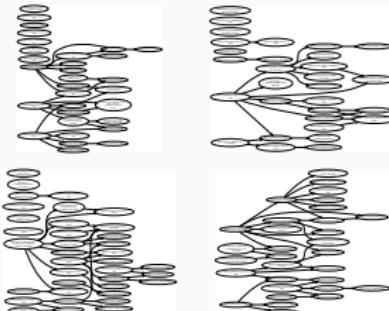
Tower building



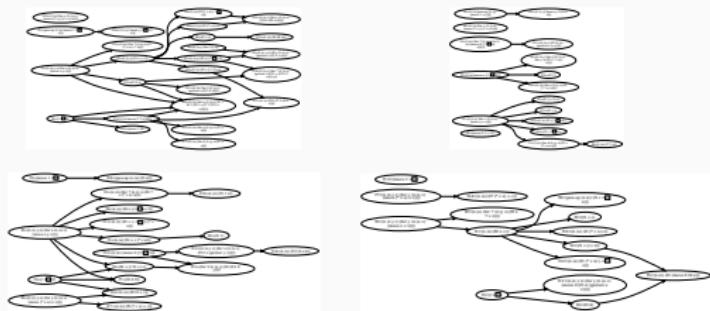
Symbolic regression



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

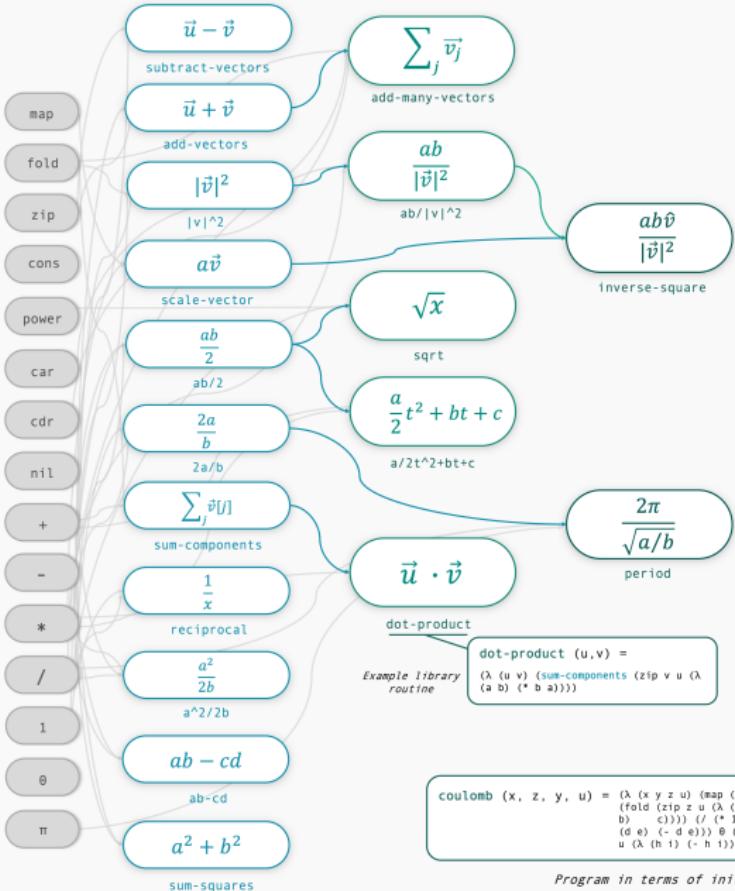
these experiments study how DreamCoder grows from a “beginner” state to “expert”:

- “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  
(λ (r) (reciprocal r)) Rs)))

Energy in a Capacitor

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

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

Ballistic Motion

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

( $a^2/2b^2+bt+c$  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

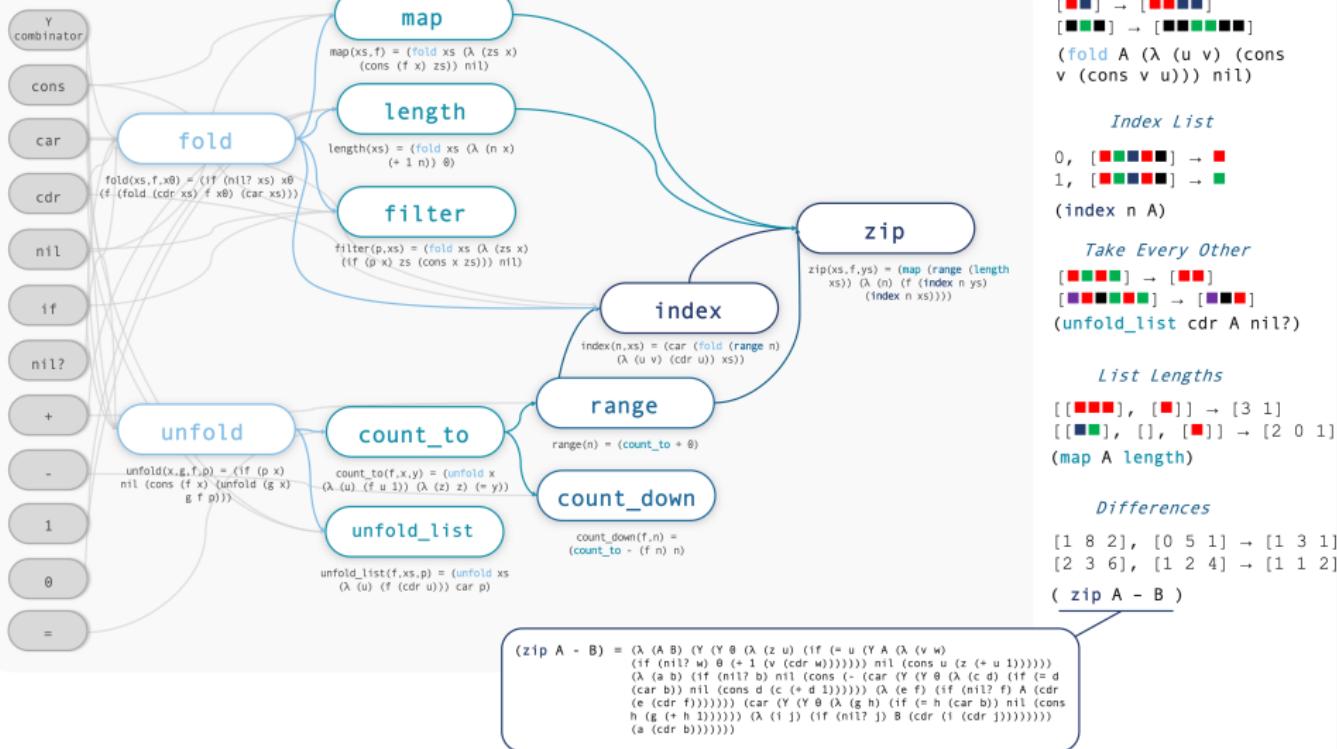
$$X_{CM} = \frac{\sum_l m_l x_l}{\sum_l m_l}$$

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

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

Program in terms of initial library

# Rediscovering origami programming



Program in terms of initial library

# From learning libraries to using interpreters

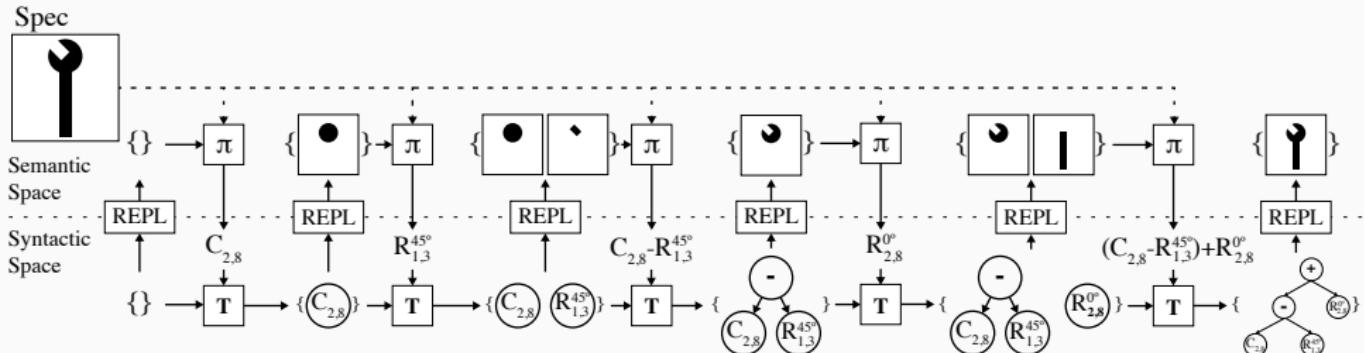
DreamCoder: library building as inspiration

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

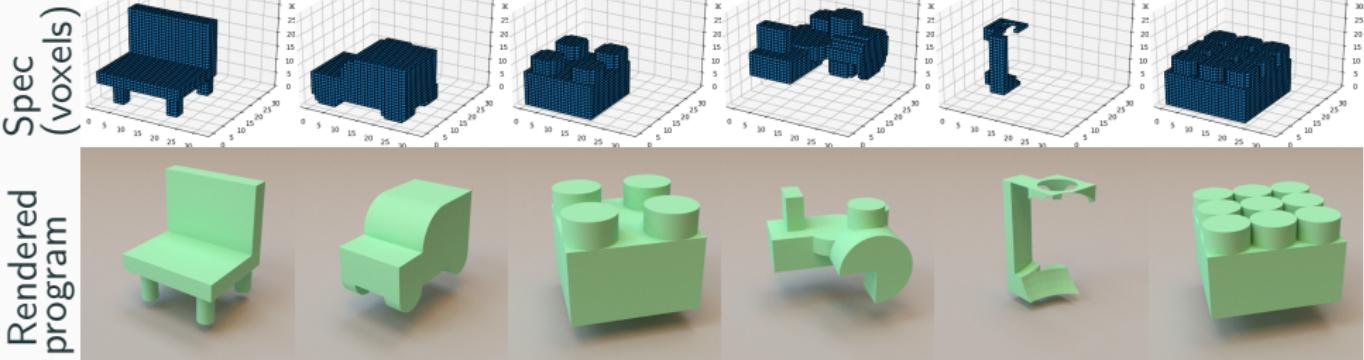
Ellis\*, Nye\*, Pu\*, Sosa\*, Tenenbaum, Solar-Lezama. NeurIPS 2019.

\*equal contribution

# Scaling to long programs

Branching factor: > 1.3 million per line of code

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



# Programmatic models of the physical world

hinge



gear



doorknob

