

# **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



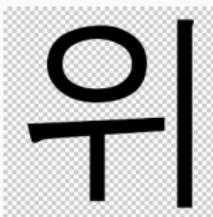
engineering



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 frameworks can contribute to this picture?

Three AI traditions

# What computational frameworks can contribute to this picture?

Three AI traditions

Symbolic



In[34]:= **Solve**[{(h w - h w^2) == Z}, h]

Out[34]= {}



In[33]:= **Solve**[(h w - h w^2) == Z, h]

Input interpretation:

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

Result:

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

# What computational frameworks can contribute to this picture?

## Three AI traditions

Symbolic



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

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



```
In[33]:= Solve[(hw-hw^2)==z],h]
```

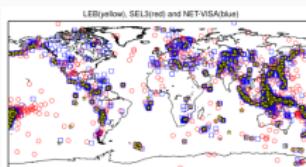
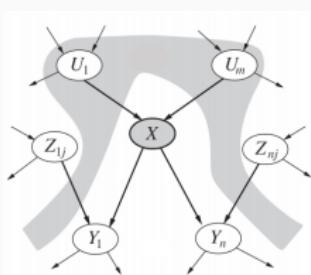
Input interpretation:

solve  $h w - h w^2 = z$  for  $h$

Result:

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

Probabilistic



# What computational frameworks can contribute to this picture?

## Three AI traditions

### Symbolic



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

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

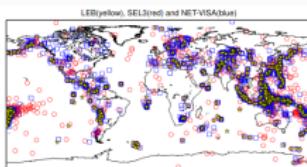
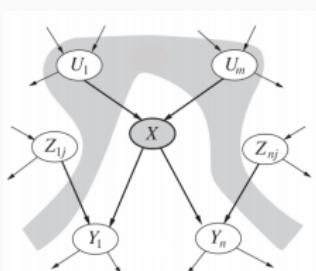
```
Input interpretation:
```

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

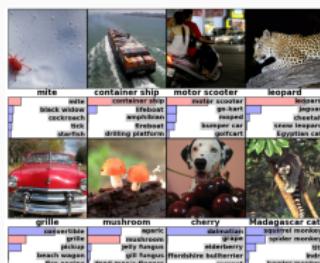
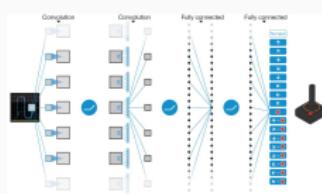
Result:

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

### Probabilistic



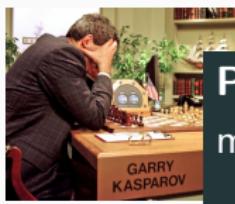
### Neural



# What computational frameworks can contribute to this picture?

Three AI traditions

Symbolic



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

Out[34]= {}

Solve[(hw-hw^2)==Z],h

Input interpretation:

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

Result:

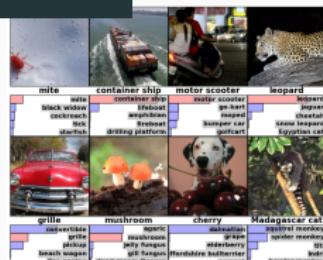
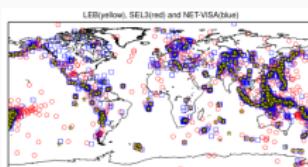
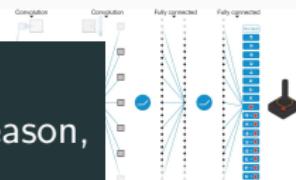
$$h = \frac{Z}{w - w^2} \text{ and } w^2 \neq w$$

Probabilistic



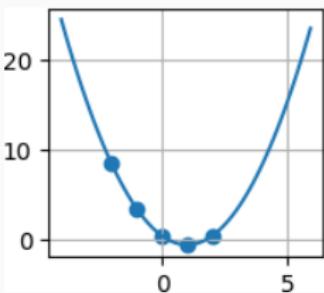
**Program induction**  
machines that learn, perceive, and reason,  
by writing their own code

Neural



# Why program induction?

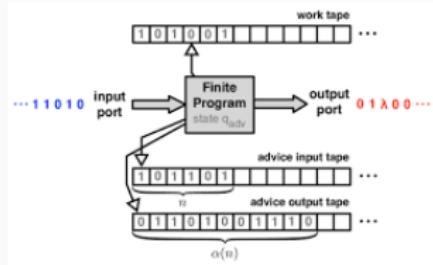
strong generalization  
+ data efficiency



interpretability



universal expressivity



# Why didn't this old idea work?

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



A PROPOSAL FOR THE  
DARTMOUTH SUMMER RESEARCH PROJECT  
ON ARTIFICIAL INTELLIGENCE

J. McCarthy, Dartmouth College  
M. L. Minsky, Harvard University  
N. Rochester, I.B.M. Corporation  
C. E. Shannon, Bell Telephone Laboratories



# Why will it work this time?

better toolkits:

## Why will it work this time?

better toolkits:

- **probabilistic** methods for uncertainty and learning-to-learn

## Why will it work this time?

better toolkits:

- **probabilistic** methods for uncertainty and learning-to-learn
- **neural** methods for guiding combinatorial search

## 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:

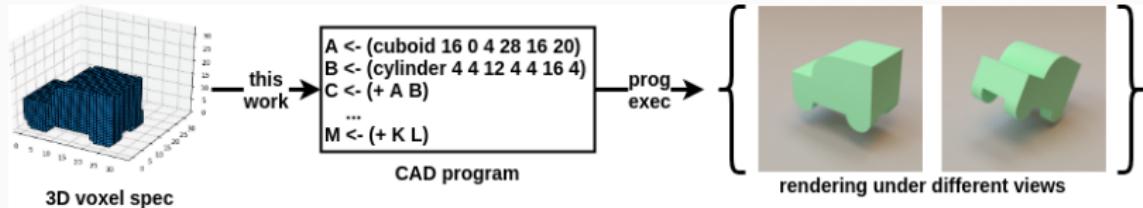
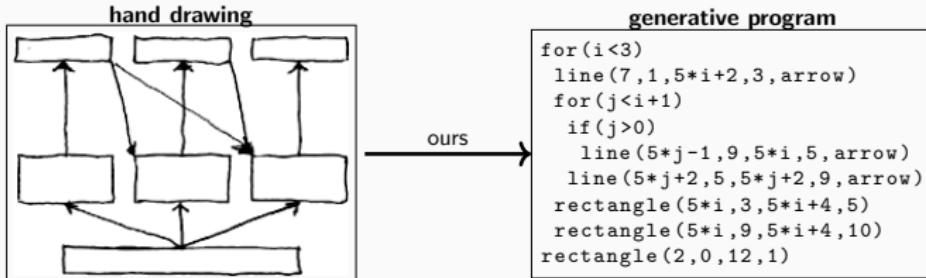
- **probabilistic** methods for uncertainty and learning-to-learn
- **neural** methods for guiding combinatorial search
- **symbolic** methods, from the **programming languages** community
  - maturing **program synthesis** techniques
  - type systems, program analysis, constraint solving, ...

better problems:

- inverse CAD [Kulkarni et al. 2015]
- synthesizing human-understandable models [Evans et al. 2019]
- natural language→code [Liang et al. 2011; Zettlemoyer et al. 2007]
- programming by examples [Gulwani 2011],  
computer-aided-programming [Solar-Lezama 2008]

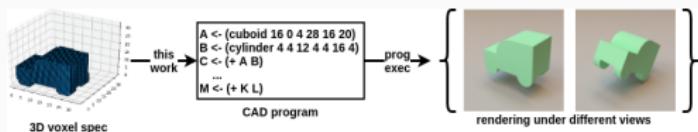
# Perception, Synthesizing models, Learning-to-Learn

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

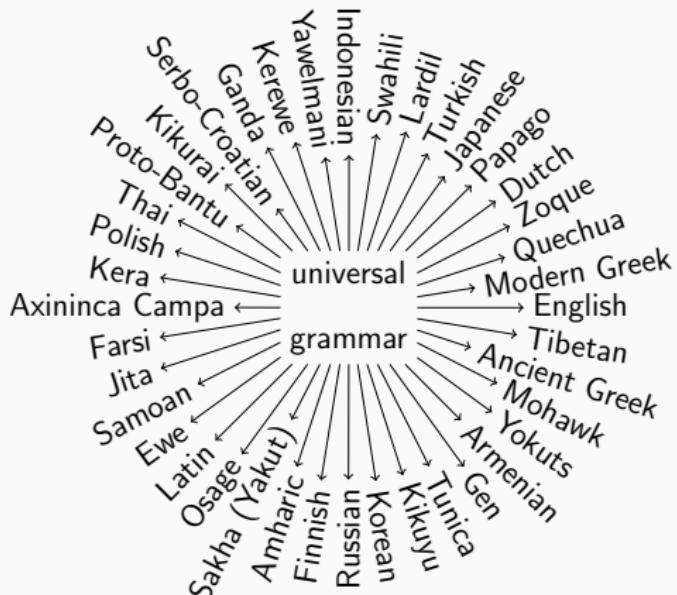


# Perception, Synthesizing models, Learning-to-Learn

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

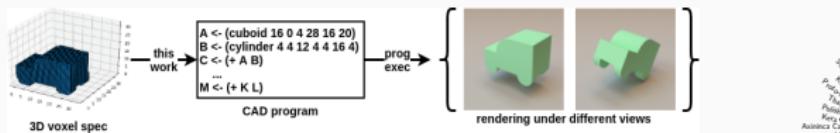


Theme #2: Synthesizing human-understandable models



## Perception, Synthesizing models, Learning-to-Learn

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



## Theme #2: Synthesizing interpretable models

## Theme #3: Learning to synthesize programs

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	  

Block Towers	Symbolic Regression	Recursive Programming	Physics
	 $\frac{(-2.4x - 0.9)}{(x - 4.4)}(x - 0.9)$	 $0.3x^3 + 1.1x^2 - 2.0x + 0.6$	<i>Filter</i> $\begin{array}{l} [\textcolor{red}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{green}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow [\textcolor{red}{\blacksquare}] \\ [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}] \\ [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}] \end{array}$ <i>Index List</i> $\begin{array}{l} 0, [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow \textcolor{red}{\blacksquare} \\ 1, [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow \textcolor{black}{\blacksquare} \\ 1, [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow \textcolor{blue}{\blacksquare} \end{array}$
	 $0.5x^4 + 2.5x^3 + 0.4x^2 - 2.2x + 2.4$	 $\frac{4.9}{x}$	$\vec{d} = \frac{1}{m} \sum_i \vec{f}_i$ $KE = \frac{1}{2} m  \vec{v} ^2$
	 $0.5x^4 + 2.5x^3 + 0.4x^2 - 2.2x + 2.4$	<i>Length</i> $\begin{array}{l} [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow 4 \\ [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow 6 \\ [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow 3 \end{array}$ <i>Every Other</i> $\begin{array}{l} [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow [\textcolor{red}{\blacksquare}] \\ [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}] \\ [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \rightarrow [\textcolor{red}{\blacksquare}\textcolor{black}{\blacksquare}\textcolor{blue}{\blacksquare}\textcolor{blue}{\blacksquare}] \end{array}$	$\vec{F}^2 = \frac{q_1 q_2}{ \vec{r}_1 - \vec{r}_2 } \vec{r}_1 \cdot \vec{r}_2$ $R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$

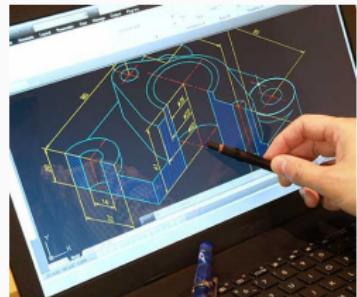
Program Induction and perception  
learning to learn  
interpretable models

# High-level, abstract visual abilities

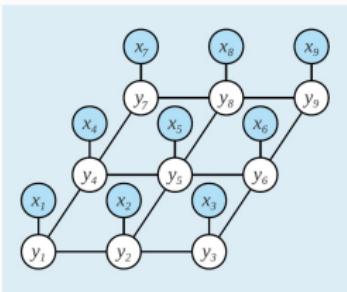
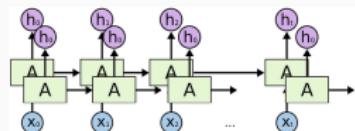
...in art



...in engineering



...in AI



# High-level, abstract visual abilities

...in art



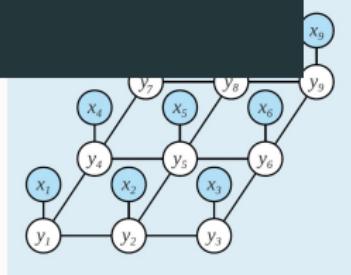
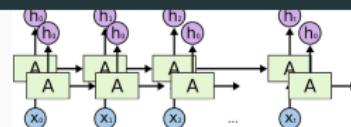
why?

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

how?

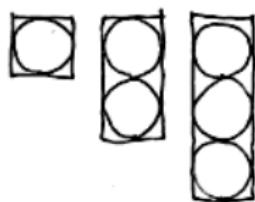
Bayesian inference of graphics program conditioned on image,  
+program synthesis  
+learning

...in AI



# Learning to infer graphics programs from hand-drawn images

model infers program from drawing



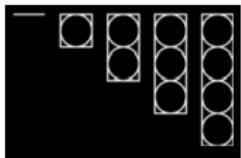
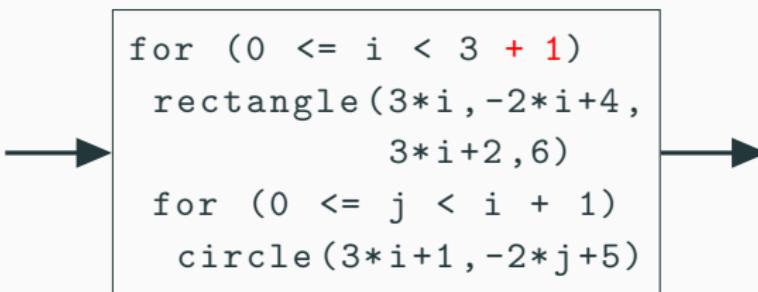
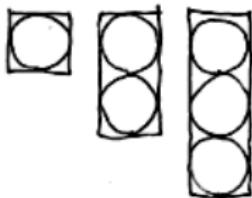
→

```
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)
```

# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

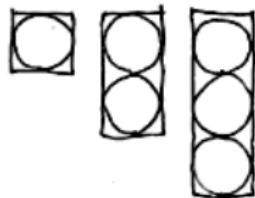
**zero-shot generalization / extrapolation**



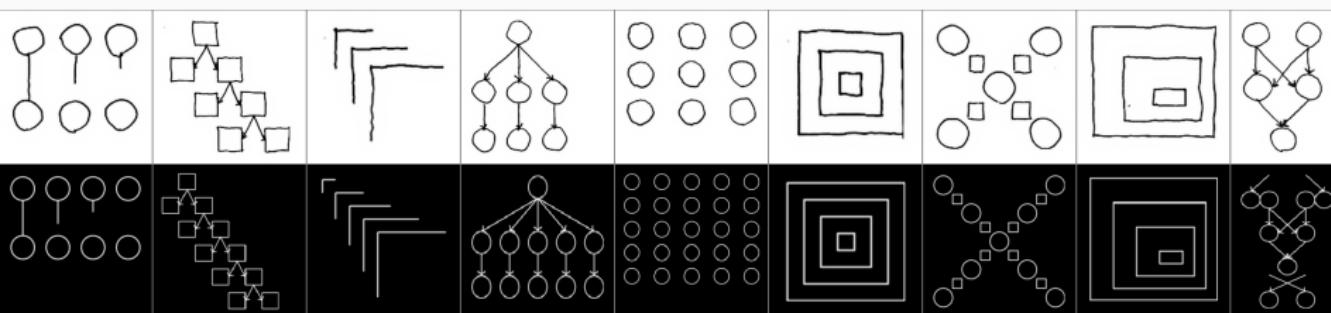
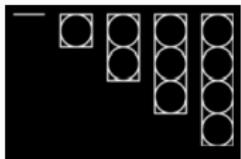
# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

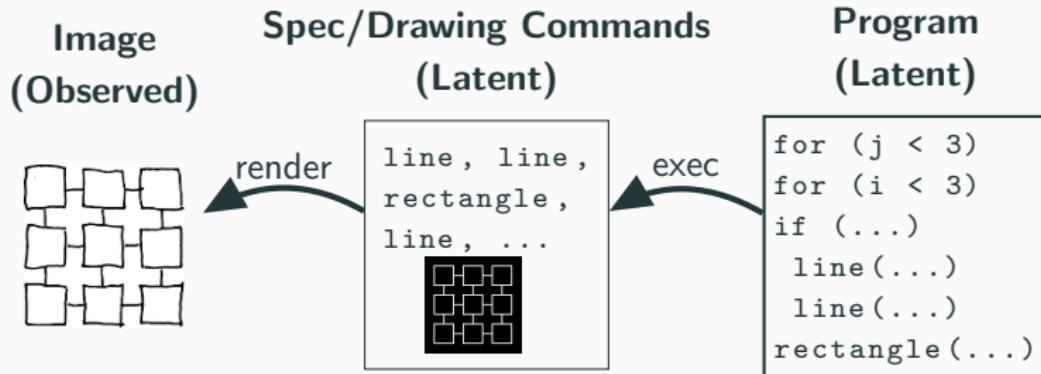
**zero-shot generalization / extrapolation**



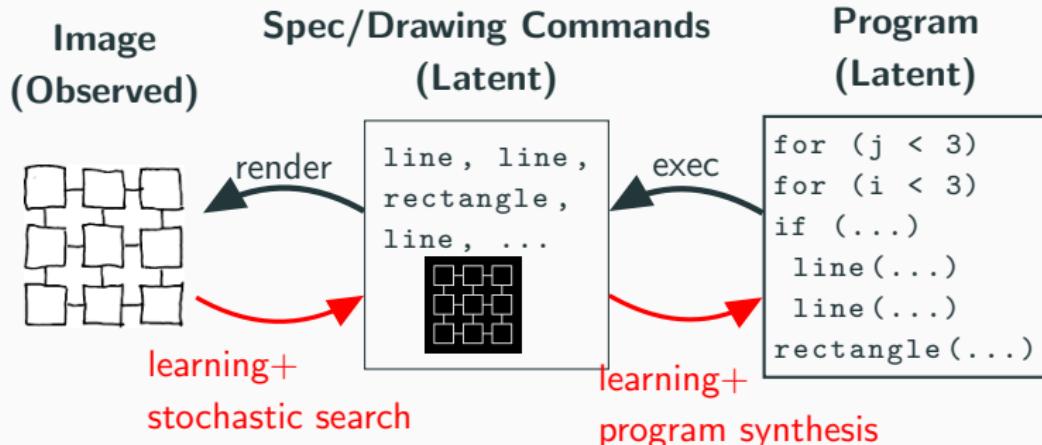
```
for (0 <= i < 3 + 1)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```



# How to infer graphics programs from hand-drawn images



# How to infer graphics programs from hand-drawn images

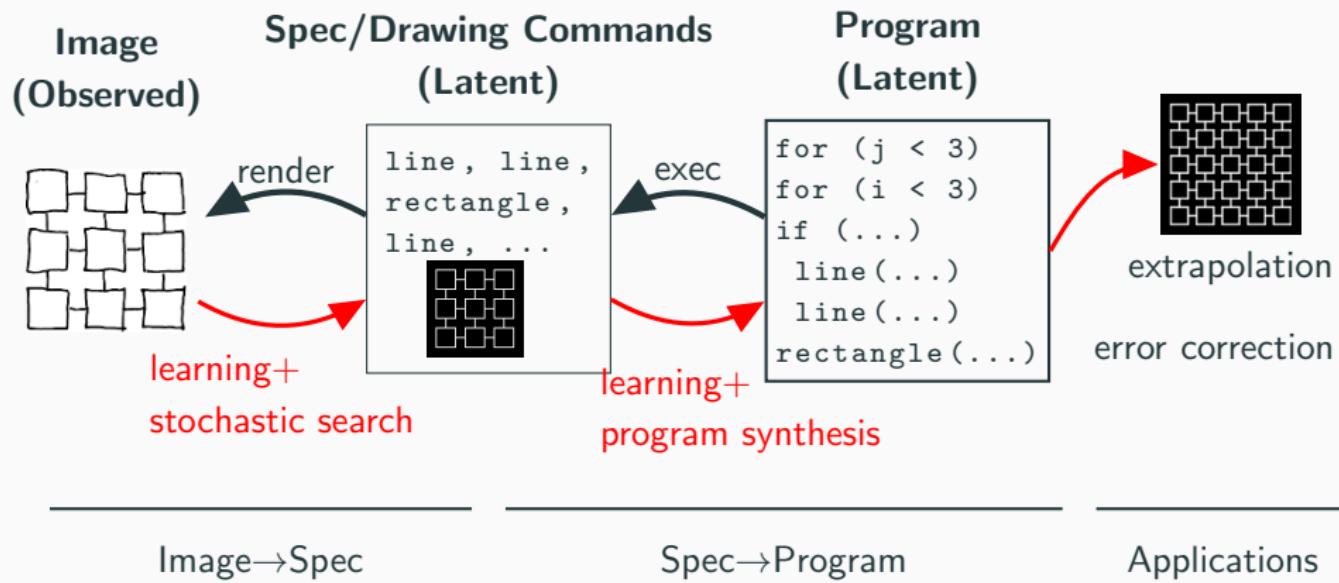


---

Image → Spec

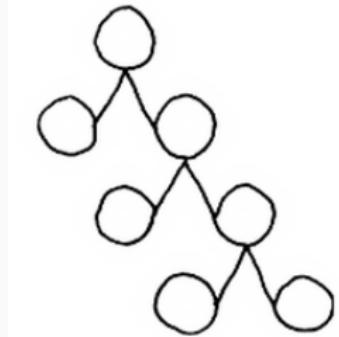
Spec → Program

# How to infer graphics programs from hand-drawn images

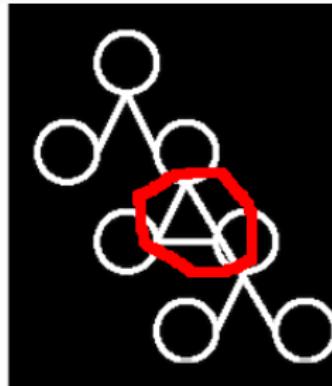


# Top-down influences on perception

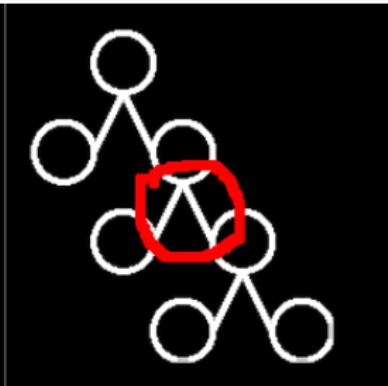
drawing



bottom-up neural net

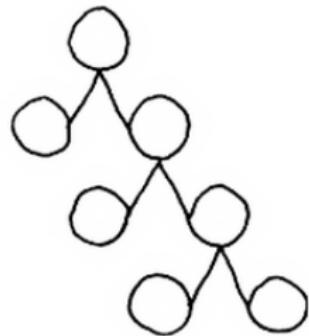


w/ top-down program bias

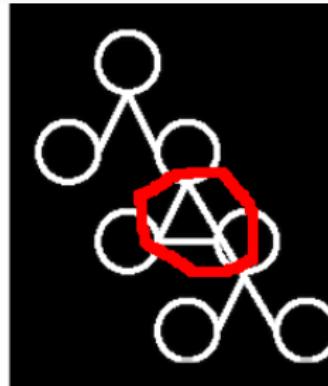


# Top-down influences on perception

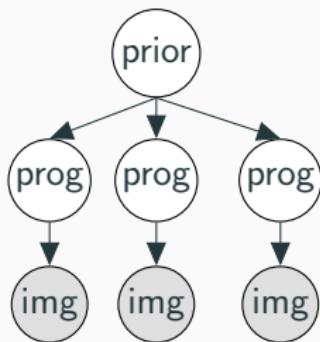
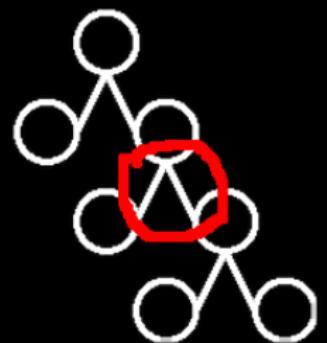
drawing



bottom-up neural net



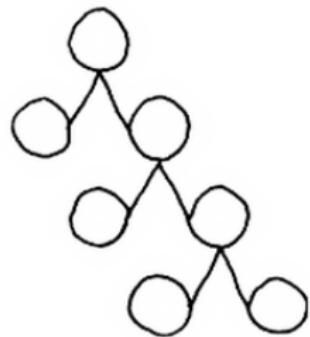
w/ top-down program bias



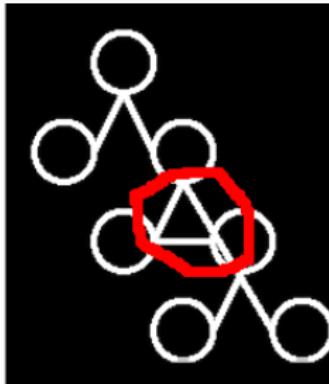
predicted program =  
 $\arg \max_{\text{progs}} \mathbb{P} [\text{img} | \text{prog}] \mathbb{P} [\text{prog} | \text{prior}]$

# Top-down influences on perception

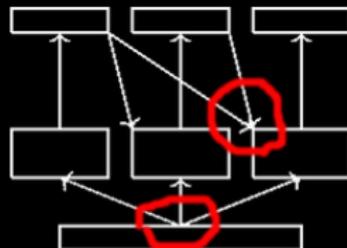
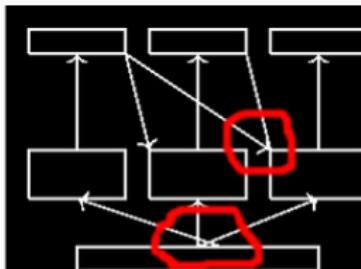
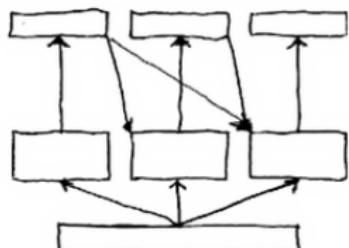
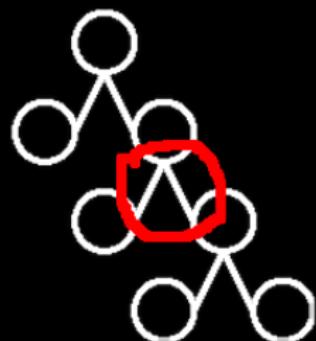
drawing



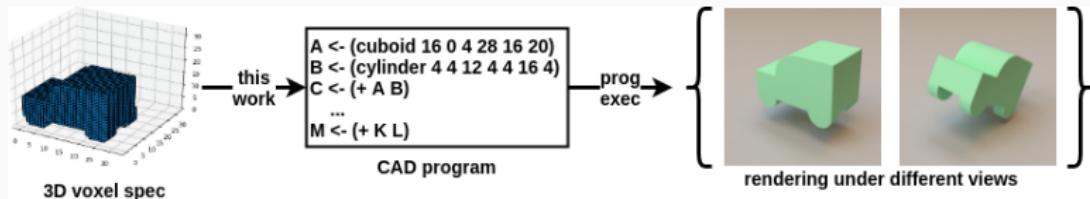
bottom-up neural net



w/ top-down program bias



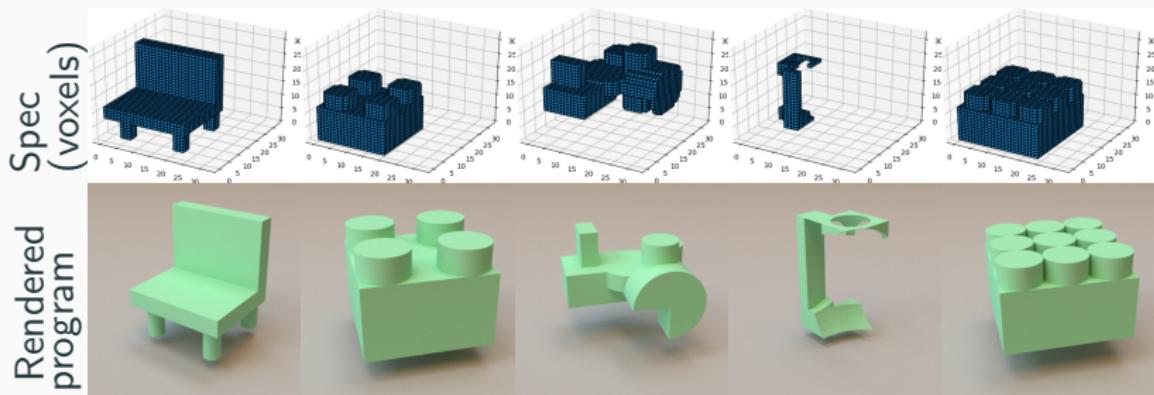
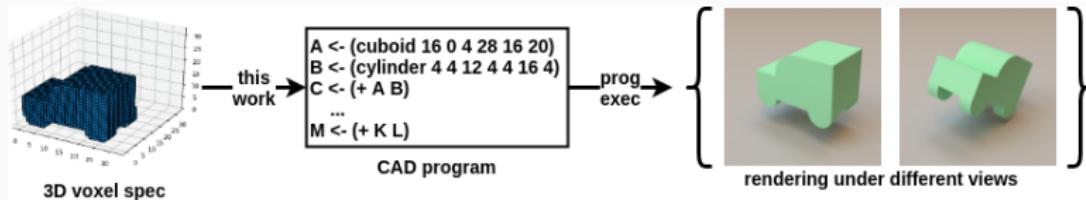
# 3D program induction



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

\*equal contribution

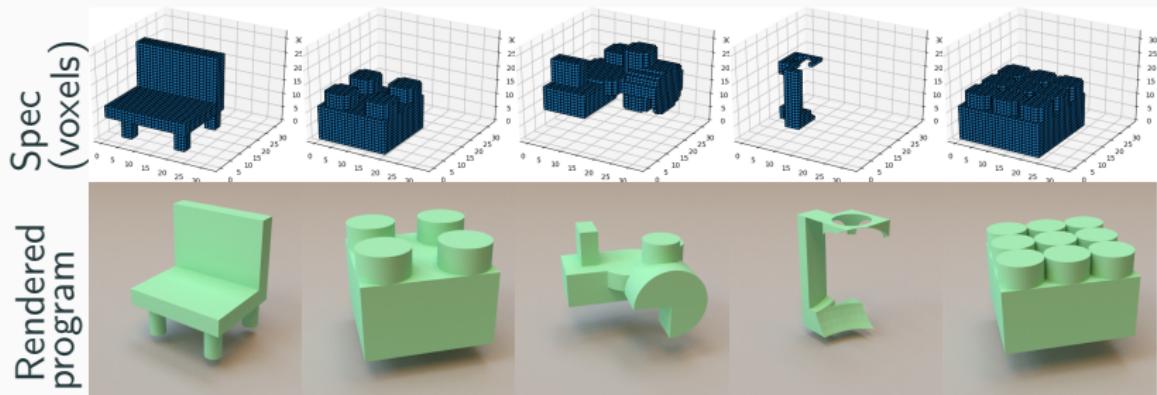
# 3D program induction



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

\*equal contribution

# 3D program induction



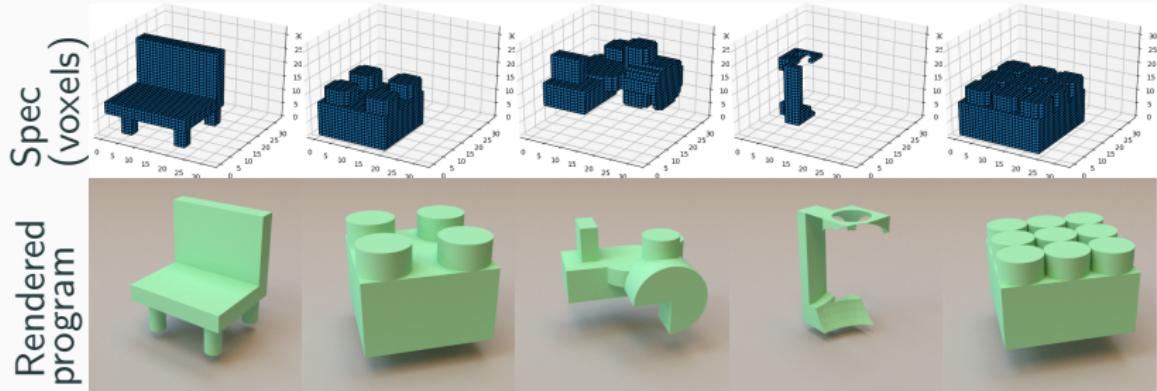
Challenge: combinatorics!

Branching factor:  $> 1.3$  million per line of code,  $\approx 20$  lines of code

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

\*equal contribution

# 3D program induction



Challenge: combinatorics!

Branching factor:  $> 1.3$  million per line of code,  $\approx 20$  lines of code

Solution: stochastic **tree search** + learn **policy** that writes code  
+ learn **value** function that assesses execution of program so far;  
analogous to AlphaGo

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

\*equal contribution

Program Induction and perception  
learning to learn  
interpretable models

## Learning to write code

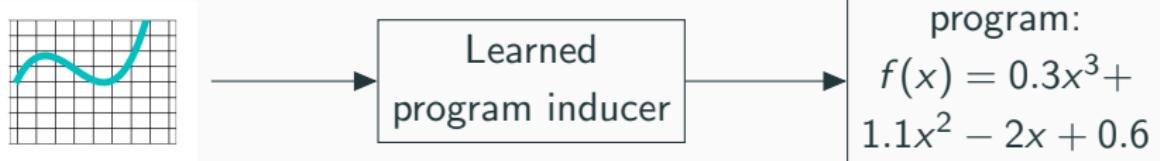
Goal: acquire domain-specific knowledge needed to induce a class of programs

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

# Learning to write code

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

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



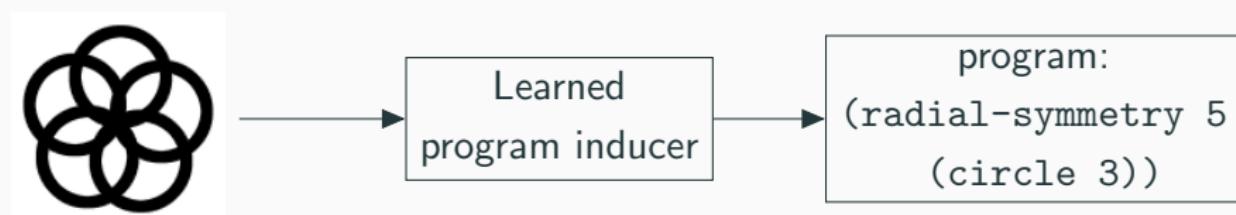
Concepts:  $x^3$ ,  $\alpha x + \beta$ , etc

Inference strategy: neurosymbolic search for programs

# Learning to write code

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

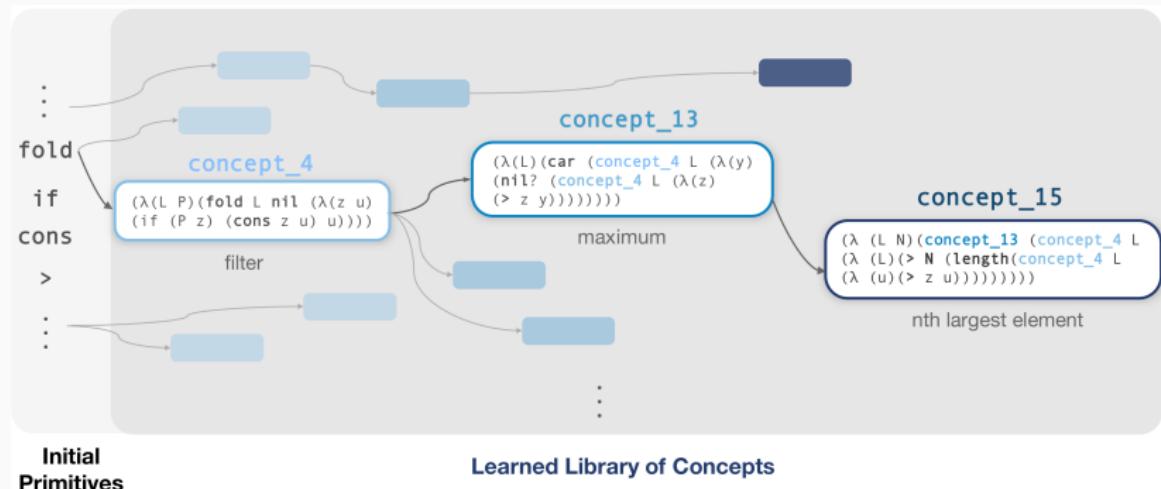
- Library of concepts (declarative knowledge; domain specific language)
- Inference strategy (procedural knowledge; synthesis algorithm)



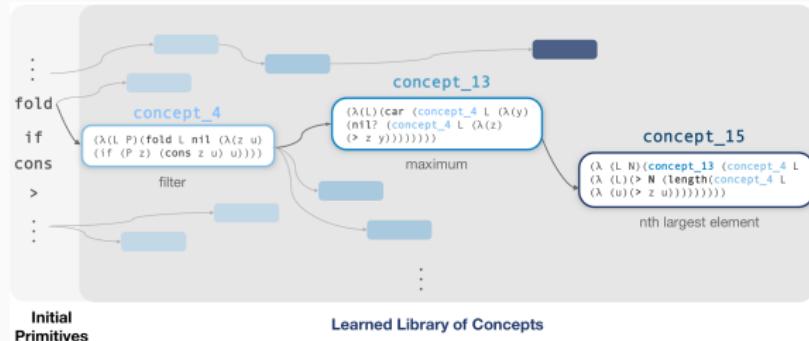
Concepts: circle, radial-symmetry, etc

Inference strategy: neurosymbolic search for programs

# Library learning



# Library learning



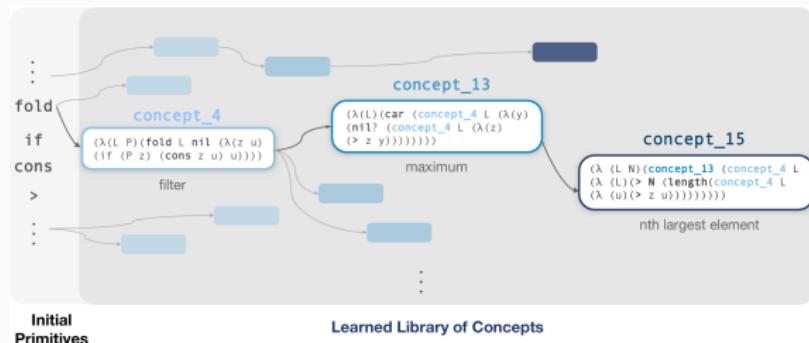
**Problem:** sort list

**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:**

```
(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

Y1775/0704  
Y2000/0101

## LOGO Graphics



Physics

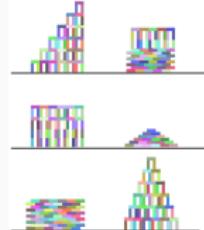
$$KE = \frac{1}{2}m|\vec{v}|^2$$

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

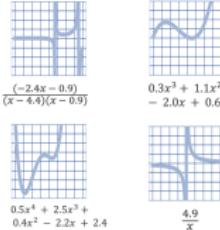
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{r}_1 - \hat{r}_2$$

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

## Block Towers



## Symbolic Regression



## Recursive Programming

### Filter

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

### Length

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

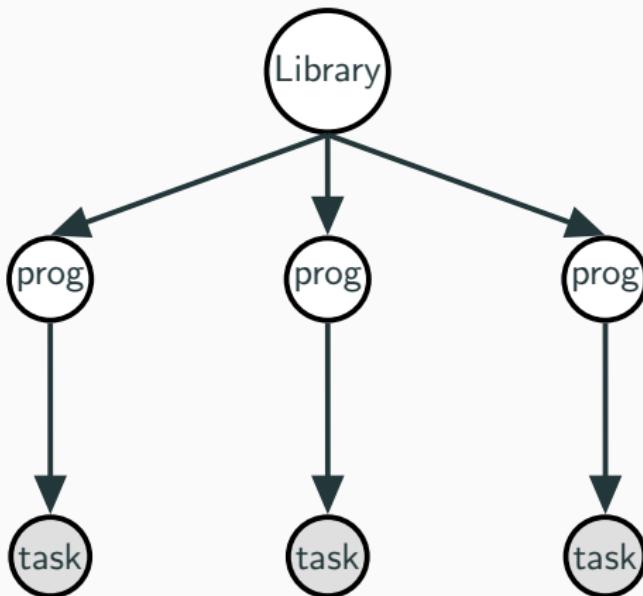
### Index List

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

### Every Other

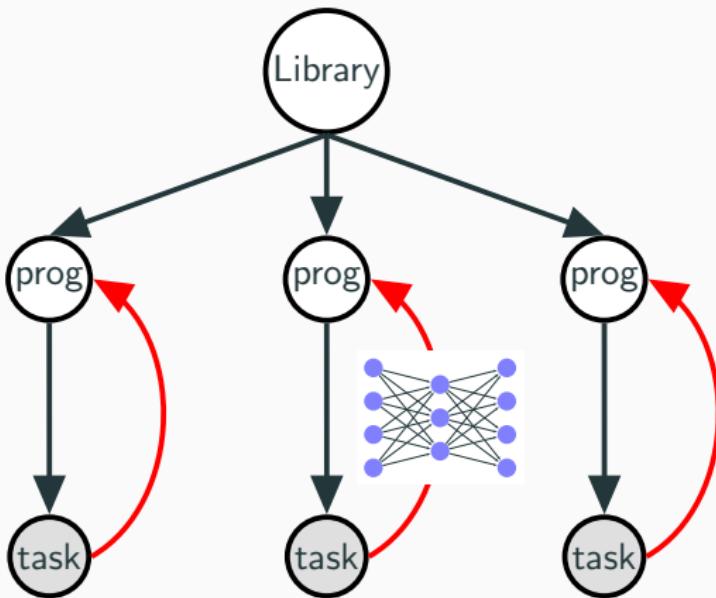
$[■■■■■■■■■] \rightarrow [■■■]$   
 $[■■■■■■■■■] \rightarrow [■■■■]$   
 $[■■■■■■■■■] \rightarrow [■■■■■]$

# Library learning as Bayesian inference

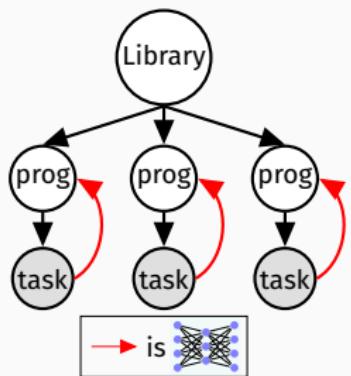


[Dechter et al., 2013] [Liang et al, 2010]; [Lake et al, 2015]

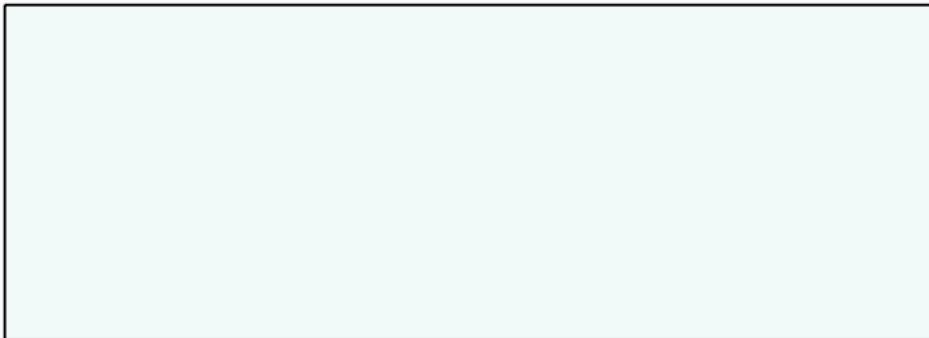
# Library learning as neurally-guided Bayesian inference



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

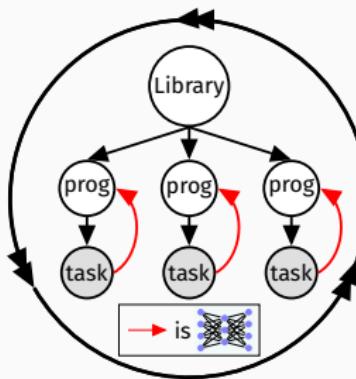


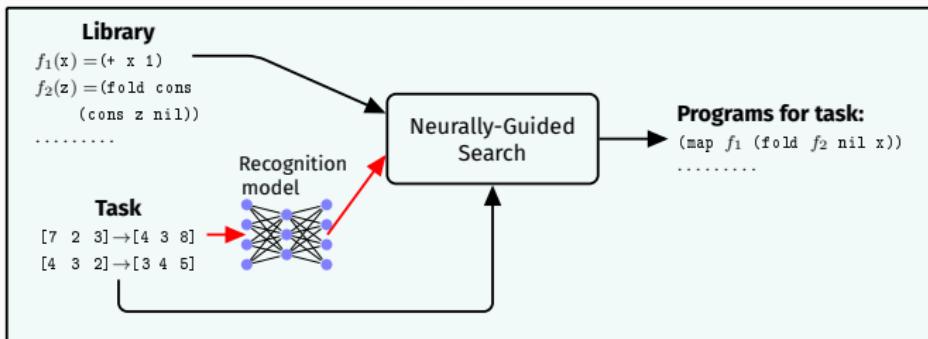
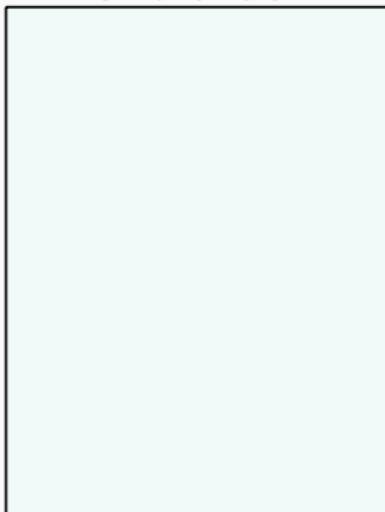
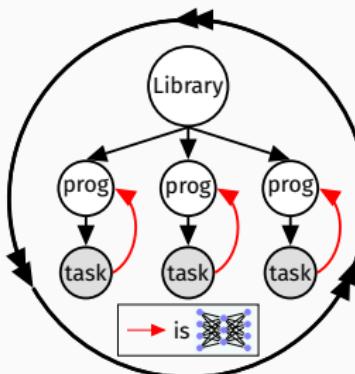
WAKE



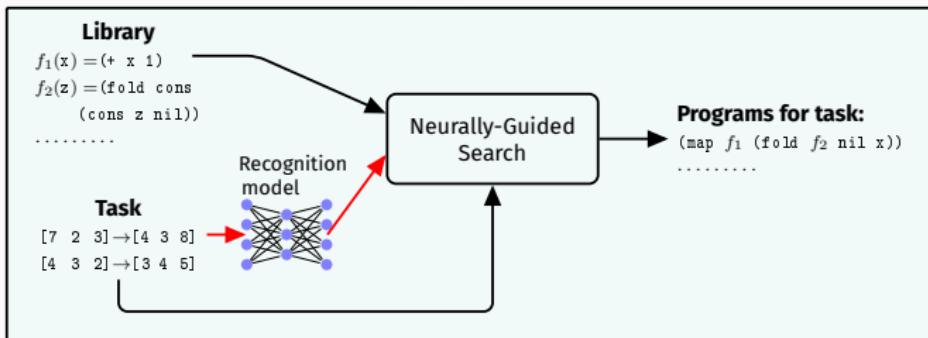
SLEEP: ABSTRACTION

SLEEP: DREAMING

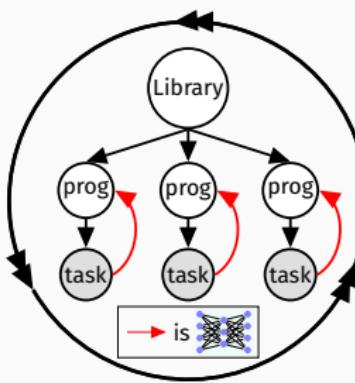


**WAKE****SLEEP: ABSTRACTION****SLEEP: DREAMING**

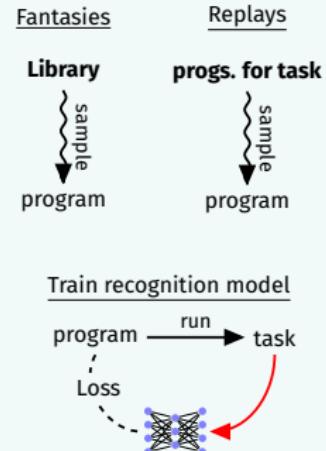
### WAKE

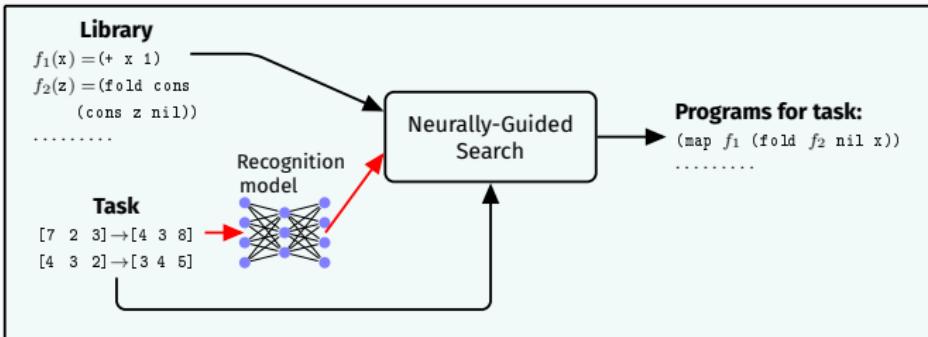
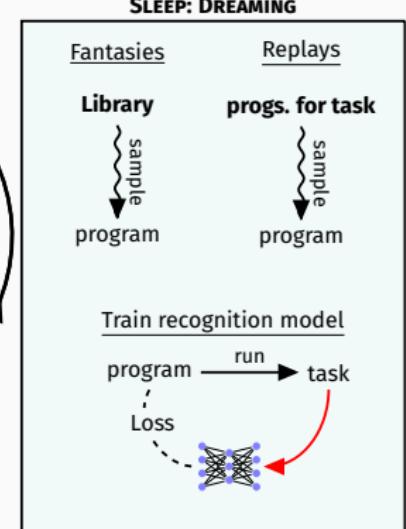
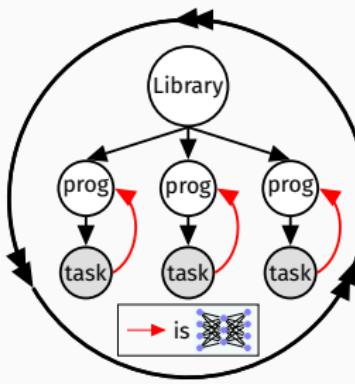
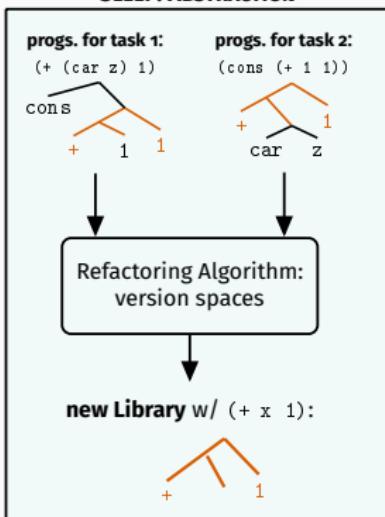


### SLEEP: ABSTRACTION



### SLEEP: DREAMING



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

# Abstraction Sleep: Growing the library via refactoring

$5 + 5$

## Abstraction Sleep: Growing the library via refactoring

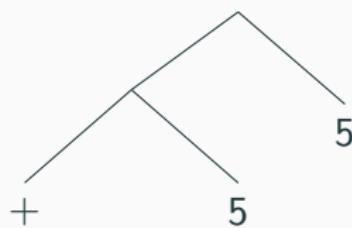
$5 + 5$

(+ 5 5)

## Abstraction Sleep: Growing the library via refactoring

$5 + 5$

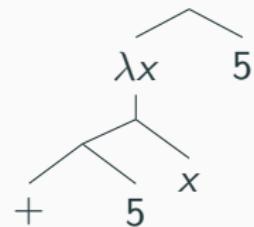
(+ 5 5)



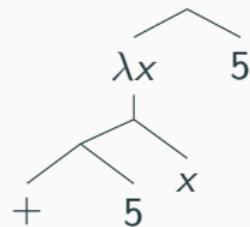
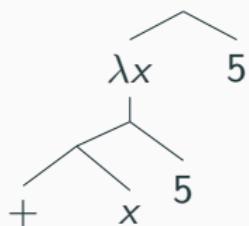
# Abstraction Sleep: Growing the library via refactoring



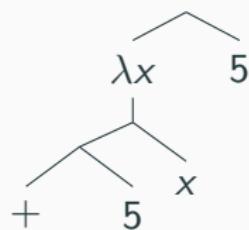
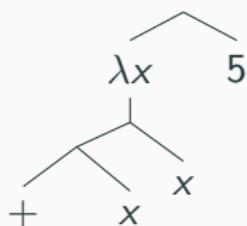
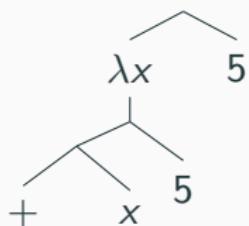
# Abstraction Sleep: Growing the library via refactoring



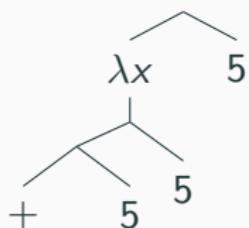
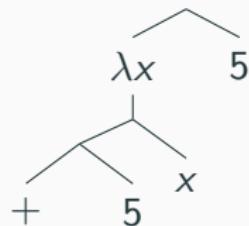
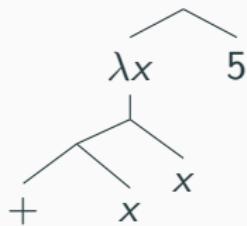
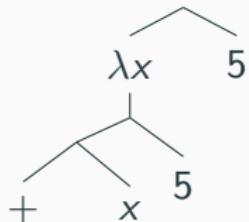
## Abstraction Sleep: Growing the library via refactoring



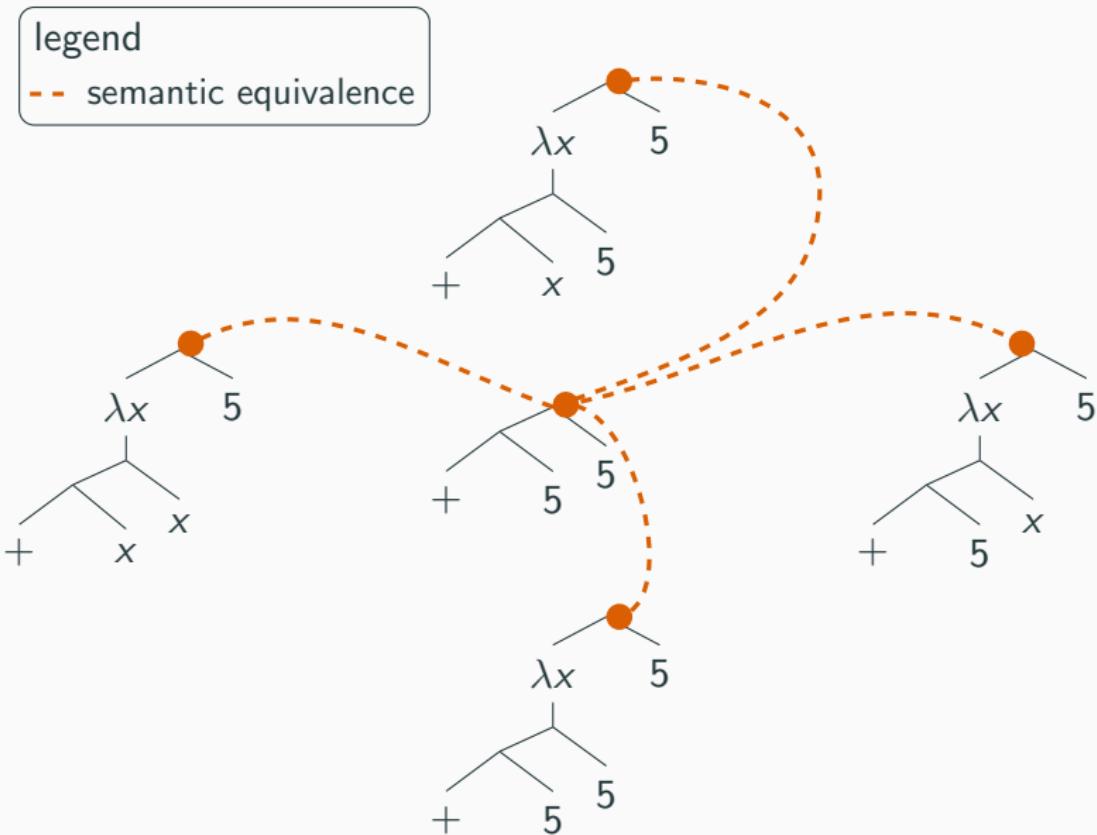
# Abstraction Sleep: Growing the library via refactoring



# Abstraction Sleep: Growing the library via refactoring



# Abstraction Sleep: Growing the library via refactoring

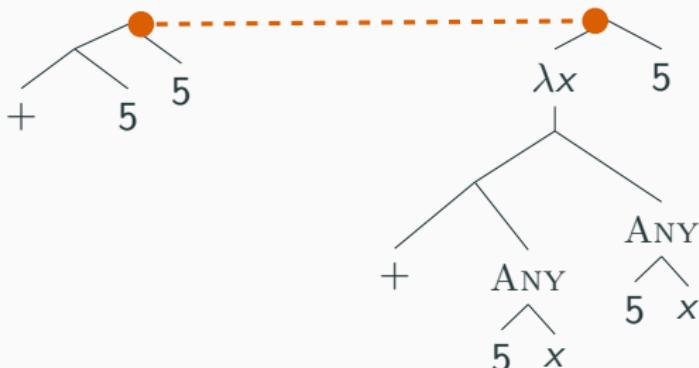


# Abstraction Sleep: Growing the library via refactoring

legend

— semantic equivalence

ANY nondeterministic choice

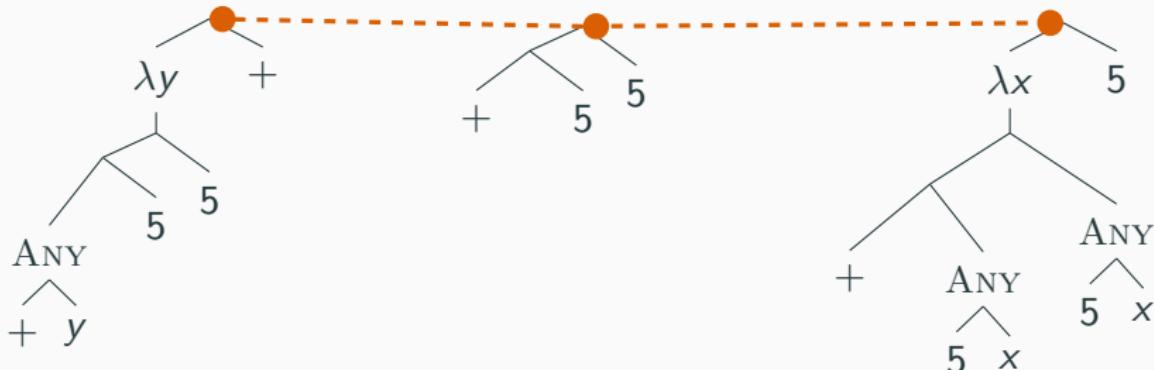


# Abstraction Sleep: Growing the library via refactoring

legend

— semantic equivalence

ANY nondeterministic choice

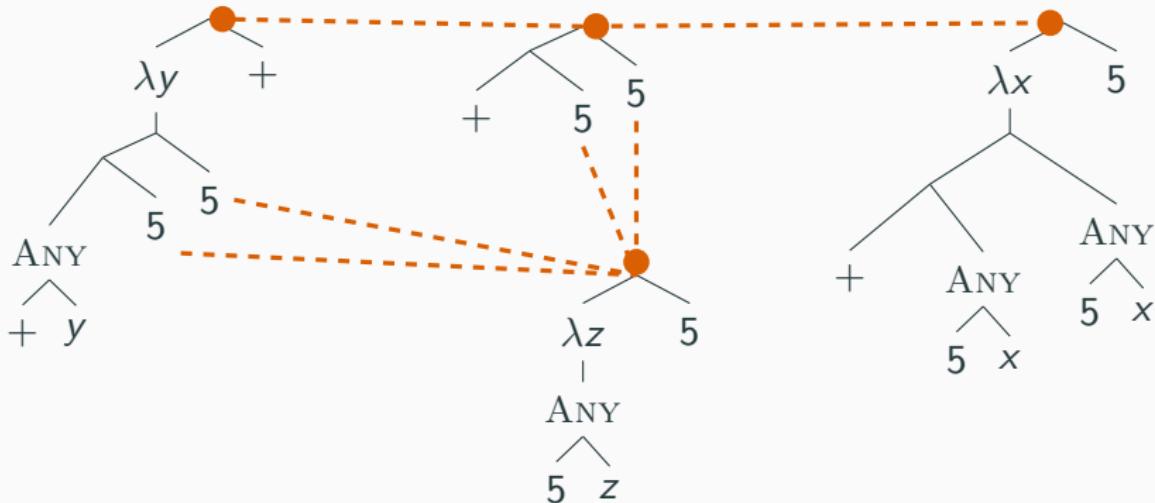


# Abstraction Sleep: Growing the library via refactoring

legend

— semantic equivalence

ANY nondeterministic choice

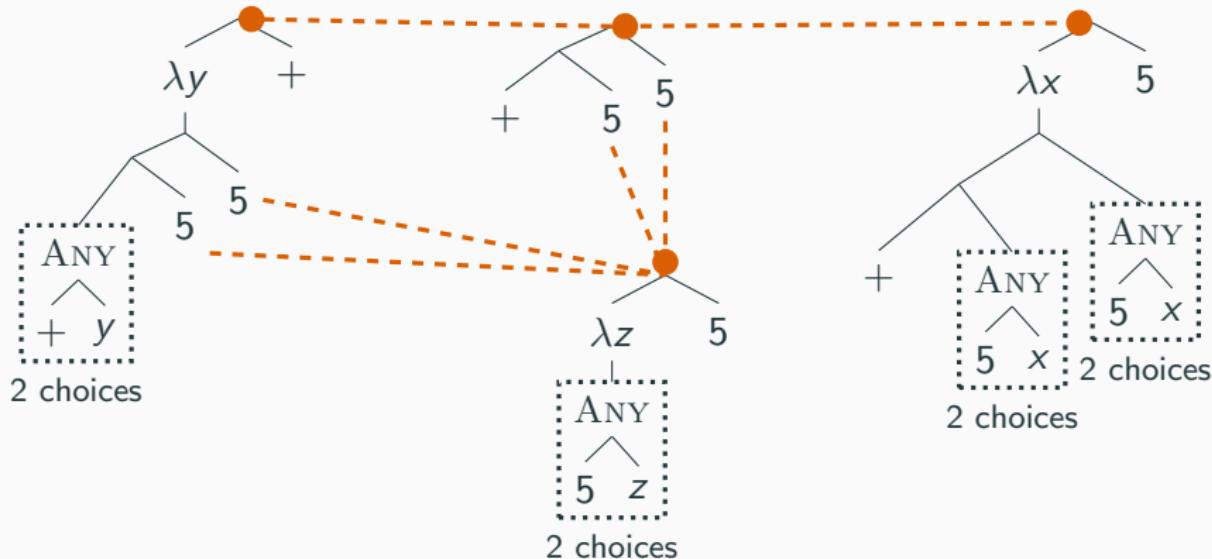


# Abstraction Sleep: Growing the library via refactoring

legend

— semantic equivalence

ANY nondeterministic choice

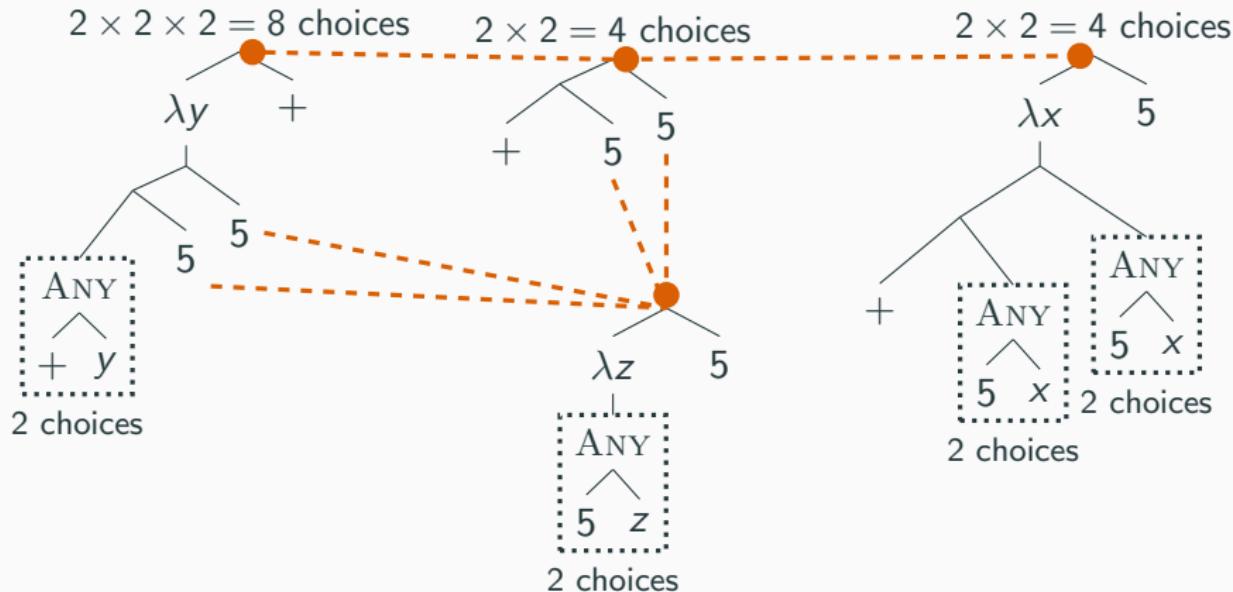


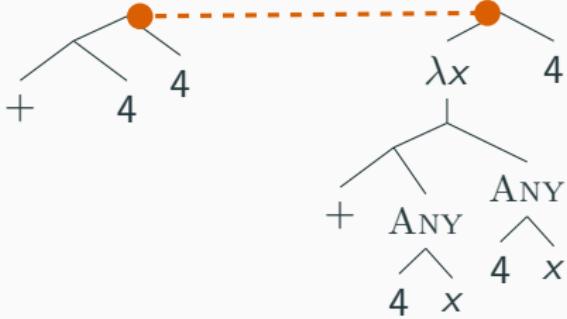
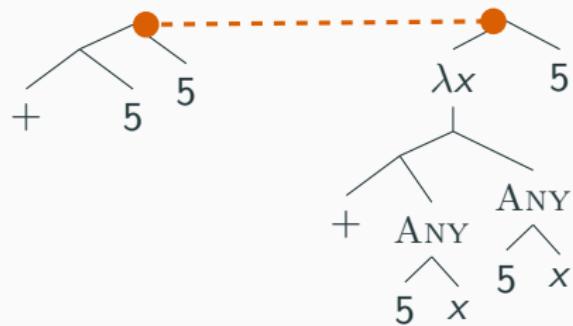
# Abstraction Sleep: Growing the library via refactoring

legend

— semantic equivalence

ANY nondeterministic choice

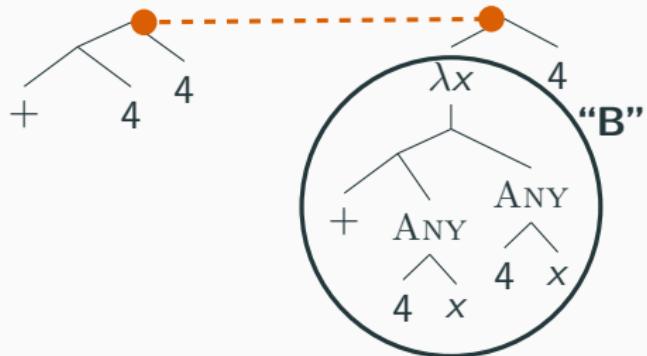
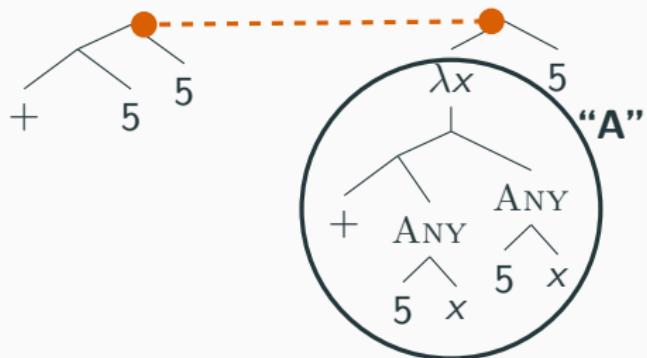




### legend

— semantic equivalence

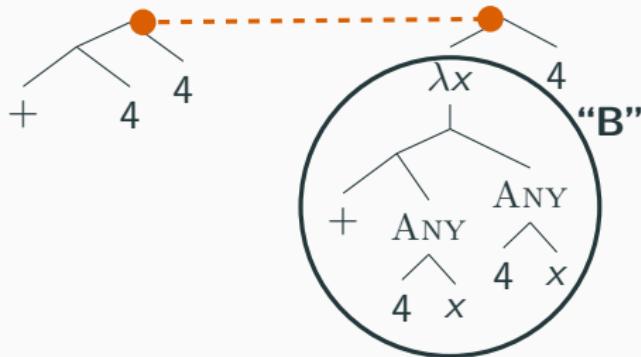
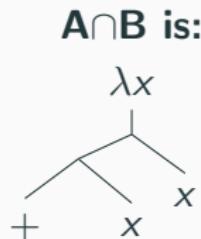
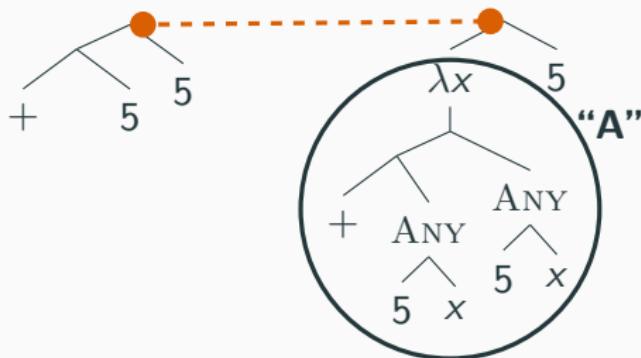
ANY nondeterministic choice



legend

— semantic equivalence

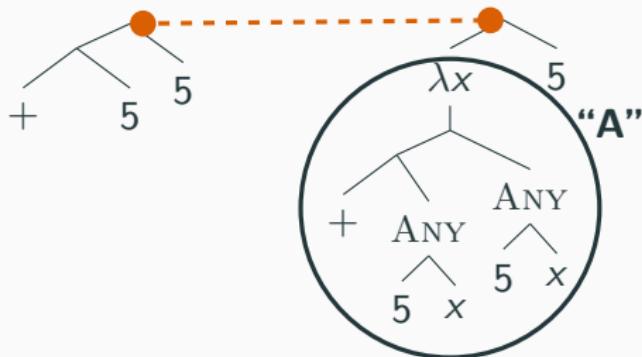
ANY nondeterministic choice



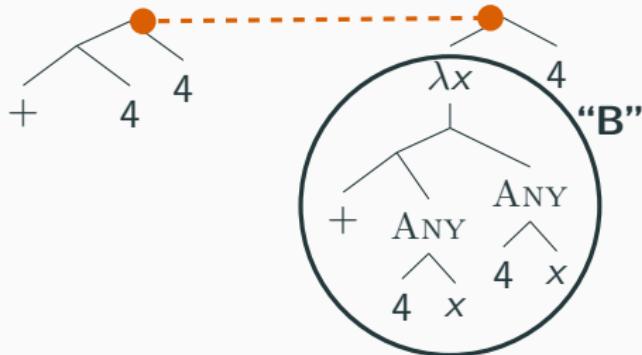
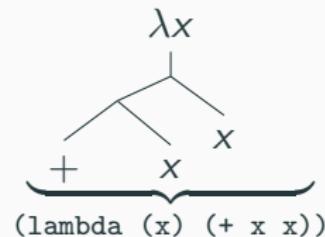
legend

— semantic equivalence

ANY nondeterministic choice



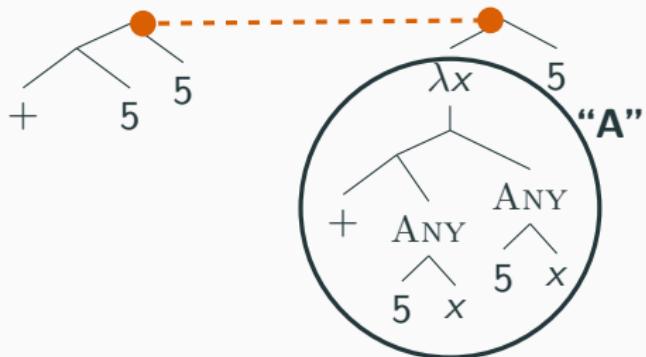
$A \cap B$  is:



legend

— semantic equivalence

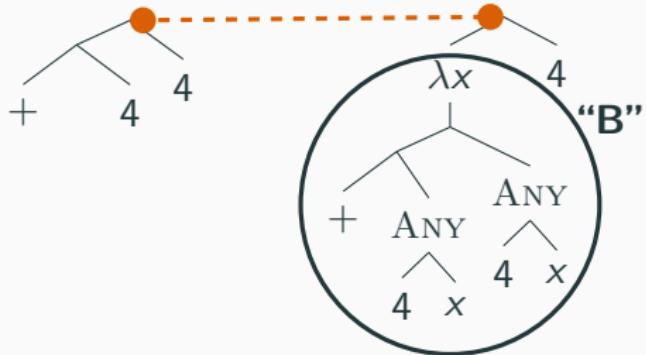
ANY nondeterministic choice



$A \cap B$  is:

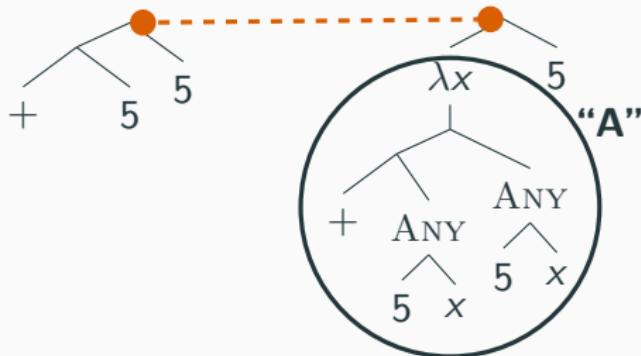
$(\lambda x \quad (+ x x))$

=double

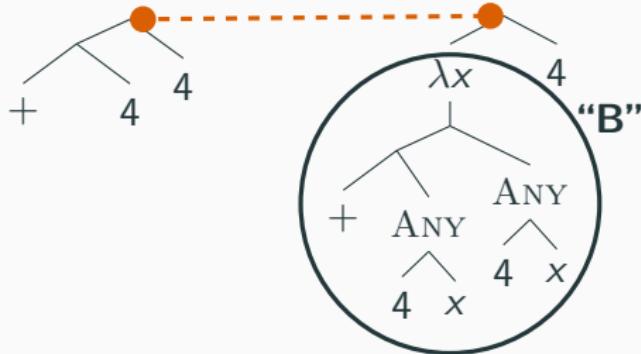
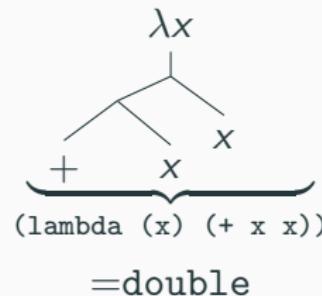


legend

- semantic equivalence
- ANY nondeterministic choice



$A \cap B$  is:



w/o double	w/ double
$(+ 5 5)$	$(double 5)$
$(+ 4 4)$	$(double 4)$
$(+ 3 3)$	$(double 3)$
...	

legend

— 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)

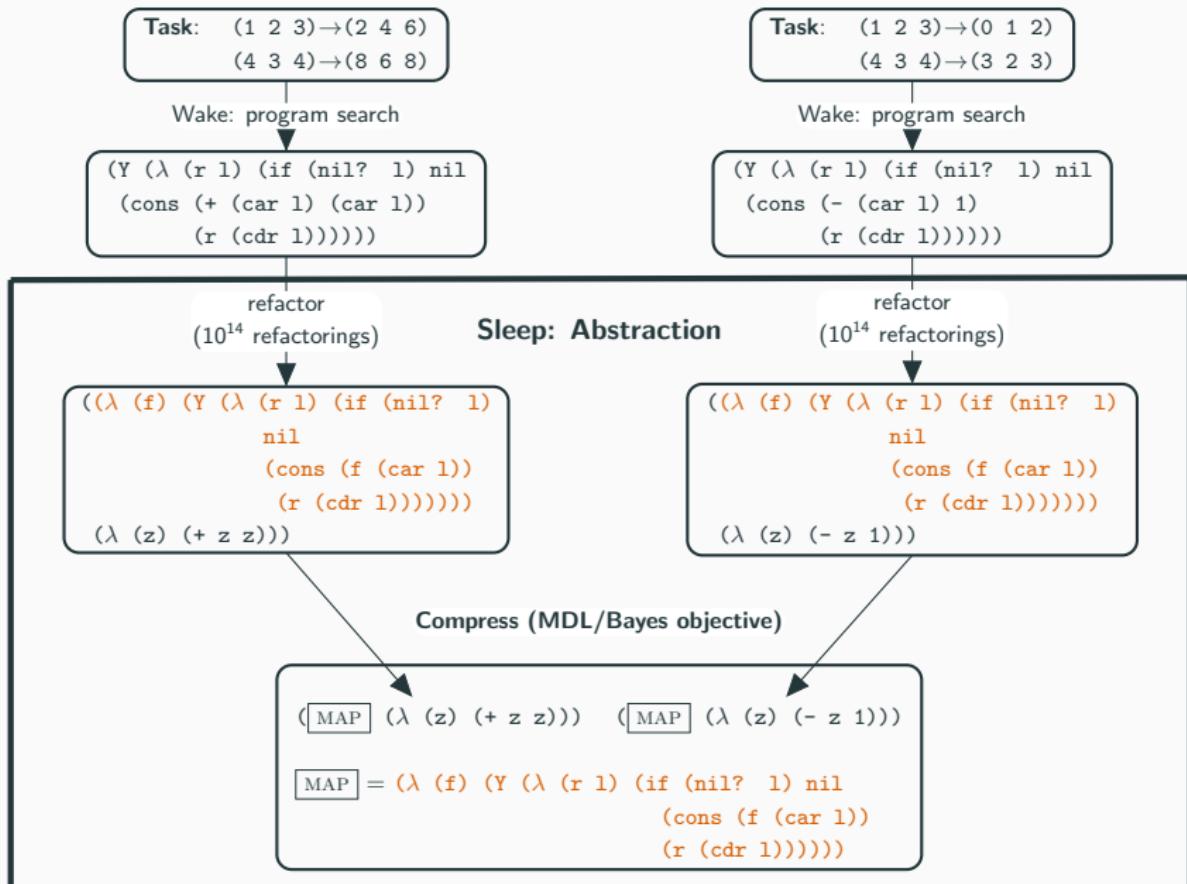
## Sleep: Abstraction

refactor  
 $(10^{14}$  refactorings)

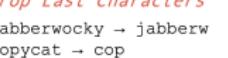
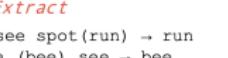
```
((λ (f) (Y (λ (r 1) (if (nil? 1)  
                           nil  
                           (cons (f (car 1))  
                                 (r (cdr 1)))))))  
  (λ (z) (+ z z)))
```

```
((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



# 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>Physics</b> $KE = \frac{1}{2} m  \vec{v} ^2$ $\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$
	 $0.5x^4 + 2.5x^3 + 0.4x^2 - 2.2x + 2.4$ $\frac{4.9}{x}$	<b>Length</b> [■■■■■] → 4 [■■■■■■] → 6 [■■■■] → 3	<b>Index List</b> 0, [■■■■■■■] → ■ 1, [■■■■■■■] → ■■ 1, [■■■■■■■■] → ■■■
		<b>Every Other</b> [■■■■■] → [■■] [■■■■■■] → [■■■■] [■■■■■■■] → [■■■■■]	$\vec{F} \propto \frac{q_1 q_2}{ \vec{r}'_1 - \vec{r}'_2 ^2} \hat{\vec{r}}_1 - \hat{\vec{r}}_2$ $R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$

# 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

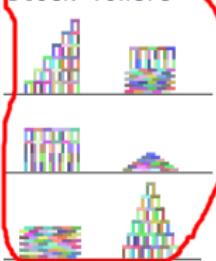
Y1775/0704

Y2000/0101

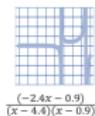
## LOGO Graphics



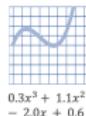
## Block Towers



## Symbolic Regression



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



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

## Recursive Programming

### Filter

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

### Length

[■■■■] → 4  
[■■■■■] → 6  
[■■■] → 3

### Index List

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

### Every Other

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

## Physics

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

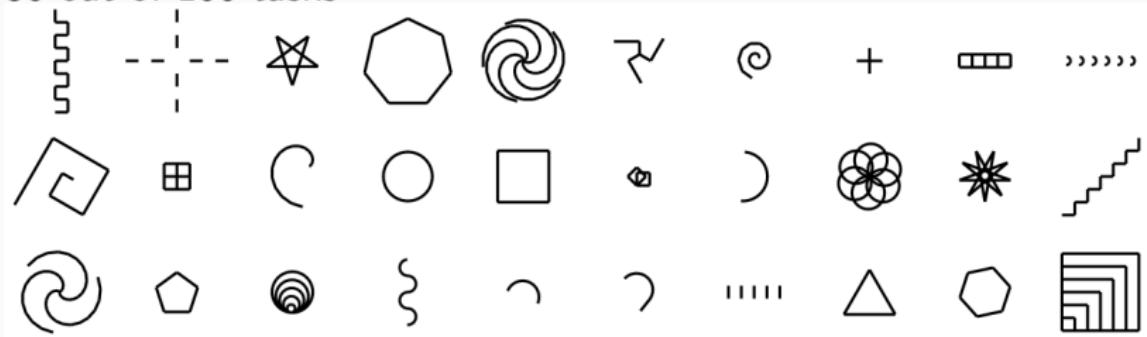
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

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

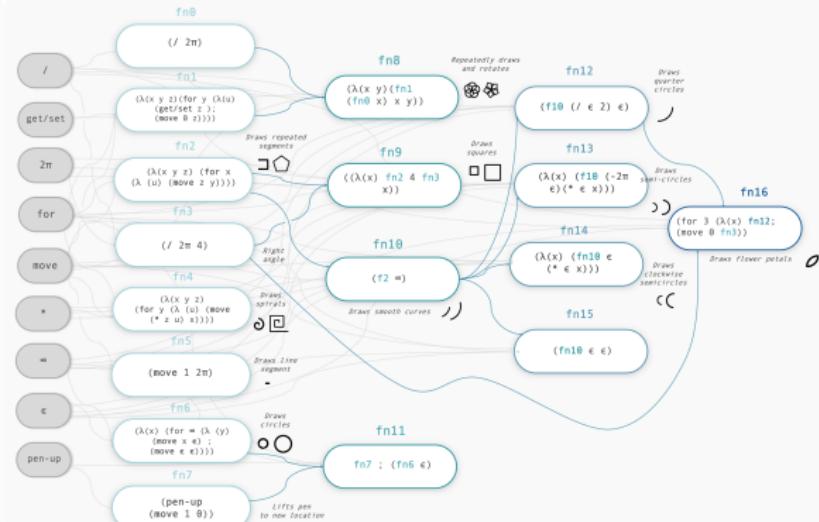
$$V_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$$

# LOGO Graphics

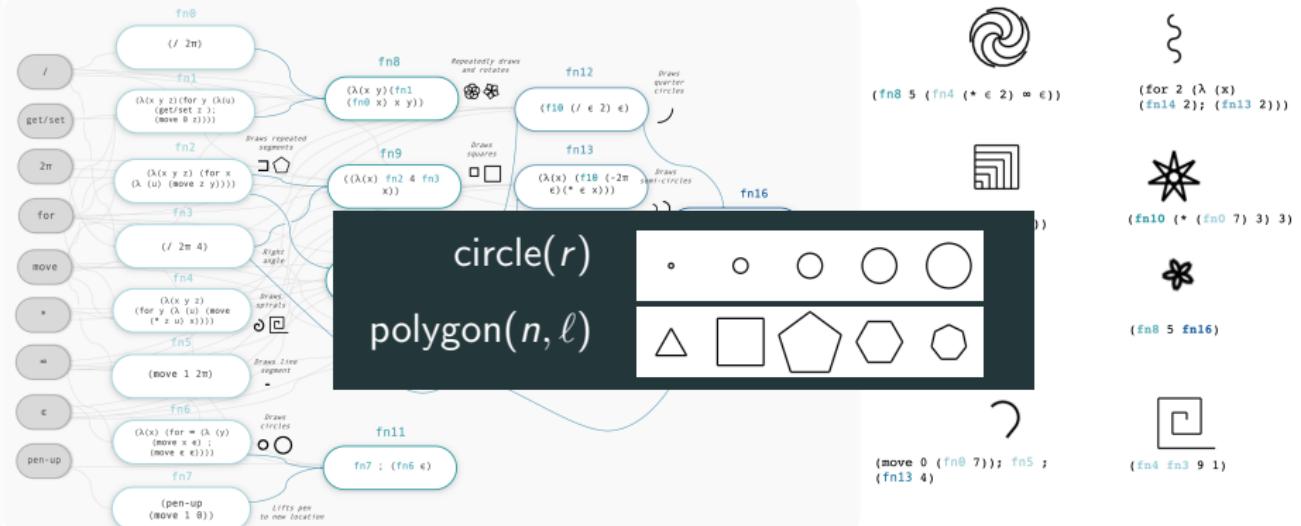
30 out of 160 tasks



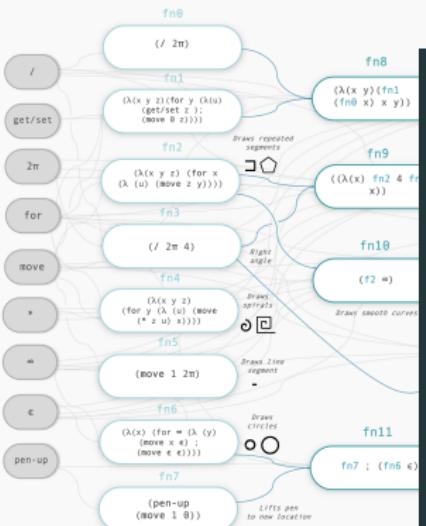
# LOGO Graphics – learning interpretable library of concepts



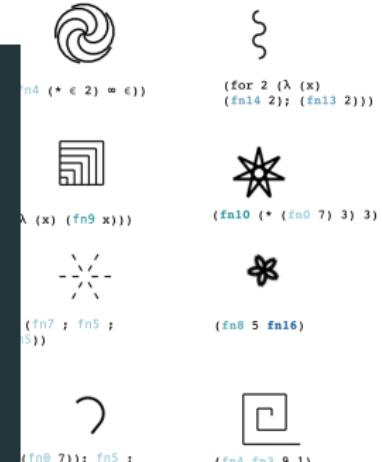
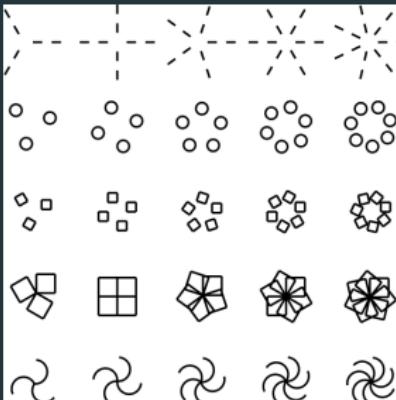
# LOGO Graphics – learning interpretable library of concepts



# LOGO Graphics – learning interpretable library of concepts



radial symmetry( $n$ , body)

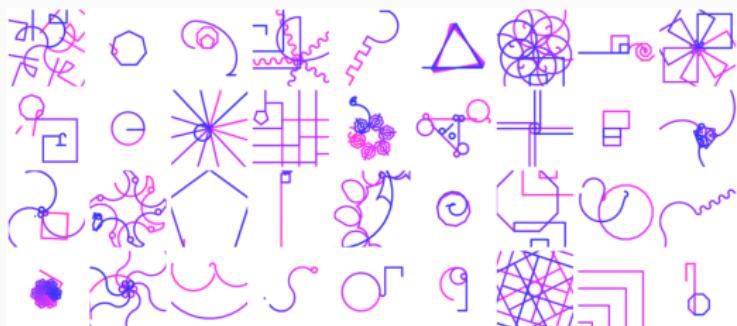


# what does DreamCoder dream of?

before learning

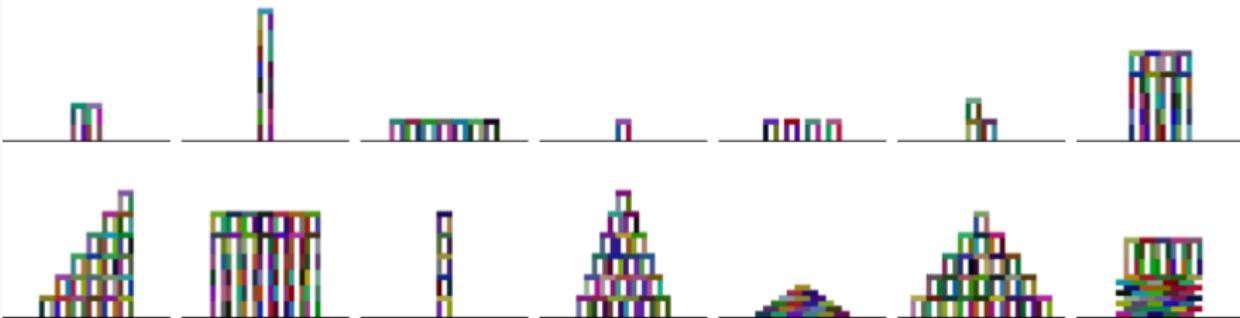


after learning



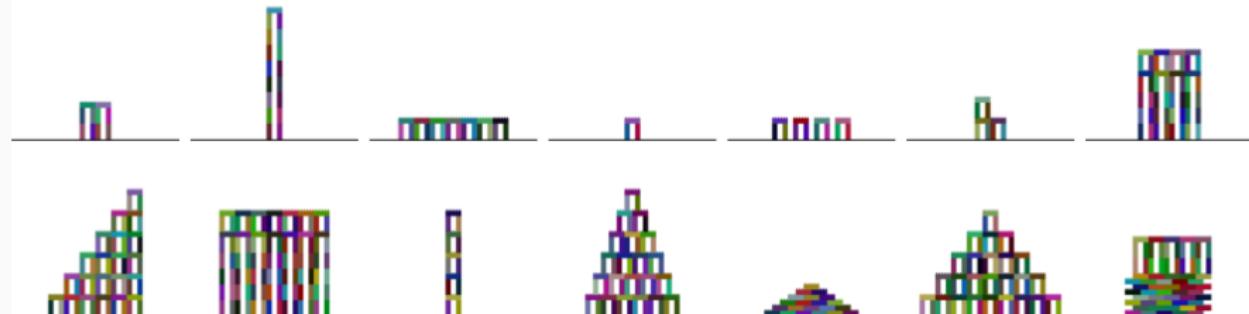
# Planning to build towers

example tasks (112 total)



# Planning to build towers

example tasks (112 total)



learned library routines ( $\approx 20$  total)

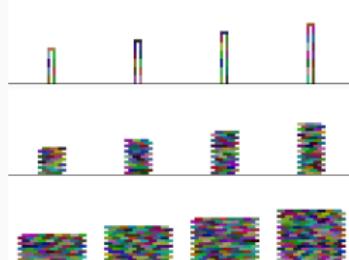
$\text{arch}(h)$



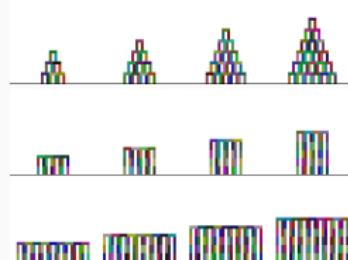
$\text{pyramid}(h)$



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

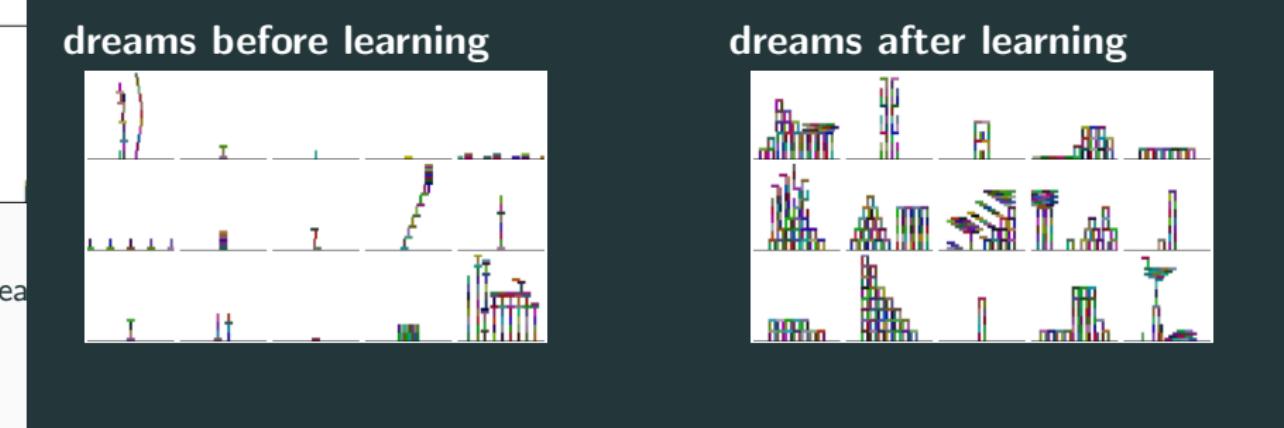


$\text{bridge}(w, h)$

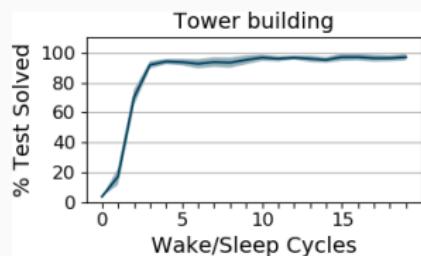


# Planning to build towers

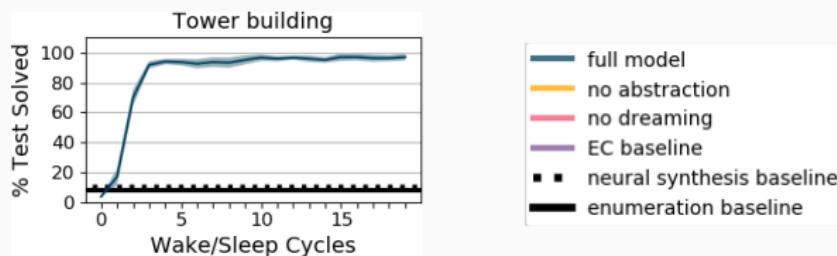
example tasks (112 total)



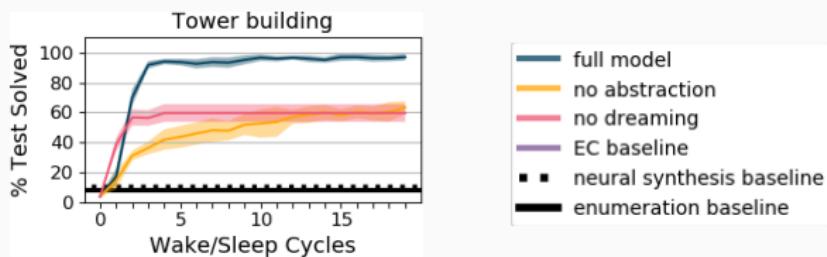
# Synergy between dreaming and library learning



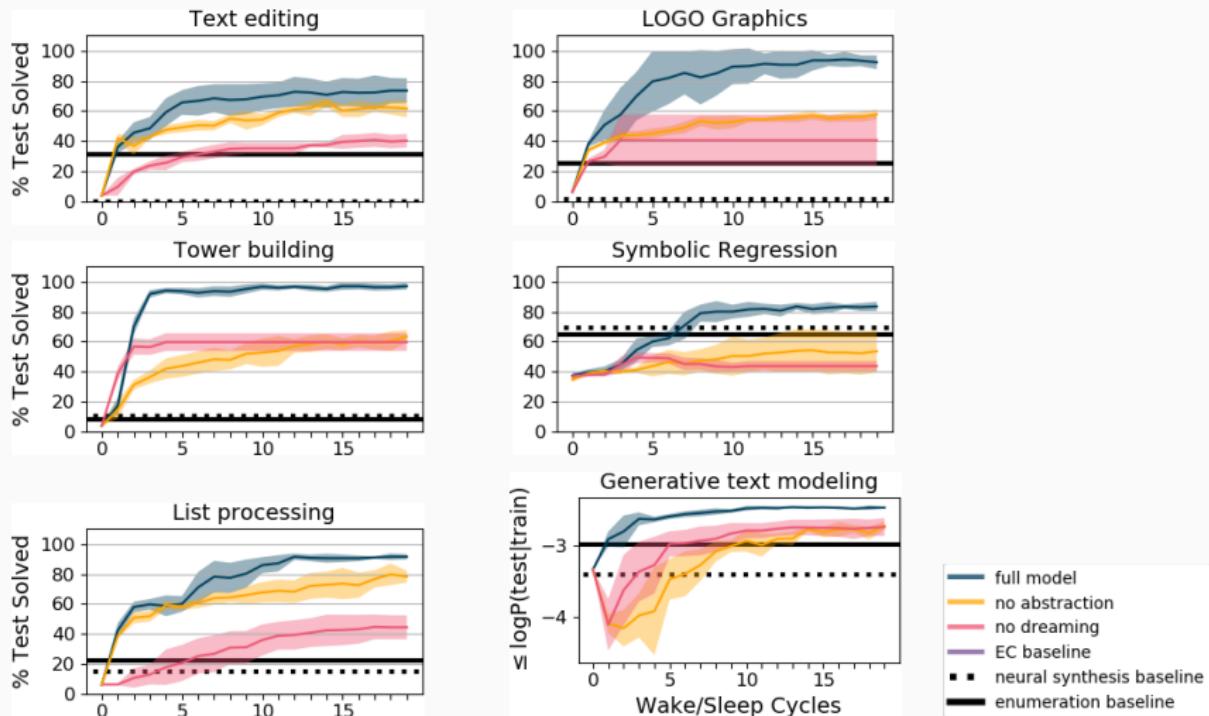
# Synergy between dreaming and library learning



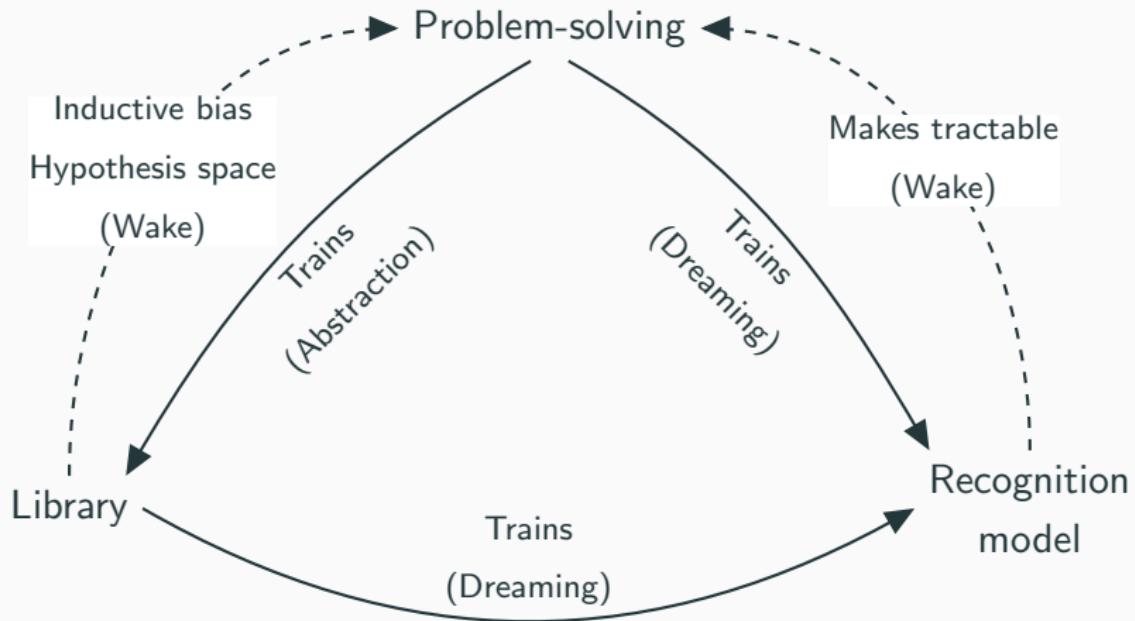
# Synergy between dreaming and library learning



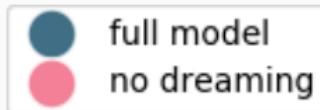
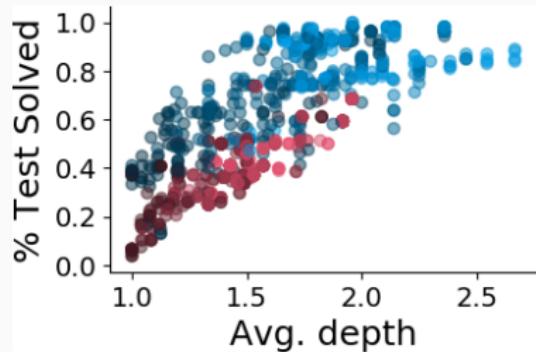
# Synergy between dreaming and library learning



# synergy between dreaming and library learning



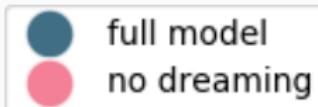
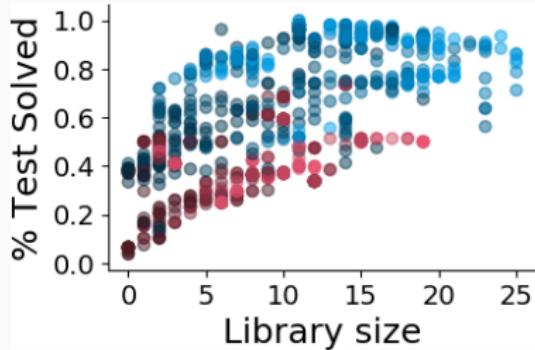
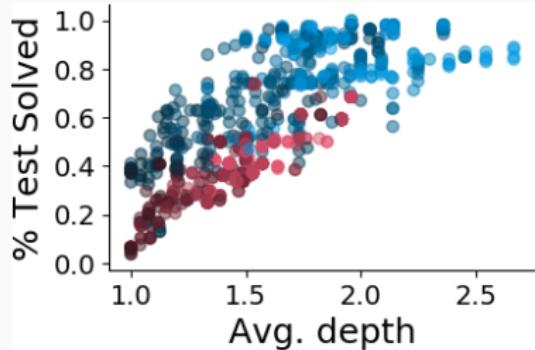
# Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Lighter: Later in learning

# Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

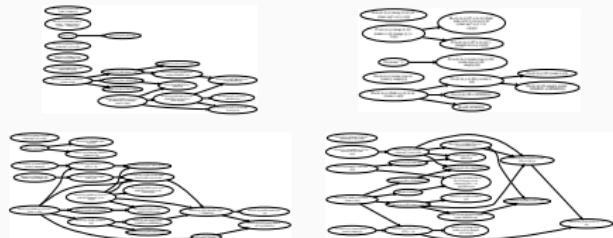
Lighter: Later in learning

# Variability in learned library

List processing



Text editing



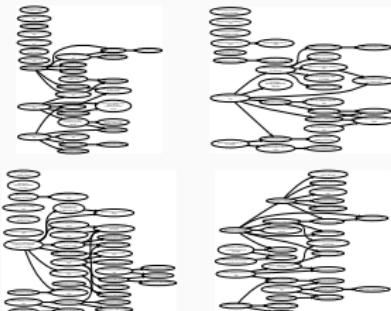
Tower building



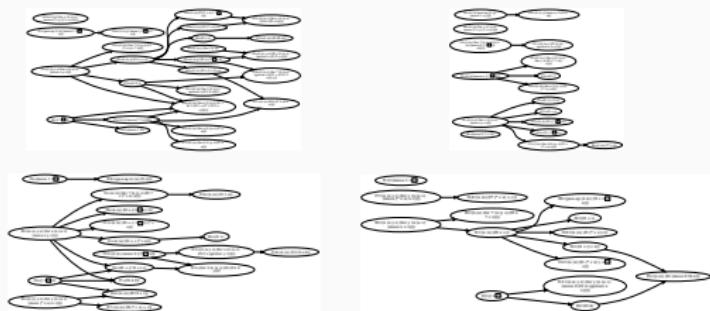
Symbolic regression



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

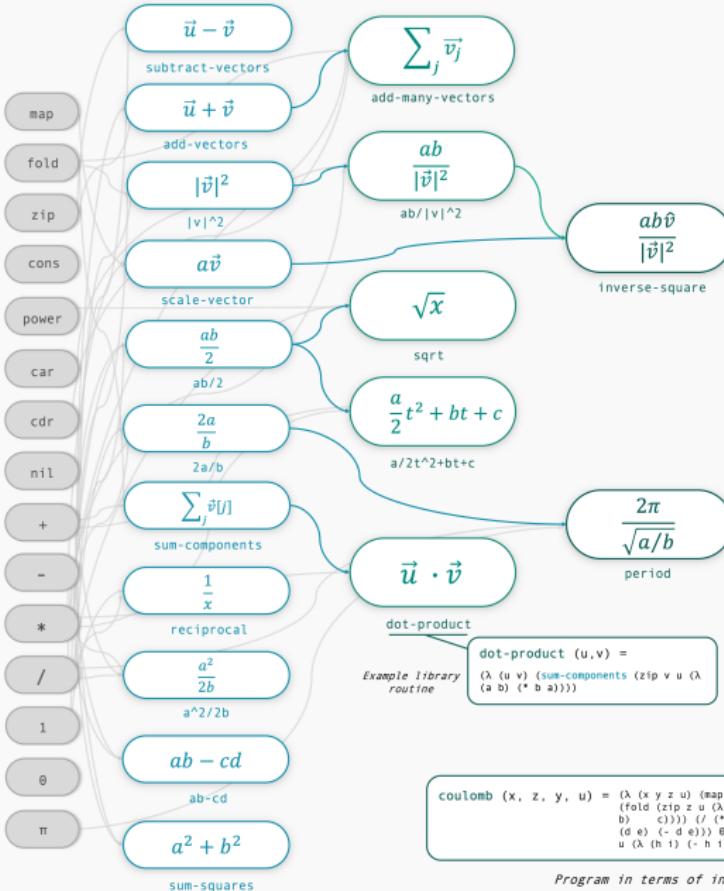
these experiments study how DreamCoder grows from a “beginner” state to “expert”:

- “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))

Parallel Resistors

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

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

Kinetic Energy

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

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

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 v (reciprocal 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

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

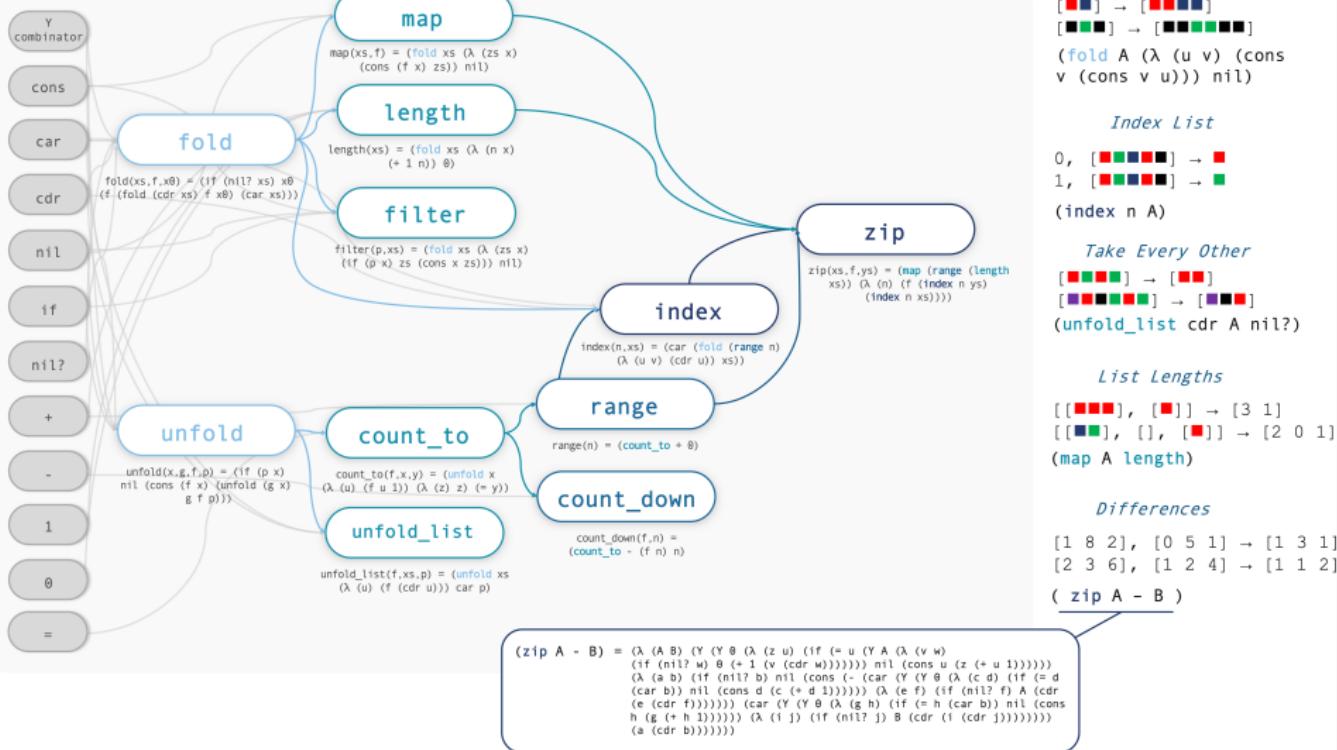
(inverse-square q\_1 q\_2 (subtract-vectors r\_1 r\_2))

Center of Mass

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

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

# Rediscovering origami programming

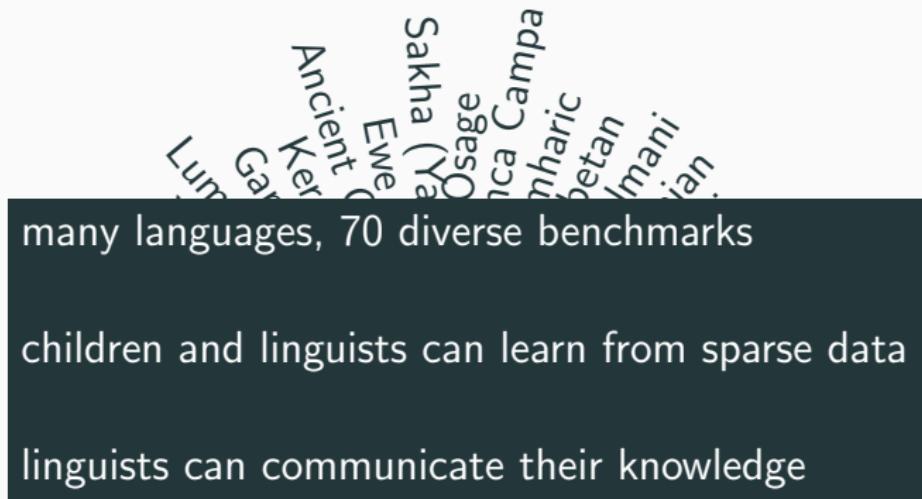


Program Induction and perception  
learning to learn  
interpretable models

# Synthesizing human-understandable models of language



# Synthesizing human-understandable models of language



# 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

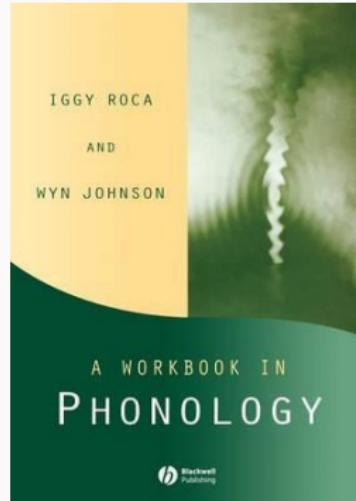
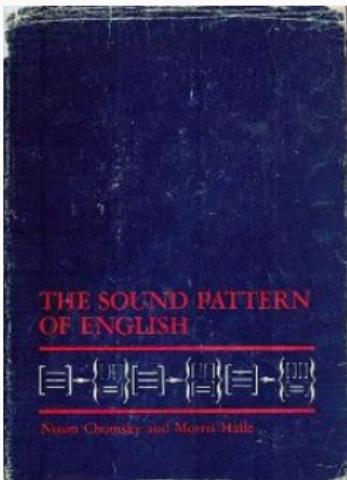
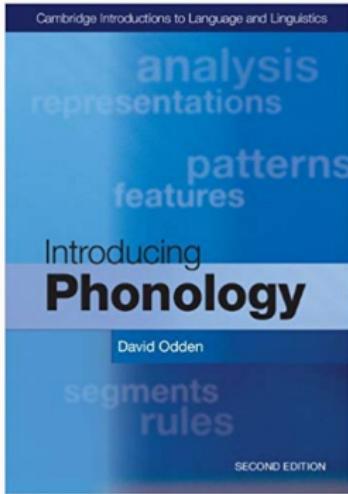
Serbo-Croatian:

	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*

$\emptyset \rightarrow a / C_C\#$



## 10 Sakha (Yakut)

Give a phonological analysis of the following case-marking paradigms of nouns in Sakha.

<i>Noun</i>	<i>Plural</i>	<i>Associative</i>	<i>oyuur</i>	<i>oyurdar</i>	<i>oyurduun</i>	<i>'forest'</i>	
aýa	aýalar	aýaliin	'father'	üçügey	üçügeyder	'good person'	
paarta	paartalar	paartaliin	'school desk'	ejiy	ejiyder	'elder sister'	
tia	tiallar	tialliin	'forest'	tomtor	tomtordor	'knob'	
kinige	kinigeler	kinigeliiñ	'book'	moyotoy	moyotoydor	'chipmunk'	
Jie	jieler	Jieliiñ	'house'	kötör	kötördör	'bird'	
iyé	iyeler	iyeliin	'mother'	bölköy	bölköydör	'islet'	
kini	kiniler	kiniliin	'3rd person'	χatiij	χatignar	'birch'	
bie	bieler	bieliin	'mare'	aan	aannar	'doo'	
oyo	oyolor	oyoluun	'child'	tiig	tiigner	'squirrel'	
χopto	χoptolor	χoptoluun	'gull'	sordoj	sordognor	'pike'	
börö	börölör	böröliün	'wolf'	olom	olomnor	'ford'	
tial	tiallar	tialliin	'wind'	oron	oronnor	'bed'	
ial	iallar	ialliin	'neighbor'	bödög	bödögör	'strong one'	
kuul	kuullar	kuulluuñ	'sack'	<i>Noun</i>	<i>Partitive</i>	<i>Comparative</i>	<i>Ablative</i>
at	attar	attiiñ	'horse'	aýa	ayataaýar	ayattan	'father'
balik	baliktar	balikiin	'fish'	paarta	paartata	paartataaýar	'school desk'
iskaap	iskaaptar	iskaaptiin	'cabinet'	tia	tiata	tiataaýar	'forest'
oyus	oyustar	oyustuuñ	'bull'	kinige	kinigete	kinigeteeyer	'book'
kus	kustar	kustuuñ	'duck'	Jie	jiete	jieteeeyer	'house'
tünnük	tünnükter	tünnüktüün	'window'	iye	iyete	iyeteeeyer	'mother'
sep	septer	septiin	'tool'	kini	kinite	kinitteeeyer	'3rd person'
et	etter	ettiiñ	'meat'	bie	biete	bieteeeyer	'mare'
örüs	örüster	örüstüün	'river'	oyo	oyoto	oyotooyor	'child'
tis	tiister	tiistiin	'tooth'	χopto	χoptoto	χoptotooyor	'gull'
soroχ	soroχtor	soroχtuun	'some person'	börö	börötö	börötööýör	'wolf'
ox	oxtor	oxtuun	'arrow'	tial	tialla	tiallaayar	'wind'
oloppos	oloppstor	oloppstuun	'chair'	ial	ialla	iallaayar	'neighbor'
ötöχ	ötöxtör	ötöxtüün	'abandoned farm'	kuul	kuulla	kuullaayar	'sack'
ubay	ubaydar	ubaydiin	'elder brother'	moχsoyol	moχsoyollo	moχsoyollooyor	'falcon'
asaray	saraydar	saraydiin	'bam'	at	atta	attayar	'horse'
tiy	tiydar	tiydiin	'foal'	balik	balikta	baliktaayar	'fish'
atiir	atiirdar	atiirdiin	'stallion'	iskaap	iskaapta	iskaaptaayar	'cabinet'
			oyus	oyusta	oyustaayar	oyustan	'bul'
			kus	kusta	kustaayar	kustan	'duck'
			tünnük	tünnükte	tünnükteeyer	tünnükten	'window'

# Turkic Sakha (Yakut)

## observed data

CABINET (PLURAL): *is̥kaaptar*

BED (PLURAL): *oronnor*

MARE (PLURAL): *bieler*

*138 total examples*

# Turkic Sakha (Yakut)

grammar (unobserved)

SINGULAR → stem  
PLURAL → stem + lar

observed data

CABINET (PLURAL): *is̥kaaptar*

BED (PLURAL): *oronnor*

MARE (PLURAL): *bieler*

*138 total examples*

# Turkic Sakha (Yakut)

## grammar (unobserved)

SINGULAR → stem  
PLURAL → stem + lar

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

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

$r_3: V \rightarrow [ \text{+rounded} ] / [ \text{+rounded} ] [ \text{-low} ]_0$

$r_4: [ \text{+continuant} \text{ } \text{-high} ] \rightarrow [ \text{-rounded} ] / u C_0$   
"harmonize" round vowels like "u", "o"

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

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

observed data

CABINET (PLURAL): *iskaaptar*

BED (PLURAL): *oronnor*

MARE (PLURAL): *bieler*

138 total examples

# Turkic Sakha (Yakut)

## grammar (unobserved)

SINGULAR → stem  
PLURAL → stem + lar

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

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

$r_3: V \rightarrow [ \text{+rounded} ] / [ \text{+rounded} ] [ \text{-low} ]_0$

$r_4: [ \text{+continuant} \text{ } \text{-high} ] \rightarrow [ \text{-rounded} ] / u \text{ } C_0$   
"harmonize" round vowels like "u", "o"

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

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

## stems (unobserved)

⟨*iskaap*, CABINET⟩

⟨*oron*, BED⟩

⟨*bie*, MARE⟩

## observed data

CABINET (PLURAL): *iskaaptar*

BED (PLURAL): *oronnor*

MARE (PLURAL): *bieler*

138 total examples

# Turkic Sakha (Yakut)

## grammar (unobserved)

SINGULAR→stem  
PLURAL→stem+lar

$r_1: l \rightarrow d / [-\text{lateral} \ -\text{tense}]$   
“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

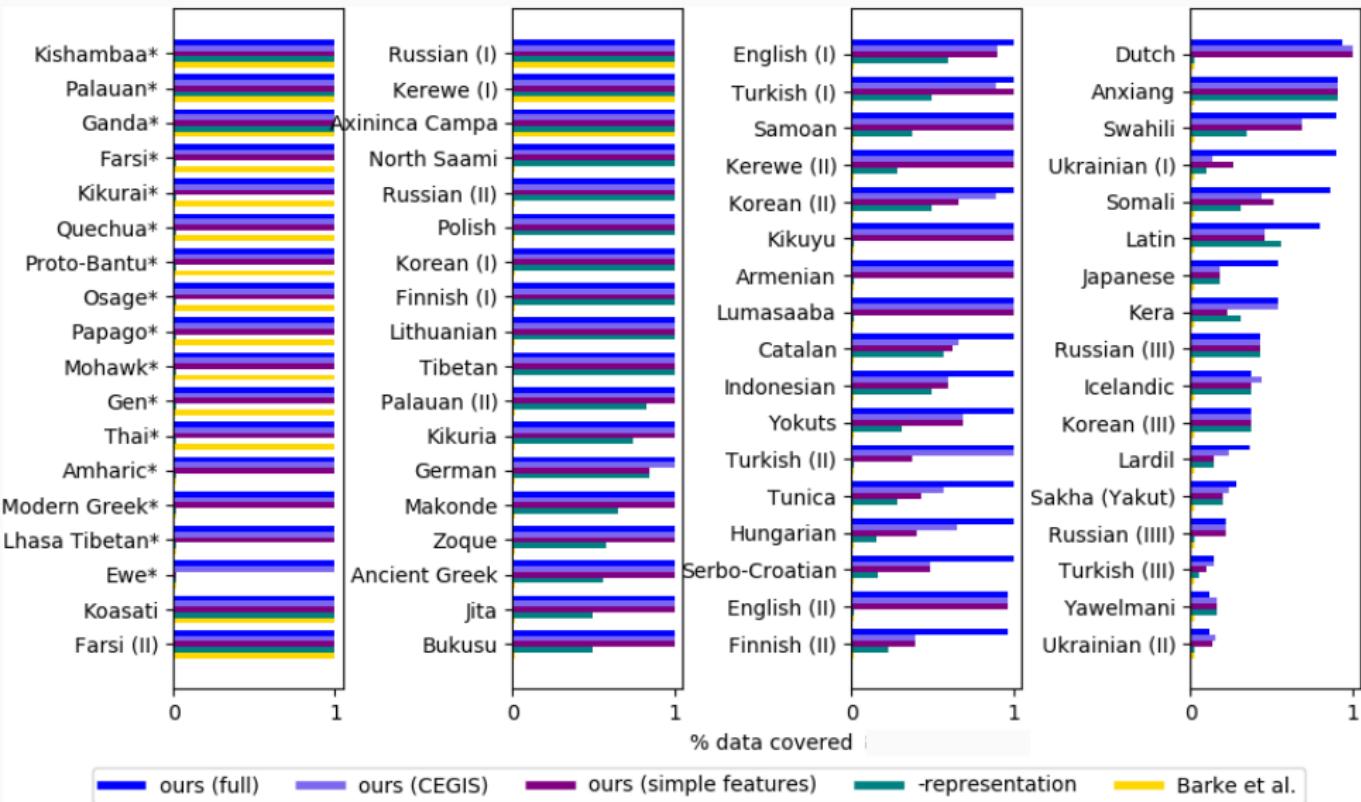
$r_6: [-\text{sonorant} \ +\text{voice}] \rightarrow [+ \text{nasal}] / [+ \text{nasal}]$   
“nasalize” consonant next to a nasal, like “m̄”

stems  
(unobserved)

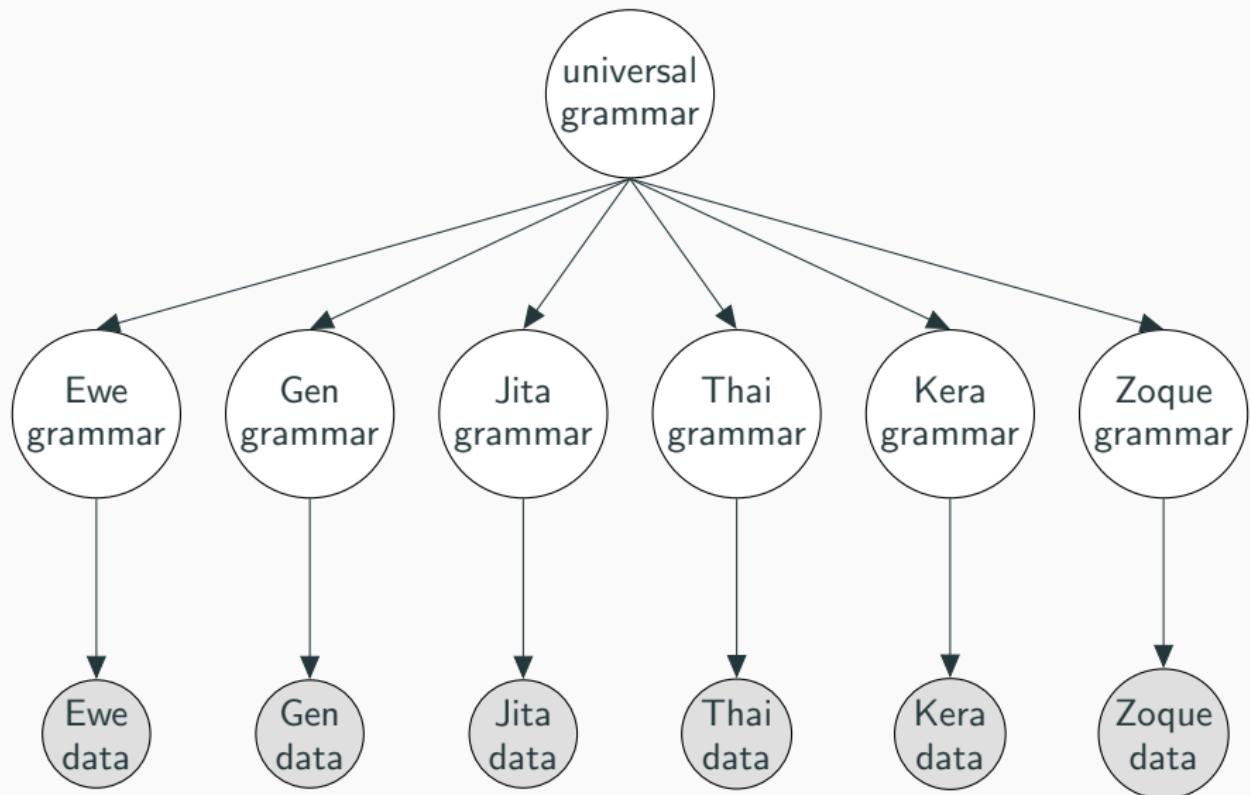
- ⟨*iskaap*, CABINET⟩
- ⟨*oron*, BED⟩
- ⟨*bie*, MARE⟩
- ⟨*örus*, RIVER⟩

observed data

- CABINETS → *iskaap+lar* → *iskaaplar*  $\xrightarrow{r_1}$  *iskaapdar*  $\xrightarrow{r_2}$  *iskaaptar*
- BEDS → *oron+lar* → *oronlar*  $\xrightarrow{r_1}$  *orondar*  $\xrightarrow{r_3}$  *orondor*  $\xrightarrow{r_6}$  *oronnor*
- MARES → *bie+lar* → *bielar*  $\xrightarrow{r_5}$  *bieler*
- RIVER (ASSOC) → *örus+län* → *öruslänn*  $\xrightarrow{r_1}$  *örusdiän*  $\xrightarrow{r_2}$  *örustiän*  $\xrightarrow{r_3}$  *örustuu*  $\xrightarrow{r_5}$  **[örüstüün]**



# Distilling higher-level knowledge



**Program Induction and** perception  
learning to learn  
interpretable models  
**the future**

# Models of the physical world

hinge



gear



doorknob



