

# **Building Machines that Discover Generalizable, Interpretable Knowledge**

---

Kevin Ellis

2020

MIT

# What computational problems are solved by intelligence?

an endless range of problems

language



using new devices



engineering



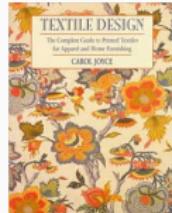
science



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]
```

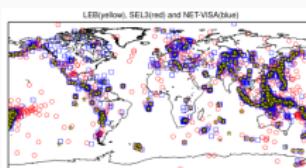
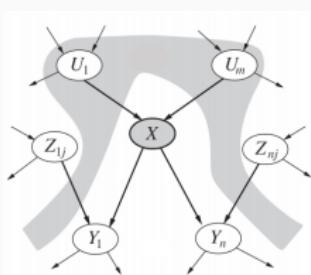
Input interpretation:

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]= {}
```

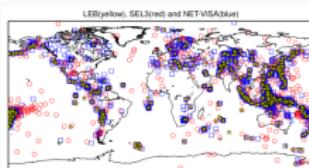
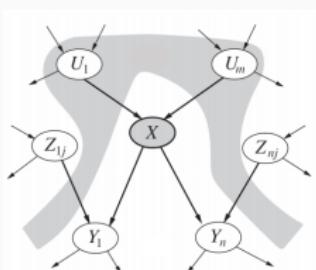
```
Input interpretation:
```

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

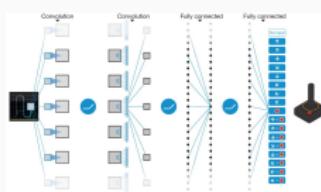
Result:

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

### Probabilistic



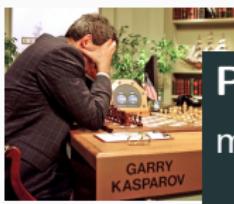
### Neural



# 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:

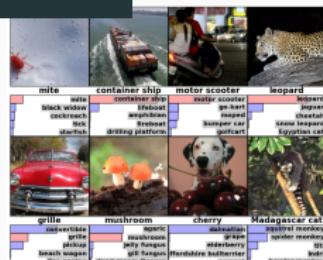
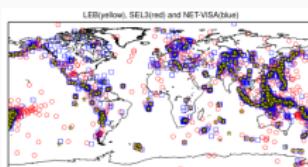
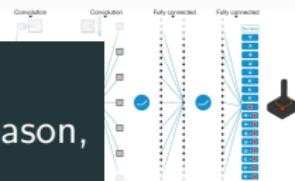
$$h = \frac{Z}{w - w^2} \text{ and } w^2 \neq w$$

Probabilistic



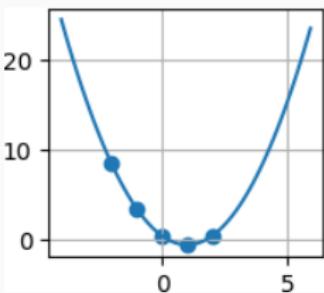
**Program induction**  
machines that learn, perceive, and reason,  
by writing their own code

Neural



# Why program induction?

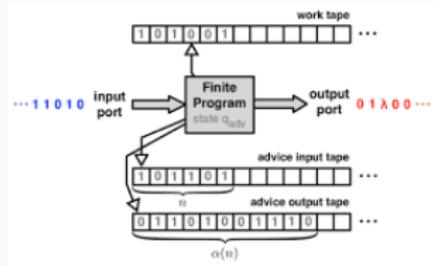
strong generalization  
+ data efficiency



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:

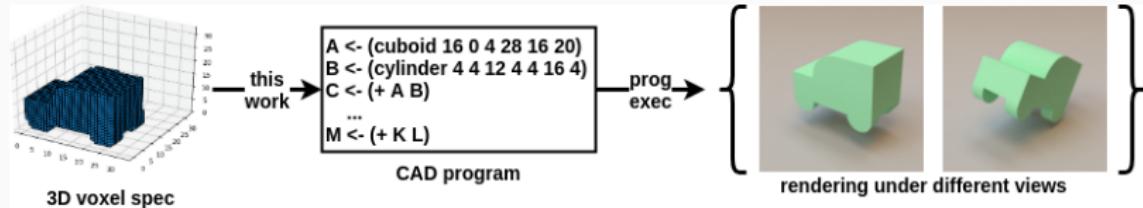
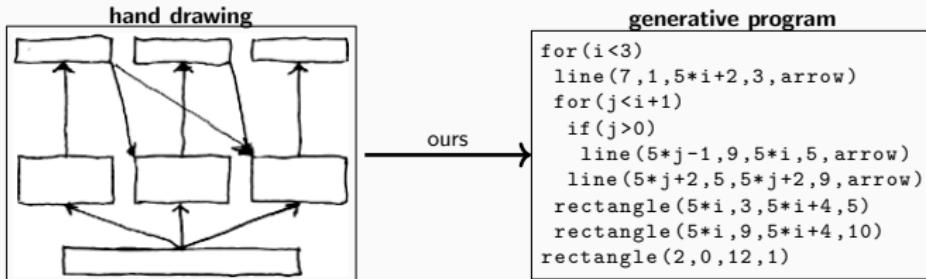
- **probabilistic** methods for uncertainty and learning-to-learn
- **neural** methods for guiding combinatorial search
- **symbolic** methods, from the **programming languages** community
  - maturing **program synthesis** techniques
  - type systems, program analysis, constraint solving, ...

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]

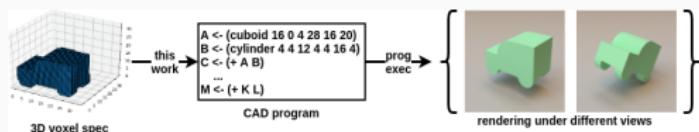
# Perception, Synthesizing models, Learning-to-Learn

Theme: high-level visual understanding, pixels→programs

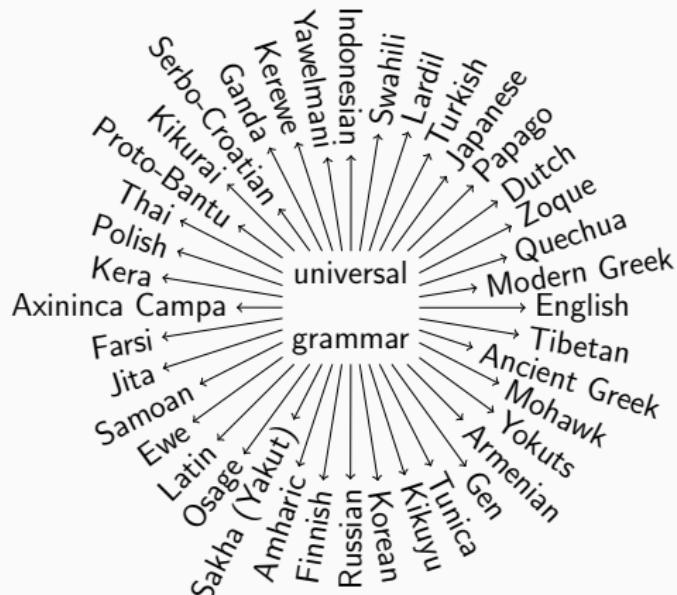


Perception, Synthesizing models, Learning-to-Learn

Theme: high-level visual understanding, pixels $\rightarrow$ programs

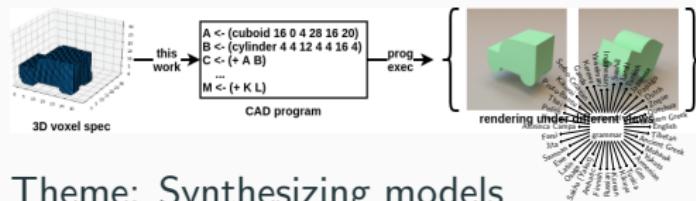


## Theme: Synthesizing human-understandable models



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

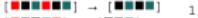
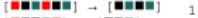
Theme: high-level visual understanding, pixels $\rightarrow$ programs



## Theme: Synthesizing models

## Theme: 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	ℳ ⋯ ⌂ ⌃
<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	ℳ ⌂ ⌃ ⌄

Block Towers	Symbolic Regression	Recursive Programming	Physics
	 $\frac{(-2x - 0.9)}{(x - 4.4)(x - 0.9)}$	 $0.3x^3 + 1.1x^2 - 2.0x + 0.6$	 $\begin{array}{l} \text{Filter} \\ [\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare] \\ [\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare\blacksquare\blacksquare] \\ [\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare\blacksquare] \end{array}$
	 $0.5x^4 + 2.5x^3 + 0.4x^2 - 2.2x + 2.4$	 $\frac{4.9}{x}$	 $\begin{array}{l} \text{Index List} \\ 0, [\blacksquare\blacksquare\blacksquare] \rightarrow \blacksquare \\ 1, [\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare] \\ 1, [\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare\blacksquare] \end{array}$
	 $0.5x^4 + 2.5x^3 + 0.4x^2 - 2.2x + 2.4$	 $\begin{array}{l} \text{Length} \\ [\blacksquare\blacksquare\blacksquare] \rightarrow 4 \\ [\blacksquare\blacksquare\blacksquare] \rightarrow 6 \\ [\blacksquare\blacksquare] \rightarrow 3 \end{array}$	 $\begin{array}{l} \text{Every Other} \\ [\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare] \\ [\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare\blacksquare\blacksquare] \\ [\blacksquare\blacksquare\blacksquare] \rightarrow [\blacksquare\blacksquare] \end{array}$

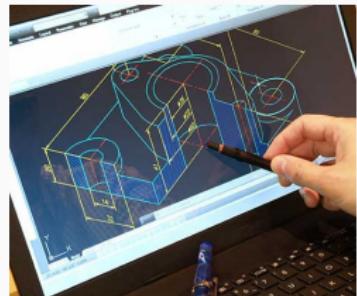
Program Induction and perception  
learning to learn  
model discovery

# High-level, abstract visual abilities

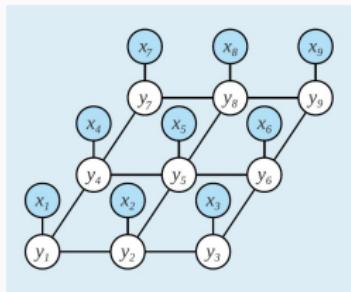
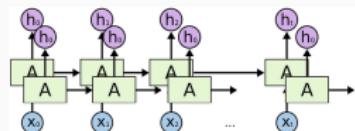
...in art



...in engineering



...in AI



# High-level, abstract visual abilities

...in art



why?

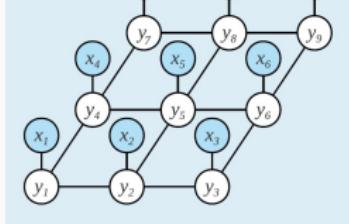
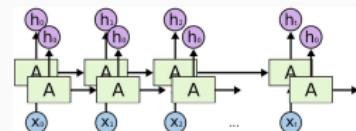
impute missing objects, extrapolate percepts,  
learn visual concepts ('arch', 'spiral', 'Ising model'),  
assist graphic design, assist 3D modeling

how?

machine learning +

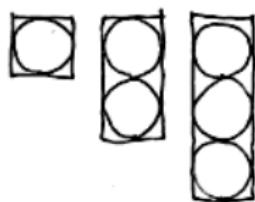
program synthesis techniques from programming languages community

...in AI



# Learning to infer graphics programs from hand-drawn images

model infers program from drawing



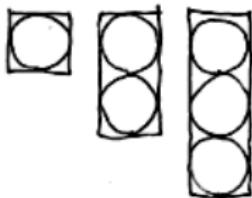
→

```
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)
```

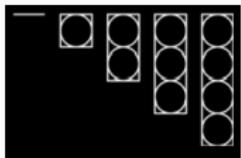
# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

**zero-shot generalization / extrapolation**



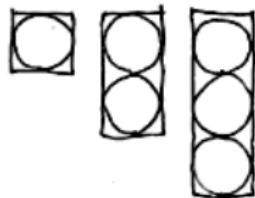
```
for (0 <= i < 3 + 1)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```



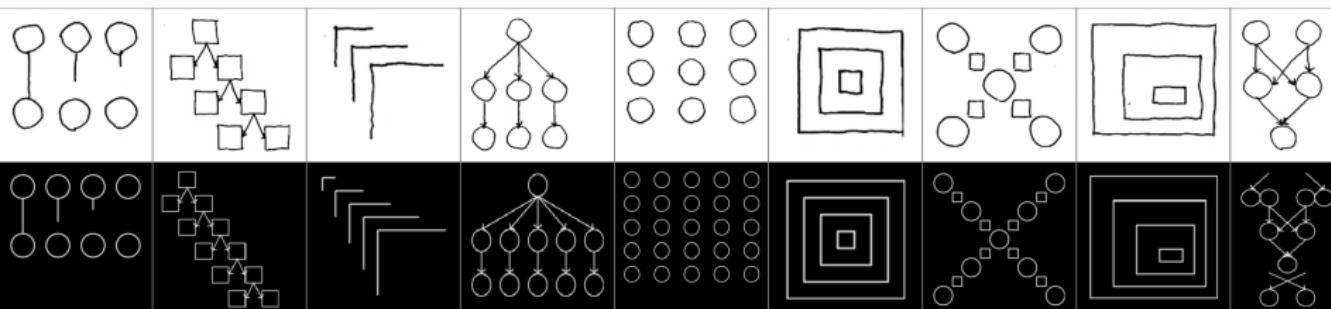
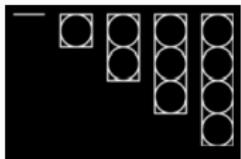
# Learning to infer graphics programs from hand-drawn images

model infers program from drawing

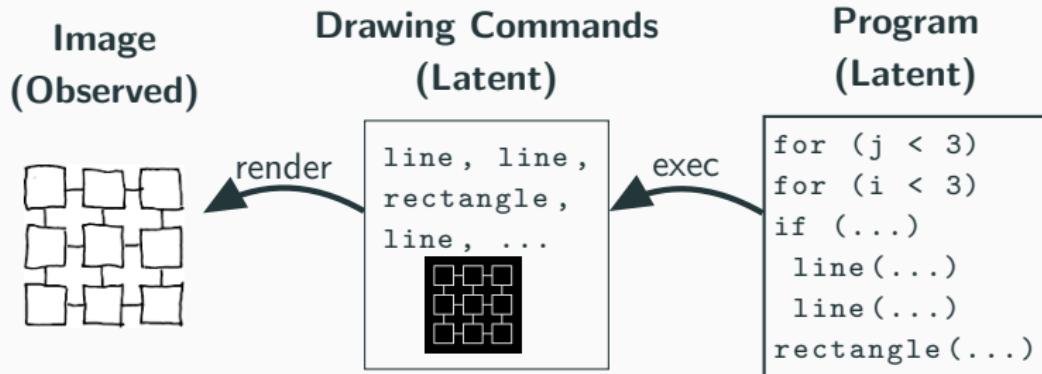
**zero-shot generalization / extrapolation**



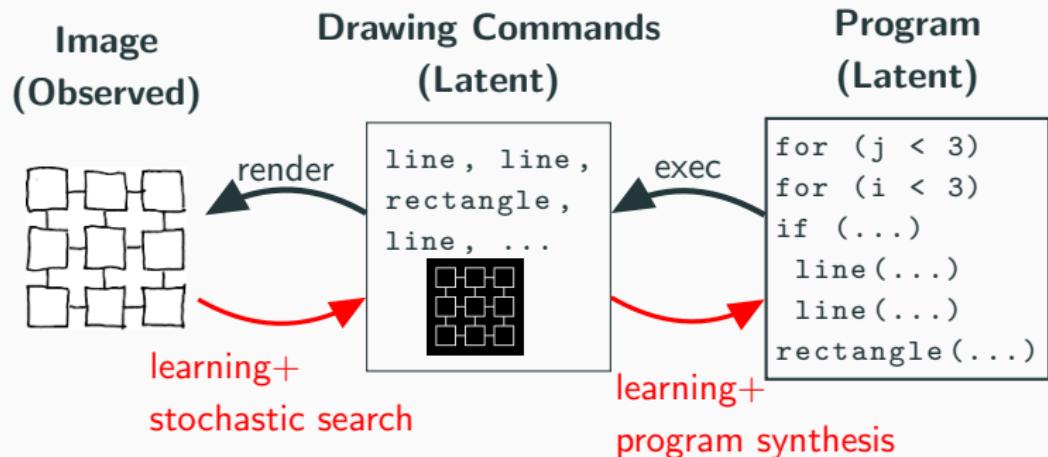
```
for (0 <= i < 3 + 1)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```



# How to infer graphics programs from hand-drawn images



# How to infer graphics programs from hand-drawn images

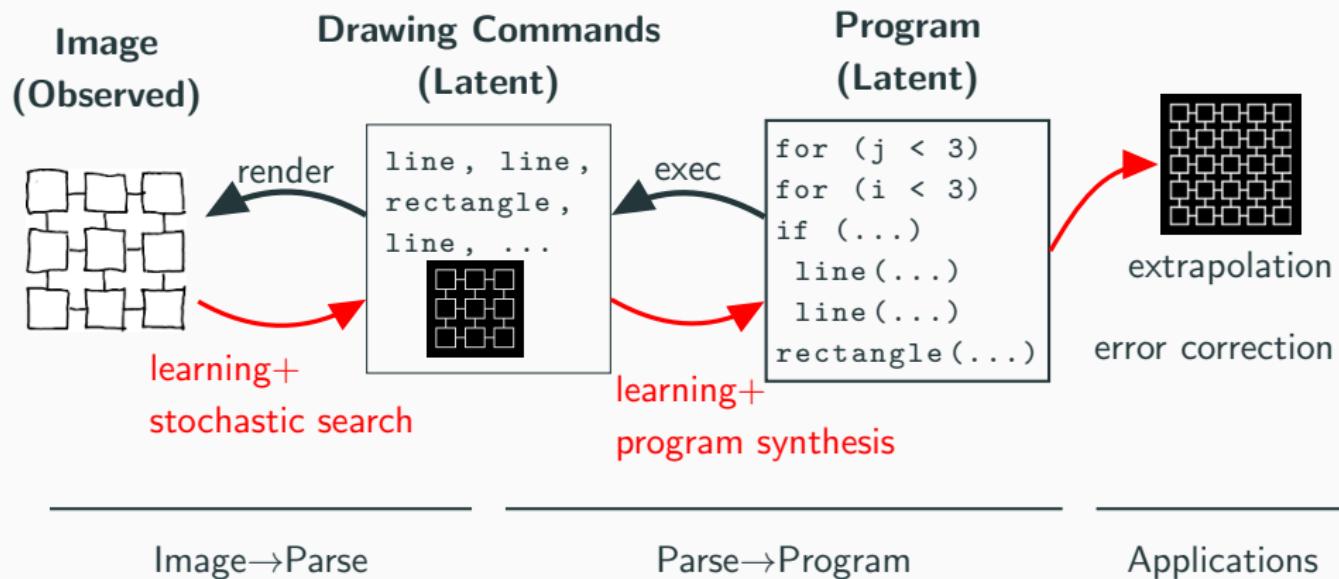


---

Image→Parse

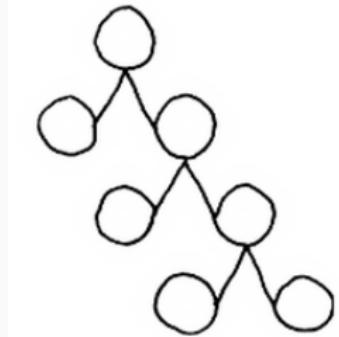
Parse→Program

# How to infer graphics programs from hand-drawn images

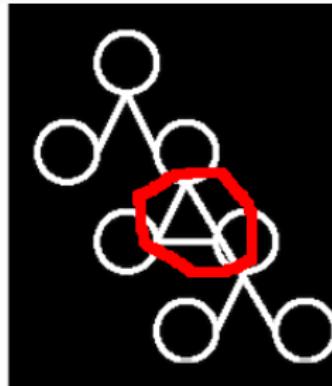


# Top-down influences on perception

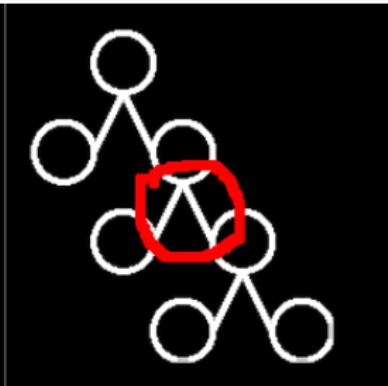
drawing



bottom-up neural net

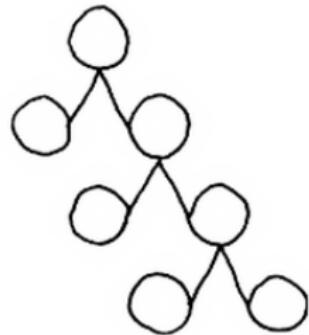


w/ top-down program bias

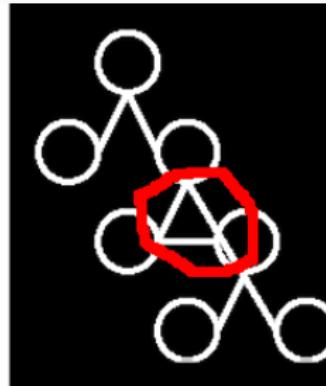


# Top-down influences on perception

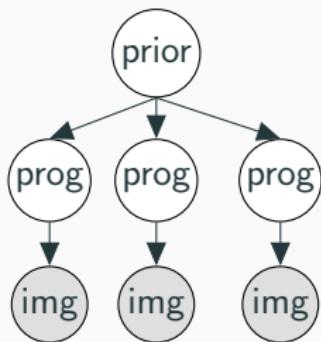
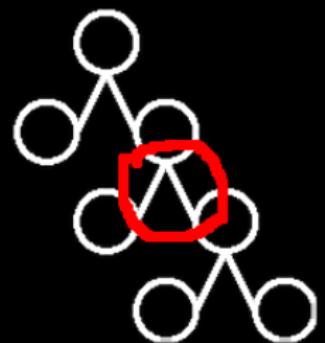
drawing



bottom-up neural net



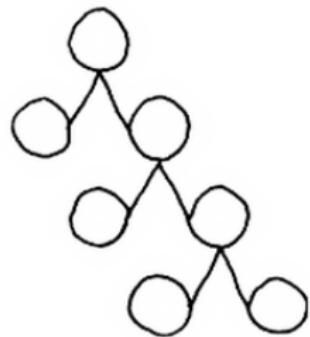
w/ top-down program bias



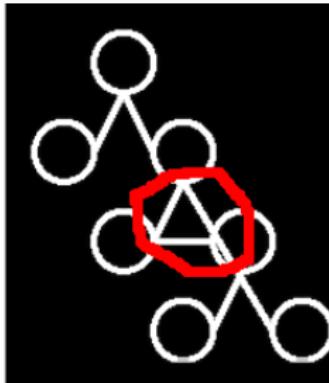
predicted program =  
 $\arg \max_{\text{progs}} \mathbb{P} [\text{img} | \text{prog}] \mathbb{P} [\text{prog} | \text{prior}]$

# Top-down influences on perception

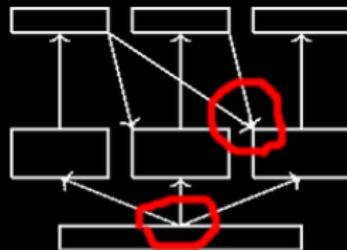
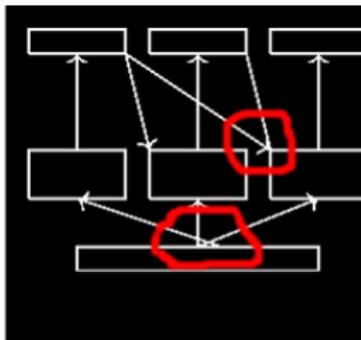
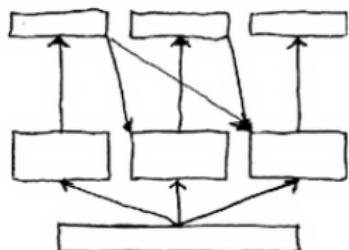
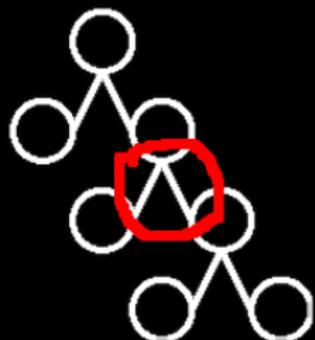
drawing



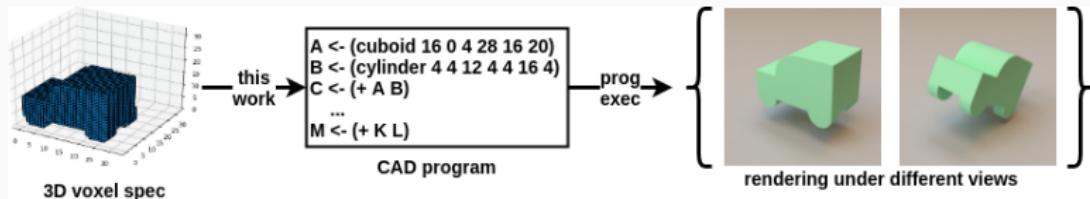
bottom-up neural net



w/ top-down program bias



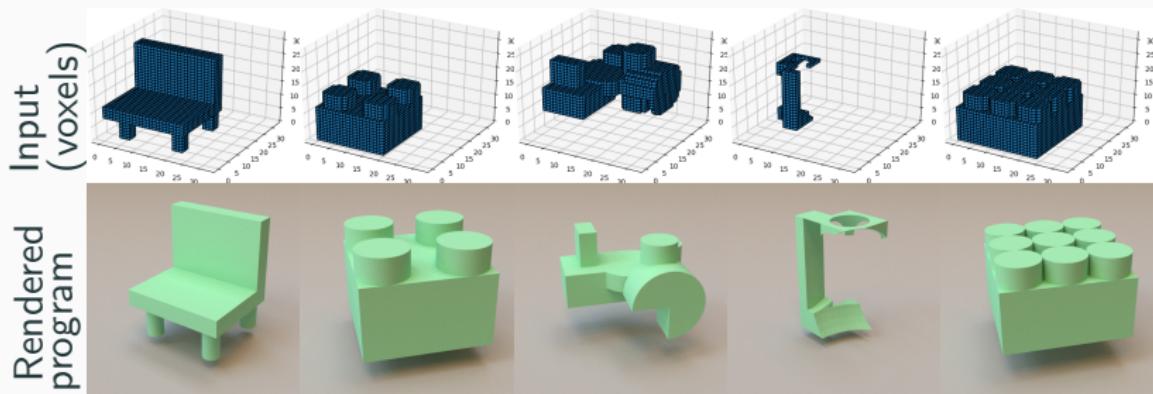
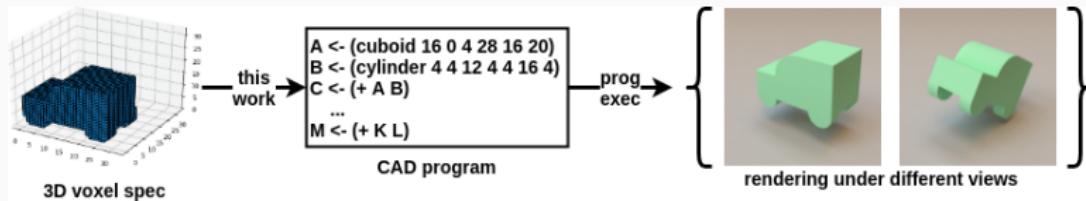
# 3D program induction



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

\*equal contribution

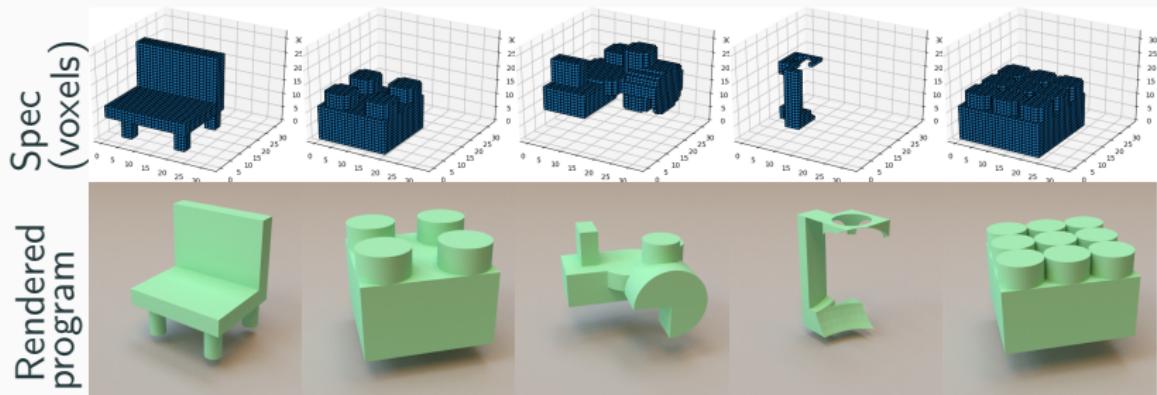
# 3D program induction



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

\*equal contribution

# 3D program induction



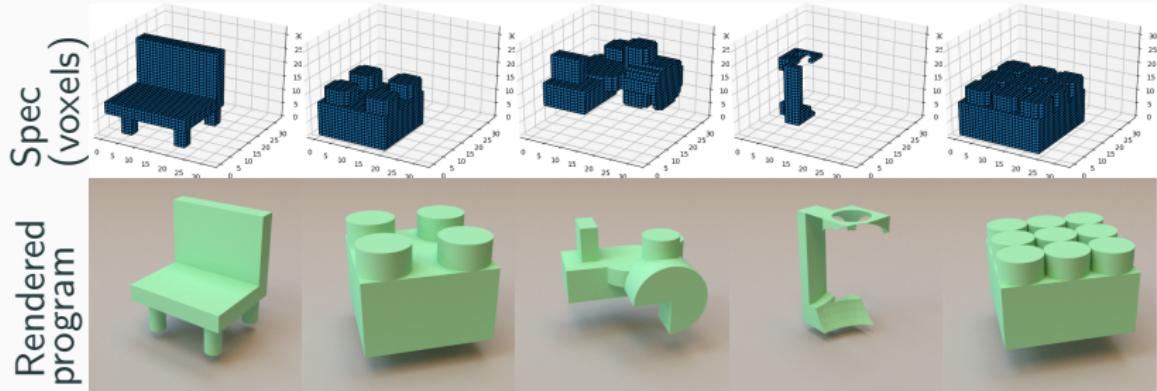
Challenge: combinatorics!

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

## Lessons

The bias from a programming language gives extrapolation, or strong generalization, sometimes even without meta-learning

Mix-and-match techniques to play to their strengths: neural nets for perception, symbols for reasoning, Bayesian methods for uncertainty

**Program Induction and perception**  
**learning to learn**  
**discovering models**

## Learning to write code

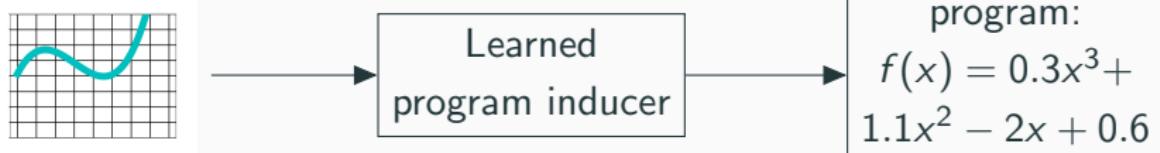
Goal: acquire domain-specific knowledge needed to induce a class of programs

- 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)



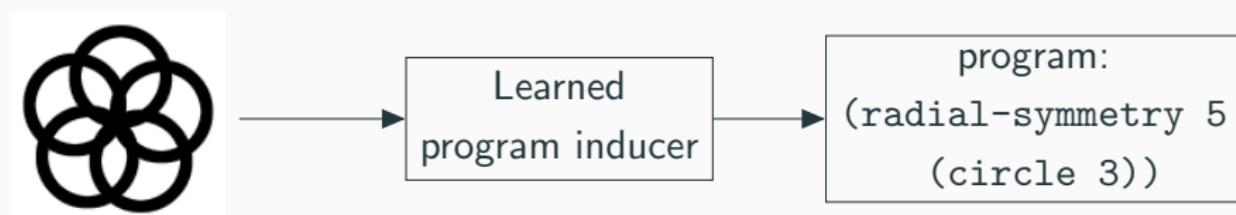
Concepts:  $x^3$ ,  $\alpha x + \beta$ , etc

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

## Initial Primitives

```
⋮  
map  
fold  
if  
cons  
>  
⋮
```

# Library learning

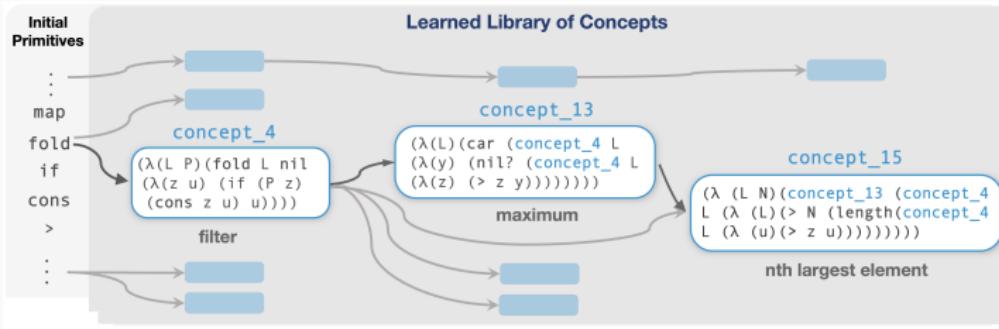
## Initial Primitives

```
:  
:  
map  
fold  
if  
cons  
>  
:  
:
```

## Sample Problem: sort list

```
[9 2 7 1] -> [1 2 7 9]  
[3 8 9 4 2] -> [2 3 4 8 9]  
[6 2 2 3 5] -> [2 3 2 3 5 6]  
...  
:
```

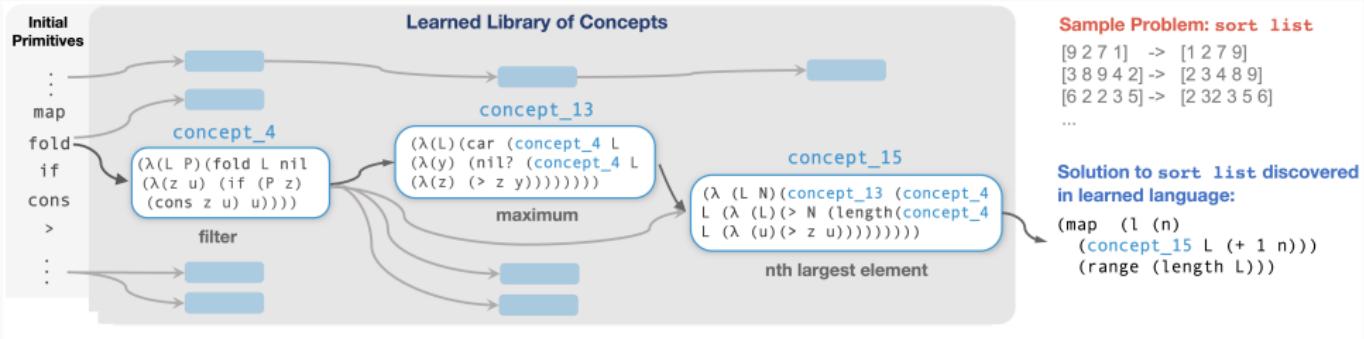
# Library learning



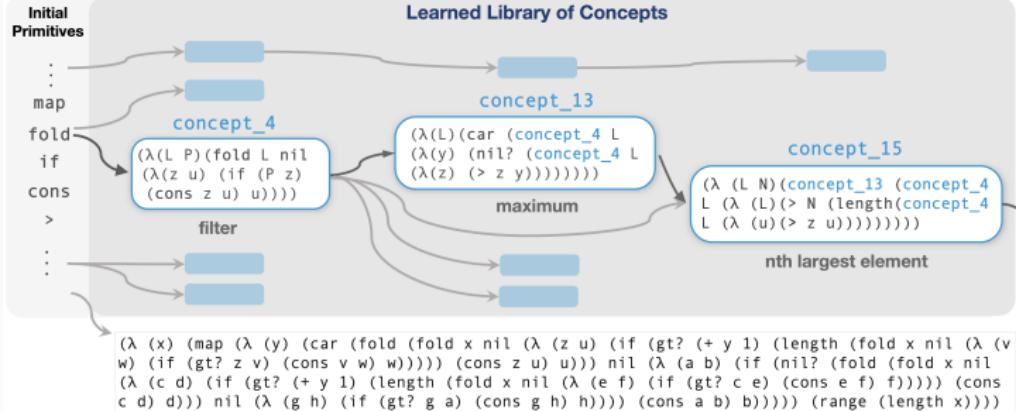
**Sample Problem: sort list**

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

# Library learning



# Library learning



**Sample Problem: sort list**

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

**Solution to sort list discovered in learned language:**

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

**Solution to sort list if expressed in initial primitives**

# DreamCoder

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

## List Processing

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

## Double

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

## Check Evens

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

## Text Editing

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

## Drop Last Characters

jabberwocky → jabberw  
copycat → cop

## Extract

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

## Regexes

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

## Currency

\$100.25  
\$4.50

## Dates

Y1775/07/04  
Y2000/01/01

## LOGO Graphics



Physics

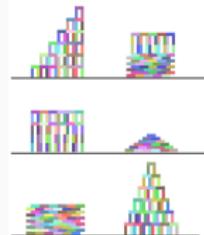
$$KE = \frac{1}{2}m|\vec{v}|^2$$

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

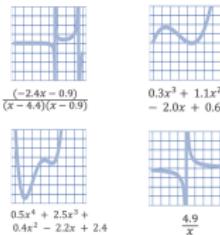
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{r}_1 - \hat{r}_2$$

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

## Block Towers



## Symbolic Regression



## Recursive Programming

### Filter

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

### Length

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

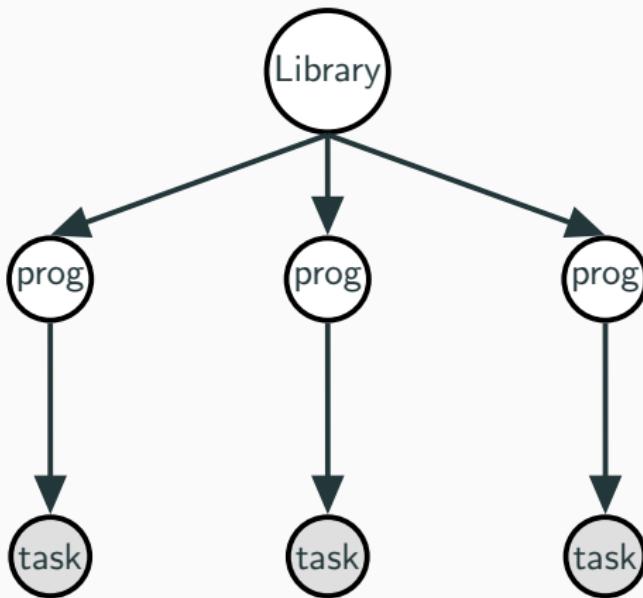
### Index List

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

### Every Other

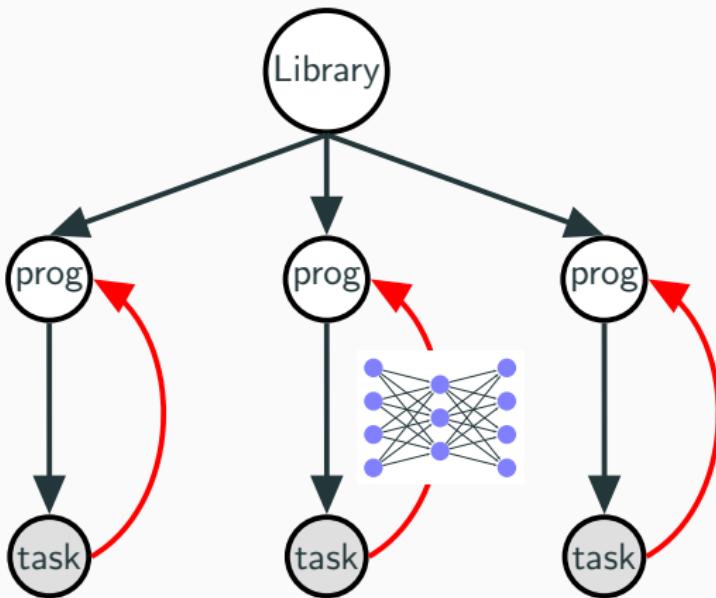
$[■■■■■■■■] \rightarrow [■■■■]$   
 $[■■■■■■■■] \rightarrow [■■■■■■]$   
 $[■■■■■■■■] \rightarrow [■■■■■■■]$

## Library learning as Bayesian inference

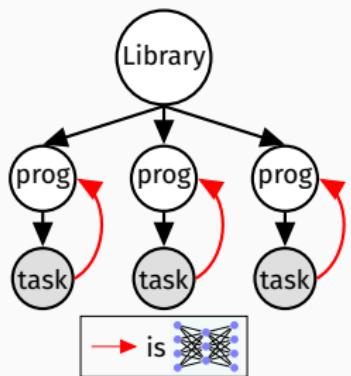


[Dechter et al., 2013] [Liang et al, 2010]; [Lake et al, 2015]

# Library learning as neurally-guided Bayesian inference



library learning via program analysis +  
new neural inference network for program synthesis +  
better program representation (Lisp+polymorphic types [Milner 1978])



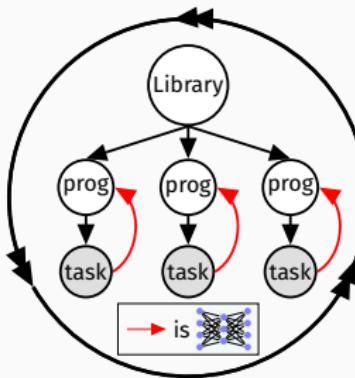
WAKE

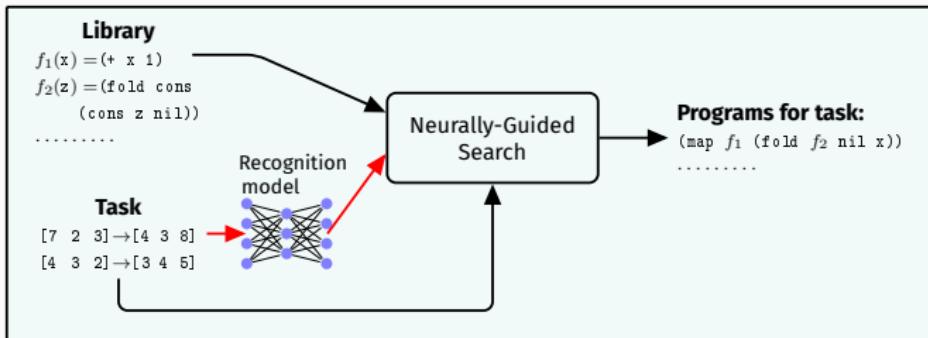
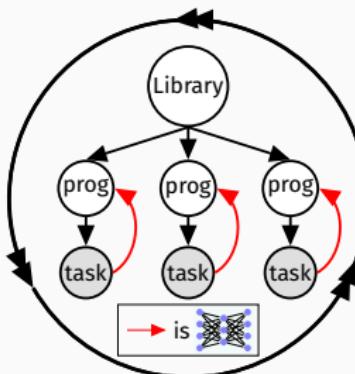


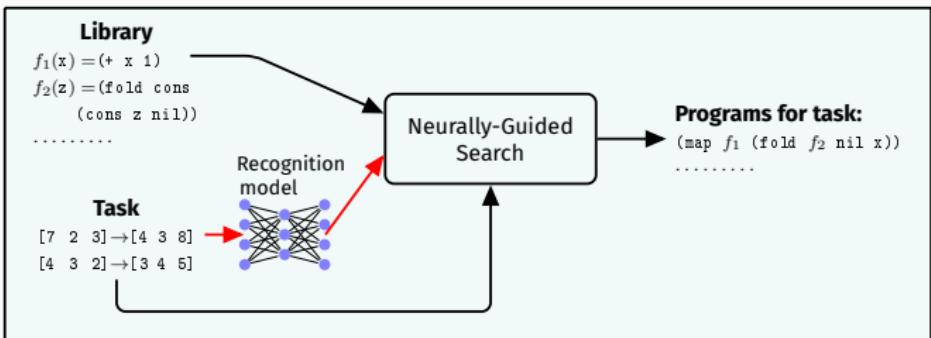
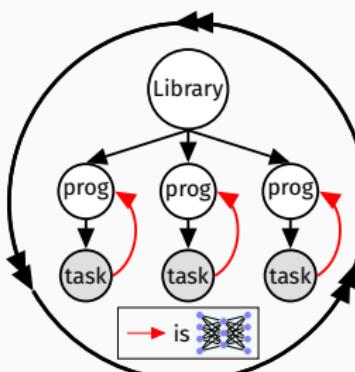
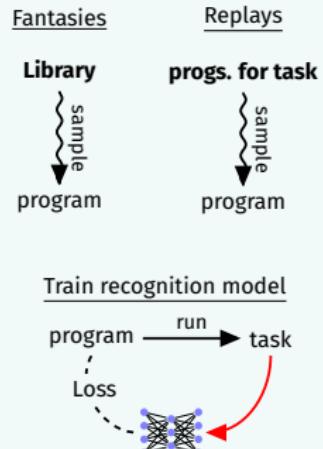
SLEEP: ABSTRACTION



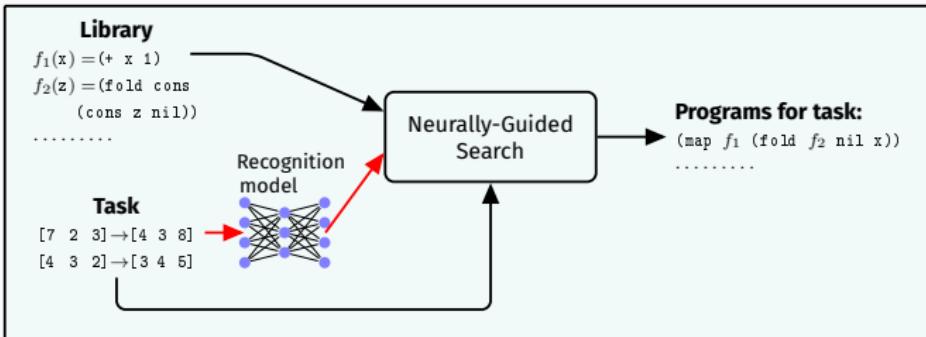
SLEEP: DREAMING



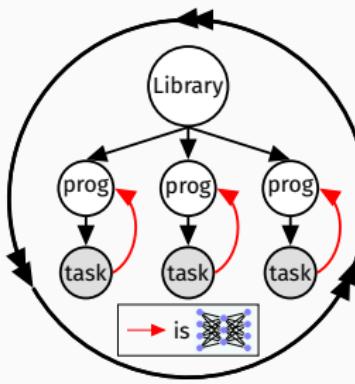
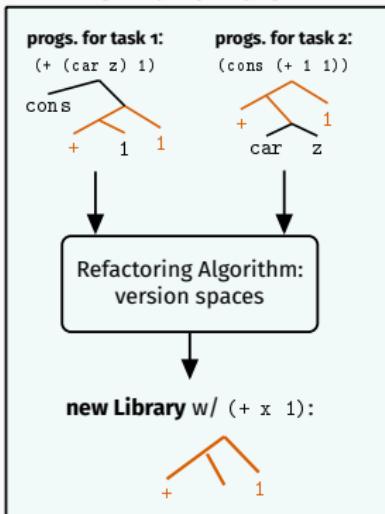
**WAKE****SLEEP: ABSTRACTION****SLEEP: DREAMING**

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

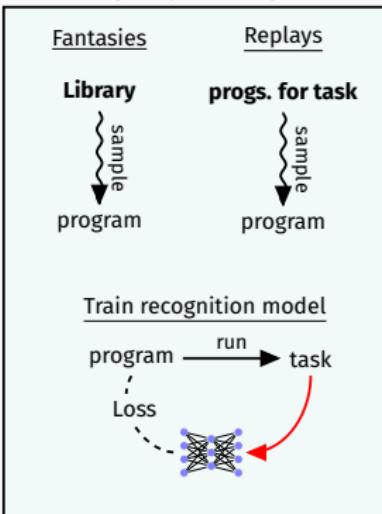
## WAKE



## SLEEP: ABSTRACTION



## SLEEP: DREAMING



## Abstraction Sleep: Growing the library via refactoring

$$5 + 5$$

## Abstraction Sleep: Growing the library via refactoring

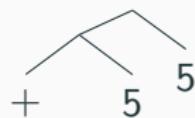
$5 + 5$

(+ 5 5)

## Abstraction Sleep: Growing the library via refactoring

$5 + 5$

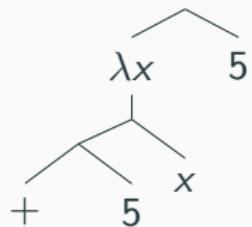
(+ 5 5)



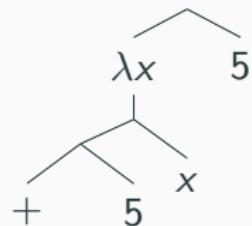
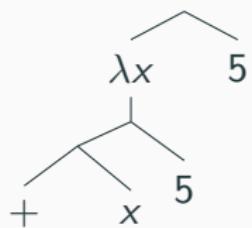
# Abstraction Sleep: Growing the library via refactoring



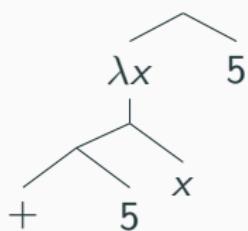
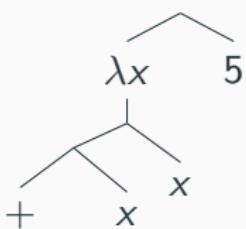
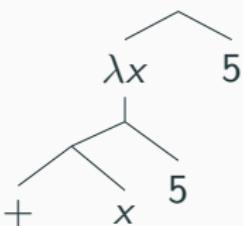
## Abstraction Sleep: Growing the library via refactoring



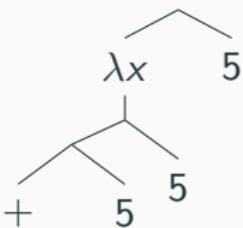
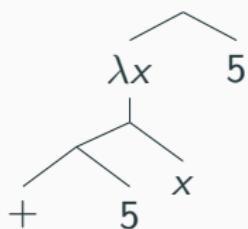
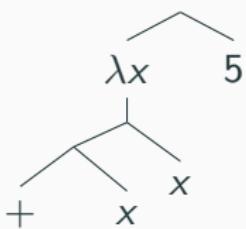
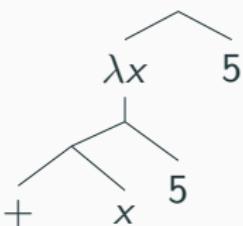
## Abstraction Sleep: Growing the library via refactoring



# Abstraction Sleep: Growing the library via refactoring

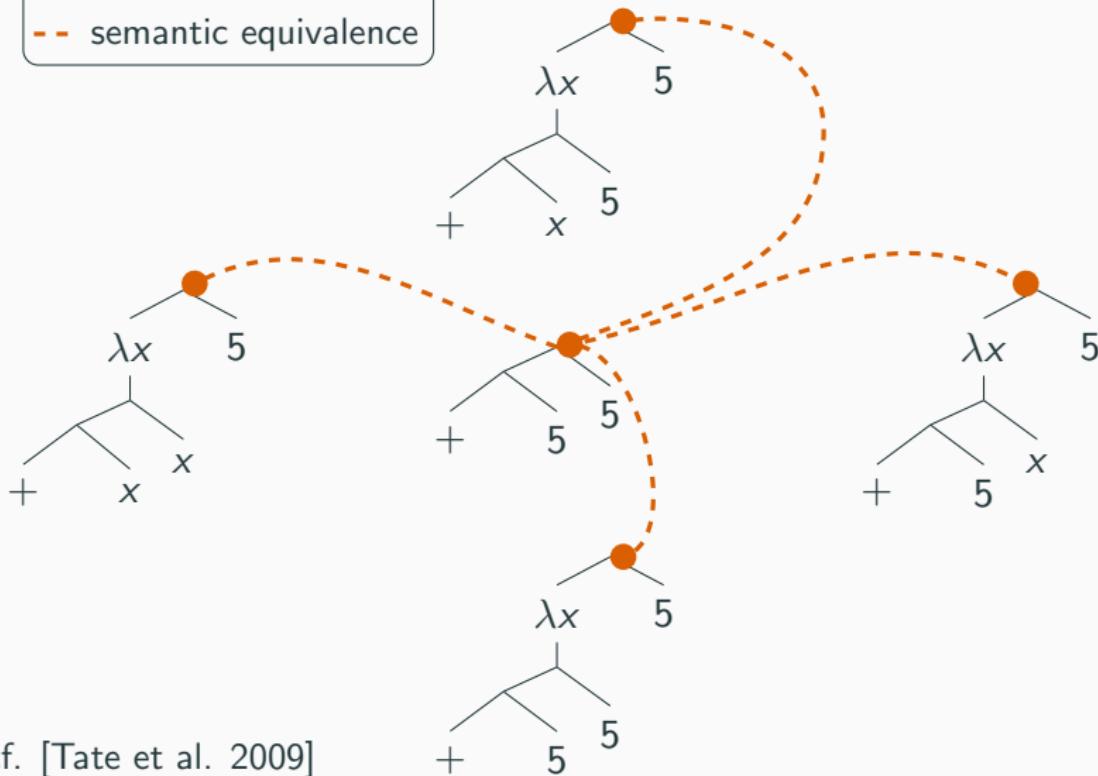


# Abstraction Sleep: Growing the library via refactoring



# Abstraction Sleep: Growing the library via refactoring

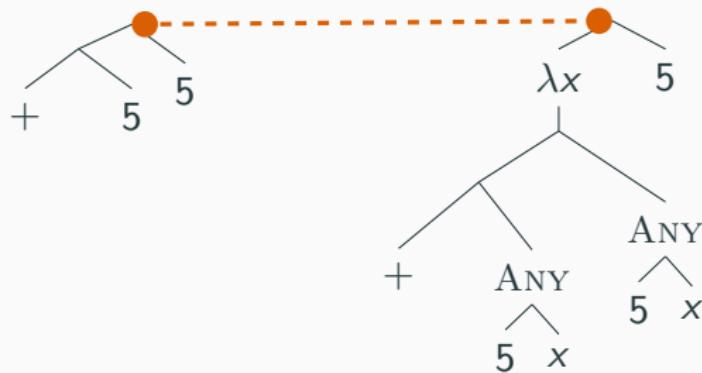
legend  
--- semantic equivalence



# Abstraction Sleep: Growing the library via refactoring

legend

- semantic equivalence
- ANY nondeterministic choice



cf. Tate et al. 2009

Gulwani 2012

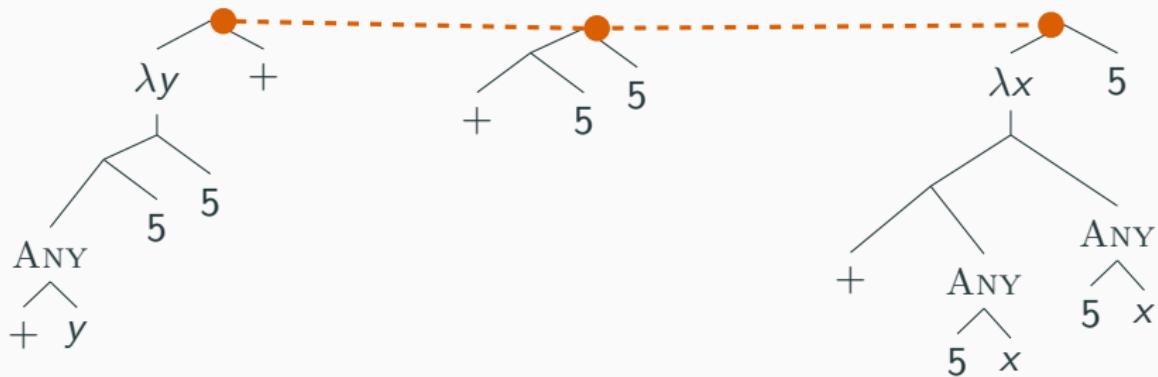
Liang et al. 2010

# Abstraction Sleep: Growing the library via refactoring

legend

semantic equivalence

ANY nondeterministic choice



cf. Tate et al. 2009

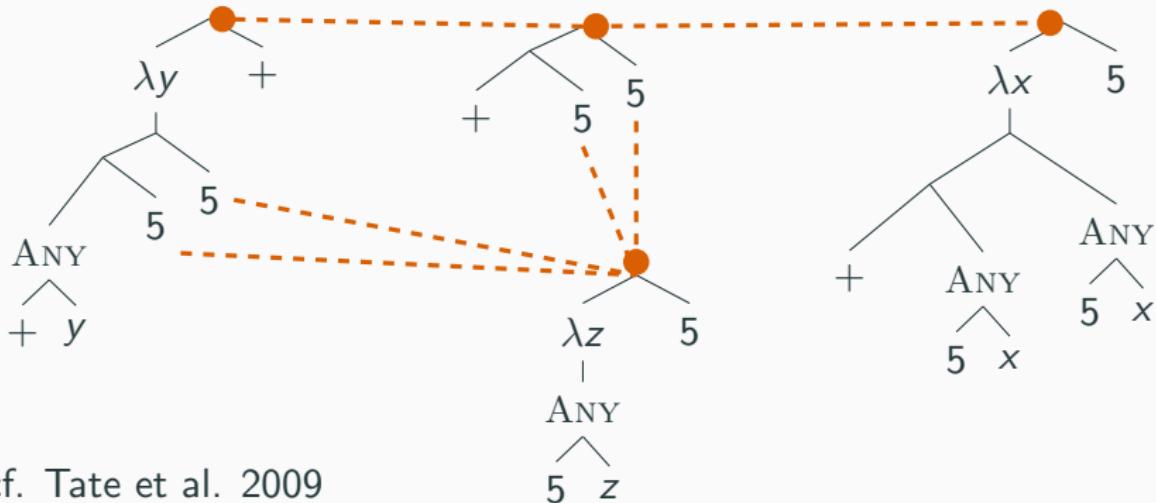
Gulwani 2012

Liang et al. 2010

# Abstraction Sleep: Growing the library via refactoring

legend

- semantic equivalence
- ANY nondeterministic choice



cf. Tate et al. 2009

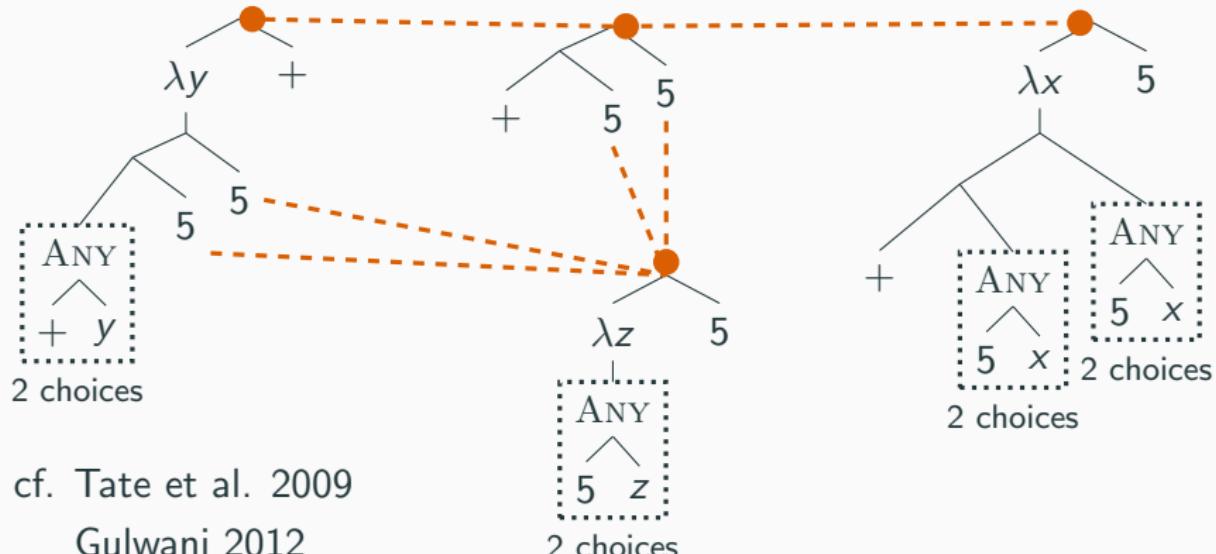
Gulwani 2012

Liang et al. 2010

# Abstraction Sleep: Growing the library via refactoring

legend

- semantic equivalence
- ANY nondeterministic choice



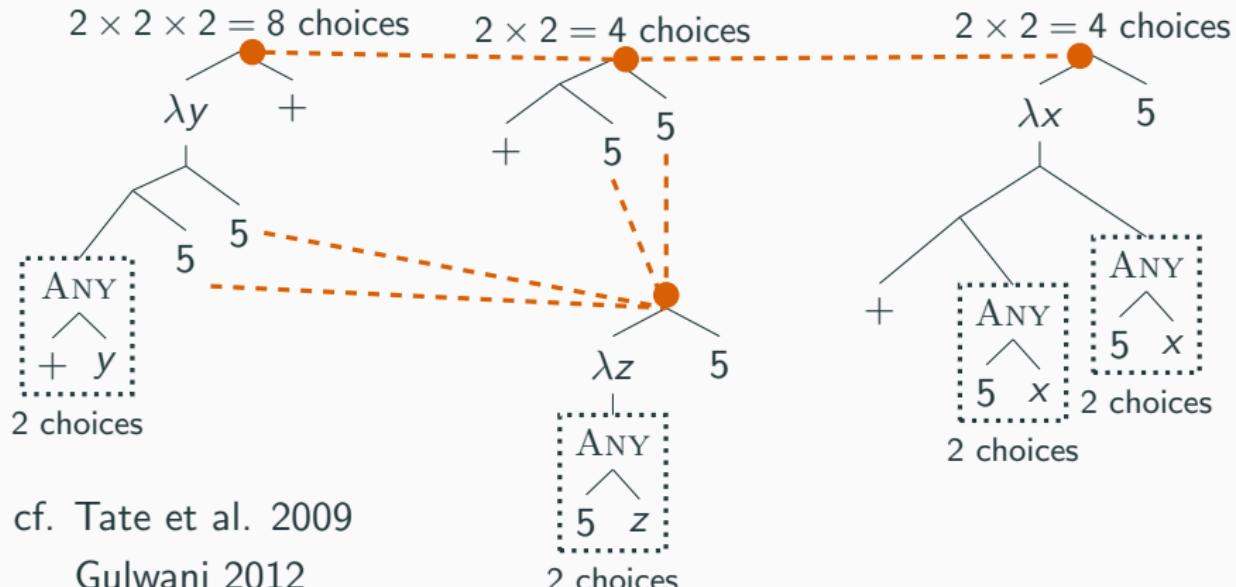
cf. Tate et al. 2009

Gulwani 2012

Liang et al. 2010

# Abstraction Sleep: Growing the library via refactoring

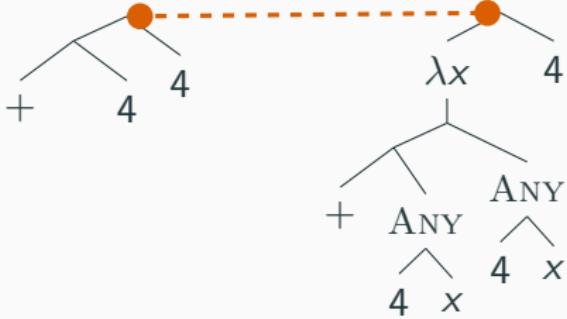
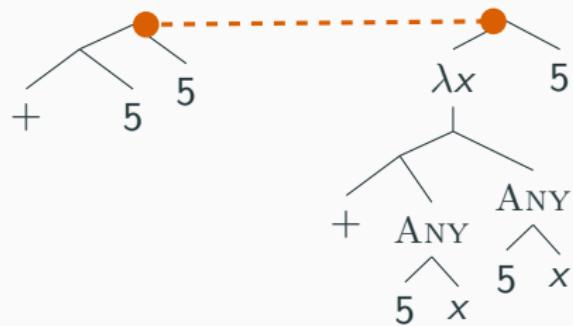
legend  
--- semantic equivalence  
ANY nondeterministic choice



cf. Tate et al. 2009

Gulwani 2012

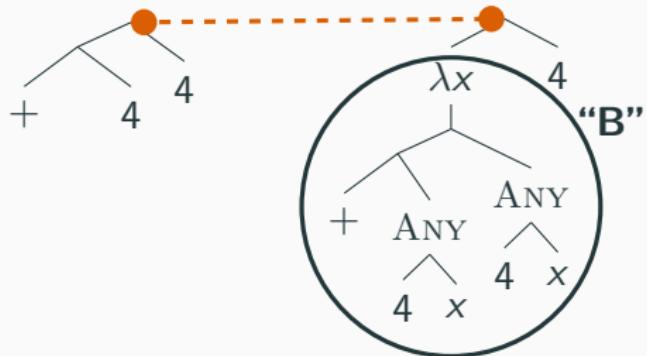
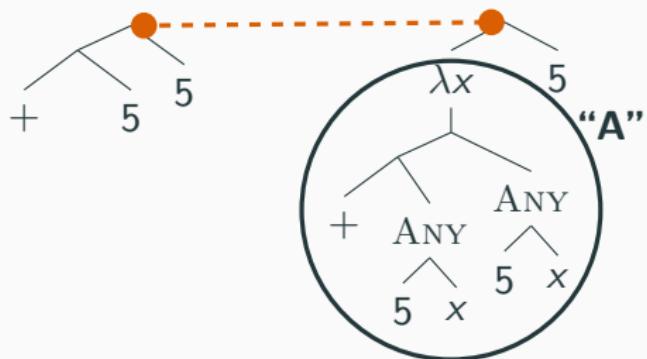
Liang et al. 2010



### legend

— semantic equivalence

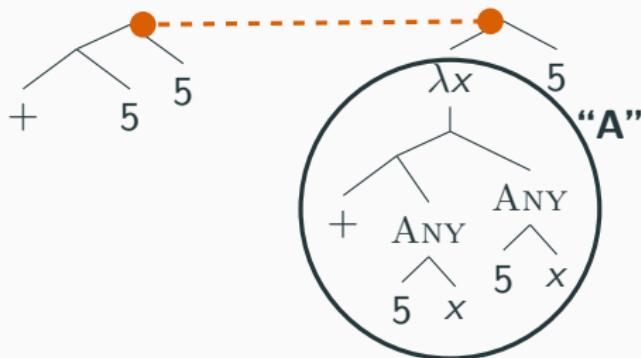
ANY nondeterministic choice



legend

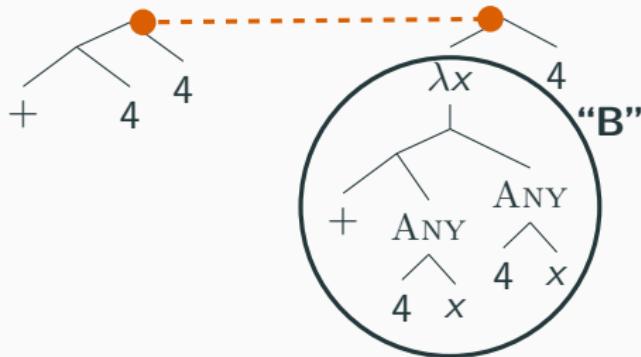
— semantic equivalence

ANY nondeterministic choice



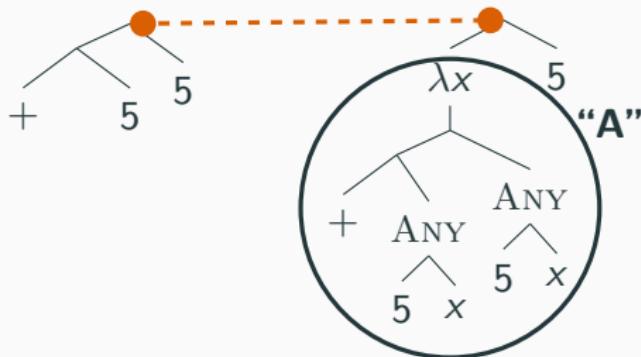
$A \cap B$  is:

A tree diagram showing the intersection  $A \cap B$ . The root node is  $\lambda x$ . It has two children: a "+" sign and the variable  $x$ .

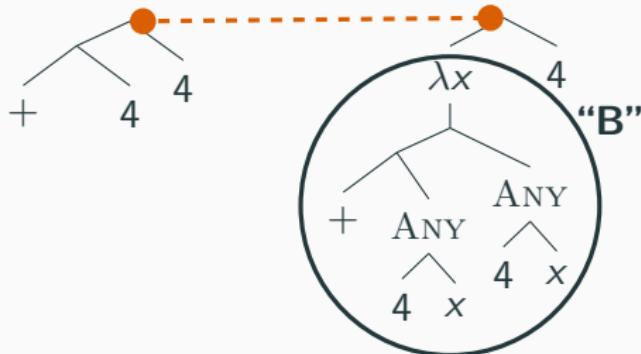
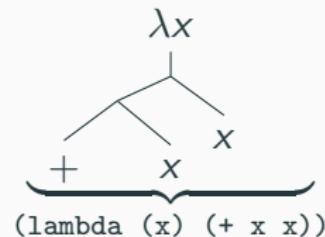


legend

- semantic equivalence
- ANY nondeterministic choice



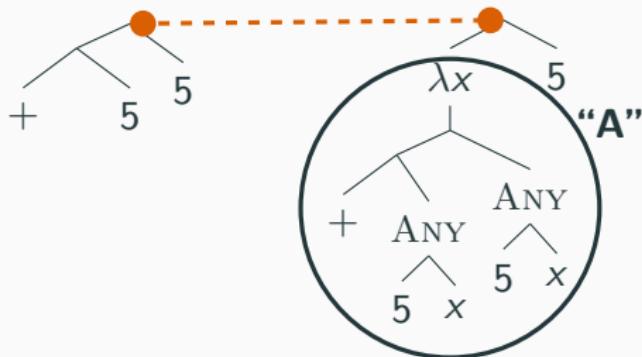
$A \cap B$  is:



legend

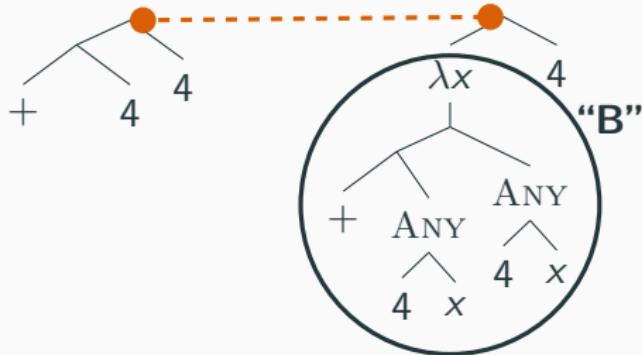
— semantic equivalence

ANY nondeterministic choice



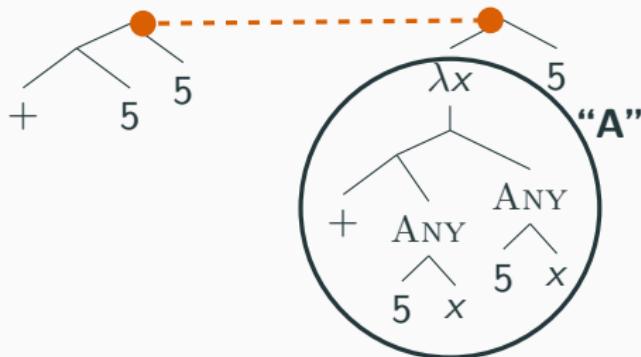
$A \cap B$  is:

The expression  $(\lambda x (+ x x))$  is labeled "double".

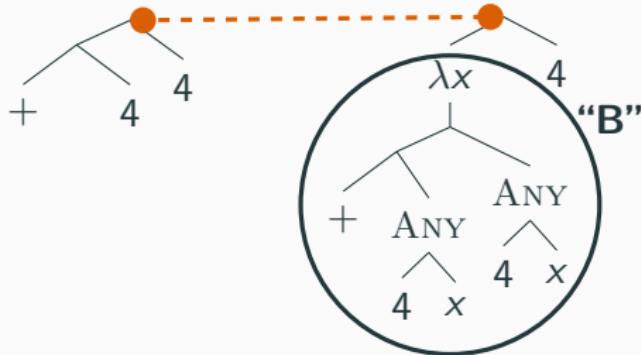
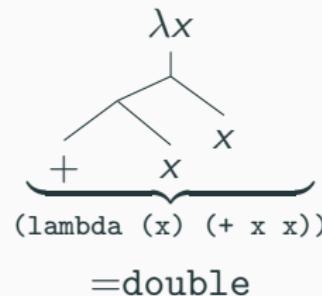


### legend

- semantic equivalence
- ANY nondeterministic choice



$A \cap B$  is:



w/o double	w/ double
$(+ 5 5)$	$(\text{double } 5)$
$(+ 4 4)$	$(\text{double } 4)$
$(+ 3 3)$	$(\text{double } 3)$
...	

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)

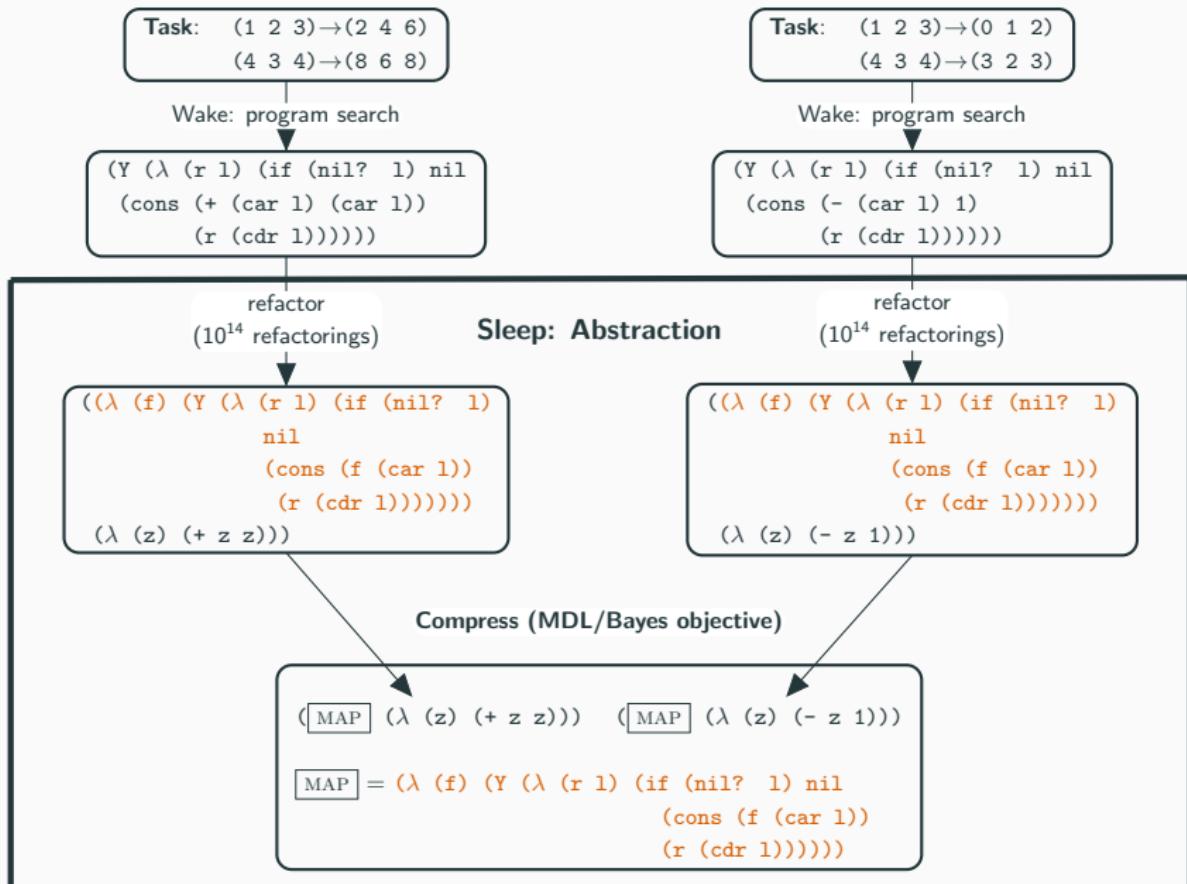
## Sleep: Abstraction

refactor  
 $(10^{14}$  refactorings)

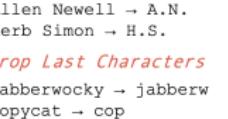
```
((λ (f) (Y (λ (r 1) (if (nil? 1)  
                           nil  
                           (cons (f (car 1))  
                                 (r (cdr 1)))))))  
  (λ (z) (+ z z)))
```

```
((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>Length</b> [■■■■■] → 4 [■■■■■■] → 6 [■■■■] → 3	<b>Physics</b> $KE = \frac{1}{2} m  \vec{v} ^2$ $\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$ $\vec{F} \propto \frac{q_1 q_2}{ \vec{r}'_1 - \vec{r}'_2 ^2} \hat{\vec{r}} = \hat{\vec{r}}$ $R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$

# DreamCoder Domains

## List Processing

### Sum List

[1 2 3] → 6

[4 6 8 1] → 17

### Double

[1 2 3 4] → [2 4 6 8]

[6 5 1] → [12 10 2]

### Check Evens

[0 2 3] → [T T F]

[2 4 9 6] → [T T F T]

## Text Editing

### Abbreviate

Allen Newell → A.N.

Herb Simon → H.S.

### Drop Last Characters

jabberwocky → jabberw

copycat → cop

### Extract

see spot(run) → run

a (bee) see → bee

## Regexes

### Phone Numbers

(555) 867-5309

(650) 555-2368

### Currency

\$100.25

\$4.50

### Dates

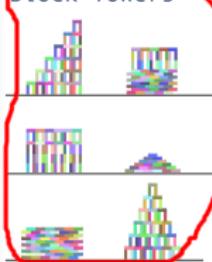
Y1775/0704

Y2000/0101

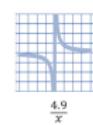
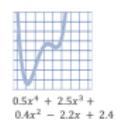
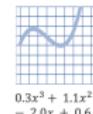
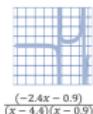
## LOGO Graphics



## Block Towers



## Symbolic Regression



## Recursive Programming

### Filter

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

### Length

[■■■■] → 4  
[■■■■■] → 6  
[■■■] → 3

### Index List

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

### Every Other

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

## Physics

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

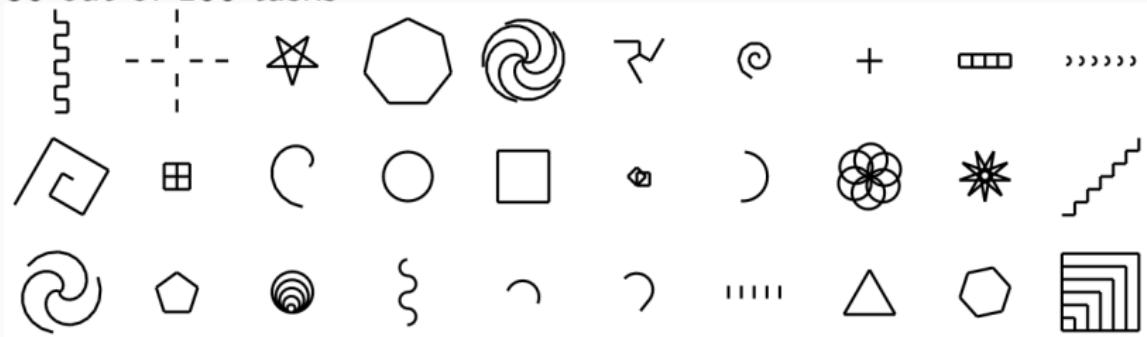
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

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

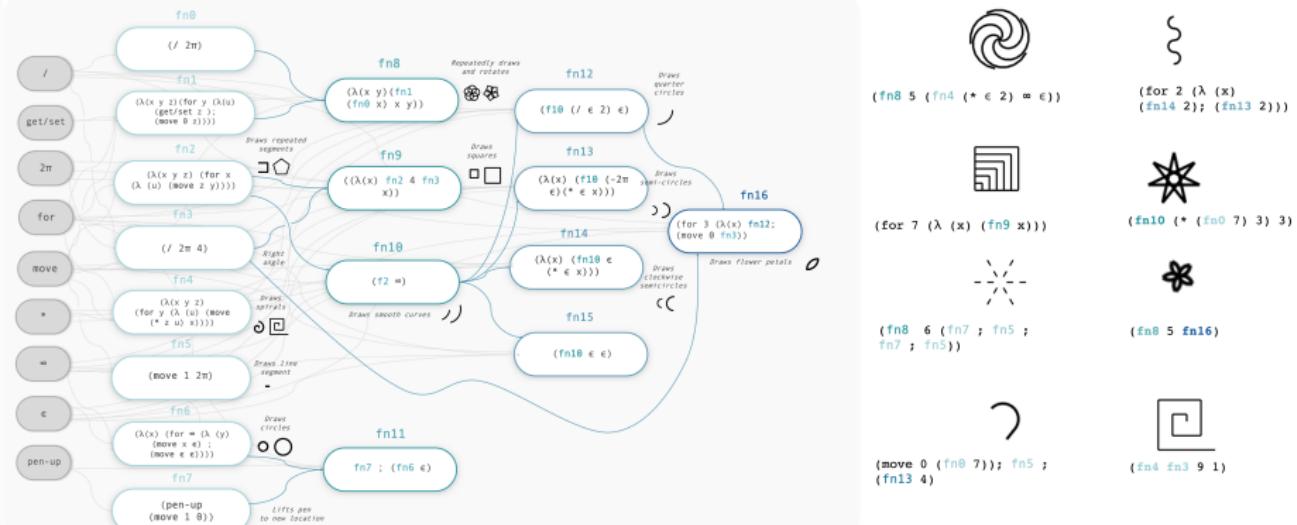
$$V_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$$

# LOGO Graphics

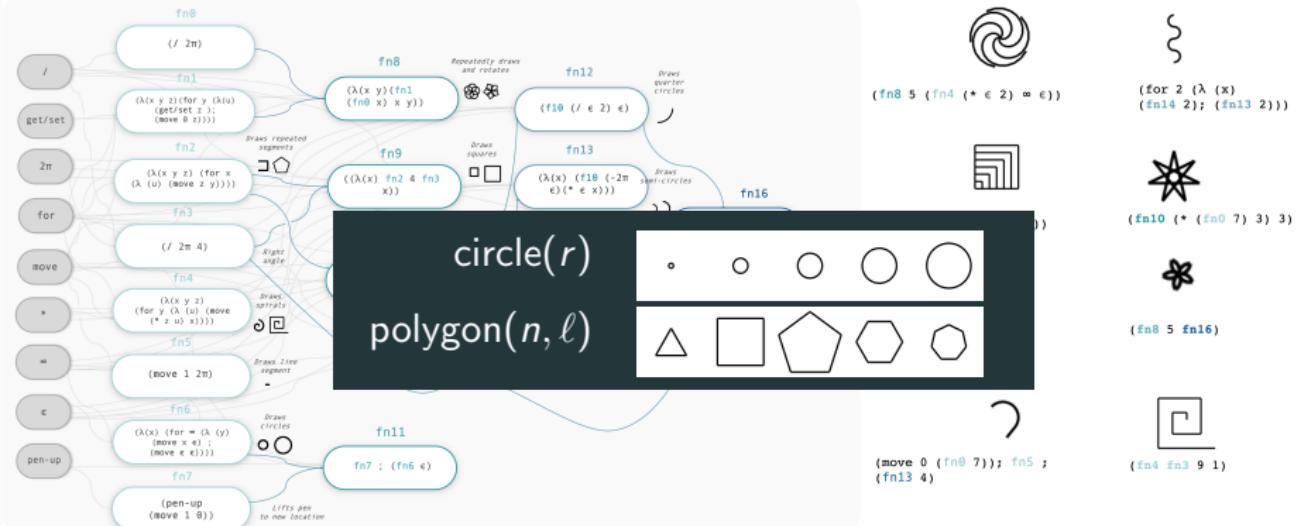
30 out of 160 tasks



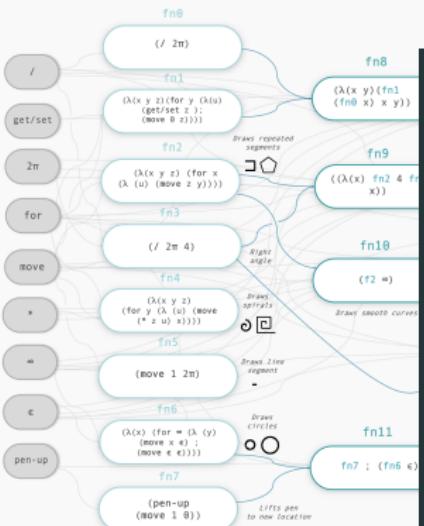
# LOGO Graphics – learning interpretable library of concepts



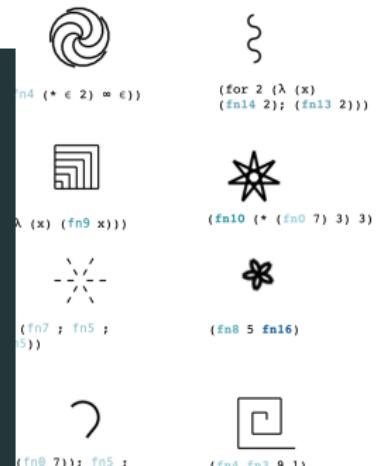
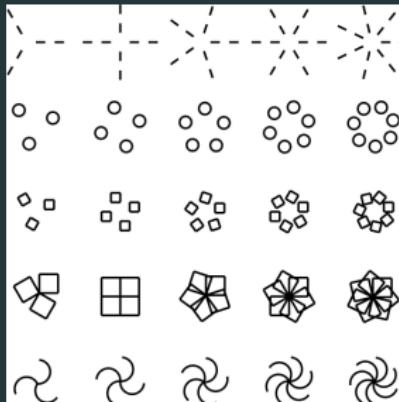
# LOGO Graphics – learning interpretable library of concepts



# LOGO Graphics – learning interpretable library of concepts



radial symmetry( $n$ , body)

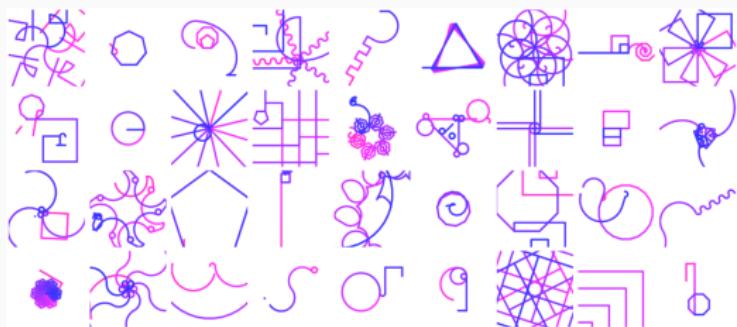


# what does DreamCoder dream of?

before learning

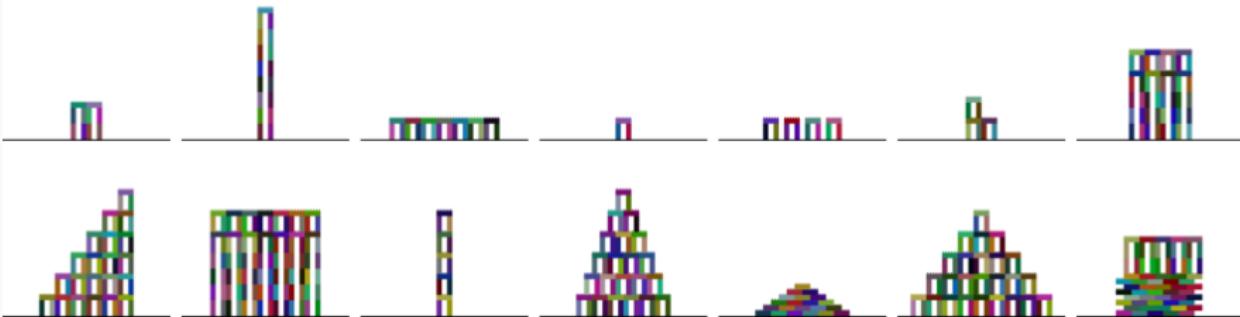


after learning



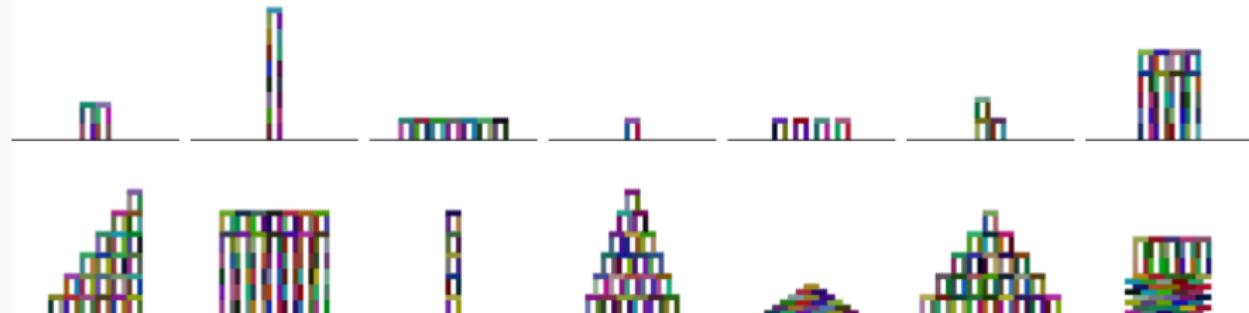
# Planning to build towers

example tasks (112 total)

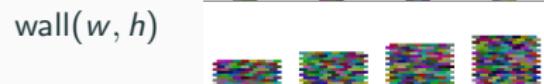
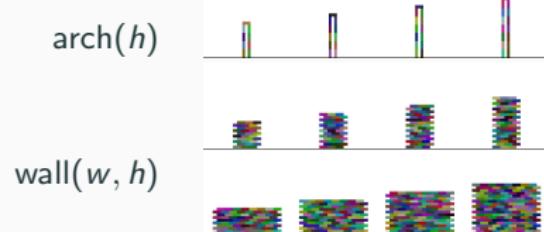


# Planning to build towers

example tasks (112 total)

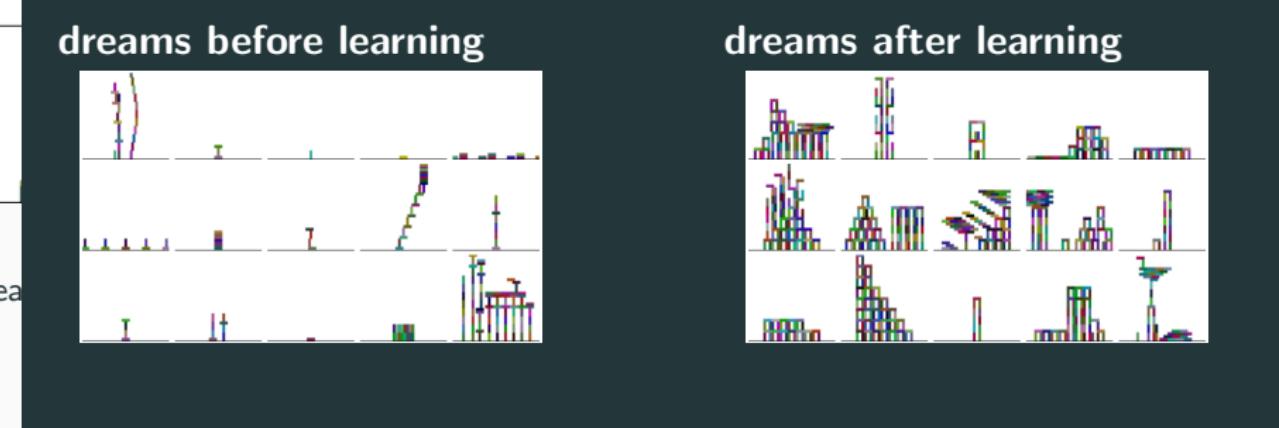


learned library routines ( $\approx 20$  total)

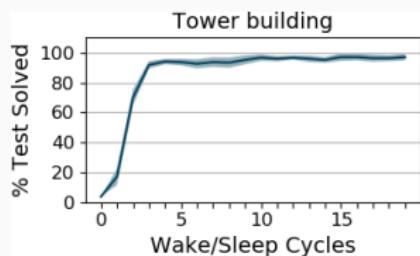


# Planning to build towers

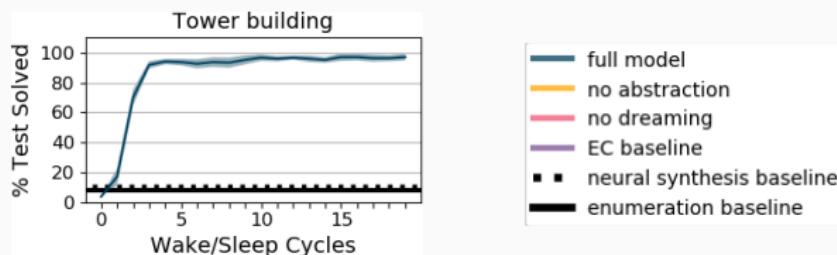
example tasks (112 total)



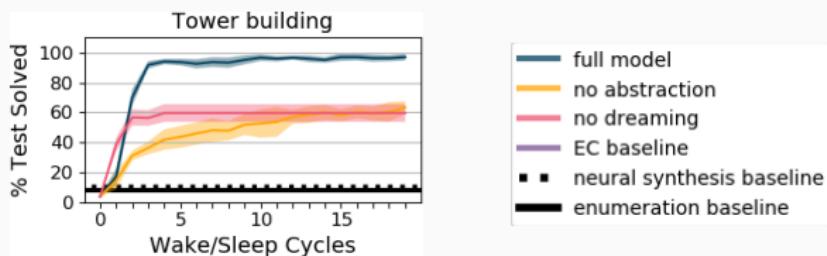
# Synergy between dreaming and library learning



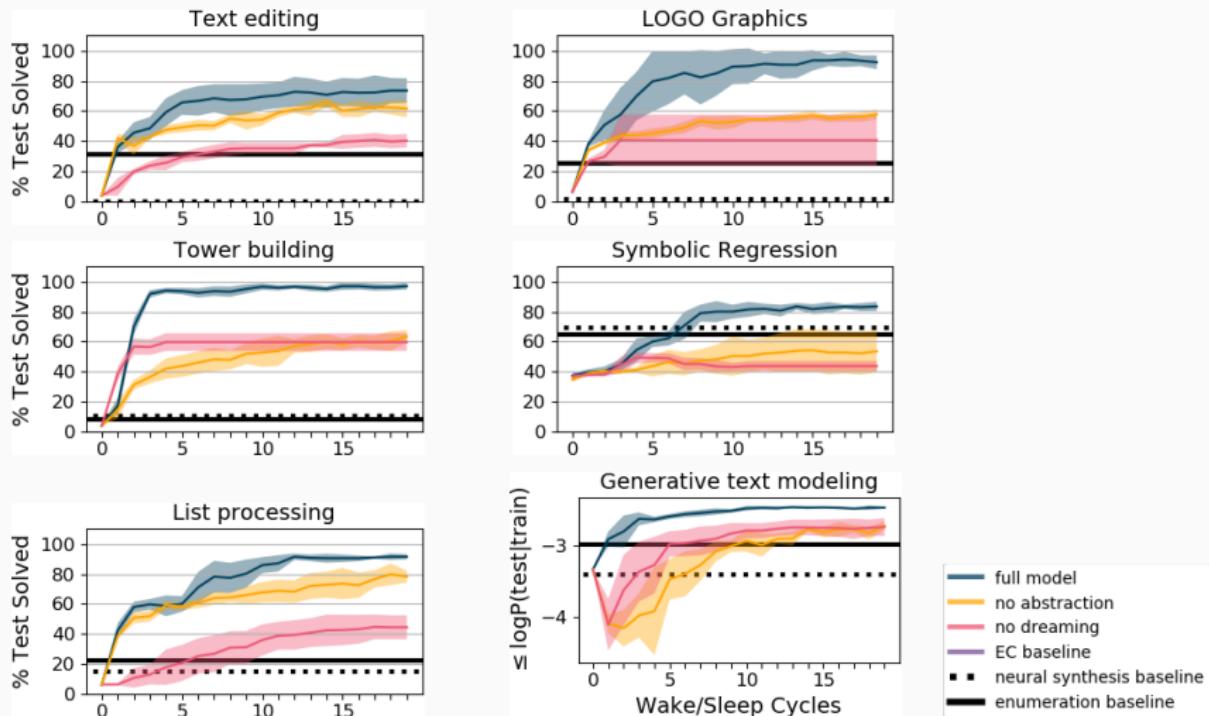
# Synergy between dreaming and library learning



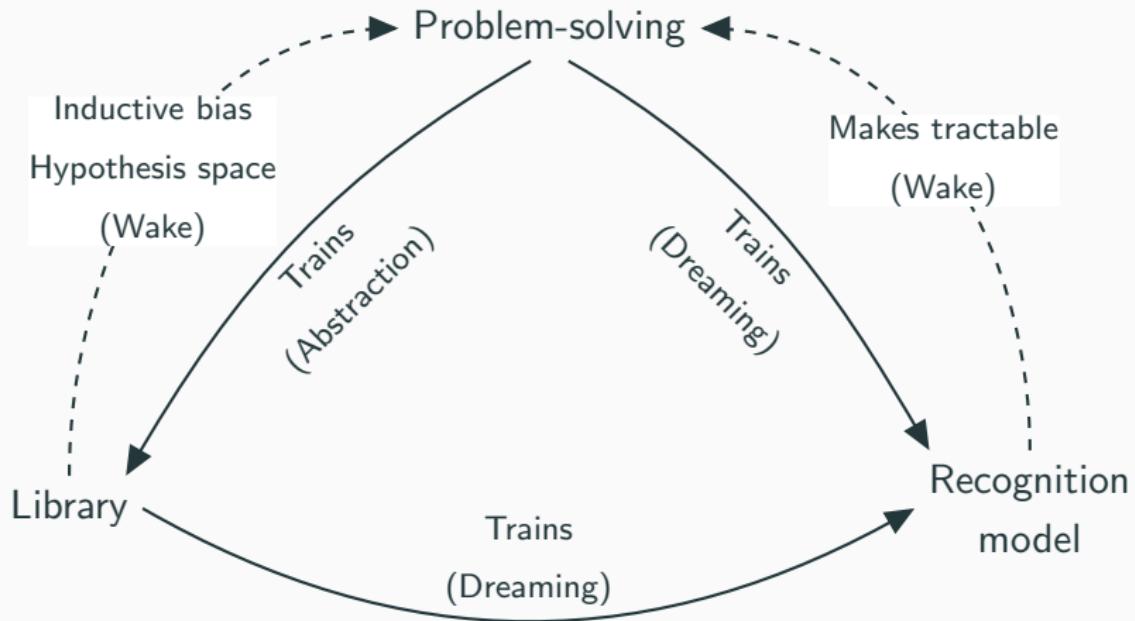
# Synergy between dreaming and library learning



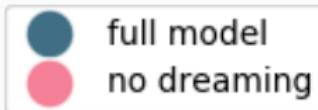
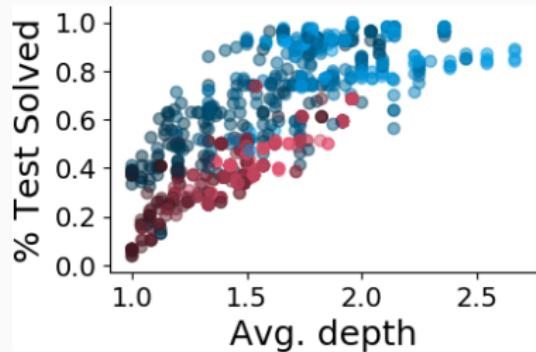
# Synergy between dreaming and library learning



# synergy between dreaming and library learning



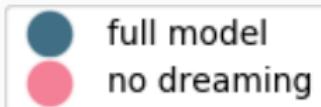
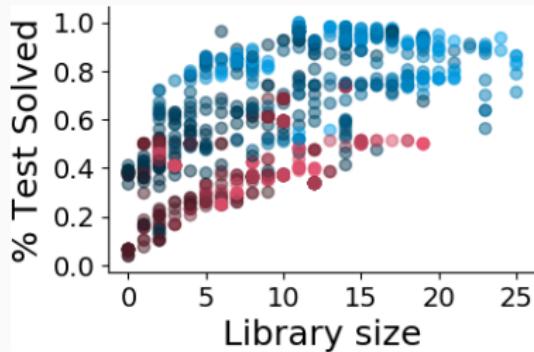
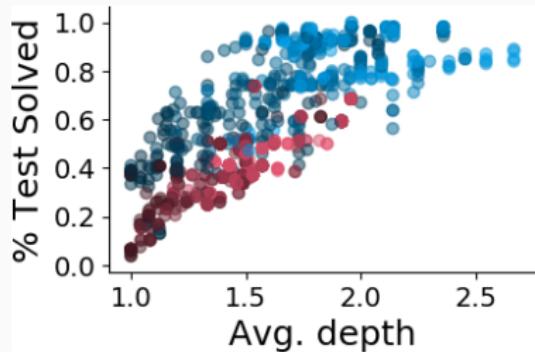
# Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Lighter: Later in learning

# Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

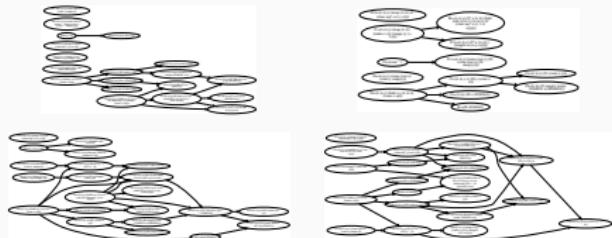
Lighter: Later in learning

# Variability in learned library

List processing



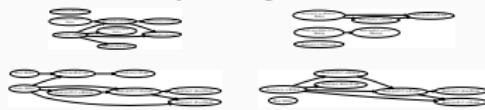
Text editing



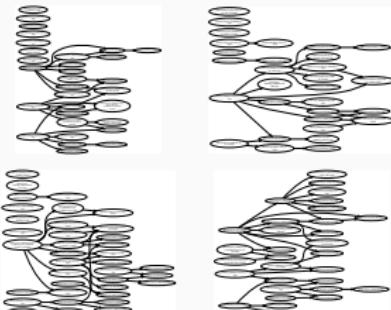
Tower building



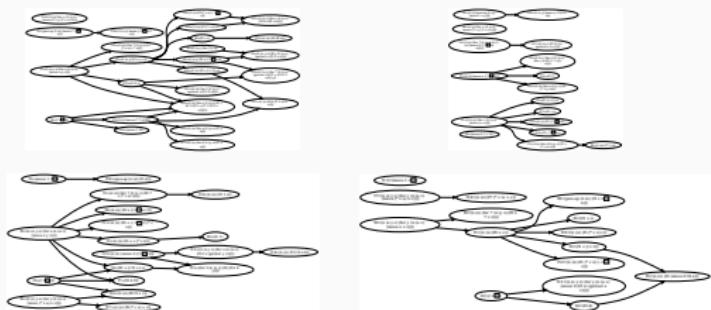
Symbolic regression



Generative regex



LOGO graphics



From learning libraries,  
to learning languages

From learning libraries,  
to learning languages

functional programming → physics

From learning libraries,  
to learning languages

1950's Lisp → modern functional programming → physics

# Growing languages for vector algebra and physics

## Initial Primitives

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

## Physics Equations

### Newton's Second Law

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

### Parallel Resistors

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

### Work

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

### Force in a Magnetic Field

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

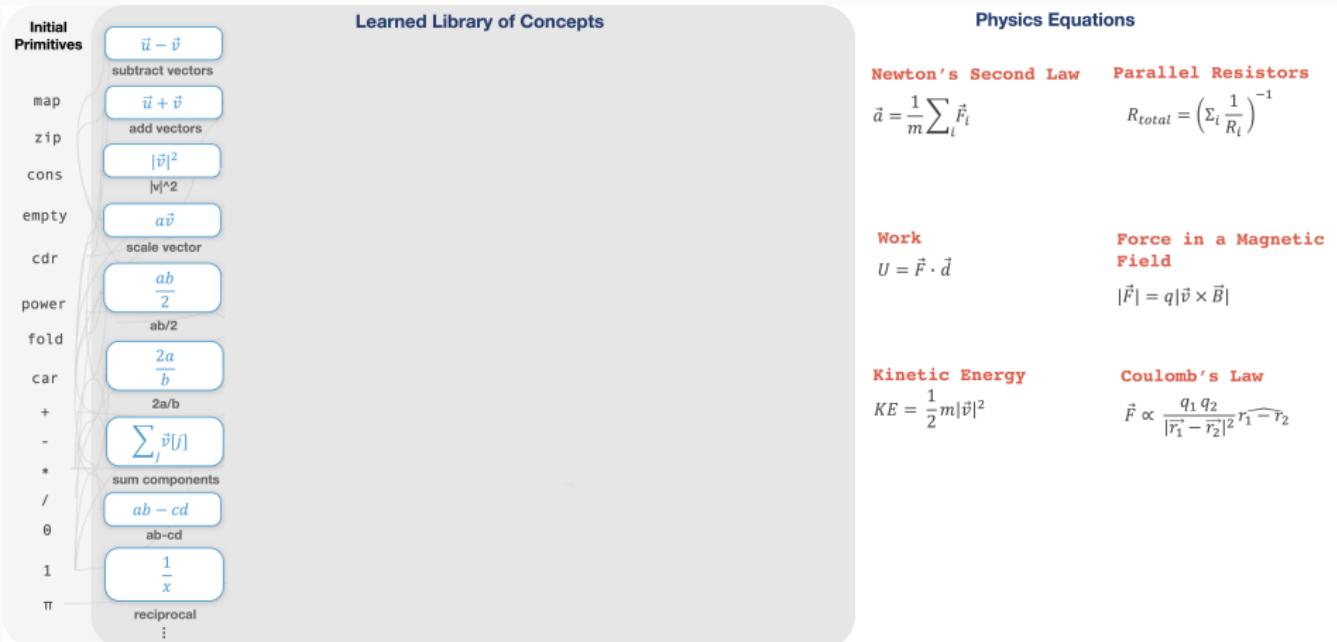
### Kinetic Energy

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

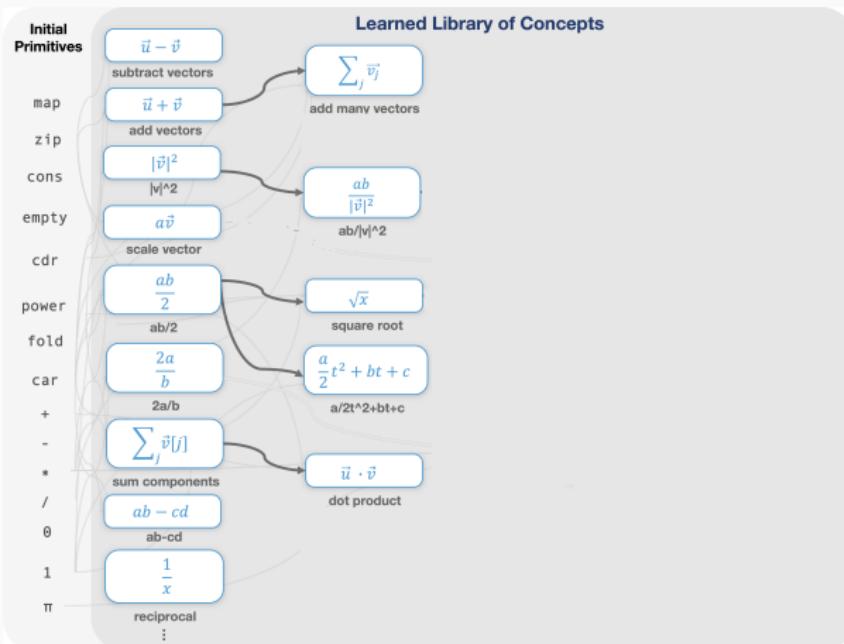
### Coulomb's Law

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

# Growing languages for vector algebra and physics

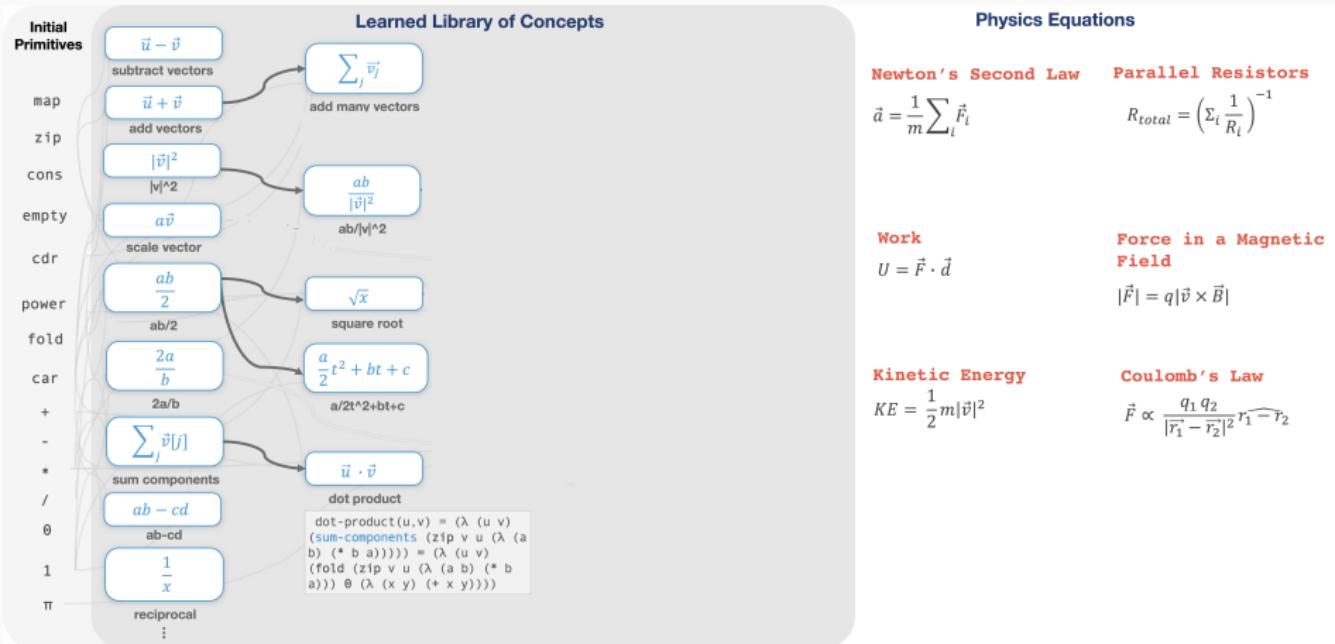


# Growing languages for vector algebra and physics

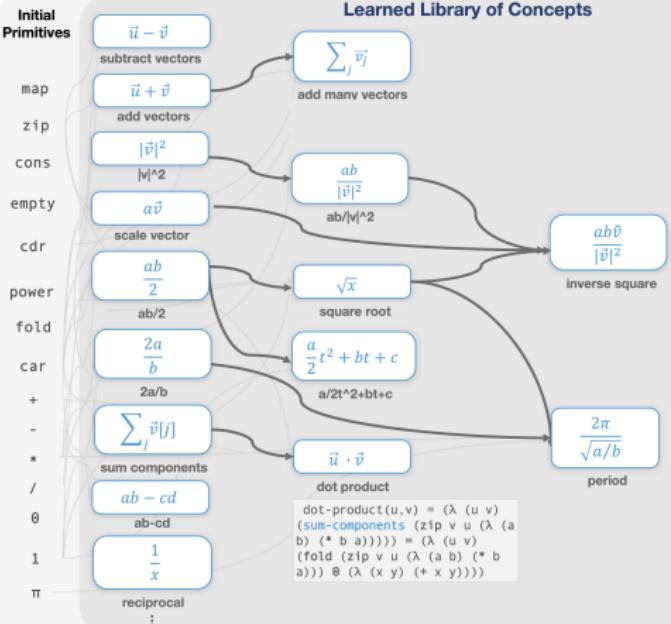


Physics Equations	
Newton's Second Law	Parallel Resistors
$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$	$R_{total} = \left( \sum_i \frac{1}{R_i} \right)^{-1}$
Work	Force in a Magnetic Field
$U = \vec{F} \cdot \vec{d}$	$ \vec{F}  = q \vec{v} \times \vec{B} $
Kinetic Energy	Coulomb's Law
$KE = \frac{1}{2} m  \vec{v} ^2$	$\vec{F} \propto \frac{q_1 q_2}{ \vec{r}_1 - \vec{r}_2 ^2} \hat{r}_1 - \hat{r}_2$

# Growing languages for vector algebra and physics



# Growing languages for vector algebra and physics



## Physics Equations

Newton's Second Law

Parallel Resistors

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

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

## Work

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

## Force in a Magnetic Field

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

## Kinetic Energy

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

## Coulomb's Law

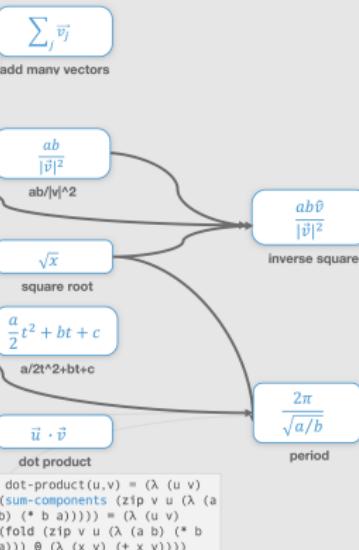
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{r}_1 - \hat{r}_2$$

# Growing languages for vector algebra and physics

Initial  
Primitives

$\vec{u} - \vec{v}$   
subtract vectors  
map  
 $\vec{u} + \vec{v}$   
add vectors  
cons  
 $|\vec{v}|^2$   
 $|v|^2$   
empty  
 $a\vec{v}$   
scale vector  
cdr  
 $\frac{ab}{2}$   
 $ab/2$   
power  
fold  
car  
 $\frac{2a}{b}$   
 $2a/b$   
+  
-  
 $\sum_j \vec{v}[j]$   
sum components  
/  
 $ab - cd$   
 $ab - cd$   
1  
 $\frac{1}{x}$   
reciprocal  
⋮

Learned Library of Concepts



Physics Equations

Newton's Second Law

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

Parallel Resistors

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

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

Work

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

(dot-product F d)

Force in a Magnetic Field

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

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

Kinetic Energy

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

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

Coulomb's Law

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

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

# Growing languages for vector algebra and physics

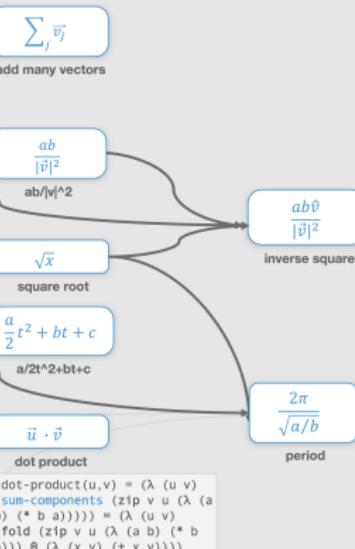
Initial  
Primitives

```

 $\vec{u} - \vec{v}$ 
subtract vectors
map
 $\vec{u} + \vec{v}$ 
add vectors
cons
 $|\vec{v}|^2$ 
 $|v|^2$ 
empty
 $a\vec{v}$ 
scale vector
power
 $\frac{ab}{2}$ 
 $ab/2$ 
fold
car
 $\frac{2a}{b}$ 
 $2a/b$ 
+
-
*
/
θ
1
π
:
reciprocal

```

Learned Library of Concepts



Physics Equations

Newton's Second Law

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

Parallel Resistors

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

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

Work

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

(dot-product F d)

Force in a Magnetic Field

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

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

Kinetic Energy

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

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

Coulomb's Law

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

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

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

Solution to Coulomb's Law if expressed in initial primitives

# Rediscovering origami programming

## Initial Primitives

Y

combinator

cons

car

cdr

nil

if

nil?

+

-

0

1

=

## Recursive Programming Algorithms

### Stutter

[■■] → [■■■■]

[■■■■] → [■■■■■■■■]

(fold A (λ (u v) (cons v (cons v u))) nil)

### Take every other

[■■■■] → [■■]

[■■■■■■■■] → [■■■■]

(unfold\_list cdr A nil?)

### List lengths

[■■■■], [■] → [3 1]

[■■■], [], [■] → [2 0 1]

(map A length)

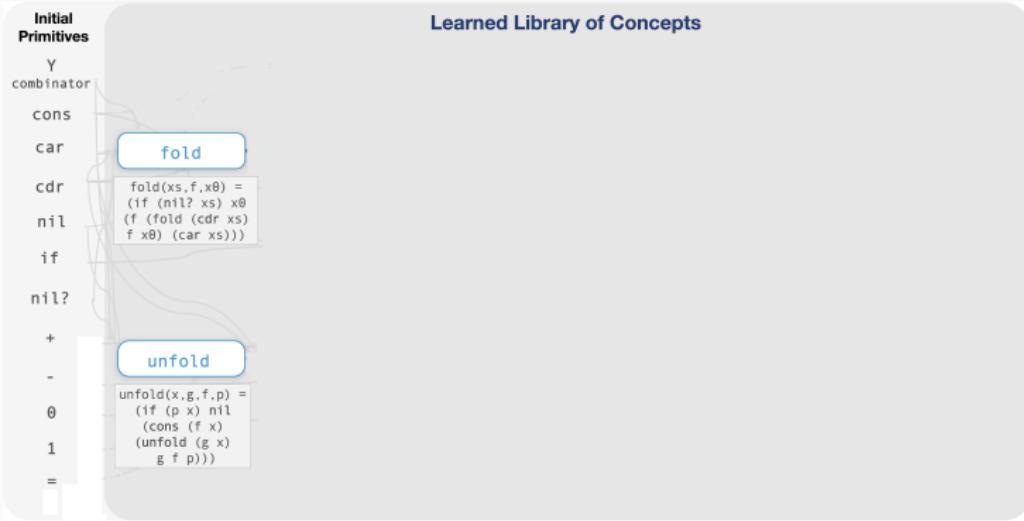
### List differences

[1 8 2], [0 5 1] → [1 3 1]

[2 3 6], [1 2 4] → [1 1 2]

(zip A - B )

# Rediscovering origami programming



**Recursive Programming Algorithms**

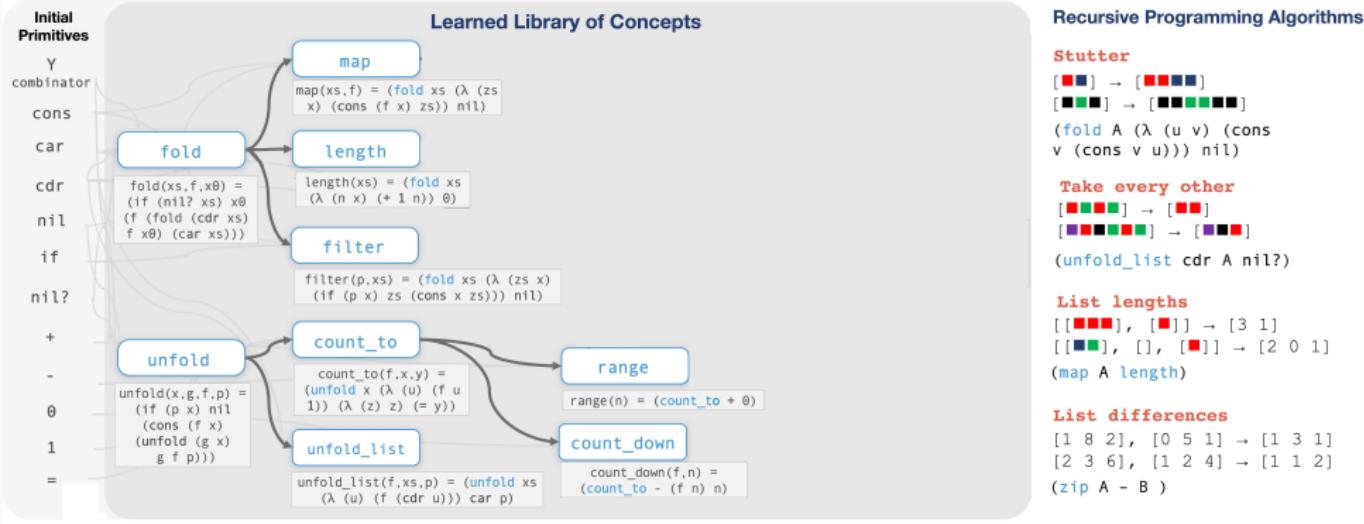
**Stutter**  
[■■] → [■■■■]  
[■■■■] → [■■■■■■]  
(fold A (λ (u v) (cons v (cons v u))) nil)

**Take every other**  
[■■■■] → [■■]  
[■■■■■■■■] → [■■■■]  
(unfold\_list cdr A nil?)

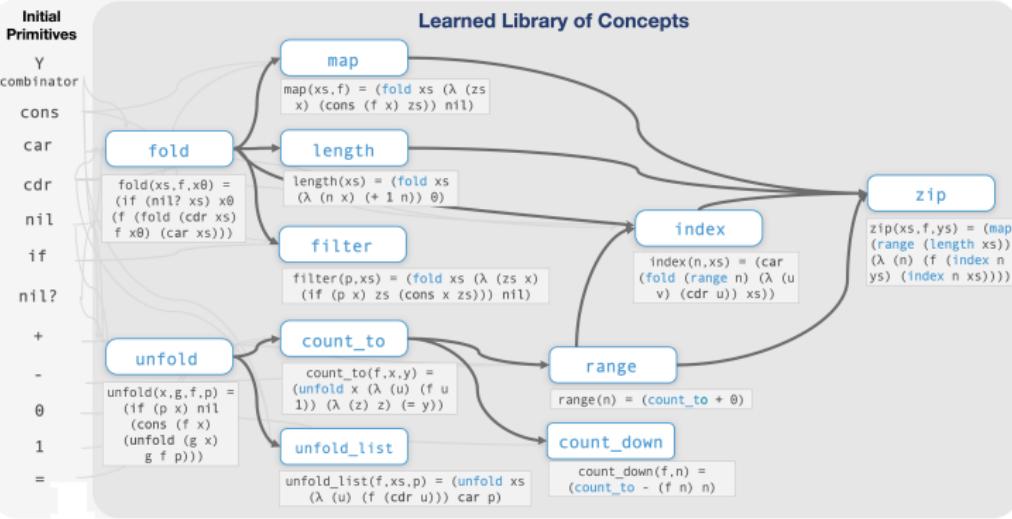
**List lengths**  
[[■■■], [■]] → [3 1]  
[[■■], [], [■]] → [2 0 1]  
(map A length)

**List differences**  
[1 8 2], [0 5 1] → [1 3 1]  
[2 3 6], [1 2 4] → [1 1 2]  
(zip A - B )

# Rediscovering origami programming



# Rediscovering origami programming



## Recursive Programming Algorithms

### Stutter

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

### Take every other

`[■■■■■■] → [■■]`  
`[■■■■■■■■■■] → [■■■■■■]`  
`(unfold_list cdr A nil?)`

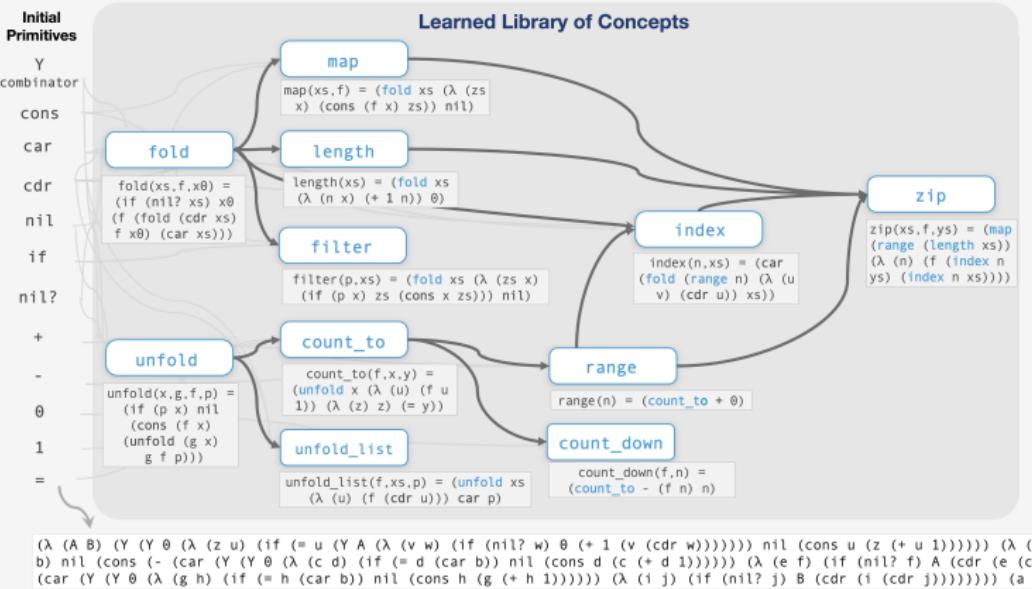
### List lengths

`[[■■■], [■]] → [3 1]`  
`[[■■■■], [], [■]] → [2 0 1]`  
`(map A length)`

### List differences

`[1 8 2], [0 5 1] → [1 3 1]`  
`[2 3 6], [1 2 4] → [1 1 2]`  
`(zip A - B )`

# Rediscovering origami programming



## Recursive Programming Algorithms

### Stutter

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

### Take every other

[■■■■■■] → [■■]  
 [■■■■■■■■] → [■■■■]  
`(unfold_list cdr A nil?)`

### List lengths

[■■■■], [■■] → [3 1]  
 [[■■■], [], [■■]] → [2 0 1]  
`(map A length)`

### List differences

[1 8 2], [0 5 1] → [1 3 1]  
 [2 3 6], [1 2 4] → [1 1 2]  
`(zip A - B )`

Solution  
if expressed in  
initial primitives

## Lessons

Symbols aren't necessarily interpretable. Flexibly grow the language based on experience to make it more powerful *and* more human understandable

Learning-from-scratch is possible in principle. Don't do it. Choose wisely what to learn and what to build in

Program Induction and perception  
learning to learn  
discovering models

# Synthesizing human-understandable models of language



# Synthesizing human-understandable models of language

many languages, 70 diverse benchmarks

children and linguists can learn from sparse data

linguists can communicate their knowledge



# 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

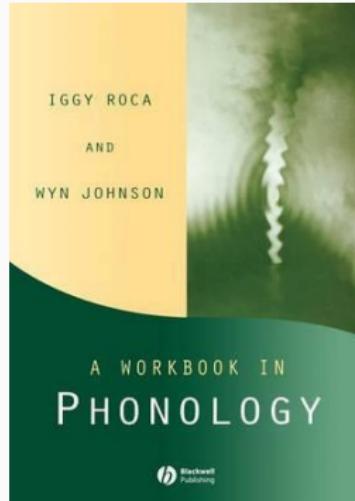
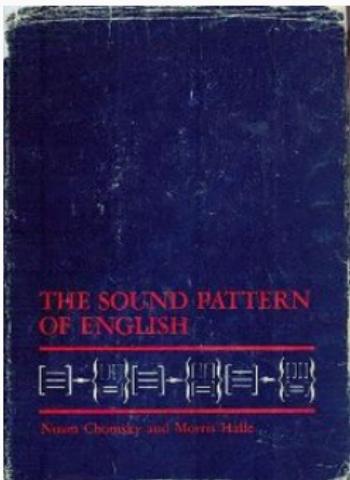
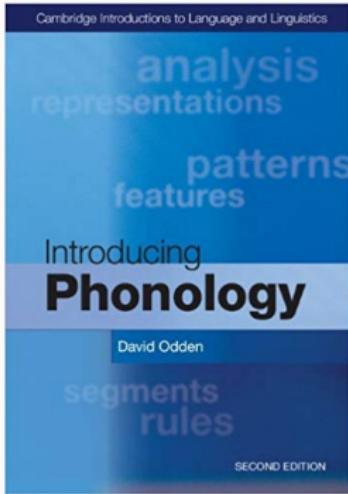
Serbo-Croatian:

	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

	SINGULAR	PLURAL
BED	orон	ороннор
MARE	bie	bieler
CABINET	ı̄skaap	ı̄skaaptartar

138 total examples

# Turkic Sakha (Yakut)

grammar (unobserved)

SINGULAR → stem  
PLURAL → stem + lar

observed data



	SINGULAR	PLURAL
BED	orон	ороннор
MARE	bie	bieler
CABINET	їскаап	їскаaptartar

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"

observed data

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	ı̄skaap	ı̄skaaptartar

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 "m"

## stems (unobserved)

BED : oron  
MARE : bie  
CABINET : ıskAAP

## observed data

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	ıskAAP	ıskAAPtar

138 total examples

## Turkic Sakha (Yakut)

## grammar (unobserved)

SINGULAR → stem  
PLURAL → stem + lar

- $r_1$ :  $I \rightarrow d / [-\text{lateral} \ -\text{tense}]$   
"I" 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

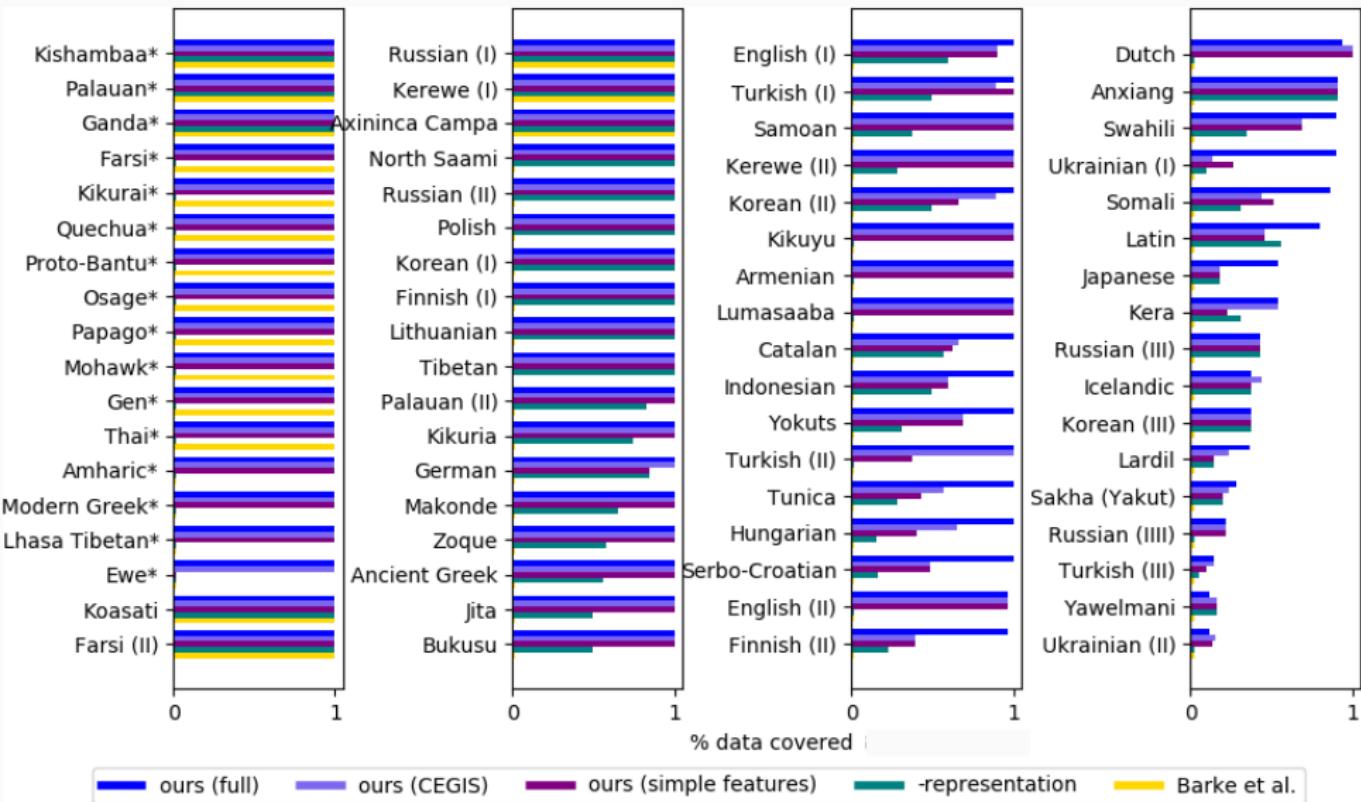
$r_6$ :  $[-\text{sonorant} \ +\text{voice}] \rightarrow [+\text{nasal}] / [+\text{nasal}]$   
"nasalize" consonant next to a nasal, like "m"

stems  
(unobserved)

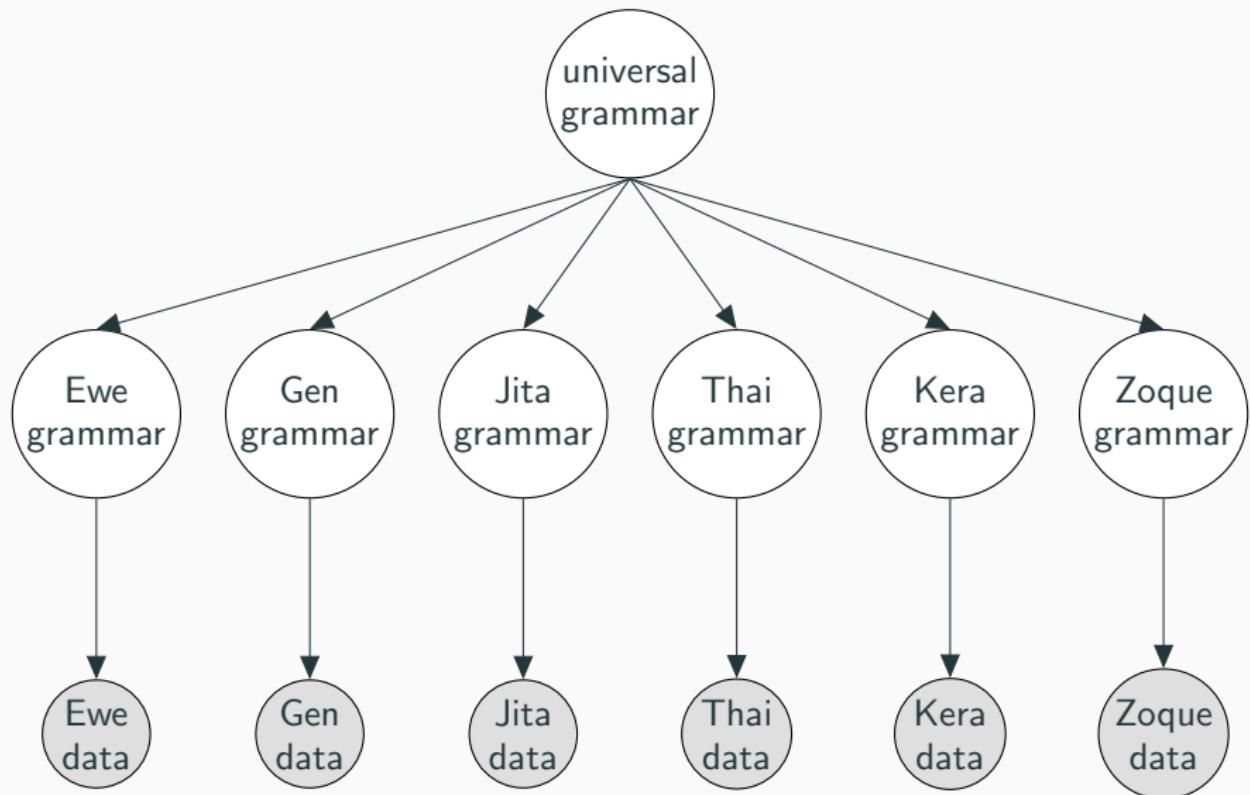
- CABINET : *iskaap*  
BED : *oron*  
MARE : *bie*  
RIVER : *örus*

## 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+liin → örusliin  $\xrightarrow{r_1}$  örusdiin  $\xrightarrow{r_2}$   
                           $\xrightarrow{r_3}$  örurstiin  $\xrightarrow{r_5}$  örurstuun → [örüstüün]



# Distilling higher-level knowledge



## Lessons

Higher-level knowledge matters (“universal grammar”). Get the basic computational substrate correct

But *some* of this higher-level knowledge can be learned. You don't need millions of examples to learn it. But it's not a one-shot learning problem either

Program Induction and perception  
learning to learn  
model discovery  
the future

# Models of the physical world

hinge



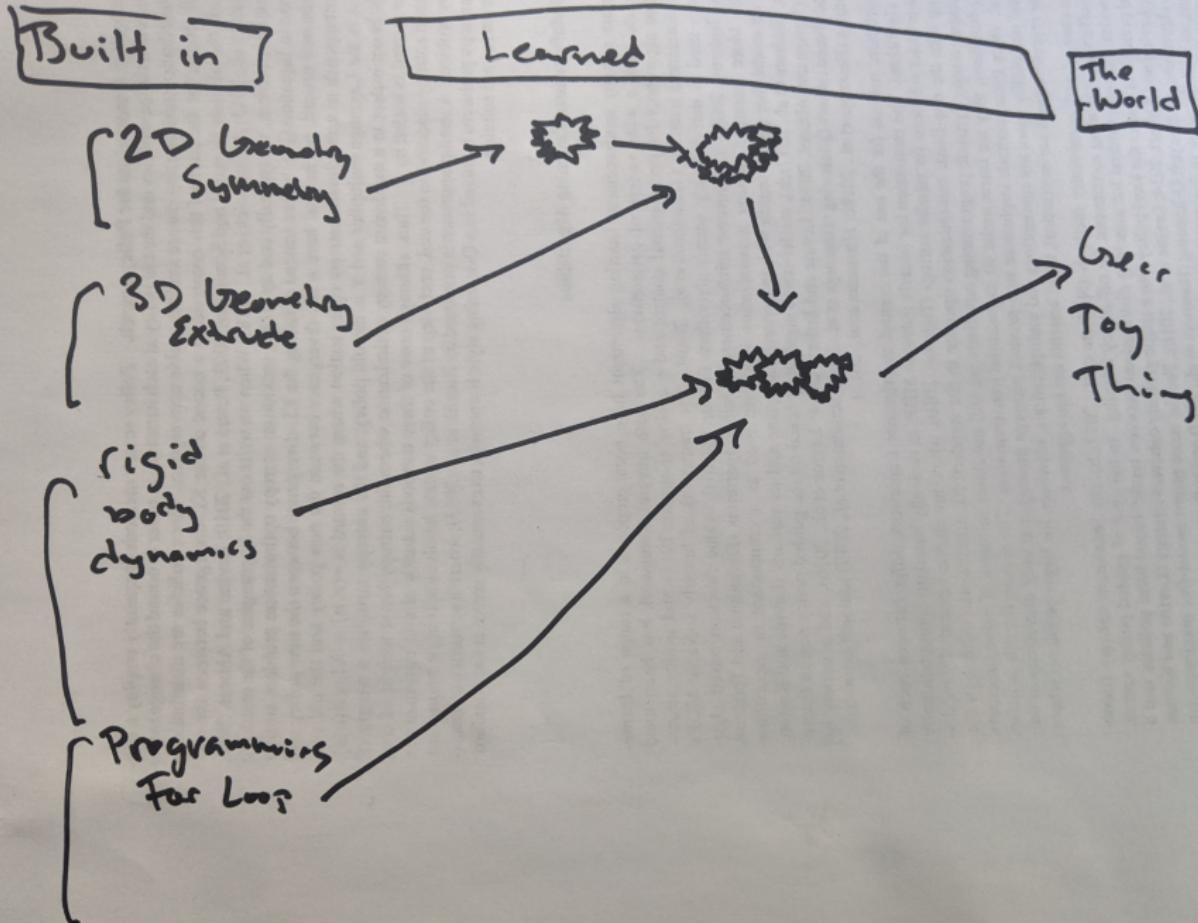
gear



doorknob







# Collaborators

Tim O'Donnell



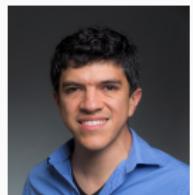
Josh Tenenbaum



Adam Albright



Armando  
Solar-Lezama



Max Nye



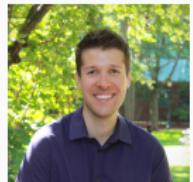
Cathy Wong



Yewen Pu



Dan Ritchie



Mathias Sable-Meyer



Lucas Morales



thank you