

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[{(hw - hw^2) == z}, h]`

Out[34]= {}

Input interpretation: solve $hw - hw^2 = z$ 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]
```

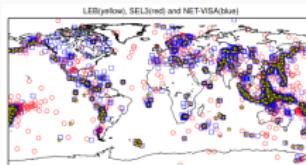
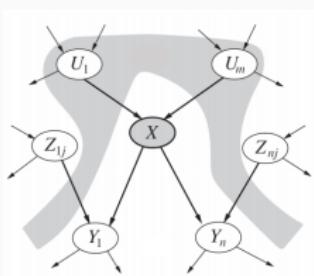
Input interpretation:

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]= {}
```

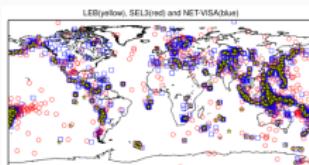
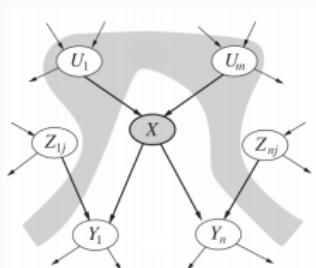
```
Input interpretation:
```

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

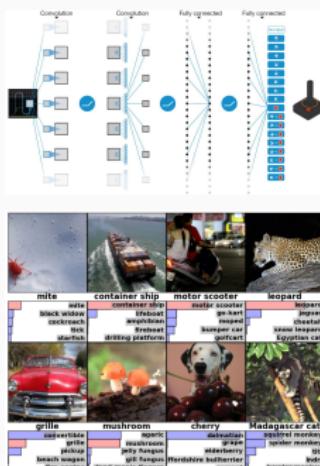
Result:

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

Probabilistic



Neural



What computational frameworks can contribute to this picture?

Three AI traditions

Symbolic



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

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

Program induction

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

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

```
Out[33]=
```

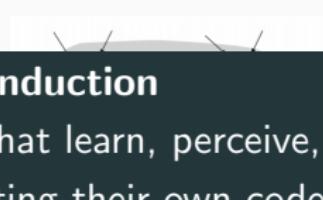
Input interpretation:

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

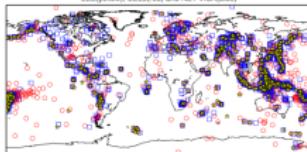
Result:

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

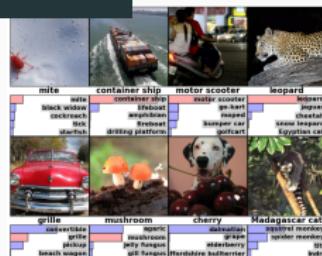
Probabilistic



LEI(yellow), SEL3(red) and NET-VISA(blue)



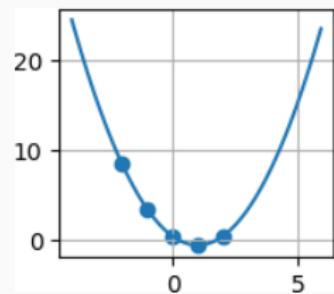
Neural



Why program induction?

Why program induction?

strong generalization
+data efficiency

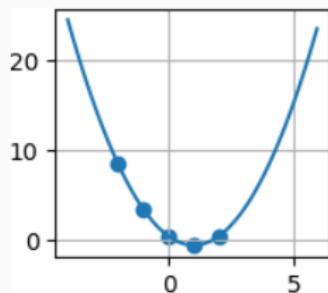


$$f(x) = (x-1)^{**2} - 0.5$$

Why program induction?

strong generalization
+data efficiency

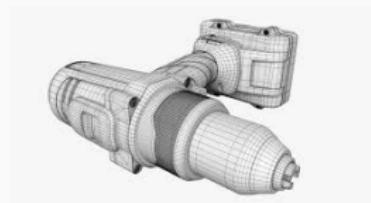
interpretability



$$f(x) = (x-1)^2 - 0.5$$

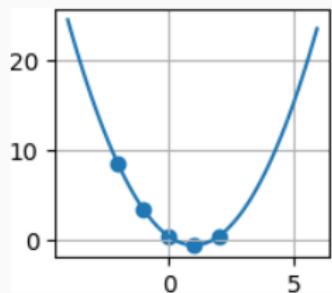


VS



Why program induction?

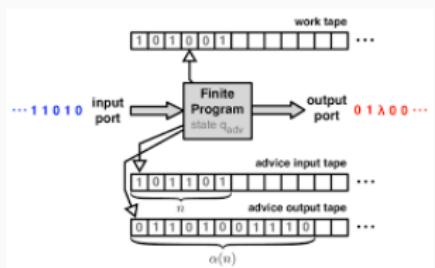
strong generalization
+data efficiency



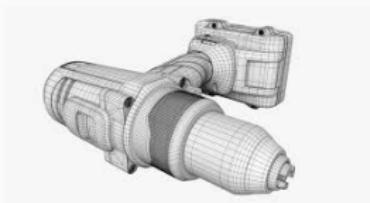
interpretability



universal expressivity



VS



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



John McCarthy, Ray Solomonoff

Why try again?

better toolkits: neural+probabilistic+symbolic, and knowing how to combine them

Why try again?

better toolkits: neural+probabilistic+symbolic, and knowing how to combine them

maturing **program synthesis** techniques

Why try again?

better toolkits: neural+probabilistic+symbolic, and knowing how to combine them

maturing **program synthesis** techniques

Program induction is already working: a roadmap of problems...

semantic parsing [Liang et al. 2011; Zettlemoyer et al. 2007]

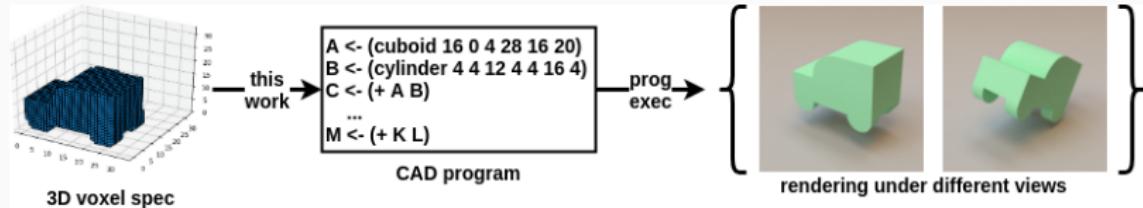
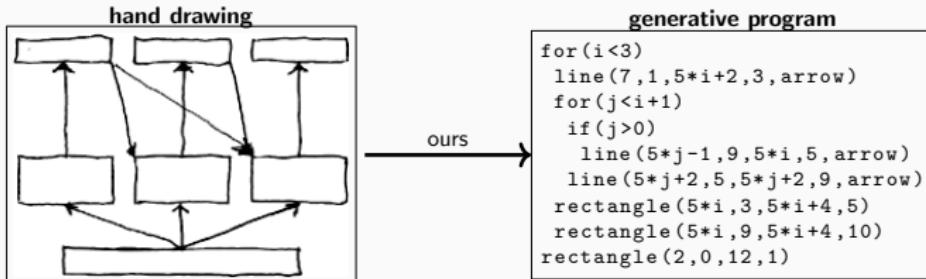
programming by examples [Gulwani 2011]

computer-aided-programming [Solar-Lezama 2008]

inverse procedural modeling [Kulkarni et al. 2015]

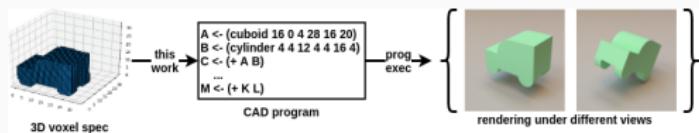
Perception, Synthesizing models, Learning-to-Learn

Theme: high-level visual understanding, pixels→programs

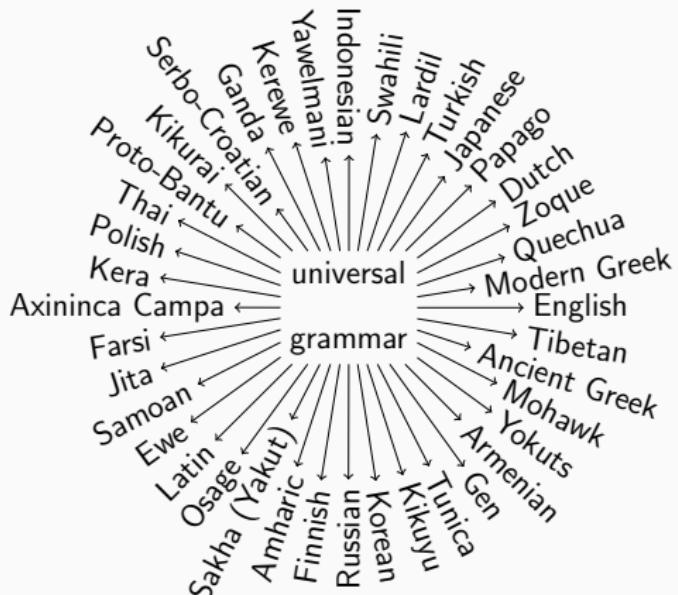


Perception, Synthesizing models, Learning-to-Learn

Theme: high-level visual understanding, pixels→programs

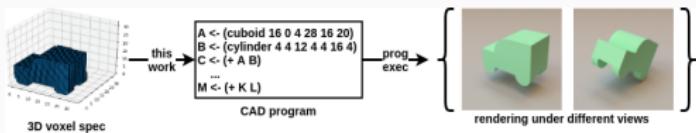


Theme: Synthesizing human-understandable models



Perception, Synthesizing models, Learning-to-Learn

Theme: high-level visual understanding, pixels→programs



Theme: Synthesizing human-understandable models

Theme: Learning to synthesize programs

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

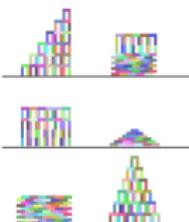
Dates

Y1775/0704
Y2000/0101

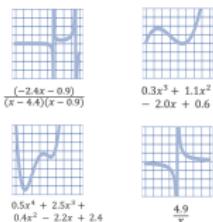
LOGO Graphics



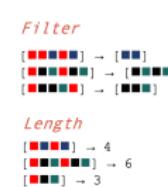
Block Towers



Symbolic Regression



Recursive Programming



Physics

$$KE = \frac{1}{2} m |\vec{v}|^2$$
$$\vec{d} = \frac{1}{m} \sum_i \vec{r}_i$$
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{\vec{r}_1} \cdot \hat{\vec{r}_2}$$
$$R_{total} = \left(\sum_i \frac{1}{R_i} \right)^{-1}$$

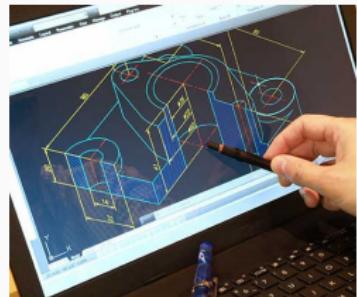
Program Induction and perception
model discovery
learning to learn

High-level, abstract visual abilities

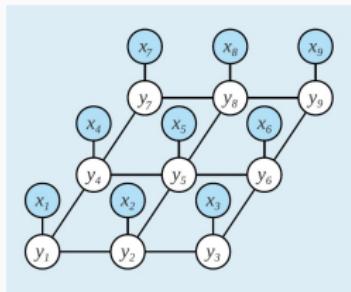
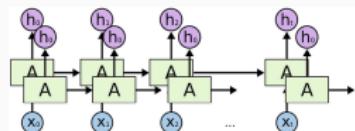
...in art



...in engineering



...in AI



High-level, abstract visual abilities

...in art



why?

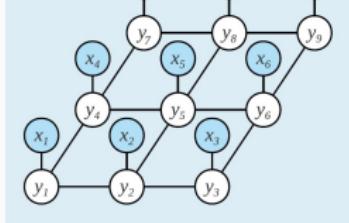
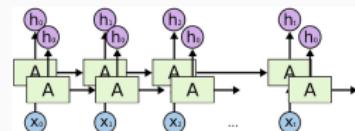
impute missing objects, extrapolate percepts,
learn visual concepts ('arch', 'spiral', 'Ising model'),
assist graphic design, assist 3D modeling

how?

machine learning +

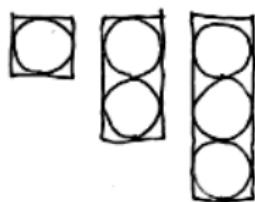
program synthesis techniques from programming languages community

...in AI



Learning to infer graphics programs from hand-drawn images

model infers program from drawing



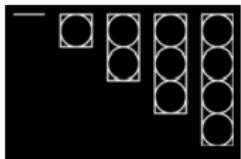
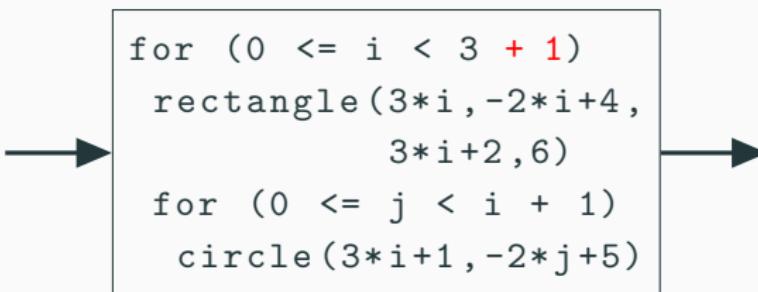
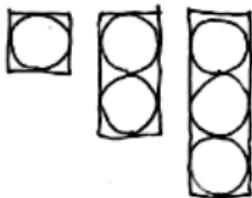
→

```
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

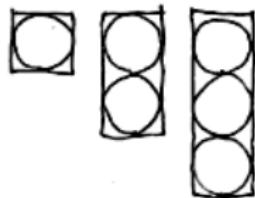
zero-shot generalization / extrapolation



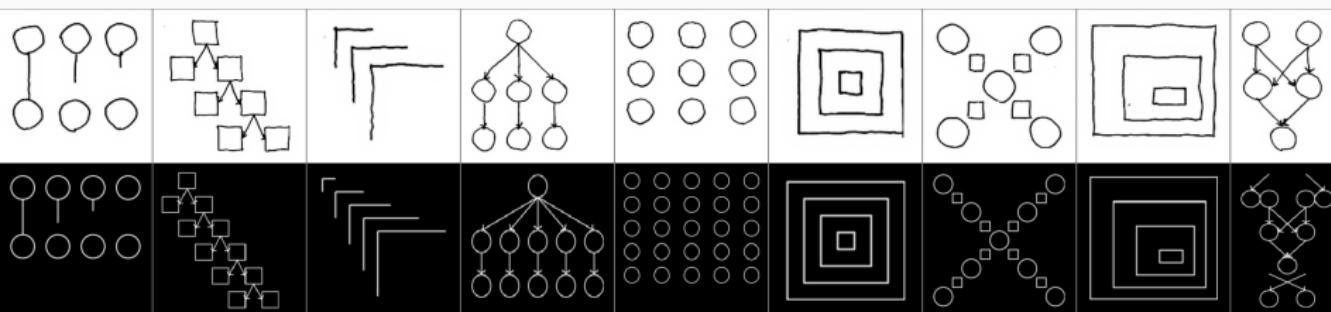
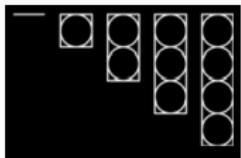
Learning to infer graphics programs from hand-drawn images

model infers program from drawing

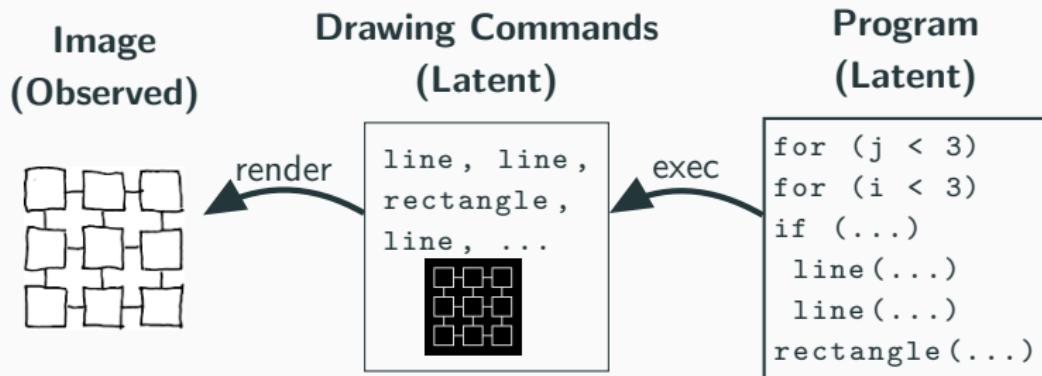
zero-shot generalization / extrapolation



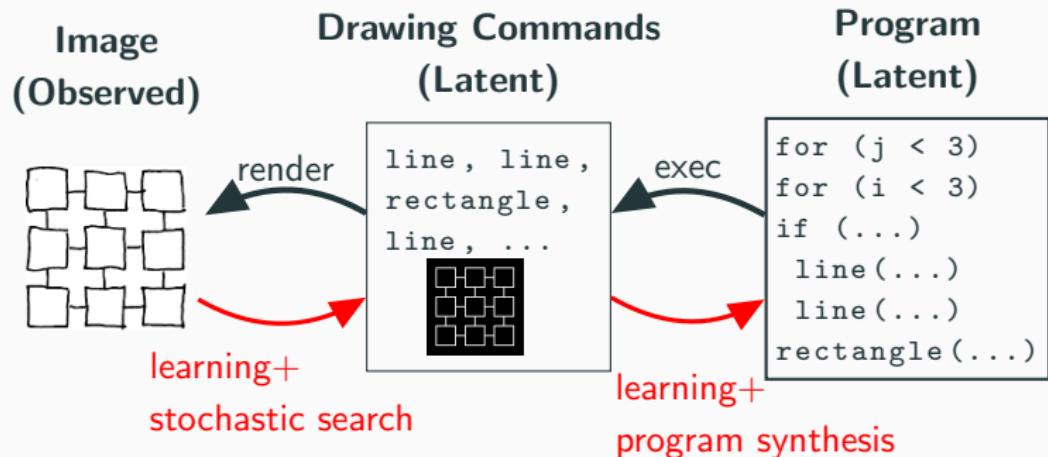
```
for (0 <= i < 3 + 1)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```



How to infer graphics programs from hand-drawn images



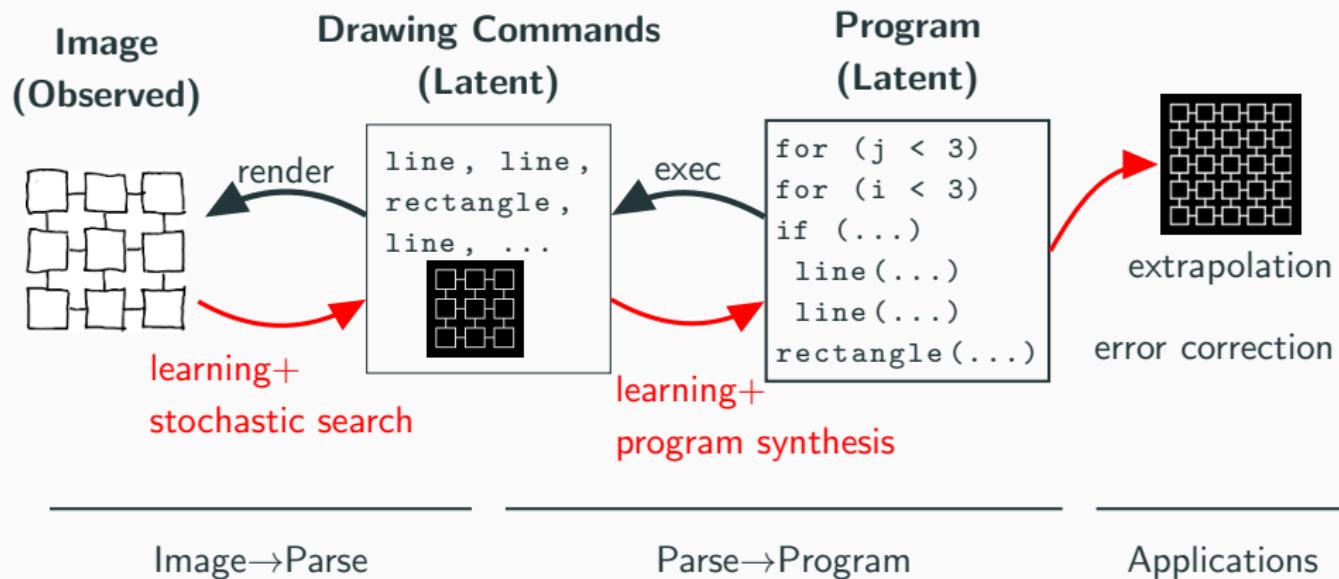
How to infer graphics programs from hand-drawn images



Image→Parse

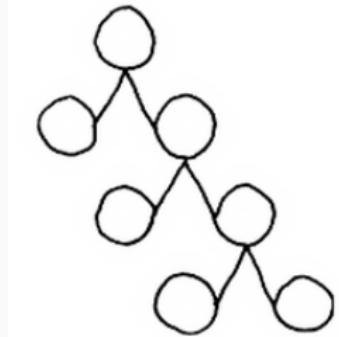
Parse→Program

How to infer graphics programs from hand-drawn images

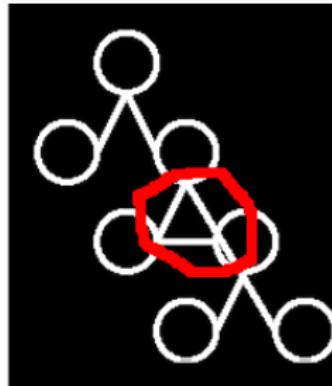


Top-down influences on perception

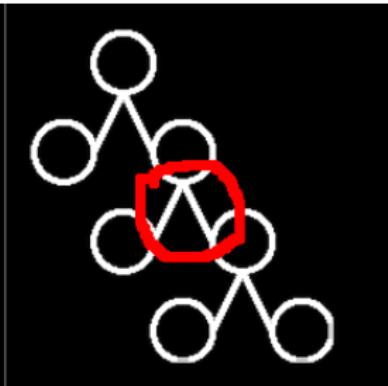
drawing



bottom-up neural net

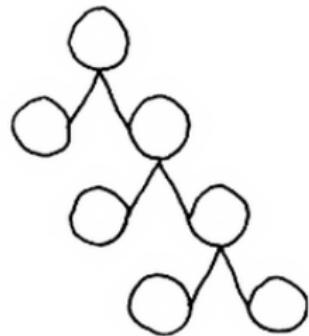


w/ top-down program bias

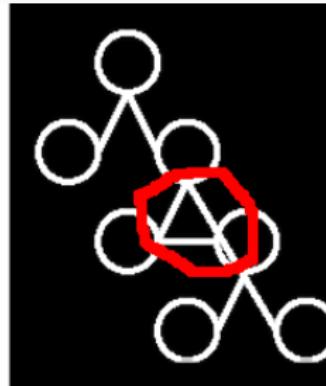


Top-down influences on perception

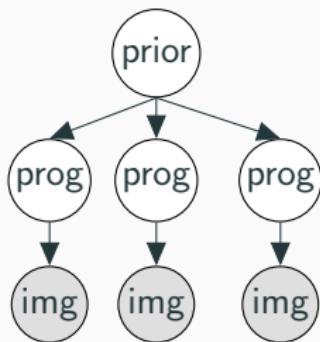
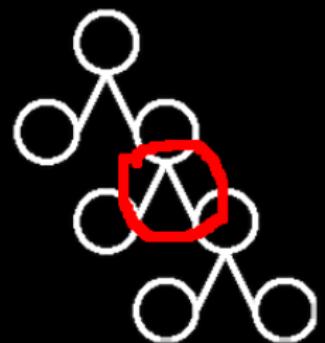
drawing



bottom-up neural net



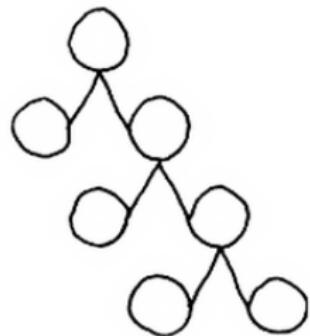
w/ top-down program bias



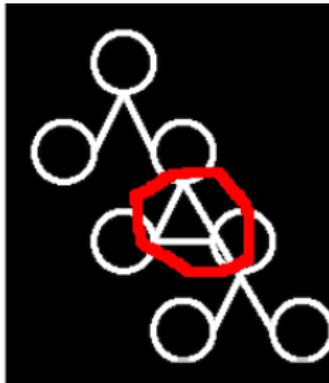
predicted program =
 $\arg \max_{\text{progs}} \mathbb{P} [\text{img} | \text{prog}] \mathbb{P} [\text{prog} | \text{prior}]$

Top-down influences on perception

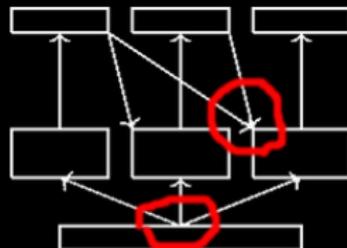
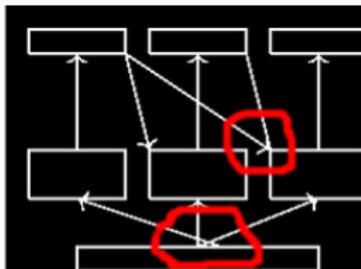
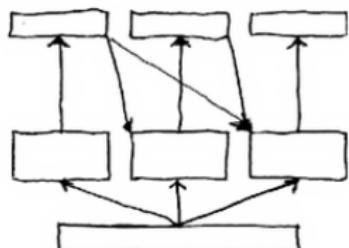
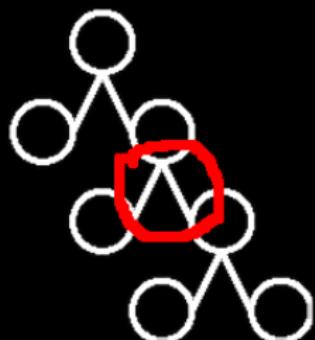
drawing



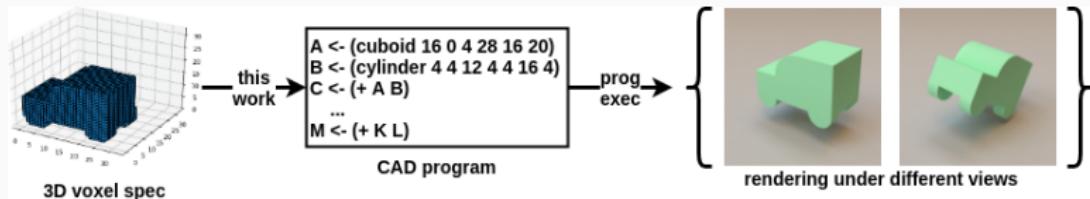
bottom-up neural net



w/ top-down program bias



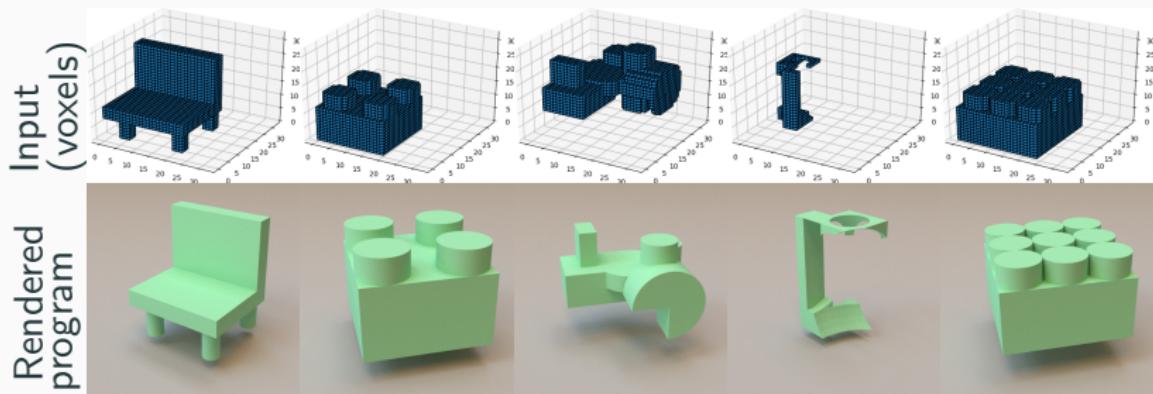
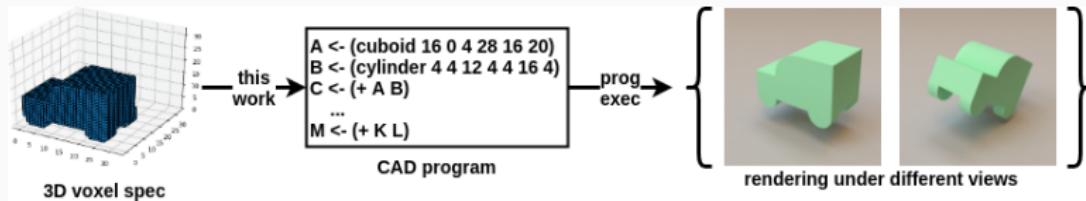
3D program induction



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

*equal contribution

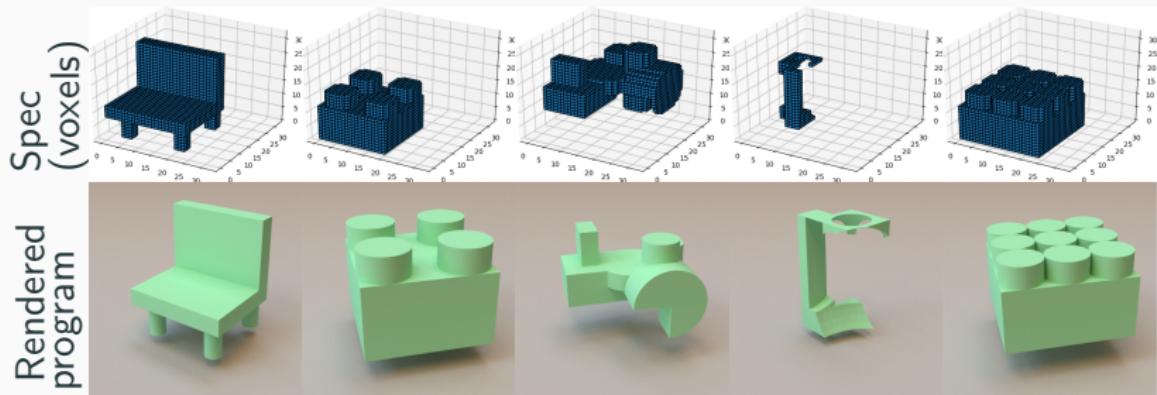
3D program induction



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

*equal contribution

3D program induction



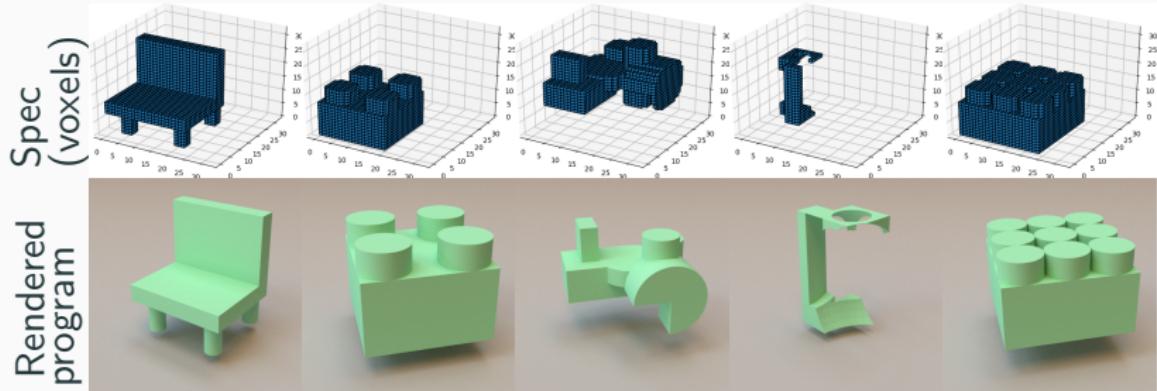
Challenge: combinatorics!

Branching factor: > 1.3 million per line of code, ≈ 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, ≈ 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

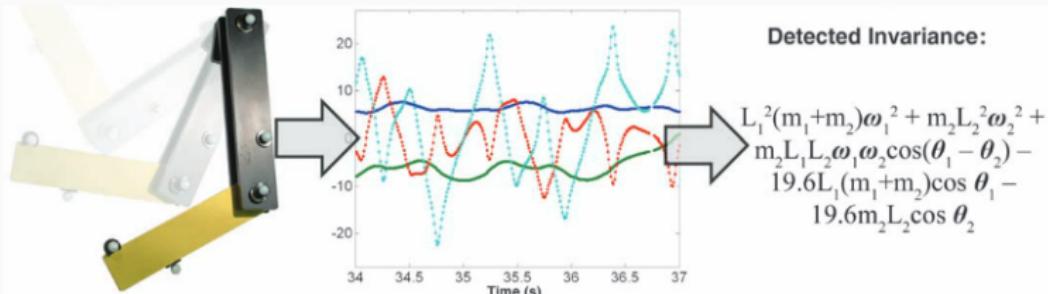
Lessons

The inductive bias from a programming language gives extrapolation, or strong generalization

Combine the best of different techniques: neural nets for perception and pattern recognition, symbols for reasoning, Bayesian methods for uncertainty

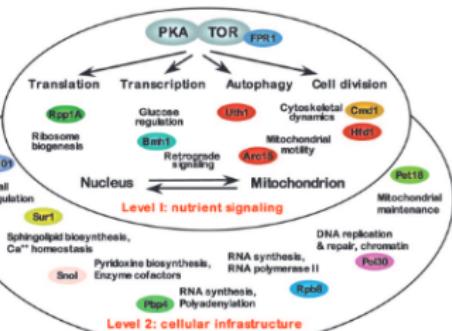
Program Induction and perception
model discovery
learning to learn

Scientific discovery



Schmidt & Lipson: "Distilling Free-Form Natural Laws from Experimental Data"

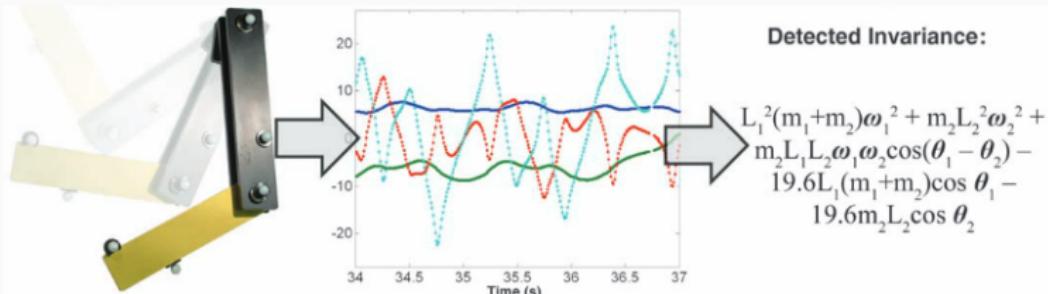
PNAS



Lezon et al. 2006

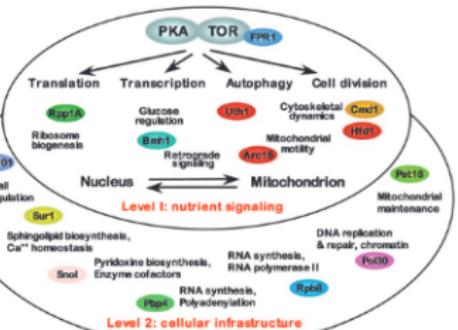
inferring genetic interaction networks

Scientific discovery



Schmidt & Lipson: "Distilling Free-Form Natural Laws from Experimental Data"

PNAS



Lezon et al. 2006

inferring genetic interaction networks

THE APPLICATION OF ALGORITHMIC PROBABILITY TO PROBLEMS IN ARTIFICIAL INTELLIGENCE

Ray J. Solomonoff

1986

This operation corresponds to Kepler's laws summarizing and compressing Tycho Brahe's empirical data on planetary motion. Algorithmic Complexity theory has this ability to synthesize, to find general laws in masses of unorganized and partially organized knowledge. It is in this area that its greatest value for A.I. lies.

Discovering human-understandable models of language



Discovering human-understandable models of language

many languages, 70 diverse benchmarks

linguists distill out higher-level knowledge: “universal grammar”

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

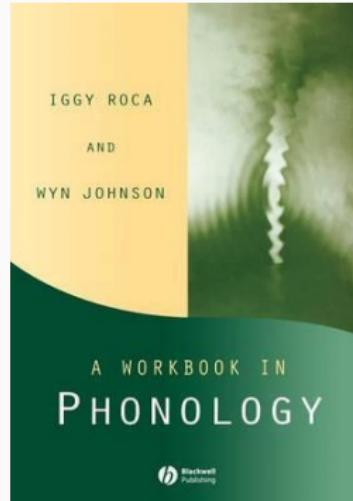
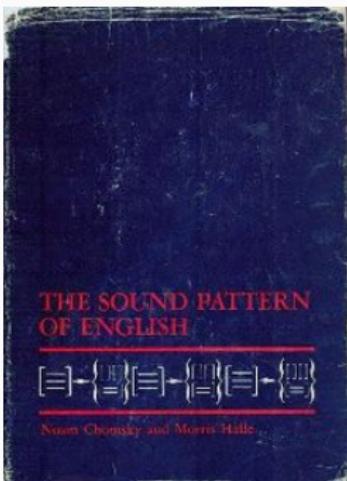
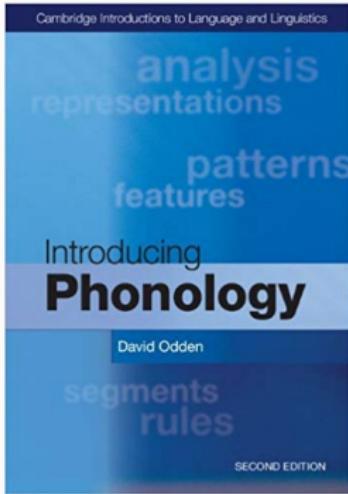
Serbo-Croatian:

	masculine	feminine	stem (unobserved)
“rich”	bogat	bogata	bogat
“mild”	blag	blaga	blag
“green”	zelen	zelena	zelen
“clear”	yasan	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>oyuurduun</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	ayata	ayataaýar	ayattan	'father'
balik	baliktar	balikiin	'fish'	paarta	paartata	paartataaýar	paattattan	'school desk'
iskaap	iskaaptar	iskaaptiin	'cabinet'	tia	tiata	tiataaýar	tiattan	'forest'
oyus	oyustar	oyustuuñ	'bull'	kinige	kinigete	kinigeteeyer	kinigetten	'book'
kus	kustar	kustuuñ	'duck'	Jie	jiete	jieteeeyer	jietten	'house'
tünnük	tünnükter	tünnüktüün	'window'	iye	iyete	iyeteeeyer	iyetten	'mother'
sep	septer	septiiñ	'tool'	kini	kinite	kinitteeeyer	kinitten	'3rd person'
et	etter	ettiiñ	'meat'	bie	biete	bieteeeyer	bietten	'mare'
örüs	örüster	örüstüün	'river'	oyo	oyoto	oyotooyor	oyotton	'child'
tis	tiister	tiistiin	'tooth'	χopto	χoptoto	χoptotooyor	χoptotton	'gull'
soroχ	soroχtor	soroχtuun	'some person'	börö	börötö	börötööyör	böröttön	'wolf'
ox	oxtor	oxtuun	'arrow'	tial	tialla	tiallaaýar	tialtan	'wind'
oloppos	oloppstor	oloppstuun	'chair'	ial	ialla	iallaaýar	ialtan	'neighbor'
ötöχ	ötöxtör	ötöxtüün	'abandoned farm'	kuul	kuulla	kuullaaýar	kuultan	'sack'
ubay	ubaydar	ubaydiin	'elder brother'	moχsoyol	moχsoyollo	moχsoyollooyor	moχsyołoton'	'falcon'
asaray	saraydar	saraydiin	'bam'	at	atta	attaayar	attan	'horse'
tiy	tiydar	tiydiin	'foal'	balik	balikta	baliktaaýar	baliktan	'fish'
atiir	atiirdar	atiirdiin	'stallion'	iskaap	iskaapta	iskaaptaaýar	iskaaptan	'cabinet'
			tünnük	oyus	oyusta	oyustaayar	oyustan	'bul'
				kus	kusta	kustaayar	kustan	'duck'
				tünnükte	tünnükte	tünnükteeyer	tünnükten	'window'

Turkic Sakha (Yakut)

observed data

	SINGULAR	PLURAL
BED	oron	<u>oronnor</u>
MARE	bie	<u>bieler</u>
CABINET	ı̄skaap	<u>ı̄skaaptar</u>

138 total examples

Turkic Sakha (Yakut)

grammar (unobserved)

SINGULAR → stem
PLURAL → stem + lar

observed data

	SINGULAR	PLURAL
BED	orон	ороннор
MARE	bie	биелер
CABINET	їскаап	їскааптар

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

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	ıskap	ıskaatpar

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)

BED : oron
MARE : bie
CABINET : ɨskaap

observed data

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	ɨskaap	ɨskaaptar

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)

CABINET : iskaap

BED : oron

MARE : bie

RIVER : örus

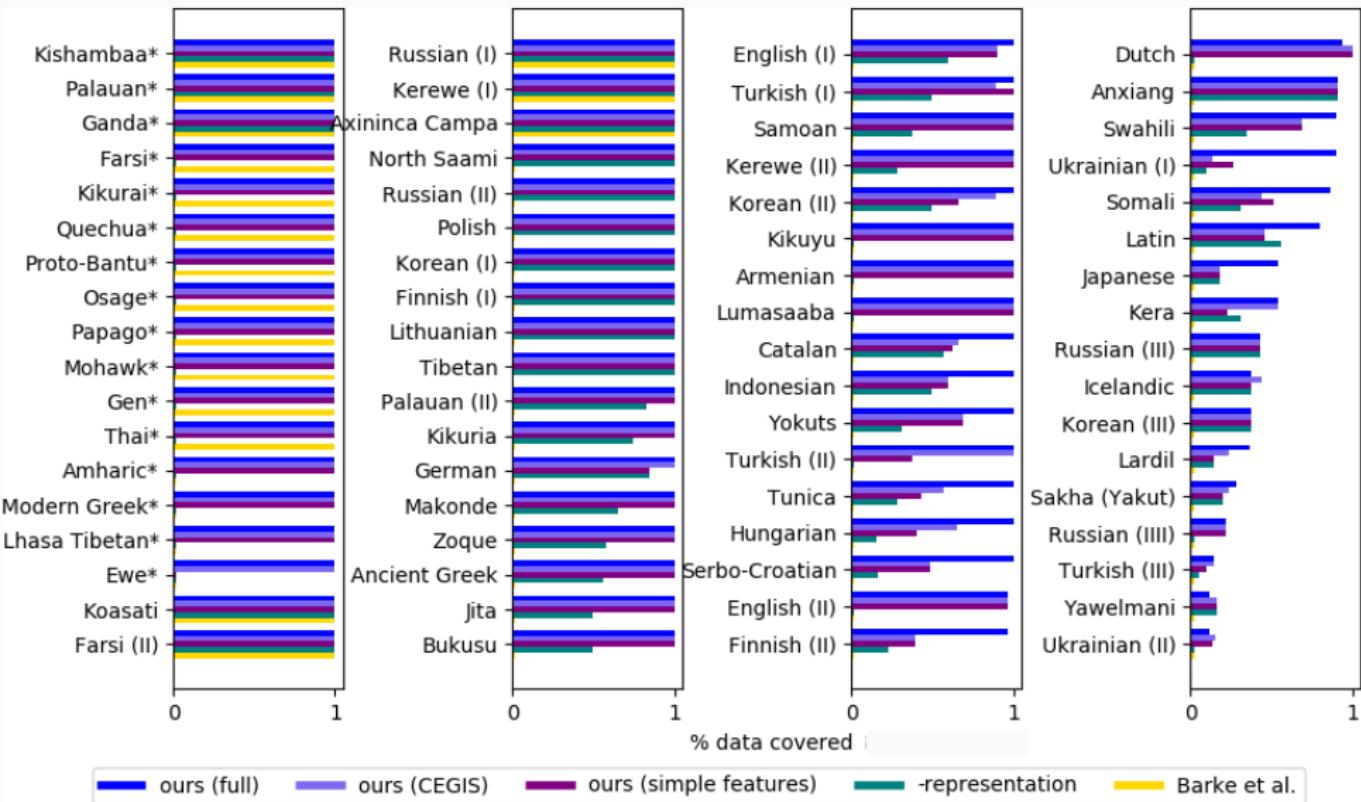
observed data

CABINETS → iskaap + lar → iskaaplar $\xrightarrow{r_1}$ iskaapdar $\xrightarrow{r_2}$ iskaaptar

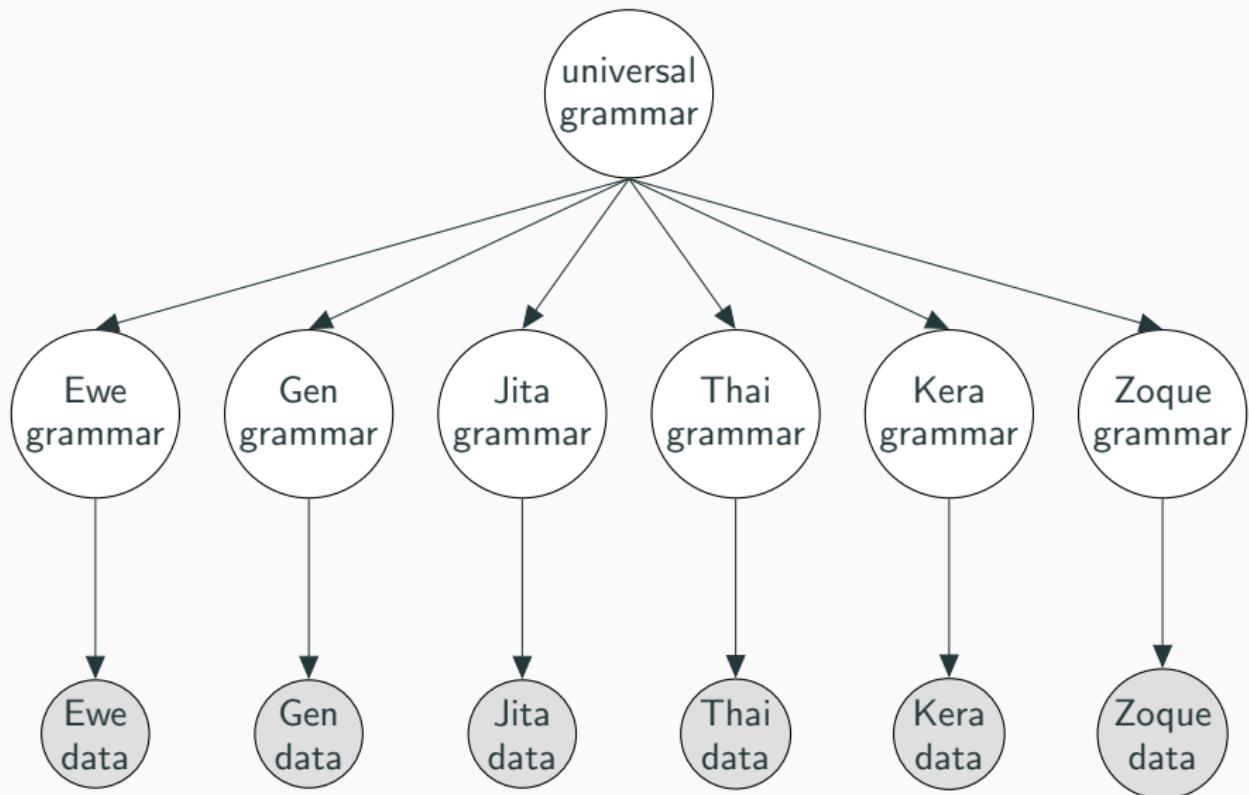
BEDS → oron + lar → oronlar $\xrightarrow{r_1}$ orondar $\xrightarrow{r_3}$ orondor $\xrightarrow{r_6}$ oronnor

MARES → bie + lar → bielar $\xrightarrow{r_5}$ bieler

RIVER (ASSOC) → örus + li+n → örusli+n $\xrightarrow{r_1}$ örusdi+n $\xrightarrow{r_2}$
örusti+n $\xrightarrow{r_3}$ örurstuu+n $\xrightarrow{r_5}$ [örüstüün]



Distilling higher-level knowledge



Lessons

Higher-level knowledge matters (“universal grammar”). Get the basics of the representation 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
model discovery
learning to learn

Learning to write code

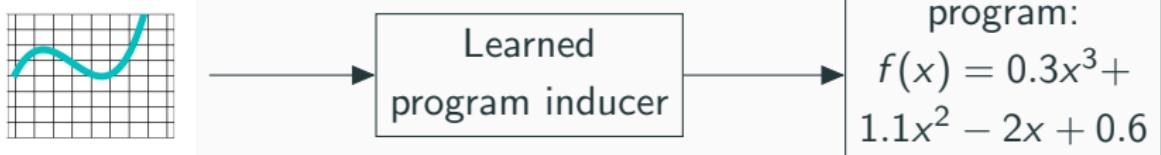
Goal: acquire domain-specific knowledge needed to induce a class of programs

- 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)



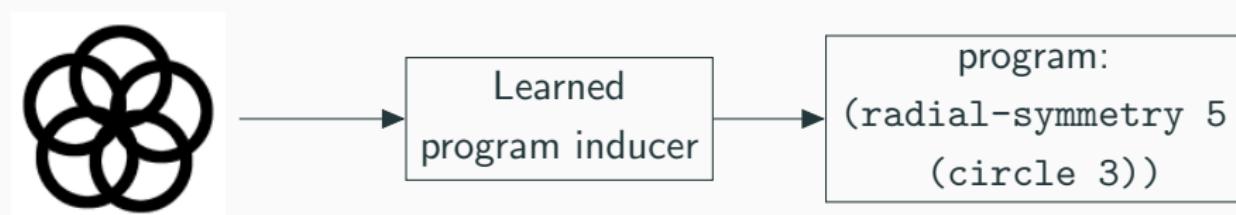
Concepts: x^3 , $\alpha x + \beta$, etc

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

>

: 

Sample Problem: sort list

[9 2 7 1] → [1 2 7 9]

[3 8 9 4 2] → [2 3 4 8 9]

[6 2 2 3 8 5] → [2 2 3 5 6 8]

...

Library learning

Initial Primitives

:

map

fold

if

cons

>

:

...

Sample Problem: sort list

[9 2 7 1] → [1 2 7 9]

[3 8 9 4 2] → [2 3 4 8 9]

[6 2 2 3 8 5] → [2 2 3 5 6 8]

...

Library learning

Initial Primitives

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

Sample Problem: sort list

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

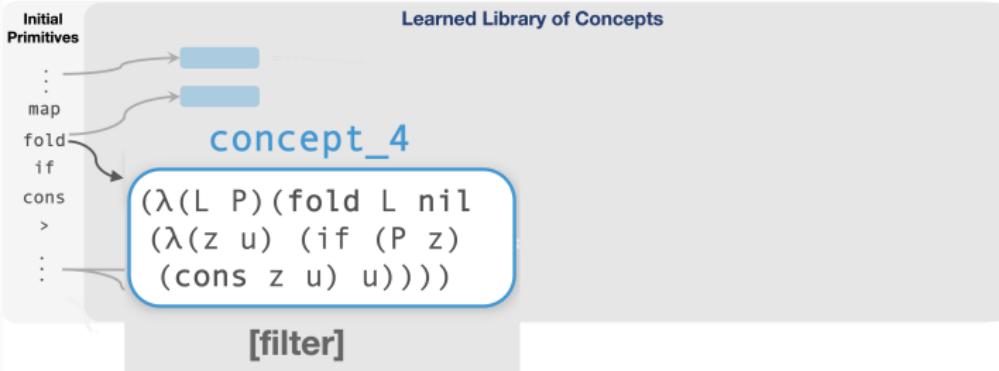
Library learning



Sample Problem: sort list

$[9 2 7 1] \rightarrow [1 2 7 9]$
 $[3 8 9 4 2] \rightarrow [2 3 4 8 9]$
 $[6 2 2 3 8 5] \rightarrow [2 2 3 5 6 8]$
...

Library learning



Sample Problem: sort list

$[9\ 2\ 7\ 1] \rightarrow [1\ 2\ 7\ 9]$
 $[3\ 8\ 9\ 4\ 2] \rightarrow [2\ 3\ 4\ 8\ 9]$
 $[6\ 2\ 2\ 3\ 8\ 5] \rightarrow [2\ 2\ 3\ 5\ 6\ 8]$
⋮

Library learning

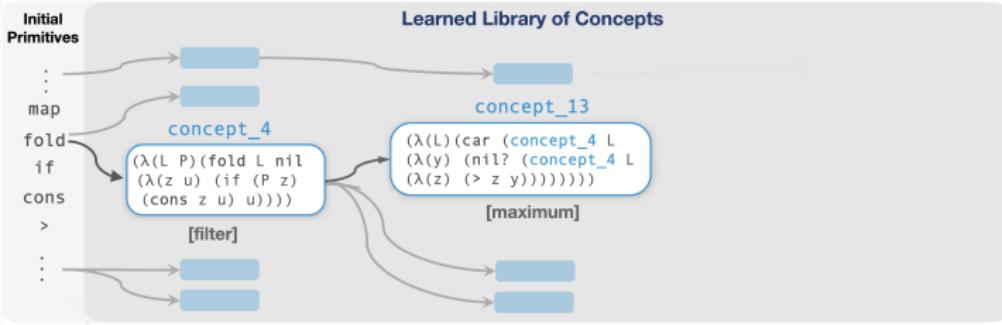


Sample Problem: sort list

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

inducing sort program found in $\leq 10\text{min}$. Brute-force search without learned library would take $\approx 10^{78}$ years to discover this solution

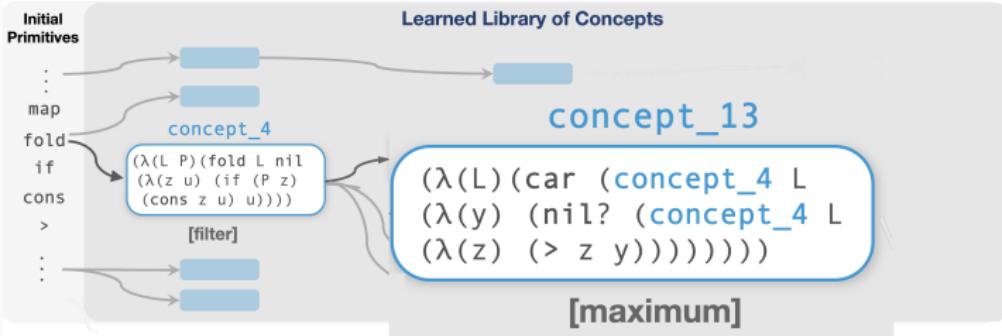
Library learning



Sample Problem: sort list

$[9 2 7 1] \rightarrow [1 2 7 9]$
 $[3 8 9 4 2] \rightarrow [2 3 4 8 9]$
 $[6 2 2 3 8 5] \rightarrow [2 2 3 5 6 8]$
...

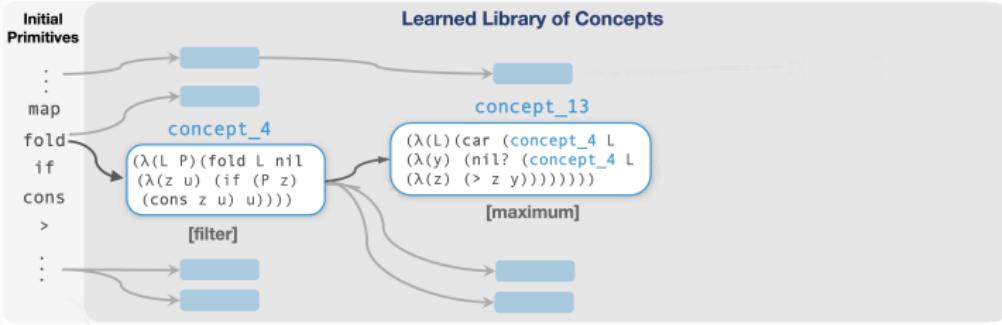
Library learning



Sample Problem: sort list

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

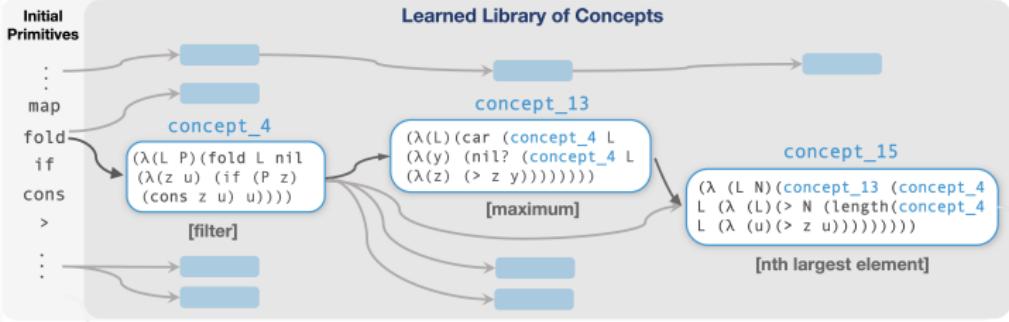
Library learning



Sample Problem: sort list

$[9 2 7 1] \rightarrow [1 2 7 9]$
 $[3 8 9 4 2] \rightarrow [2 3 4 8 9]$
 $[6 2 2 3 8 5] \rightarrow [2 2 3 5 6 8]$
...

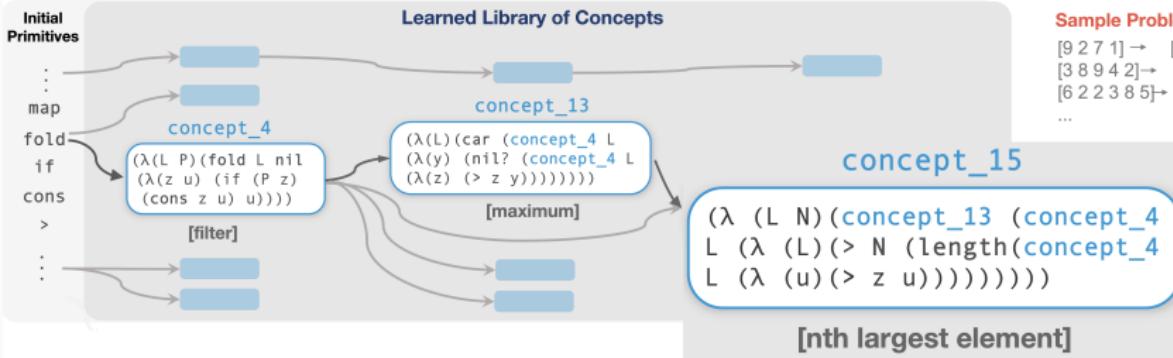
Library learning



Sample Problem: sort list

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

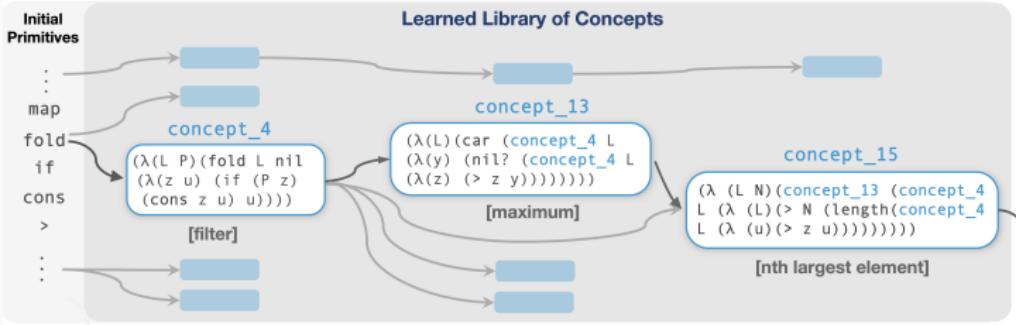
Library learning



Sample Problem: sort list

$[9 2 7 1] \rightarrow [1 2 7 9]$
 $[3 8 9 4 2] \rightarrow [2 3 4 8 9]$
 $[6 2 2 3 8 5] \rightarrow [2 2 3 5 6 8]$
...

Library learning



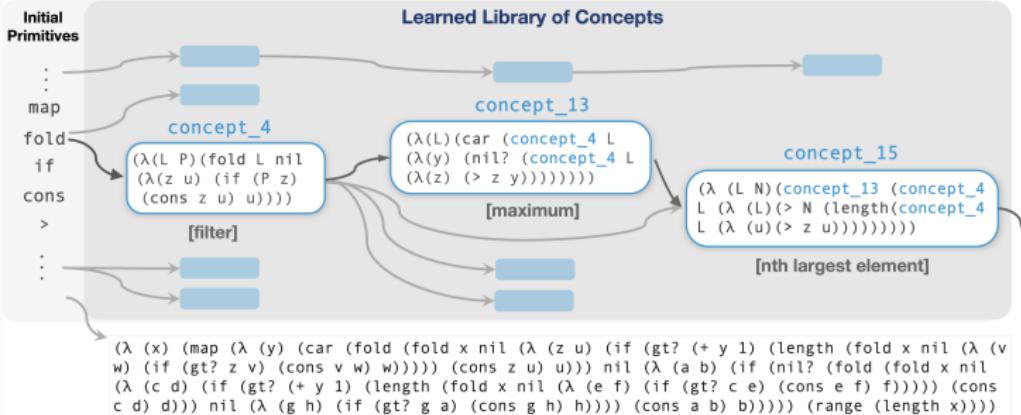
Sample Problem: sort list

$[9 \ 2 \ 7 \ 1] \rightarrow [1 \ 2 \ 7 \ 9]$
 $[3 \ 8 \ 9 \ 4 \ 2] \rightarrow [2 \ 3 \ 4 \ 8 \ 9]$
 $[6 \ 2 \ 2 \ 3 \ 8 \ 5] \rightarrow [2 \ 2 \ 3 \ 5 \ 6 \ 8]$
...

Solution to sort list discovered in learned language:

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

Library learning



Sample Problem: sort list

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

Solution to sort list discovered in learned language:

```
(map\ (\lambda\ (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



cf. Helmholtz machine, wake/sleep neural network training

DreamCoder

- **Wake:** Solve problems by writing programs
- **Sleep:** Improve library and neural recognition model:
 - **Abstraction sleep:** Improve library
 - **Dream sleep:** Improve neural inference model

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

$$\text{Allen Newell} \rightarrow \text{A.N.}$$

$$\text{Herb Simon} \rightarrow \text{H.S.}$$

Drop Last Characters

$$\text{jabberwocky} \rightarrow \text{jabberw}$$

$$\text{copycat} \rightarrow \text{cop}$$

Extract

$$\text{see spot(run)} \rightarrow \text{run}$$

$$\text{a (bee) see} \rightarrow \text{bee}$$

Regexes

Phone Numbers

$$(555) \ 867-5309$$

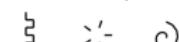
$$(650) \ 555-2368$$

Currency

$$\$100.25$$

$$\$4.50$$

LOGO Graphics



Physics

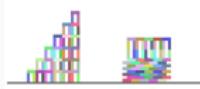
$$KE = \frac{1}{2}m|\vec{v}|^2$$

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

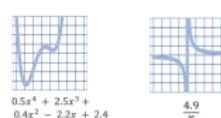
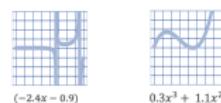
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{r}_1 \hat{r}_2$$

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

Block Towers



Symbolic Regression



Recursive Programming

Filter

$$[\blacksquare \blacksquare \blacksquare] \rightarrow [\blacksquare \blacksquare]$$

$$[\blacksquare \blacksquare \blacksquare \blacksquare] \rightarrow [\blacksquare \blacksquare \blacksquare]$$

$$[\blacksquare \blacksquare \blacksquare \blacksquare \blacksquare] \rightarrow [\blacksquare \blacksquare \blacksquare]$$

Index List

$$0, [\blacksquare \blacksquare \blacksquare \blacksquare] \rightarrow \blacksquare$$

$$1, [\blacksquare \blacksquare \blacksquare \blacksquare \blacksquare] \rightarrow \blacksquare$$

$$1, [\blacksquare \blacksquare \blacksquare \blacksquare \blacksquare] \rightarrow \blacksquare$$

Length

$$[\blacksquare \blacksquare \blacksquare] \rightarrow 4$$

$$[\blacksquare \blacksquare \blacksquare \blacksquare] \rightarrow 6$$

$$[\blacksquare \blacksquare] \rightarrow 3$$

Every Other

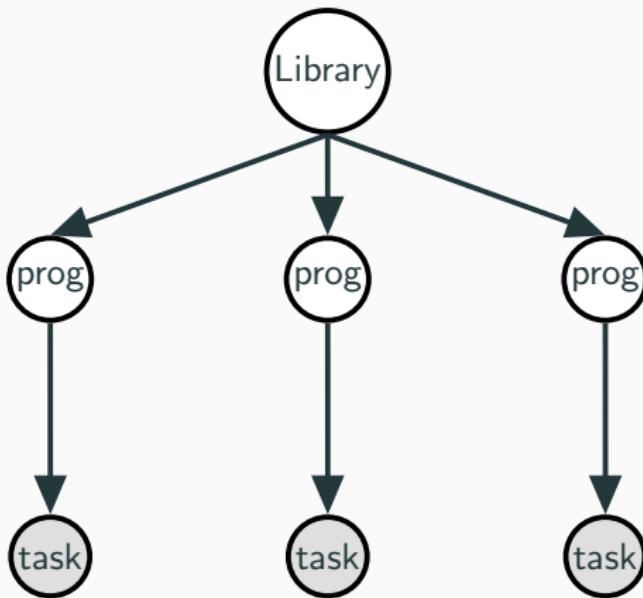
$$[\blacksquare \blacksquare \blacksquare \blacksquare] \rightarrow [\blacksquare \blacksquare]$$

$$[\blacksquare \blacksquare \blacksquare \blacksquare \blacksquare] \rightarrow [\blacksquare \blacksquare]$$

$$[\blacksquare \blacksquare \blacksquare \blacksquare \blacksquare] \rightarrow [\blacksquare \blacksquare]$$

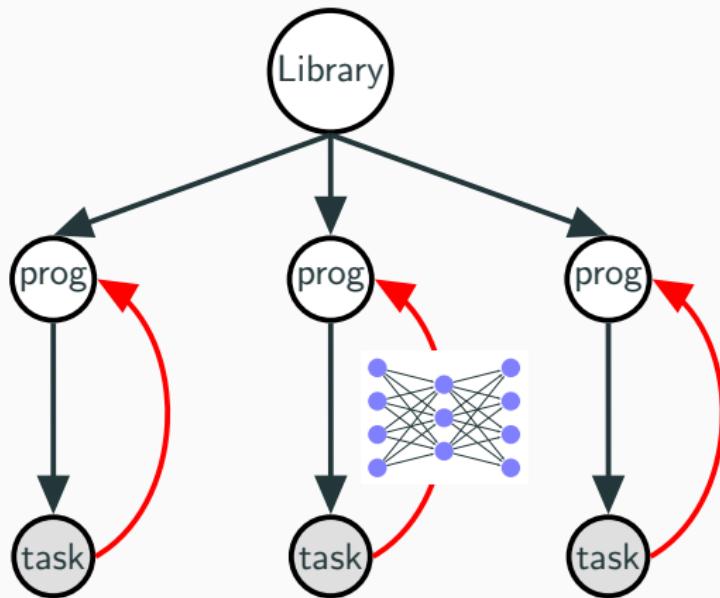
cf. Helmholtz machine, wake/sleep neural network training

Library learning as Bayesian inference

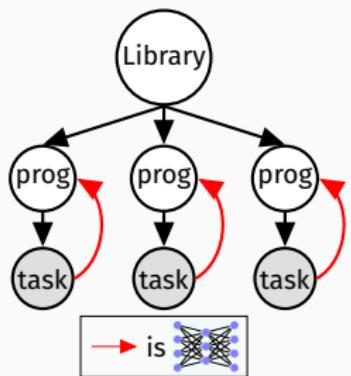


[Dechter et al., 2013] [Liang et al, 2010]; [Lake et al, 2015]

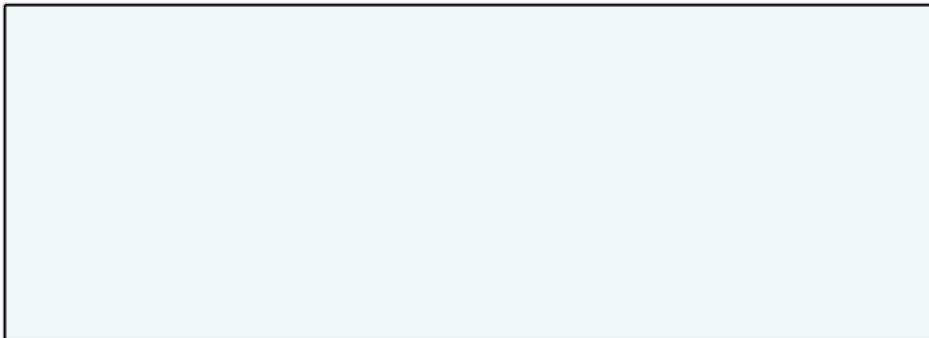
Library learning as neurally-guided Bayesian inference



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

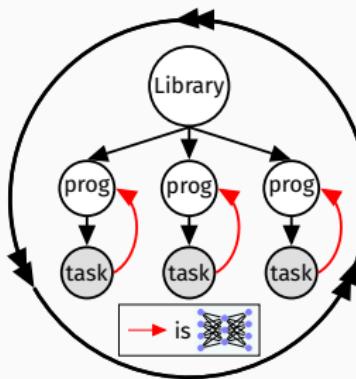


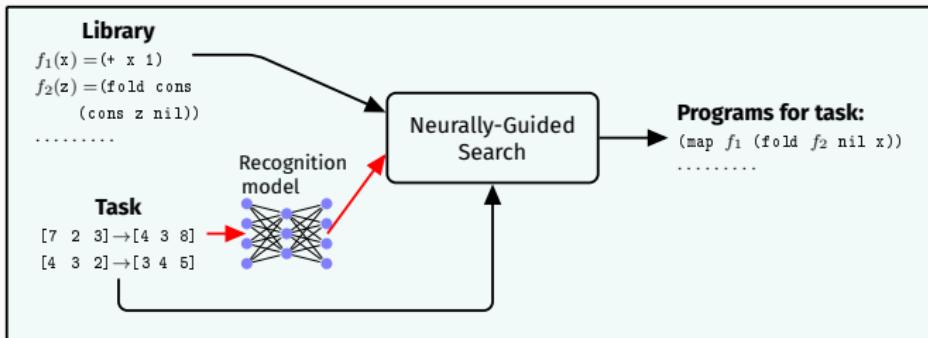
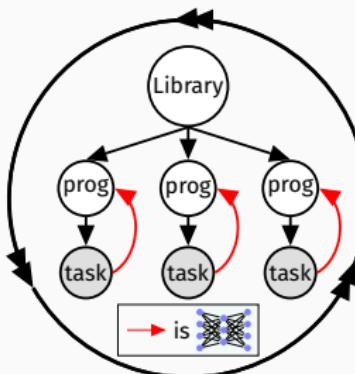
WAKE

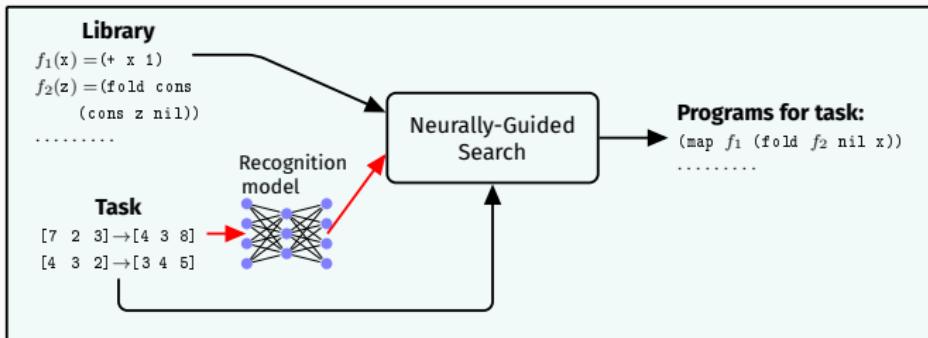
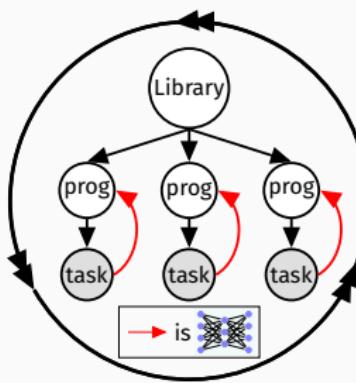
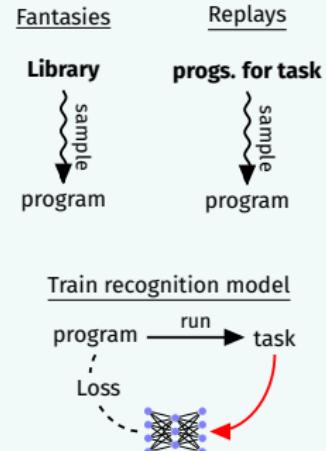


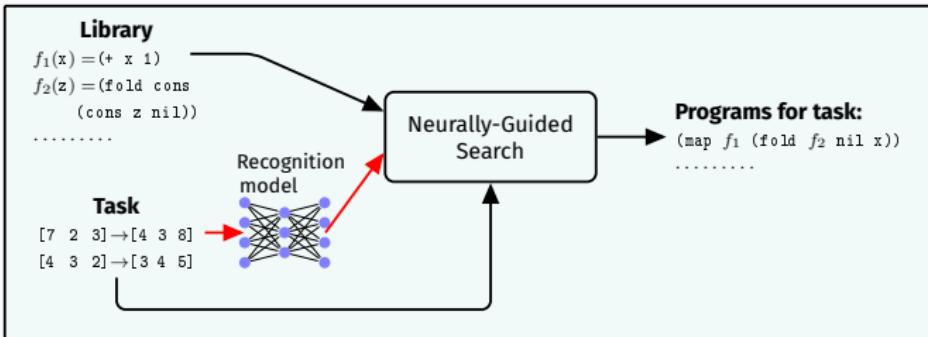
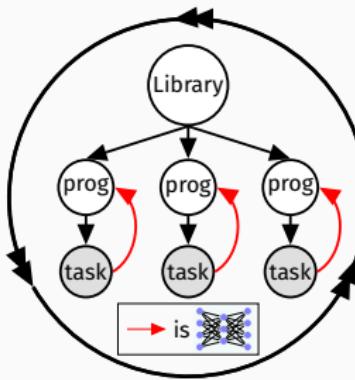
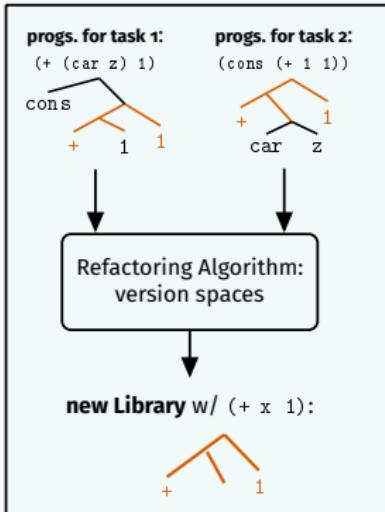
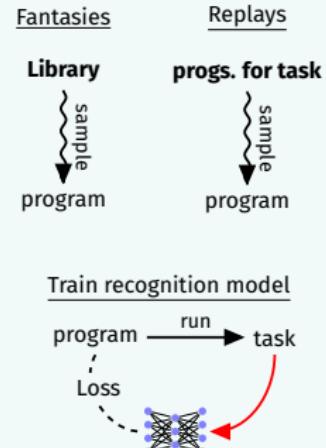
SLEEP: ABSTRACTION

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

$5 + 5$

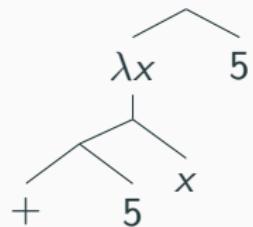
(+ 5 5)



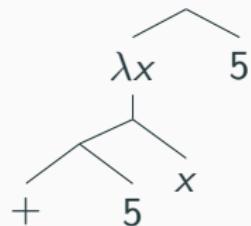
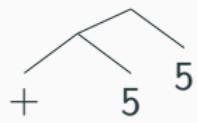
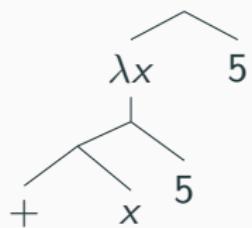
Abstraction Sleep: Growing the library via refactoring



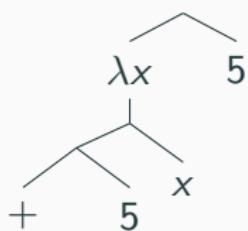
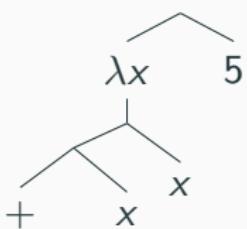
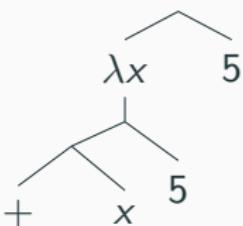
Abstraction Sleep: Growing the library via refactoring



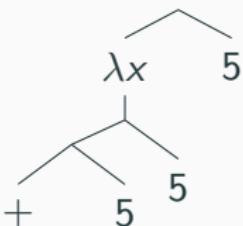
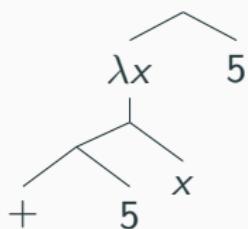
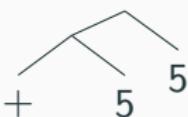
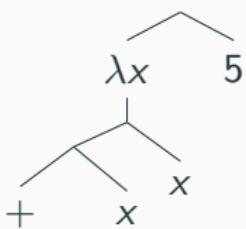
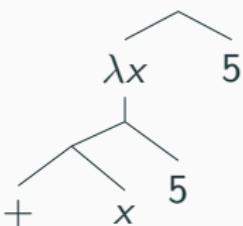
Abstraction Sleep: Growing the library via refactoring



Abstraction Sleep: Growing the library via refactoring

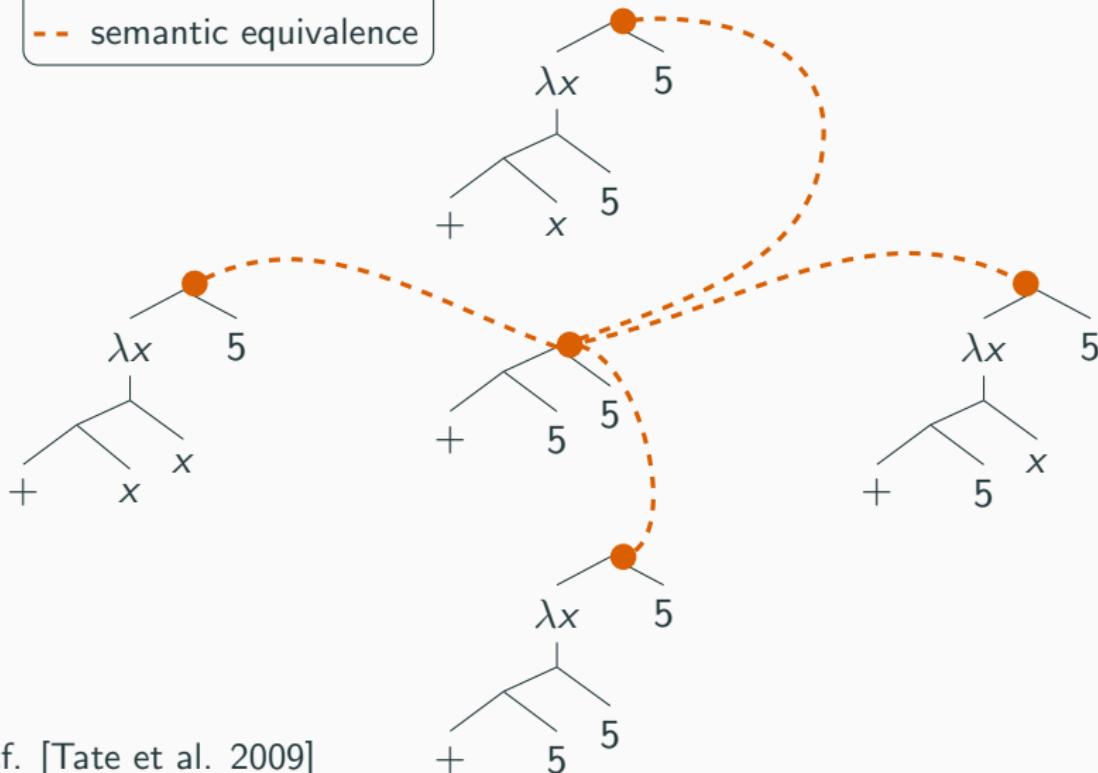


Abstraction Sleep: Growing the library via refactoring



Abstraction Sleep: Growing the library via refactoring

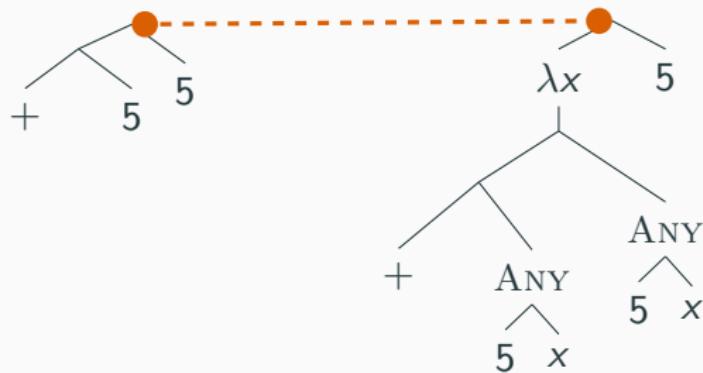
legend
--- semantic equivalence



Abstraction Sleep: Growing the library via refactoring

legend

- semantic equivalence
- ANY nondeterministic choice



cf. Tate et al. 2009

Gulwani 2012

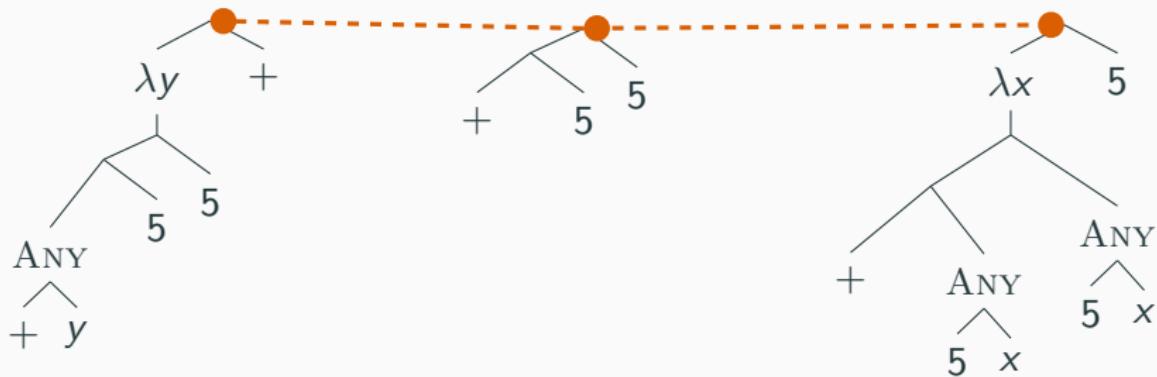
Liang et al. 2010

Abstraction Sleep: Growing the library via refactoring

legend

semantic equivalence

ANY nondeterministic choice



cf. Tate et al. 2009

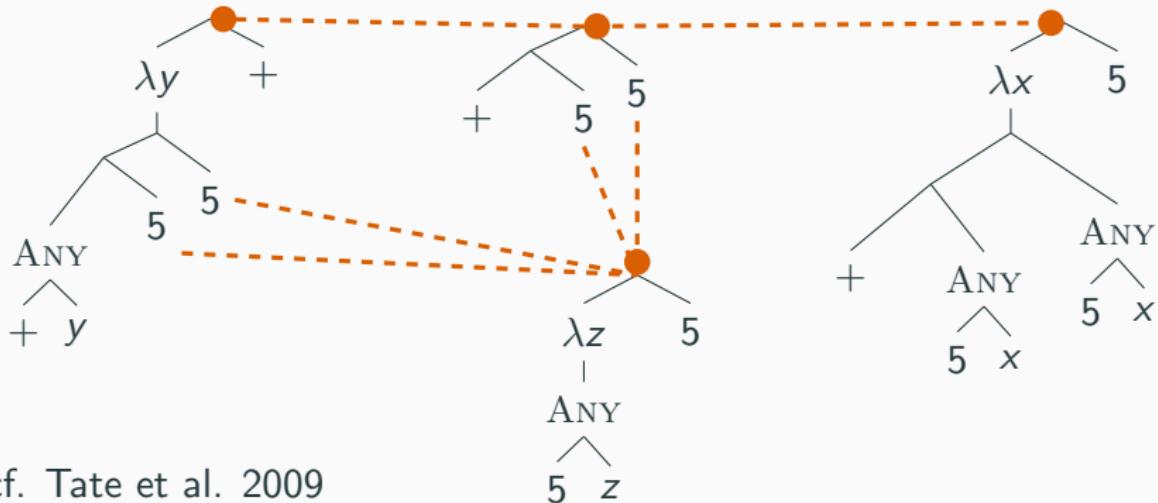
Gulwani 2012

Liang et al. 2010

Abstraction Sleep: Growing the library via refactoring

legend

- semantic equivalence
- ANY nondeterministic choice



cf. Tate et al. 2009

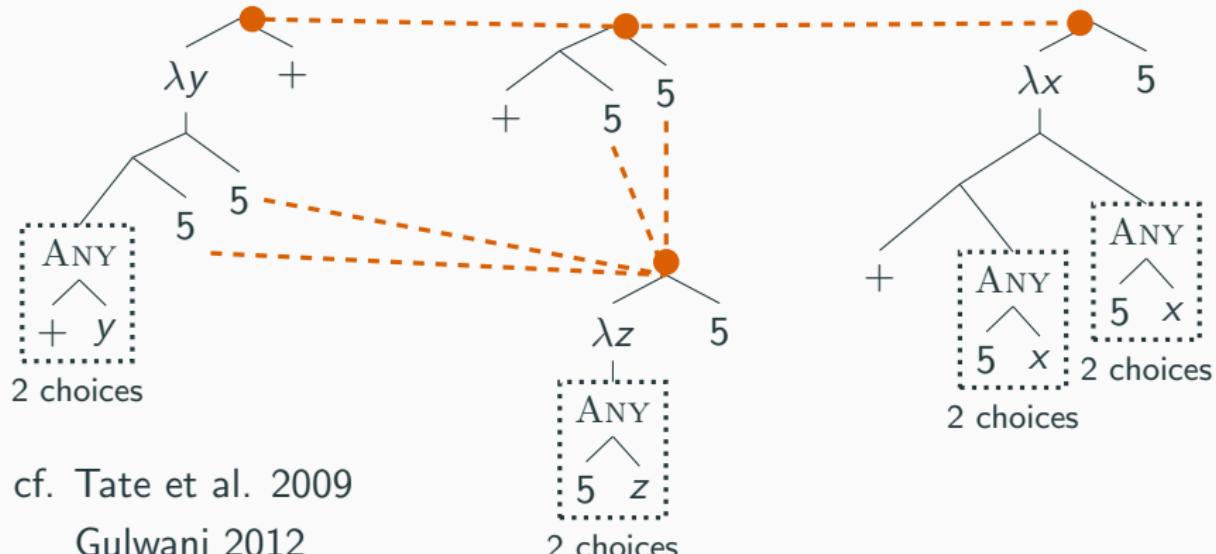
Gulwani 2012

Liang et al. 2010

Abstraction Sleep: Growing the library via refactoring

legend

dashed orange line semantic equivalence
ANY nondeterministic choice



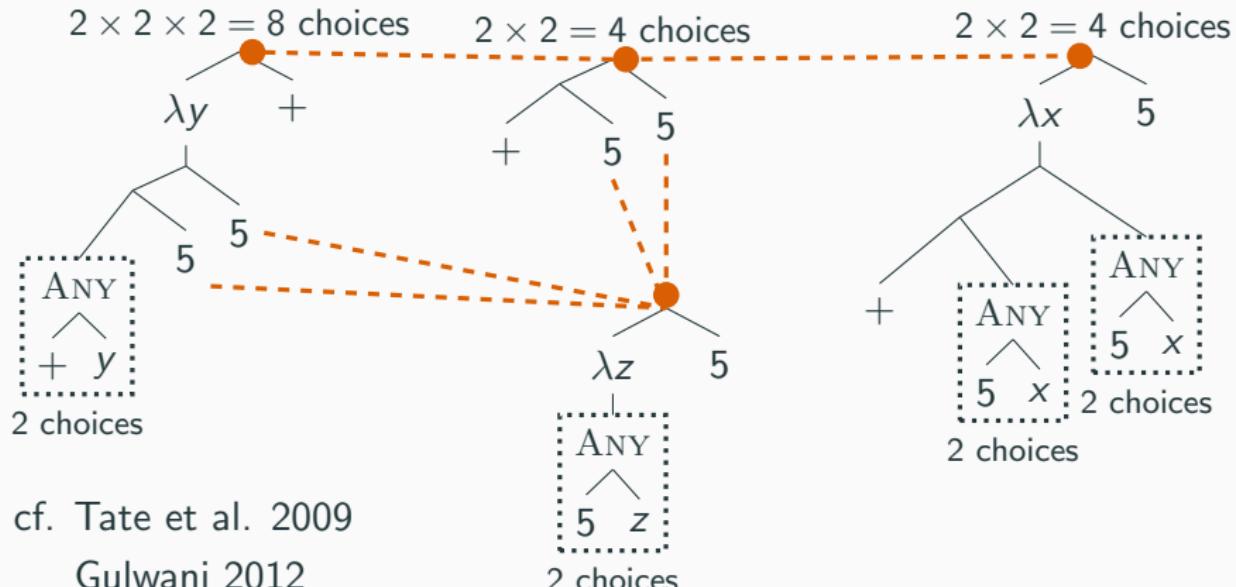
cf. Tate et al. 2009

Gulwani 2012

Liang et al. 2010

Abstraction Sleep: Growing the library via refactoring

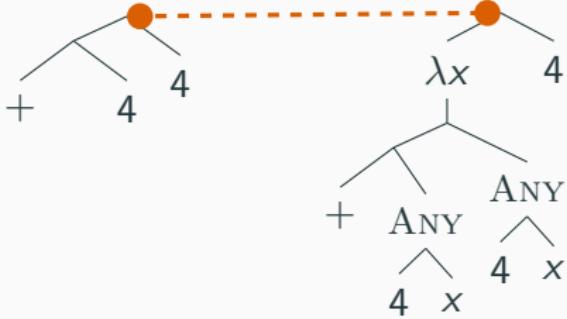
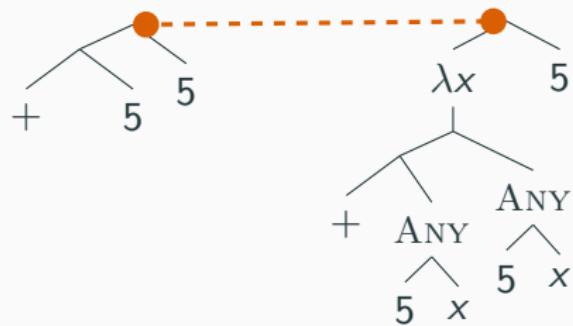
legend
--- semantic equivalence
ANY nondeterministic choice



cf. Tate et al. 2009

Gulwani 2012

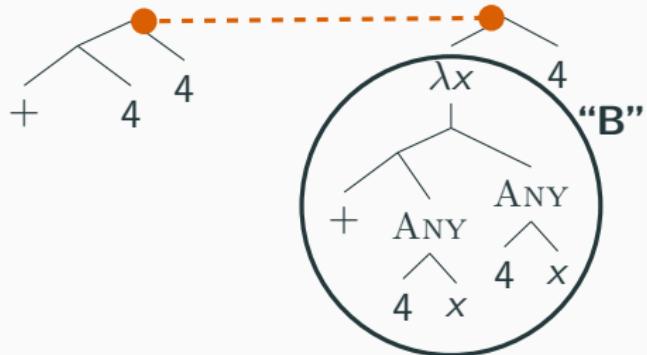
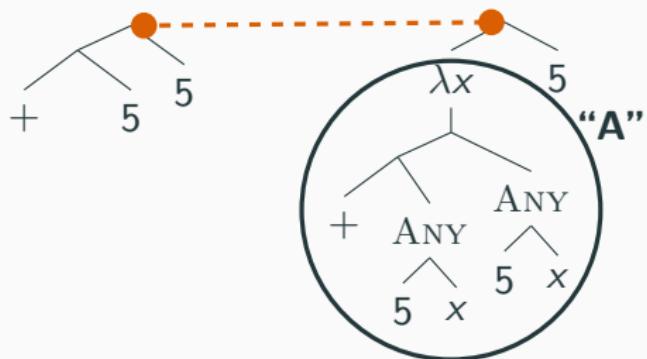
Liang et al. 2010



legend

— semantic equivalence

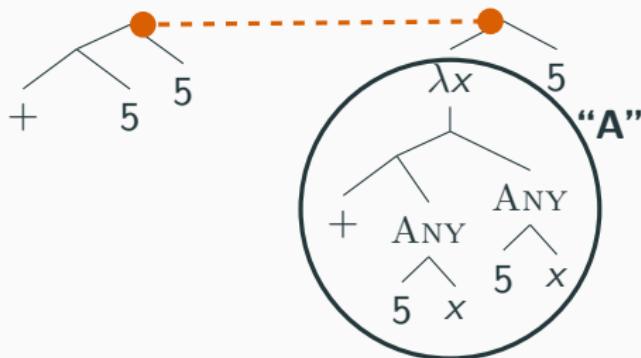
ANY nondeterministic choice



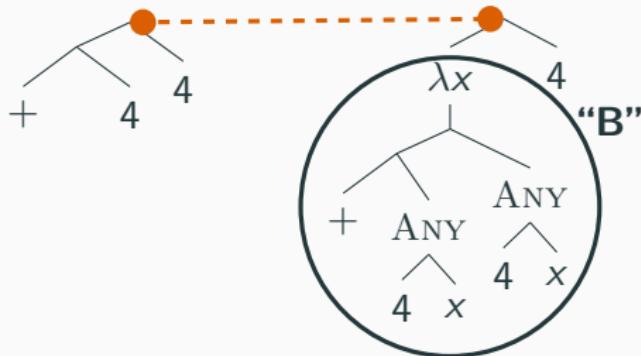
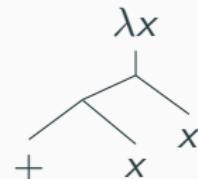
legend

— semantic equivalence

ANY nondeterministic choice

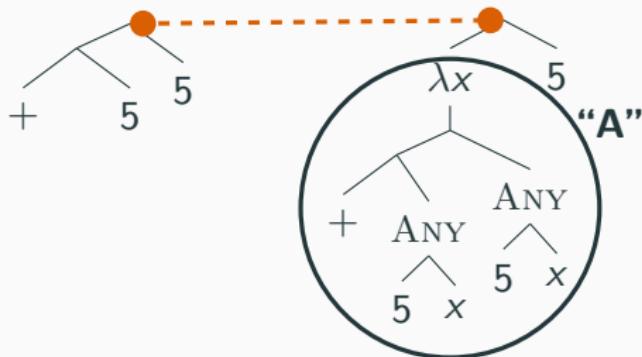


A intersect B:

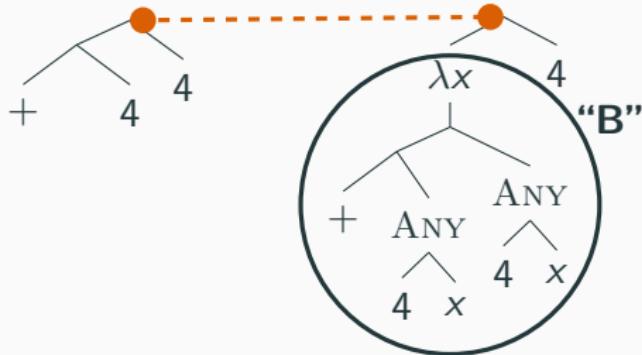
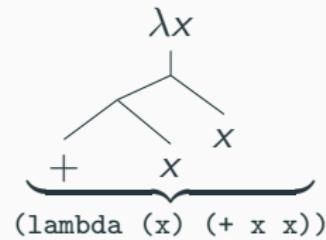


legend

- semantic equivalence
- ANY nondeterministic choice



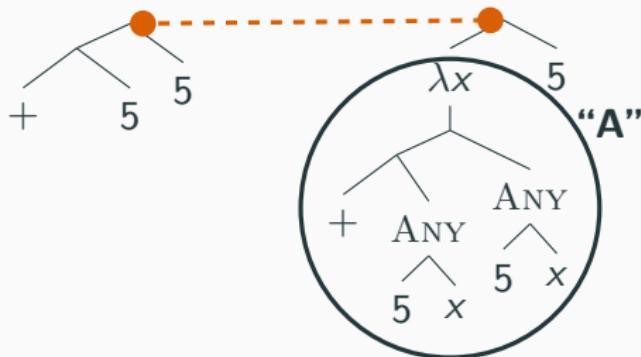
A intersect B:



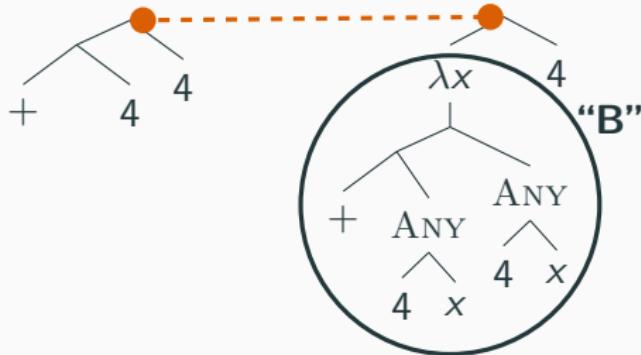
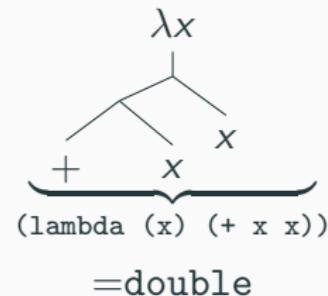
legend

— semantic equivalence

ANY nondeterministic choice



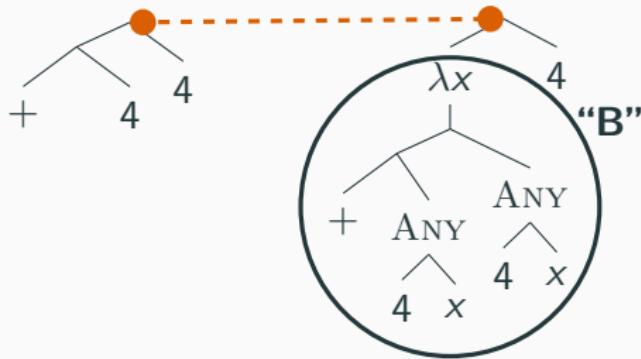
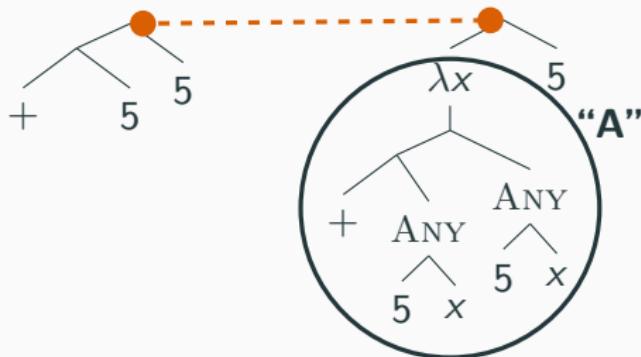
A intersect B:



legend

— semantic equivalence

ANY nondeterministic choice

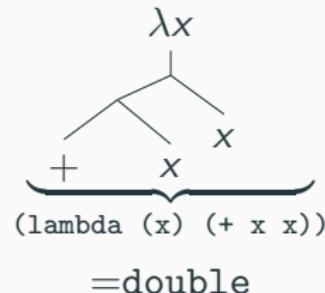


legend

— semantic equivalence

ANY nondeterministic choice

A intersect B:



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

Abstraction Sleep: Growing the library via refactoring

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

Wake: program search

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

Task: $[1\ 2\ 3] \rightarrow [0\ 1\ 2]$
 $[4\ 3\ 4] \rightarrow [3\ 2\ 3]$

Wake: program search

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

Abstraction Sleep: Growing the library via refactoring

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

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)

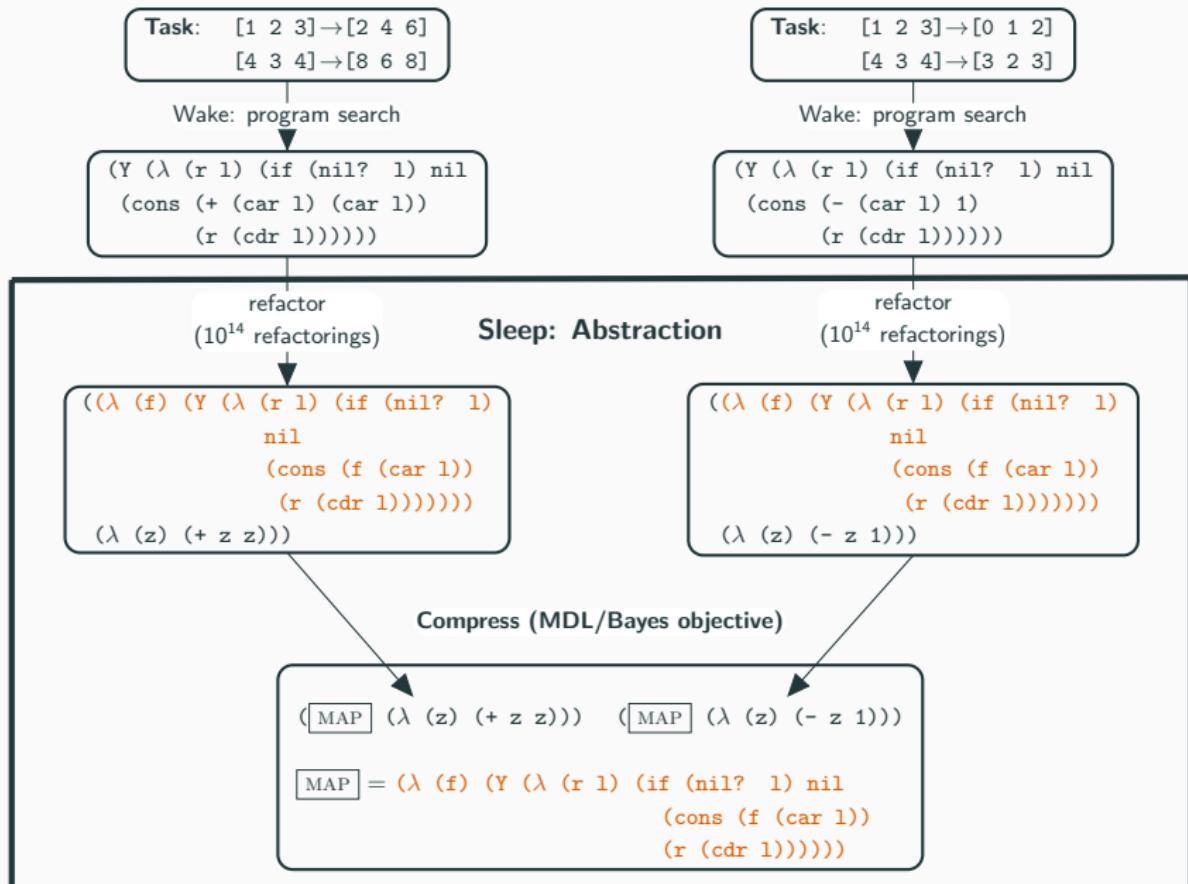
Sleep: Abstraction

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

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

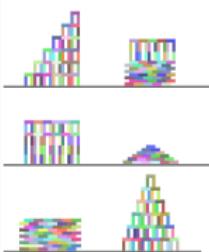
Dates

Y1775/0704
 Y2000/0101

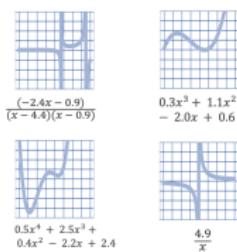
LOGO Graphics



Block Towers



Symbolic Regression



Recursive Programming

Filter

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

Length

[■■■■■] → 4
 [■■■■■■■] → 6
 [■■■■] → 3

Index List

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

Every Other

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

Physics

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

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

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

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

DreamCoder Domains

List Processing

Sum List

[1 2 3] → 6

[4 6 8 1] → 17

Double

[1 2 3 4] → [2 4 6 8]

[6 5 1] → [12 10 2]

Check Evens

[0 2 3] → [T T F]

[2 4 9 6] → [T T F T]

Text Editing

Abbreviate

Allen Newell → A.N.

Herb Simon → H.S.

Drop Last Characters

jabberwocky → jabberw

copycat → cop

Extract

see spot(run) → run

a (bee) see → bee

Regexes

Phone Numbers

(555) 867-5309

(650) 555-2368

Currency

\$100.25

\$4.50

Dates

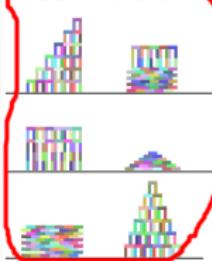
Y1775/0704

Y2000/0101

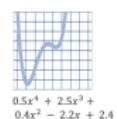
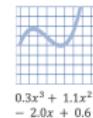
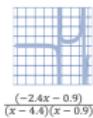
LOGO Graphics



Block Towers



Symbolic Regression



Recursive Programming

Filter

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

Length

[■■■■] → 4
[■■■■■] → 6
[■■■] → 3

Index List

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

Every Other

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

Physics

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

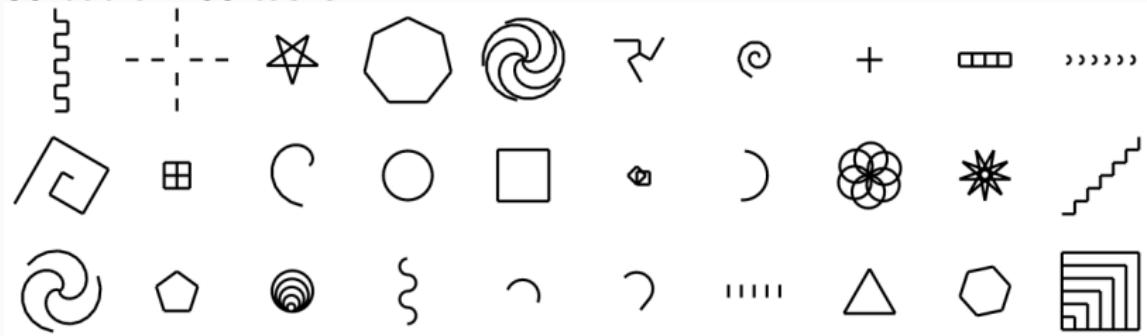
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

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

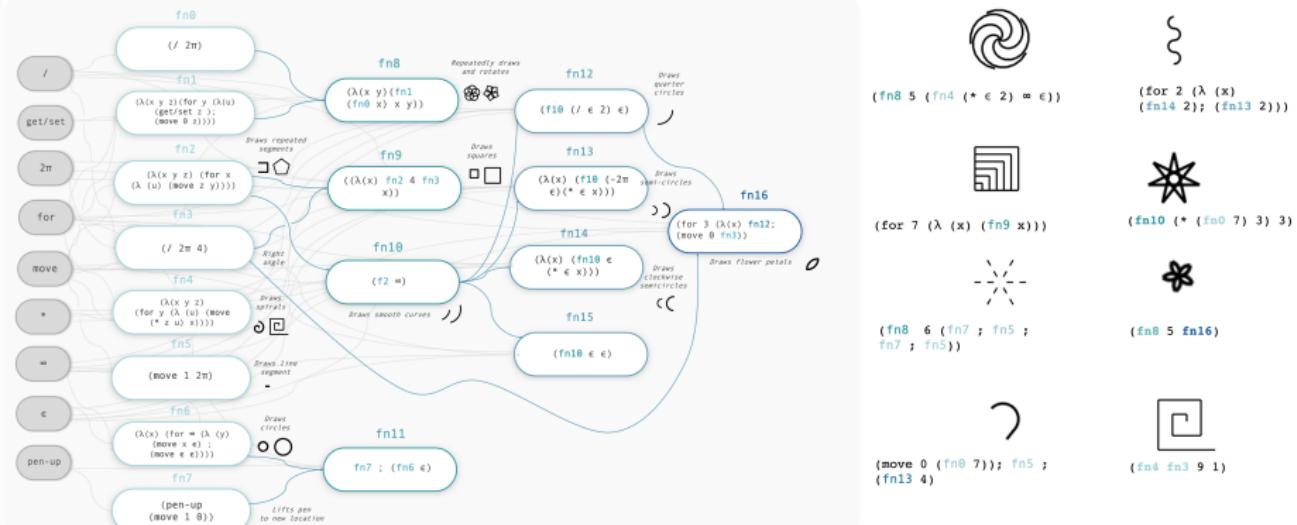
$$V_{total} = \left(\sum_i \frac{1}{R_i} \right)^{-1}$$

LOGO Graphics

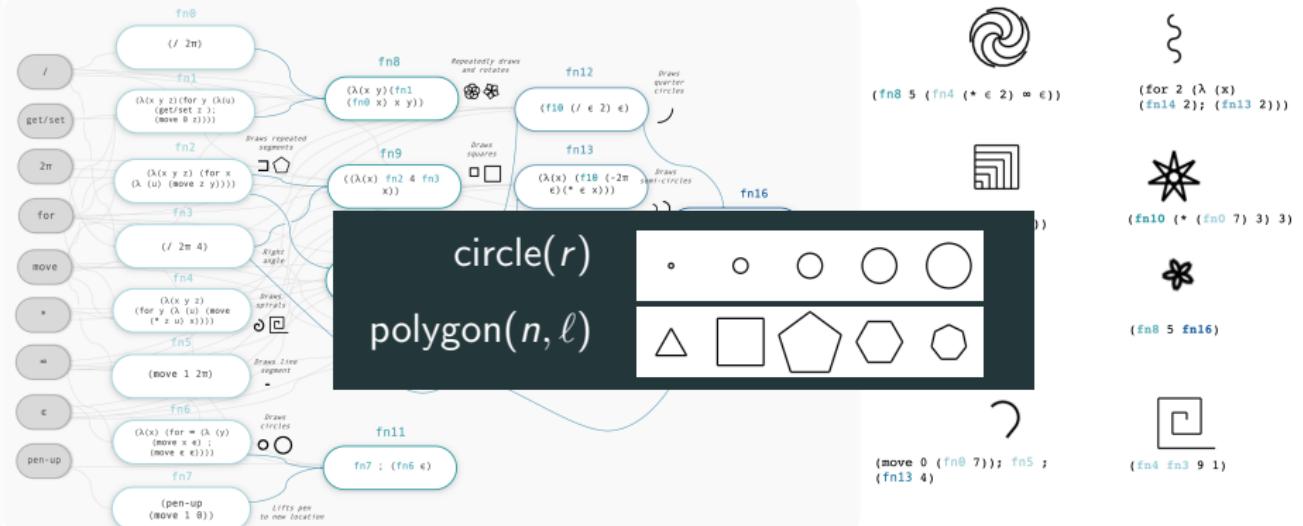
30 out of 160 tasks



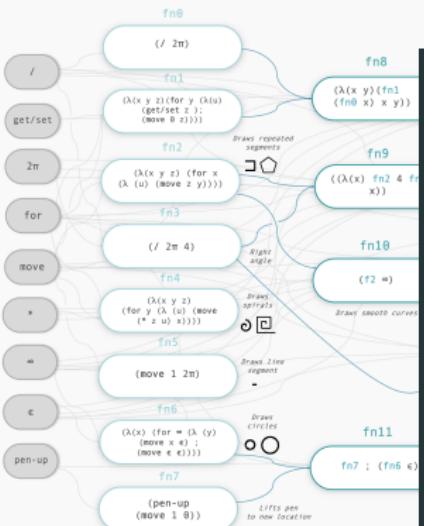
LOGO Graphics – learning interpretable library of concepts



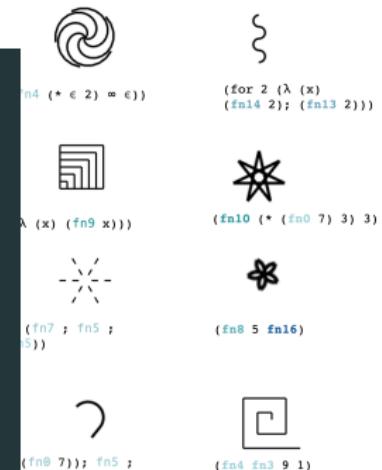
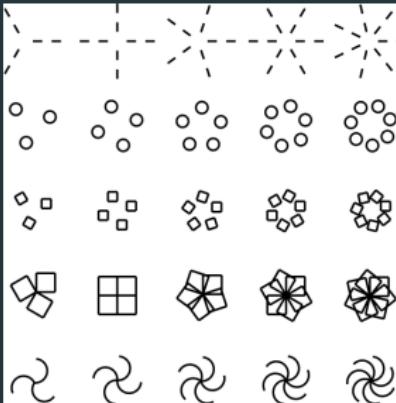
LOGO Graphics – learning interpretable library of concepts



LOGO Graphics – learning interpretable library of concepts



radial symmetry(*n*, body)

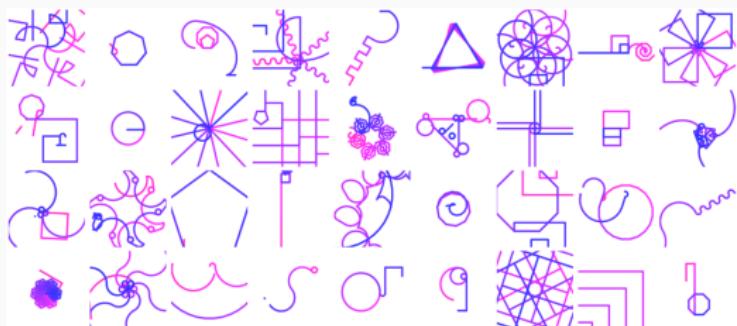


what does DreamCoder dream of?

before learning

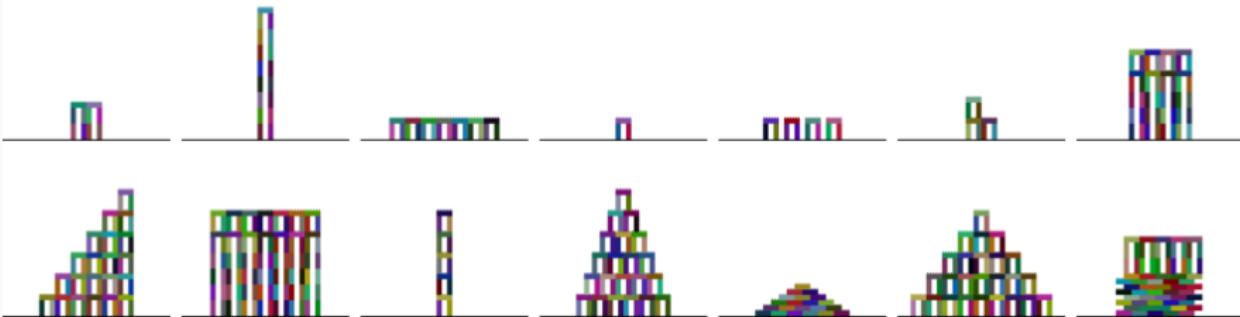


after learning



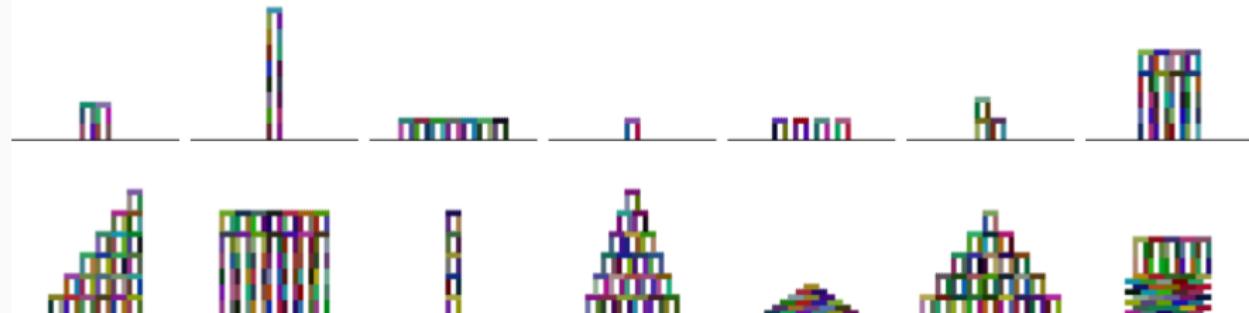
Planning to build towers

example tasks (112 total)

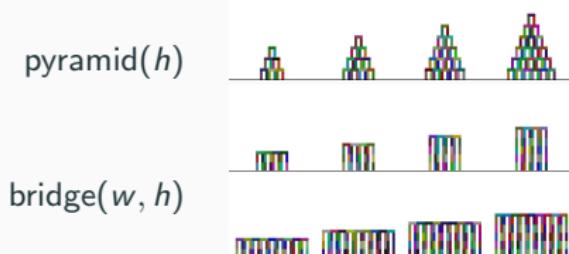
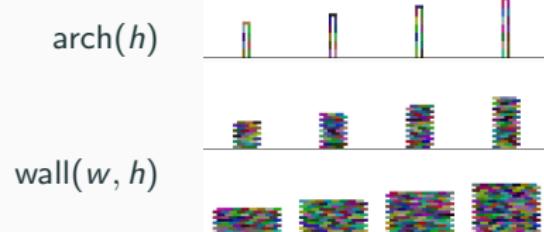


Planning to build towers

example tasks (112 total)

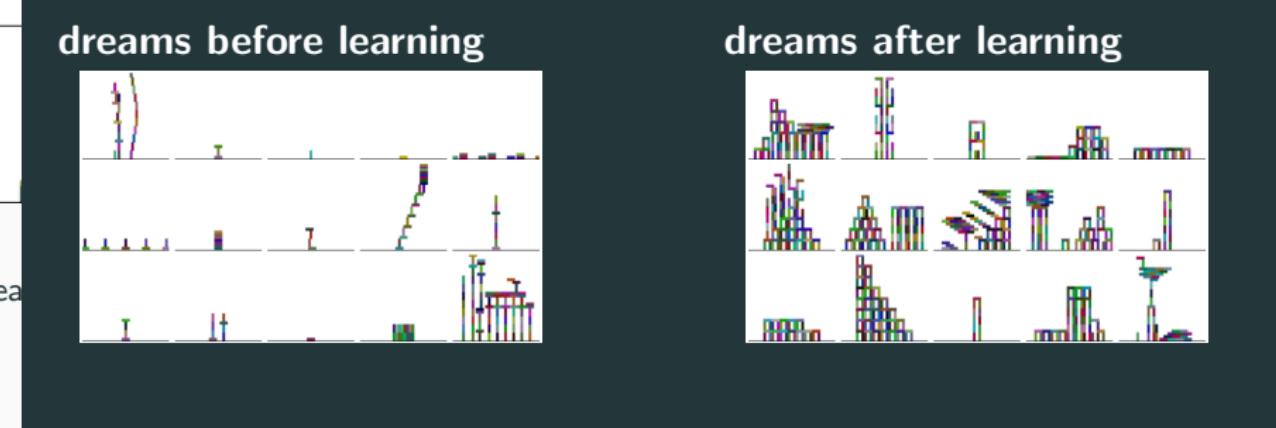


learned library routines (≈ 20 total)



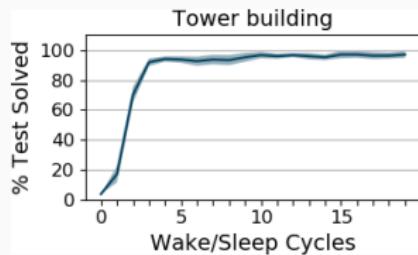
Planning to build towers

example tasks (112 total)

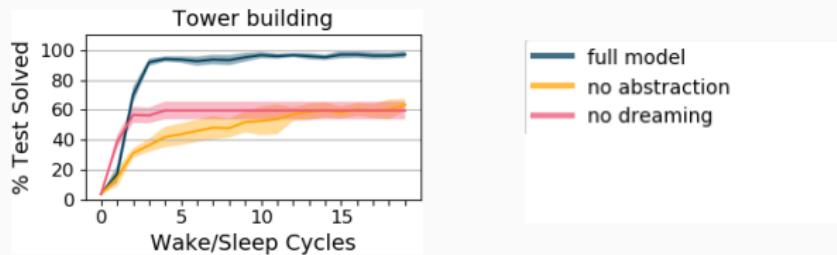


112 tasks

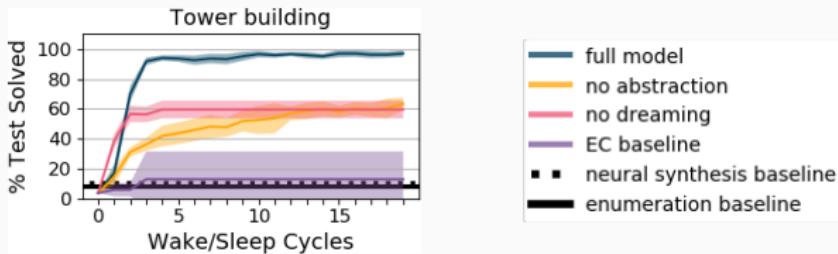
Learning dynamics



Learning dynamics

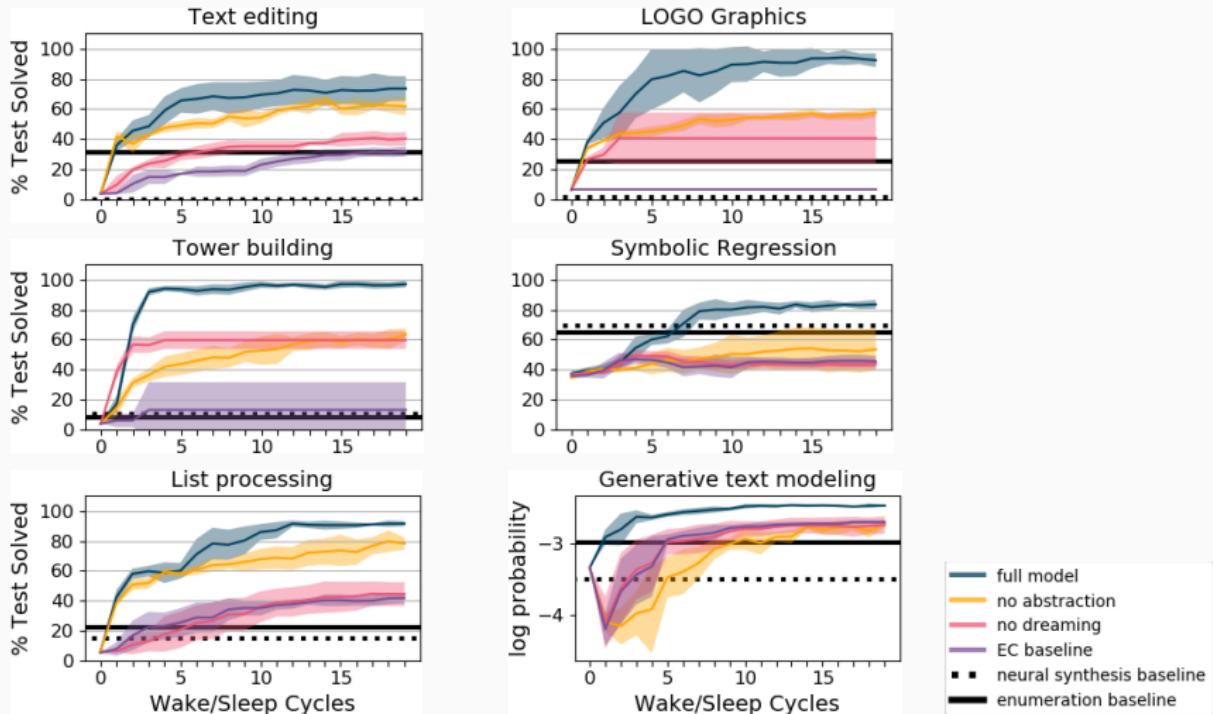


Learning dynamics

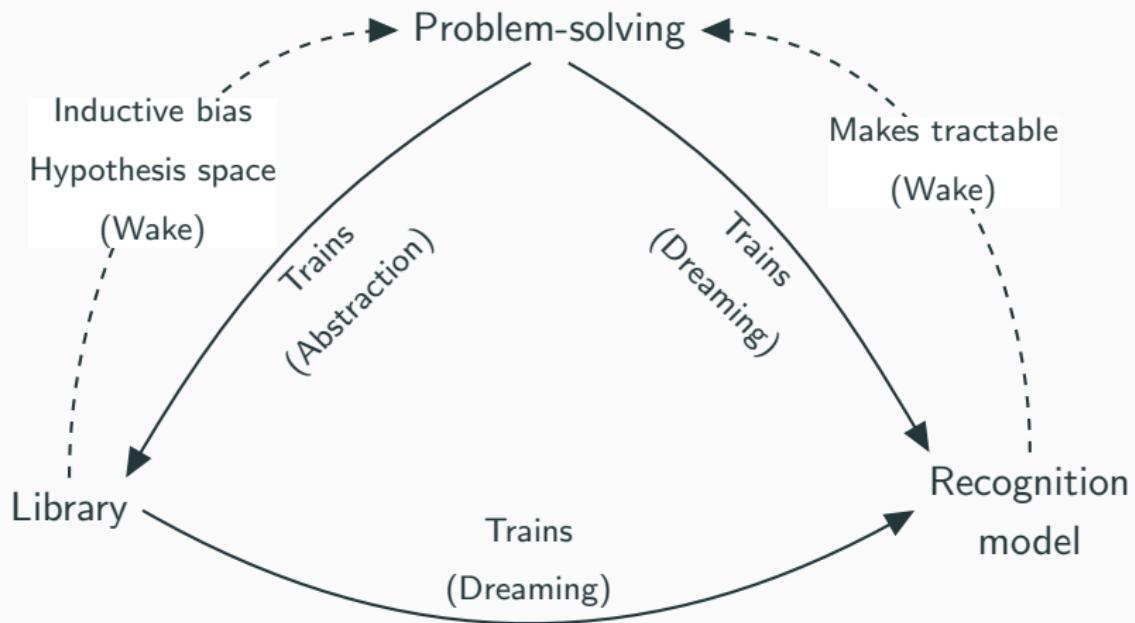


baselines: Exploration-Compression, EC (Dechter et al. 2013)
neural program synthesis, RobustFill (Devlin et al. 2017)
24 hours of brute-force enumeration

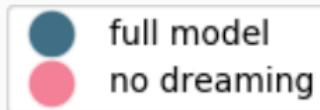
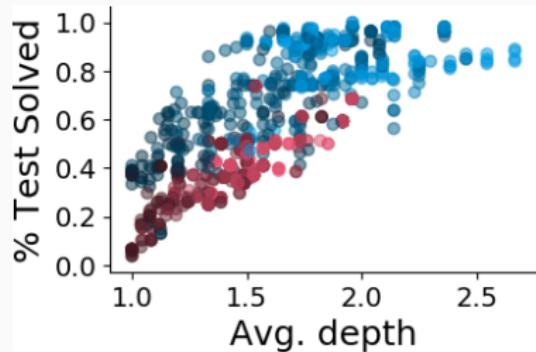
Learning dynamics



Synergy between dreaming and library learning



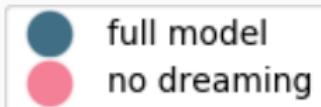
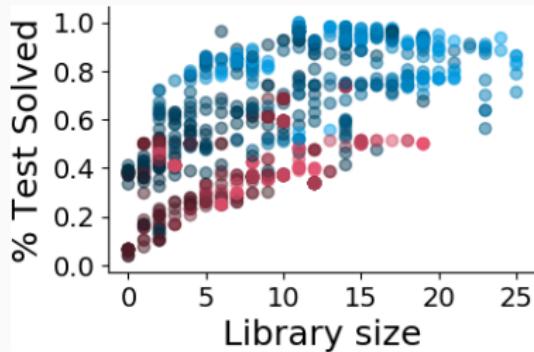
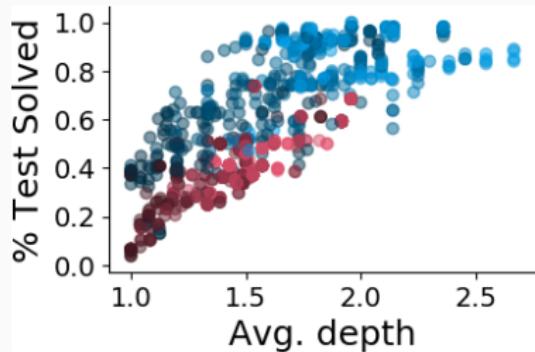
Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Lighter: Later in learning

Evidence for dreaming bootstrapping better libraries

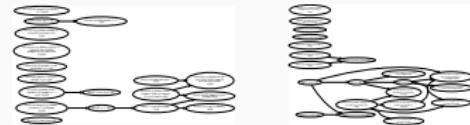


Darker: Early in learning

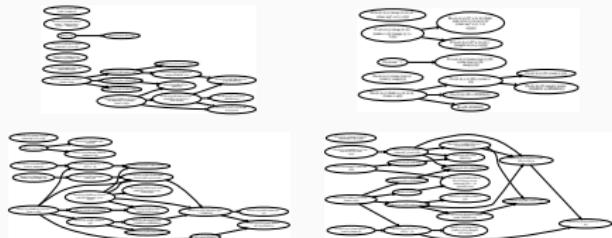
Lighter: Later in learning

Variability in learned library

List processing



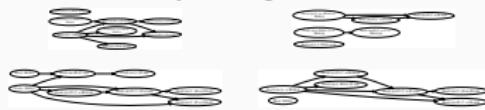
Text editing



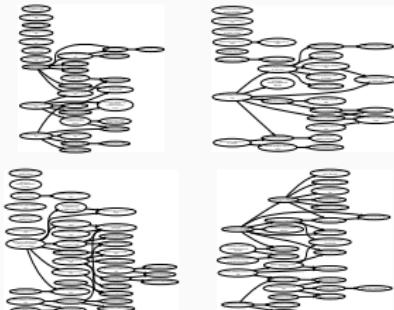
Tower building



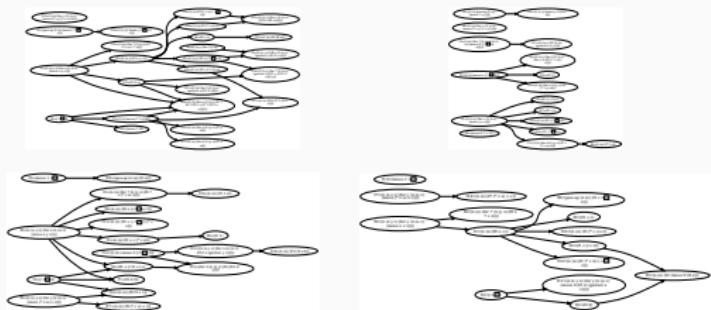
Symbolic regression



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

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

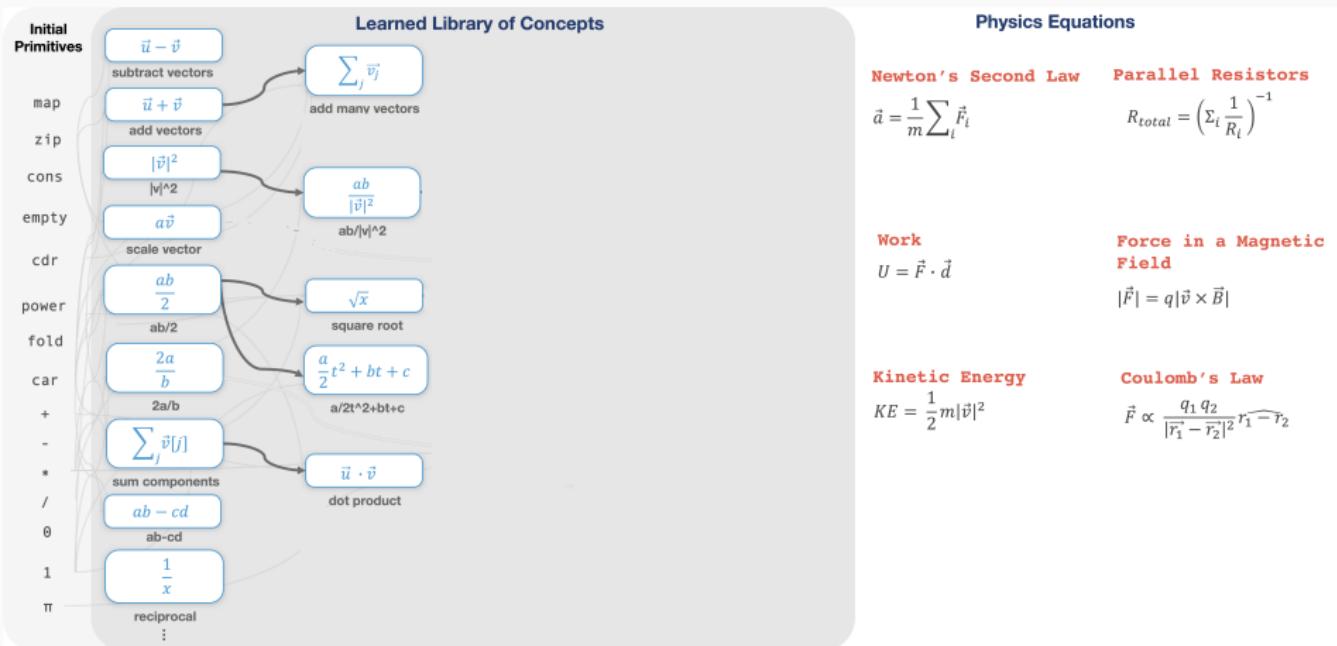
Coulomb's Law

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

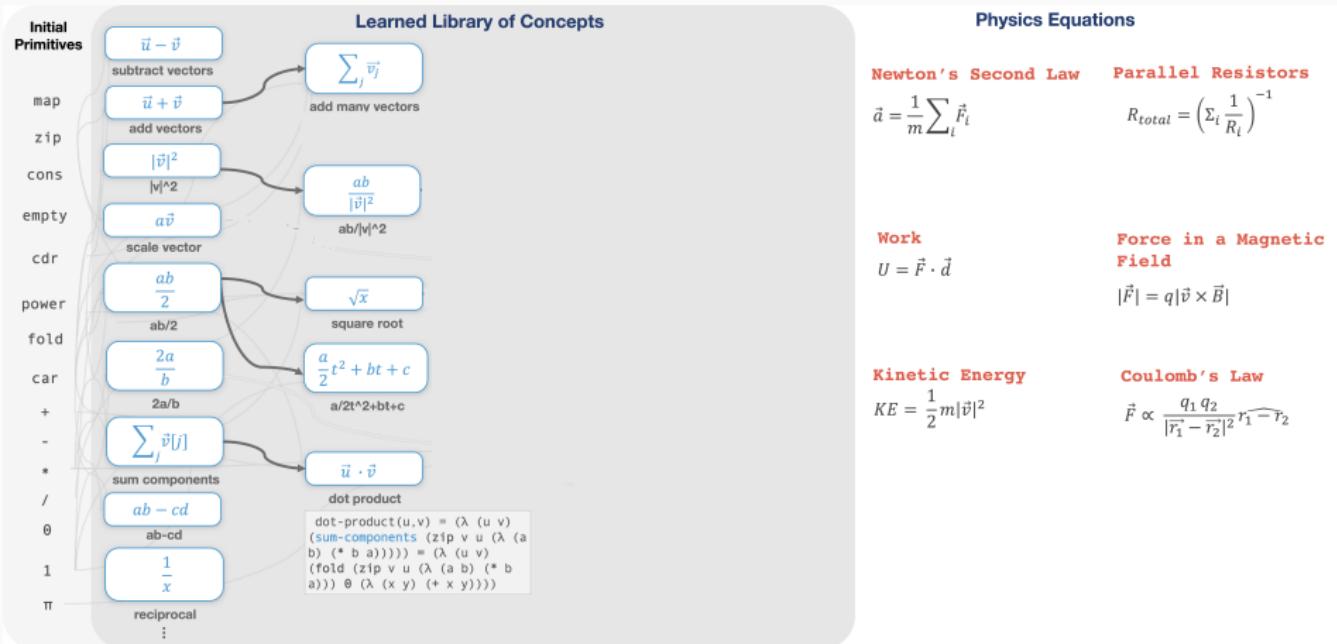
Growing languages for vector algebra and physics

Initial Primitives	Learned Library of Concepts	Physics Equations
	$\vec{u} - \vec{v}$ subtract vectors	
map	$\vec{u} + \vec{v}$ add vectors	Newton's Second Law Parallel Resistors
zip	$ \vec{v} ^2$ $ M ^2$	$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$
cons		$R_{total} = \left(\sum_i \frac{1}{R_i} \right)^{-1}$
empty	$a\vec{v}$	
cdr		
power	$\frac{ab}{2}$ ab/2	Work
fold		$U = \vec{F} \cdot \vec{d}$
car	$\frac{2a}{b}$ 2a/b	Force in a Magnetic Field
+		$ \vec{F} = q \vec{v} \times \vec{B} $
-	$\sum_j \vec{v}[j]$ sum components	
*		Kinetic Energy
/	$ab - cd$ ab-cd	$KE = \frac{1}{2} m \vec{v} ^2$
0		Coulomb's Law
1	$\frac{1}{x}$ reciprocal	$\vec{F} \propto \frac{q_1 q_2}{ \vec{r}_1 - \vec{r}_2 ^2} \hat{r}_1 - \hat{r}_2$
π		
	:	

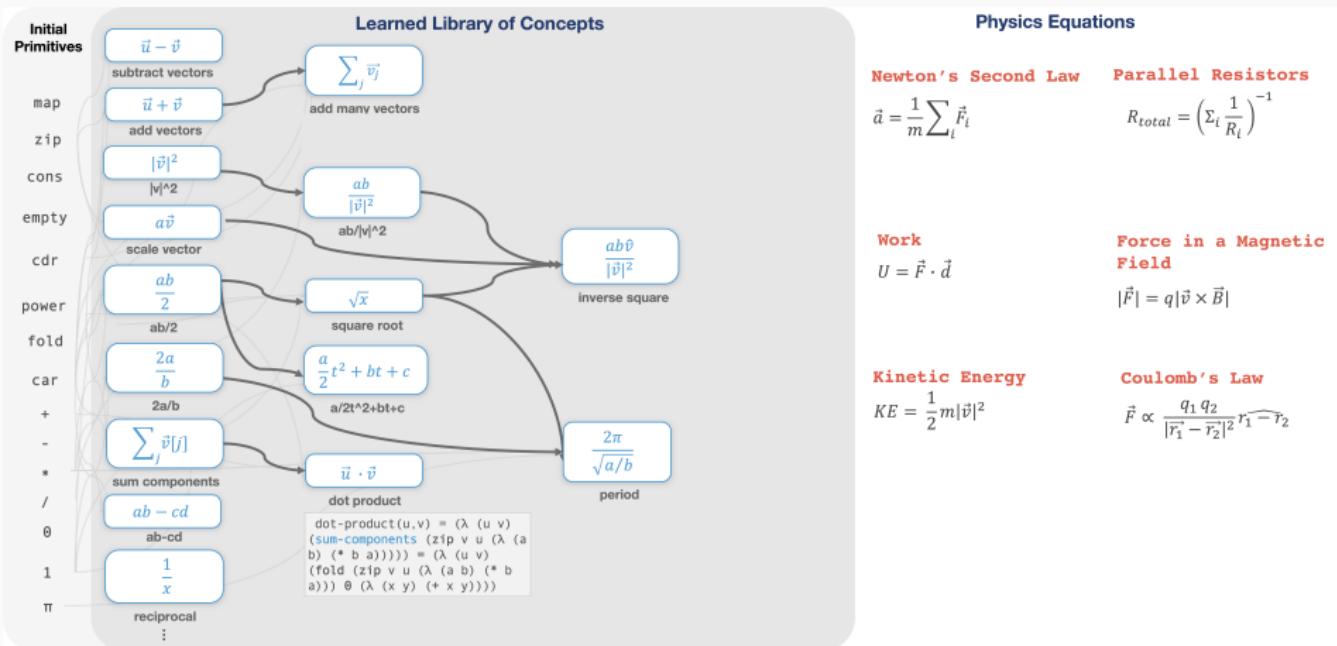
Growing languages for vector algebra and physics



Growing languages for vector algebra and physics



Growing languages for vector algebra and physics

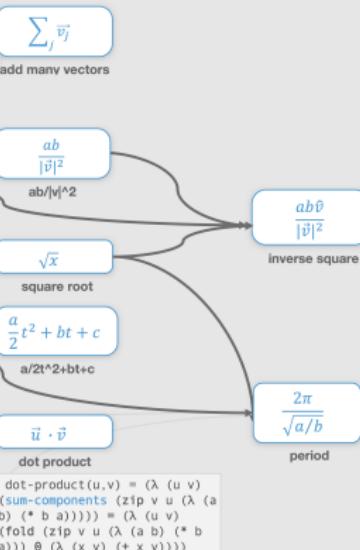


Growing languages for vector algebra and physics

Initial
Primitives

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

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

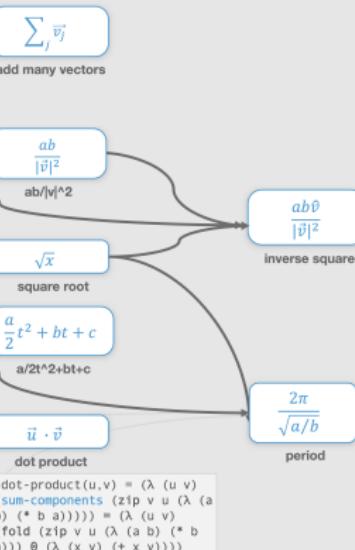
Initial
Primitives

```

 $\vec{u} - \vec{v}$ 
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))
```

Solution to Coulomb's Law if expressed in initial primitives

Growing a language for recursive programming

Initial Primitives

Y

combinator

cons

car

cdr

nil

if

nil?

+

-

0

1

=

Recursive Programming Algorithms

Stutter

[] → []
[] → []

Take every other

[] → []
[] → []

List lengths

[, []] → [3 1]
[[], [], []] → [2 0 1]

List differences

[1 8 2], [0 5 1] → [1 3 1]
[2 3 6], [1 2 4] → [1 1 2]

Growing a language for recursive programming

Initial Primitives

λ

combinator

cons

car

cdr

nil

if

nil?

+

-

0

1

=

Learned Library of Concepts

fold

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

unfold

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

Recursive Programming Algorithms

Stutter

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

Take every other

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

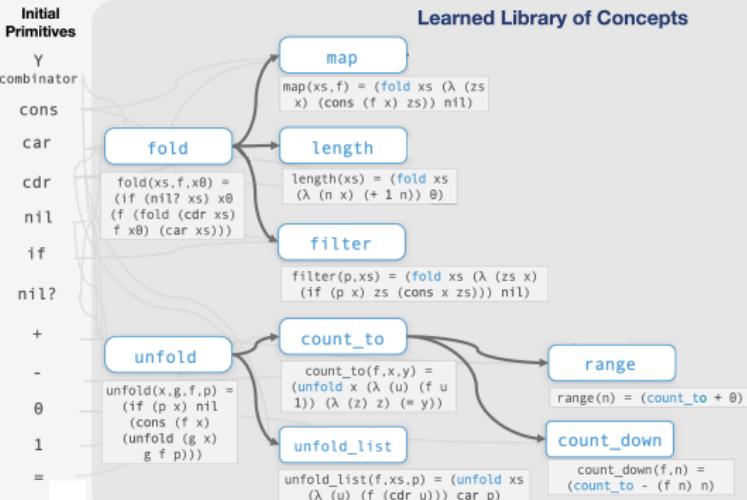
List lengths

```
[[■■■], [■]] → [3 1]  
[[■■■], [], [■]] → [2 0 1]
```

List differences

```
[1 8 2], [0 5 1] → [1 3 1]  
[2 3 6], [1 2 4] → [1 1 2]
```

Growing a language for recursive programming



Recursive Programming Algorithms

Stutter

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

Take every other

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

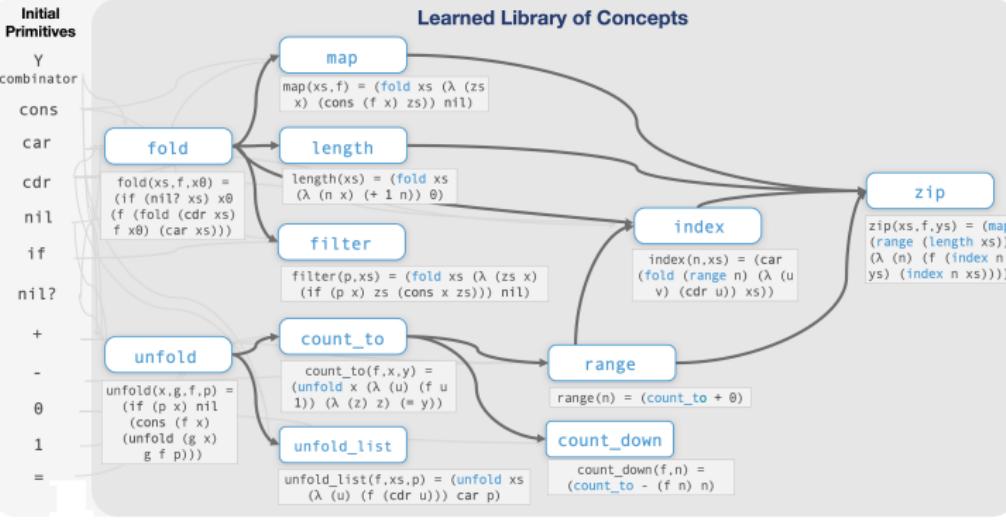
List lengths

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

List differences

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

Growing a language for recursive programming



Recursive Programming Algorithms

Stutter

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

Take every other

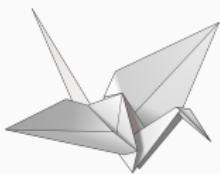
$[\text{■■■■■■■■}] \rightarrow [\text{■■}]$
 $[\text{■■■■■■■■■■}] \rightarrow [\text{■■■■}]$
 $(\text{unfold_list } \text{cdr } A \text{ nil?})$

List lengths

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

List differences

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

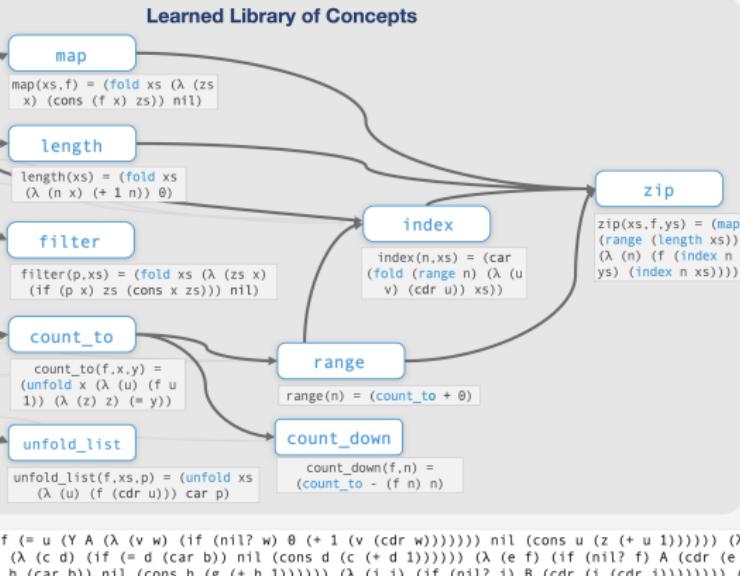


Origami Programming: Jeremy Gibbons, 2003

Growing a language for recursive programming

Initial Primitives

Y
 combinator
 cons
 car
 cdr
 nil
 if
 nil?
 +
 -
 0
 1
 =
 ~



Recursive Programming Algorithms

Stutter

$[\text{red blue}] \rightarrow [\text{red red blue blue}]$
 $[\text{black green black}] \rightarrow [\text{black black green green green green}]$
 $(\text{fold } A (\lambda(u v) (\text{cons } v (\text{cons } v u))) \text{ nil})$

Take every other

$[\text{red red black black}] \rightarrow [\text{red black}]$
 $[\text{purple purple black black green green}] \rightarrow [\text{purple black}]$
 $(\text{unfold_list } \text{cdr } A \text{ nil?})$

List lengths

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

List differences

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

Solution
if expressed in
initial primitives



Origami Programming: Jeremy Gibbons, 2003

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 carefully what to learn and what to build in

Program Induction and perception
learning to learn
model discovery
the future

Models of the physical world

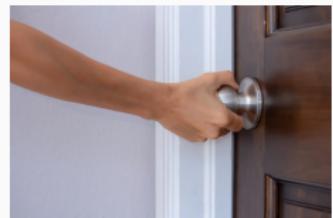
hinge

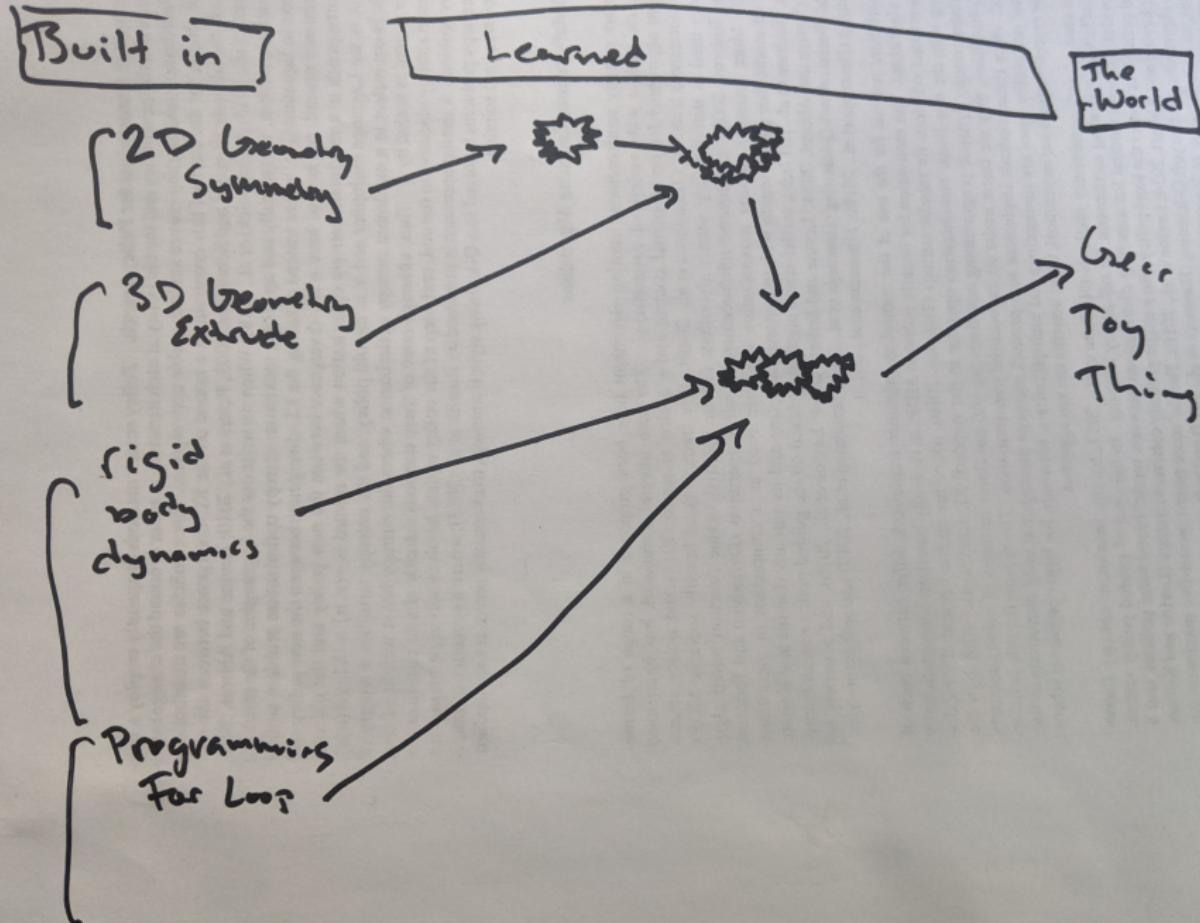


gear



doorknob





Collaborators

Tim O'Donnell



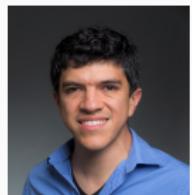
Josh Tenenbaum



Adam Albright



Armando
Solar-Lezama



Max Nye



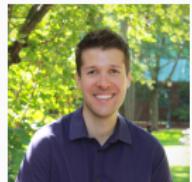
Cathy Wong



Yewen Pu



Dan Ritchie



Mathias Sable-Meyer



Lucas Morales



thank you