

~~Algorithms for Learning to Induce Programs~~

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



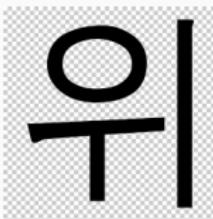
engineering



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 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?



Symbolic

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

Input interpretation:

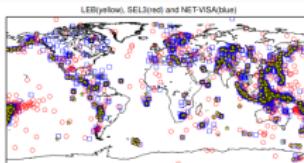
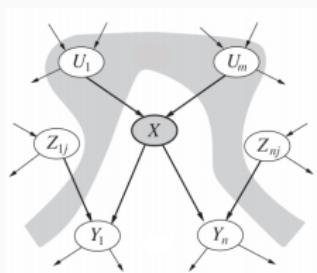
```
solve h w - h w^2 = Z for h
```

Result:

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

Three AI traditions

Probabilistic



What computational frameworks can contribute to this picture?



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

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

Input interpretation:

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

Result:

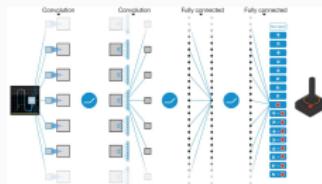
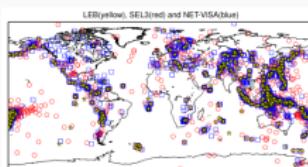
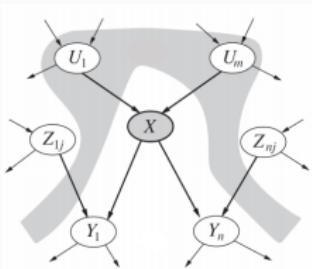
$$h = \frac{Z}{w - w^2} \text{ and } w^2 \neq w$$

Three AI traditions

Symbolic

Probabilistic

Neural



What computational frameworks can contribute to this picture?

Three AI traditions

Symbolic



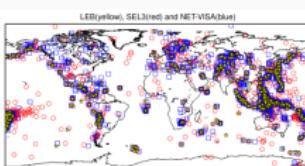
Program induction
machines that learn, perceive, and reason,
by writing their own code

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

Input interpretation:

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

Result:

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


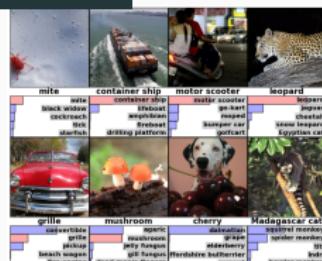
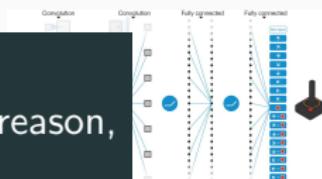
LEB(yellow), SEL3(red) and NETViSA(blue)

mite container ship motor scooter leopard

black widow cockroach ge-hart cheetah

cockroach arachnid cheetah

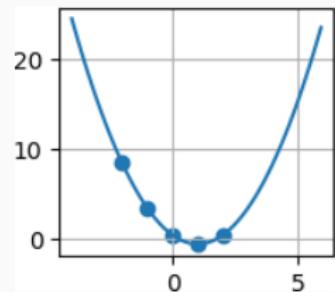
arachnid driving platform



Why program induction?

Why program induction?

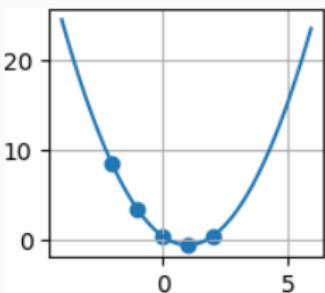
strong generalization
+data efficiency



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

Why program induction?

strong generalization
+data efficiency

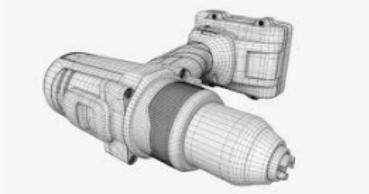


$$f(x) = (x-1)^2 - 0.5$$

interpretability

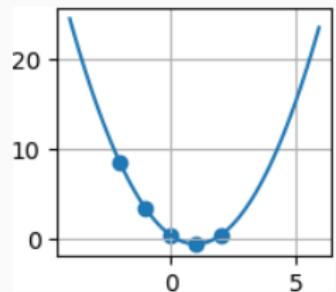


vs

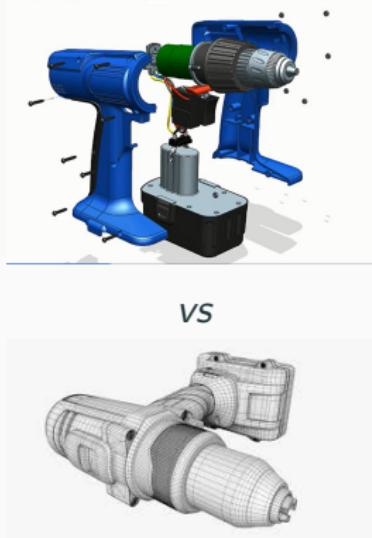


Why program induction?

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



John McCarthy, Ray Solomonoff

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

main obstacle: combinatorial search is hard

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

better compute+parallel algorithms

A lesson from the AI winter

We need an on-ramp of practical, tractable problems:

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

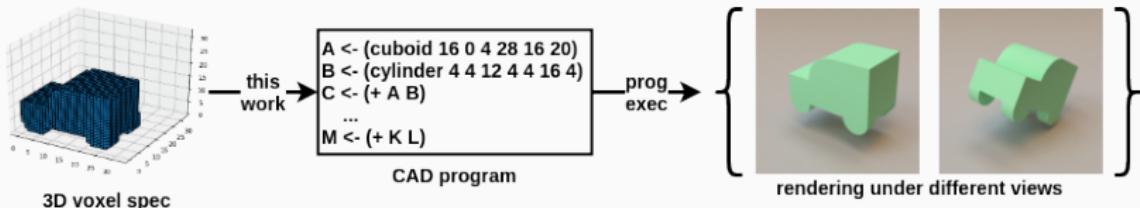
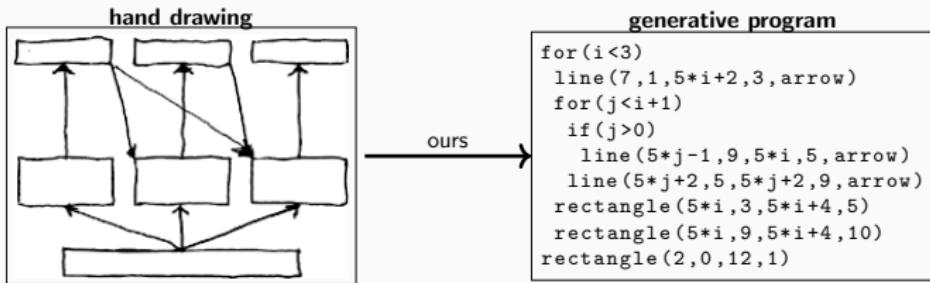
programming by examples [Gulwani 2011]

computer aided programming [Solar-Lezama 2008]

inverse procedural modeling [Kulkarni et al. 2015]

Perception, Synthesizing models, Learning-to-Learn

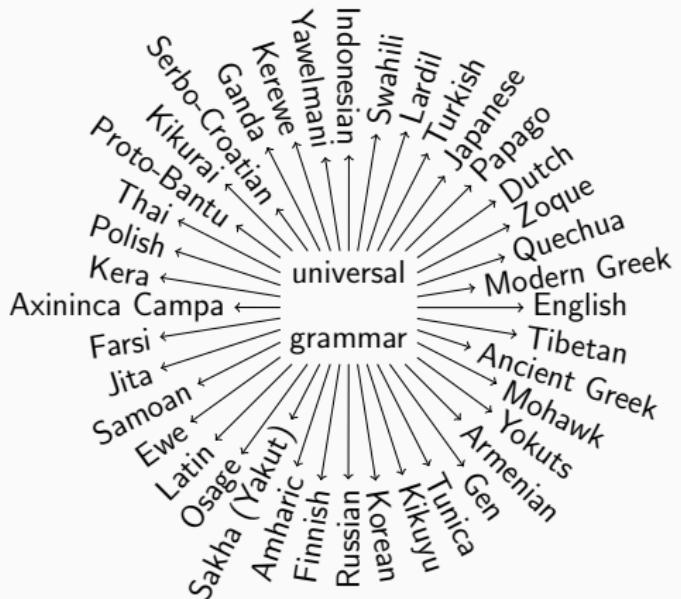
Theme #1: high-level visual understanding, pixels→programs



Perception, Synthesizing models, Learning-to-Learn

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

Theme #2: synthesizing human-understandable models



Perception, Synthesizing models, Learning-to-Learn

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

Theme #2: Synthesizing human-understandable models

Theme #3: learning to synthesize programs

List Processing

Sum List

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

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

Double

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

$[4 \ 5 \ 1] \rightarrow [8 \ 10 \ 2]$

Text Editing

Abbreviate

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

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

Drop Last Three

$\text{shrdlu} \rightarrow \text{shr}$

$\text{shakey} \rightarrow \text{sha}$

Regexes

Phone numbers

$(555) \ 867-5309$

$(650) \ 555-2368$

Currency

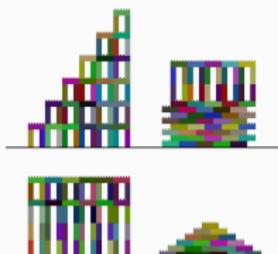
\$100.25

\$4.50

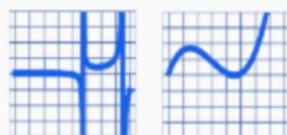
LOGO Graphics



Block Towers



Symbolic Regression



$$y = f(x)$$

Recursive Programming

Filter Red

$[\textcolor{red}{\blacksquare} \ \textcolor{blue}{\blacksquare} \ \textcolor{red}{\blacksquare} \ \textcolor{blue}{\blacksquare}] \rightarrow [\textcolor{blue}{\blacksquare} \ \textcolor{blue}{\blacksquare}]$

$[\textcolor{red}{\blacksquare} \ \textcolor{teal}{\blacksquare} \ \textcolor{red}{\blacksquare} \ \textcolor{teal}{\blacksquare}] \rightarrow [\textcolor{black}{\blacksquare} \ \textcolor{black}{\blacksquare} \ \textcolor{teal}{\blacksquare}]$

$[\textcolor{red}{\blacksquare} \ \textcolor{black}{\blacksquare} \ \textcolor{red}{\blacksquare} \ \textcolor{black}{\blacksquare}] \rightarrow [\textcolor{black}{\blacksquare} \ \textcolor{black}{\blacksquare}]$

Physical Laws

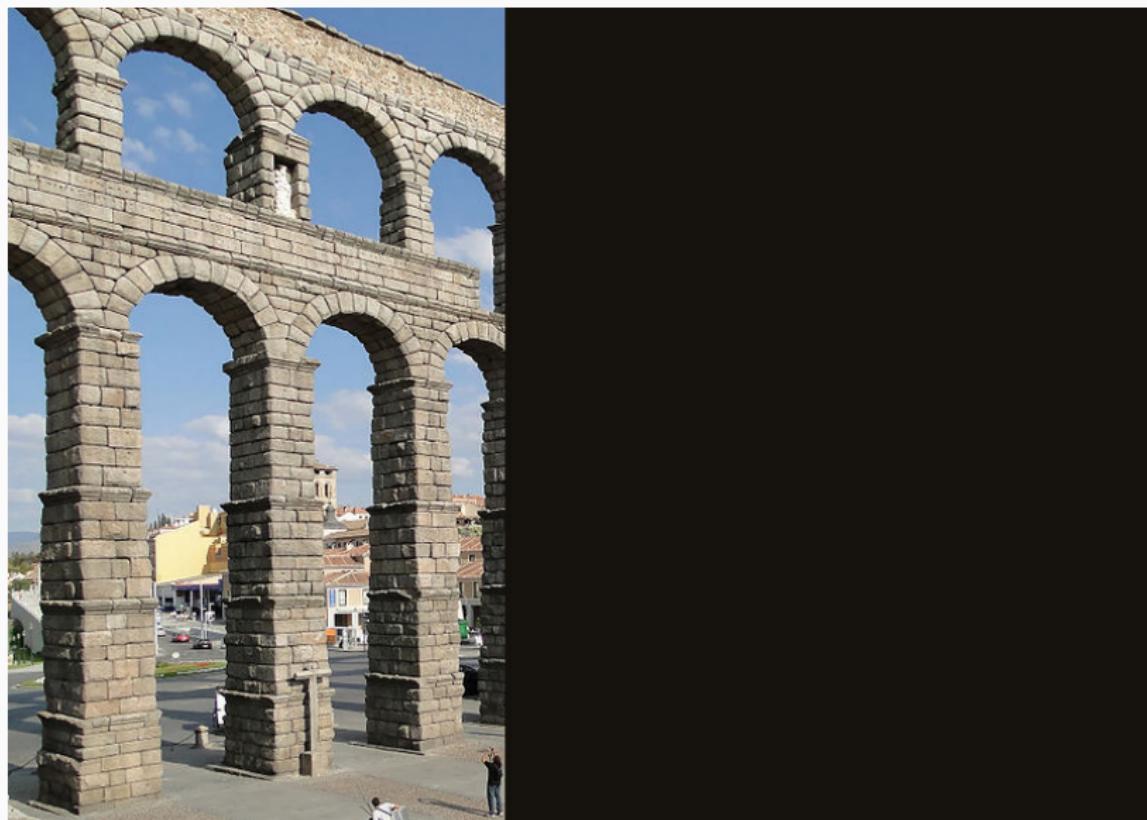
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

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

Program Induction and perception
model discovery
learning to learn

Vision is more than knowing what is where

Can you visually extrapolate this aqueduct?



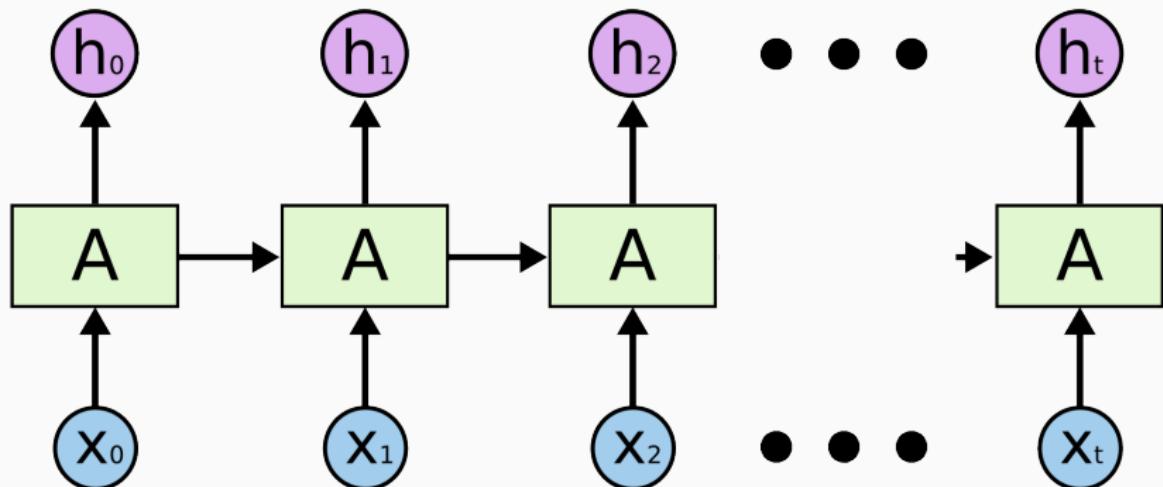
Vision is more than knowing what is where

Can you visually extrapolate this aqueduct?

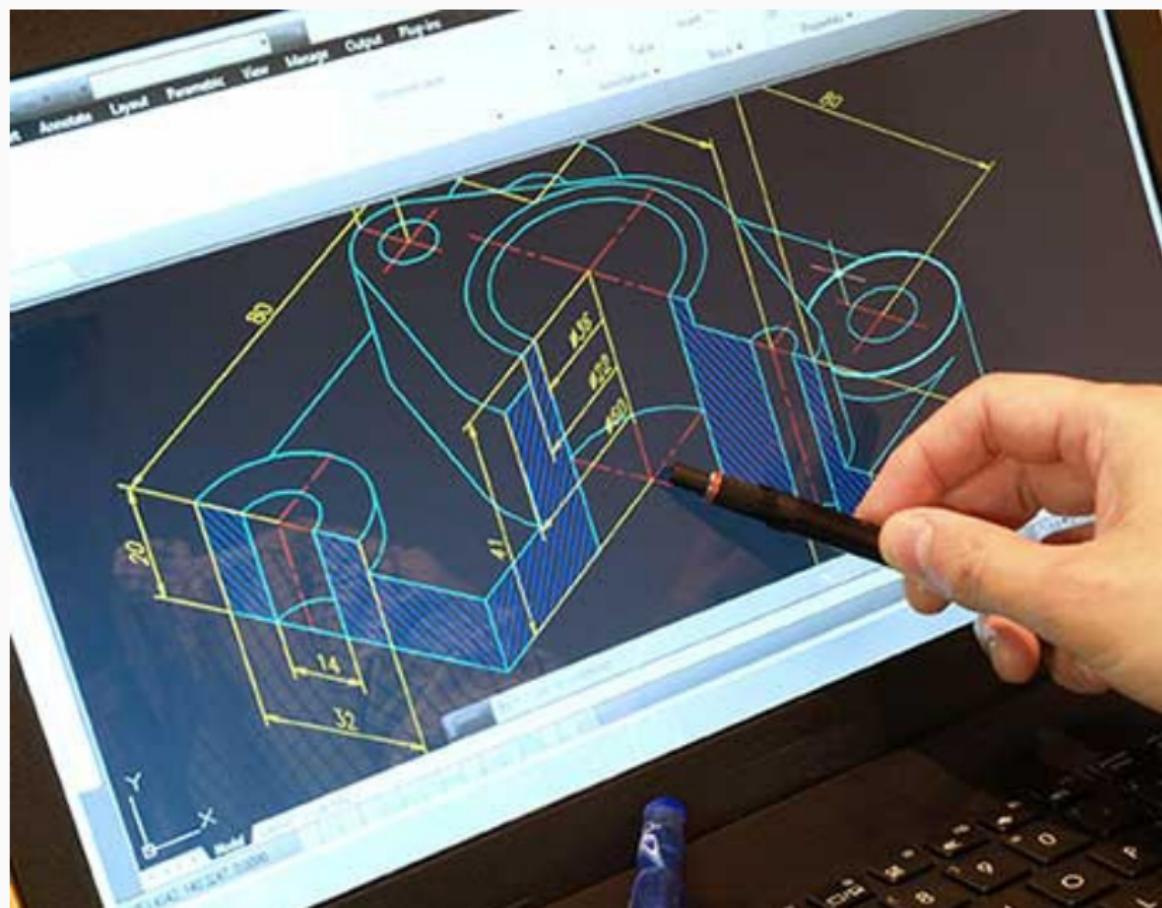


Vision is more than knowing what is where

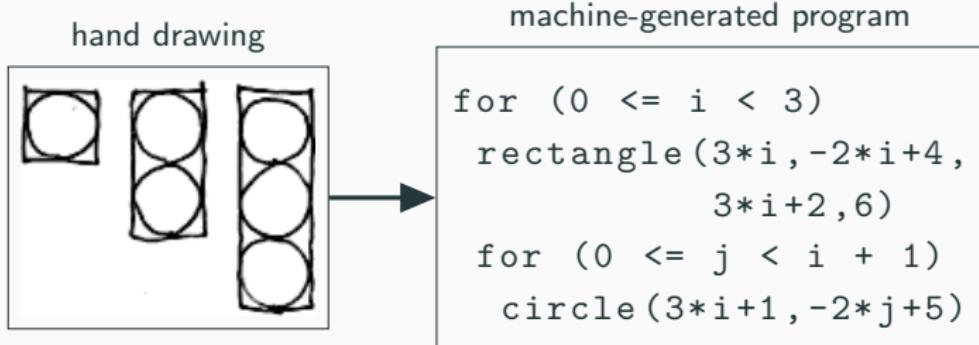
Can you infer what goes in the ellipses?



Vision is more than knowing what is where



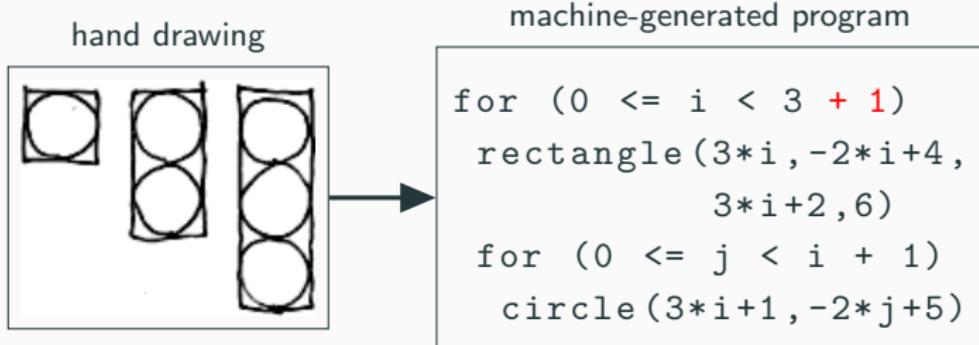
Learning to infer graphics programs from hand-drawn images



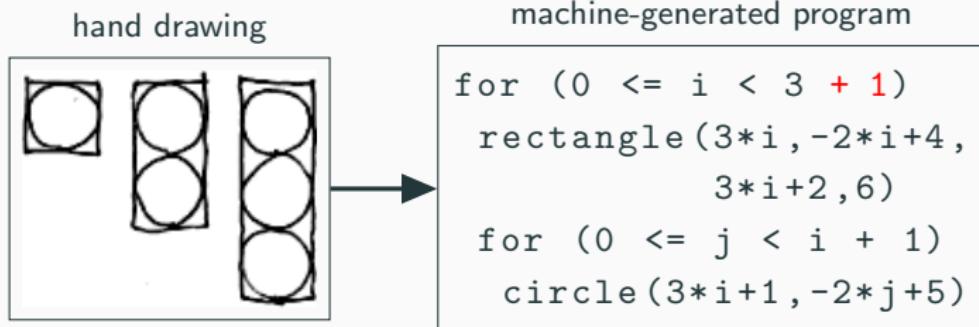
Dan Ritchie



Learning to infer graphics programs from hand-drawn images

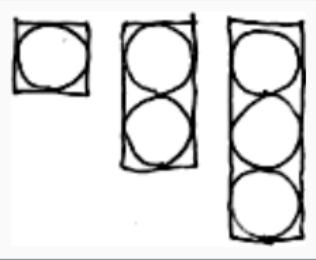


Learning to infer graphics programs from hand-drawn images



Learning to infer graphics programs from hand-drawn images

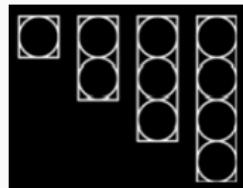
hand drawing



machine-generated program

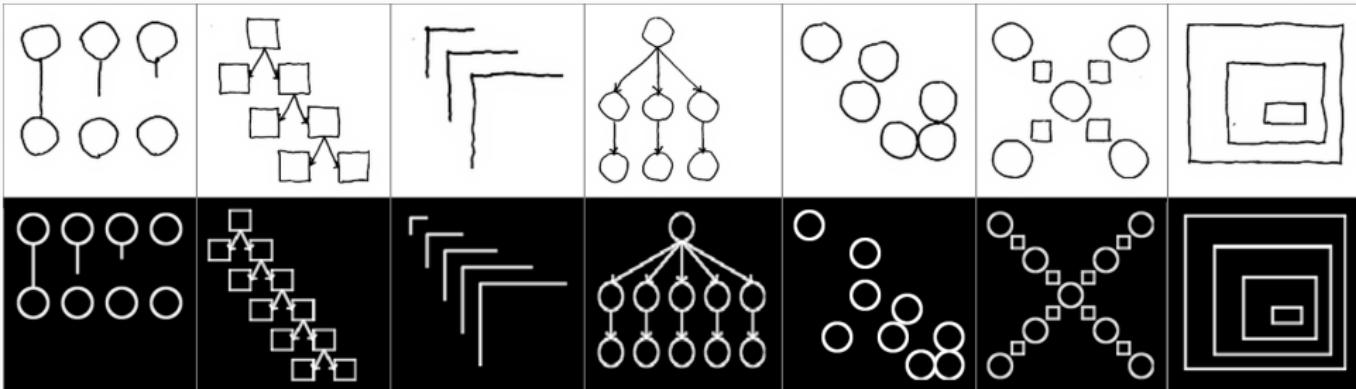
```
for (0 <= i < 3 + 1)
    rectangle(3*i, -2*i+4,
              3*i+2, 6)
    for (0 <= j < i + 1)
        circle(3*i+1, -2*j+5)
```

autogenerated
extrapolation

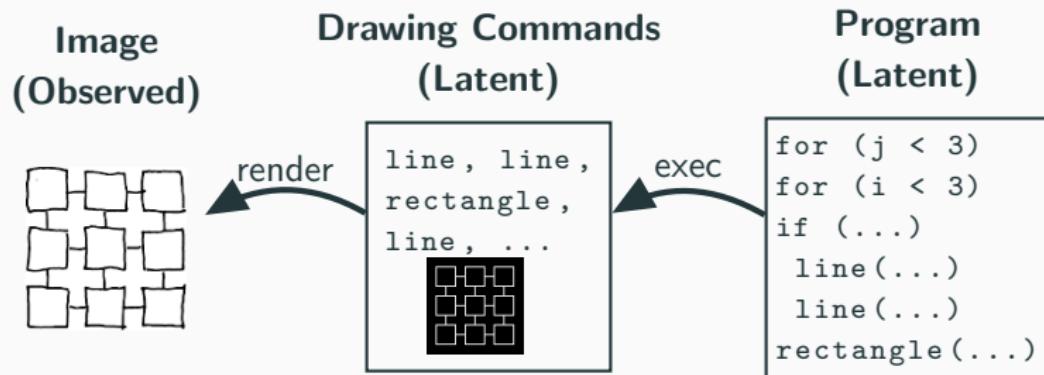


one-shot generalization / extrapolation

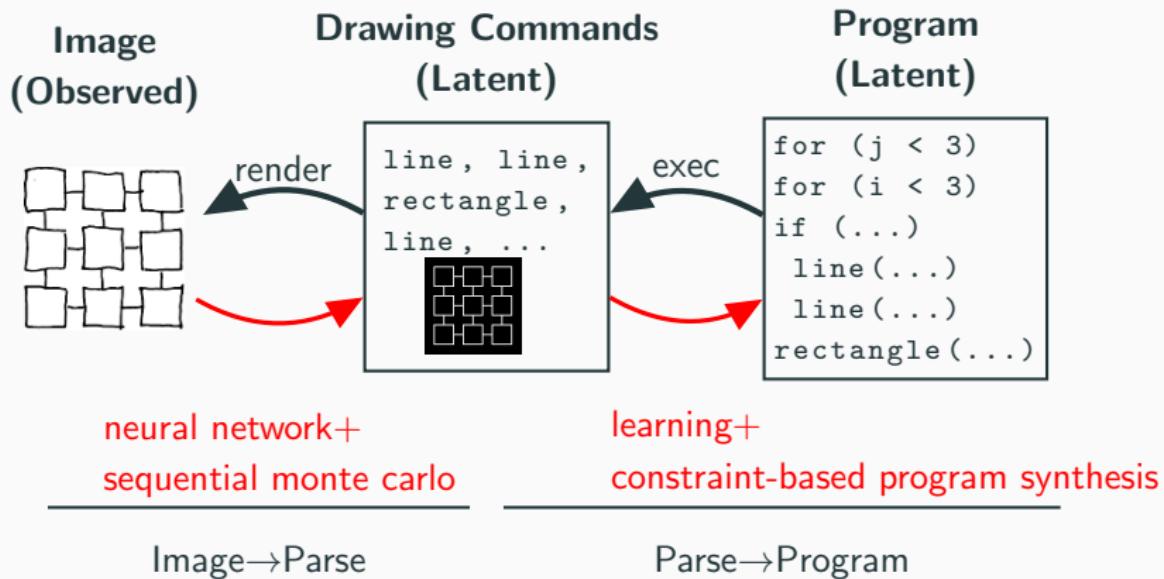
Extrapolation from a single image



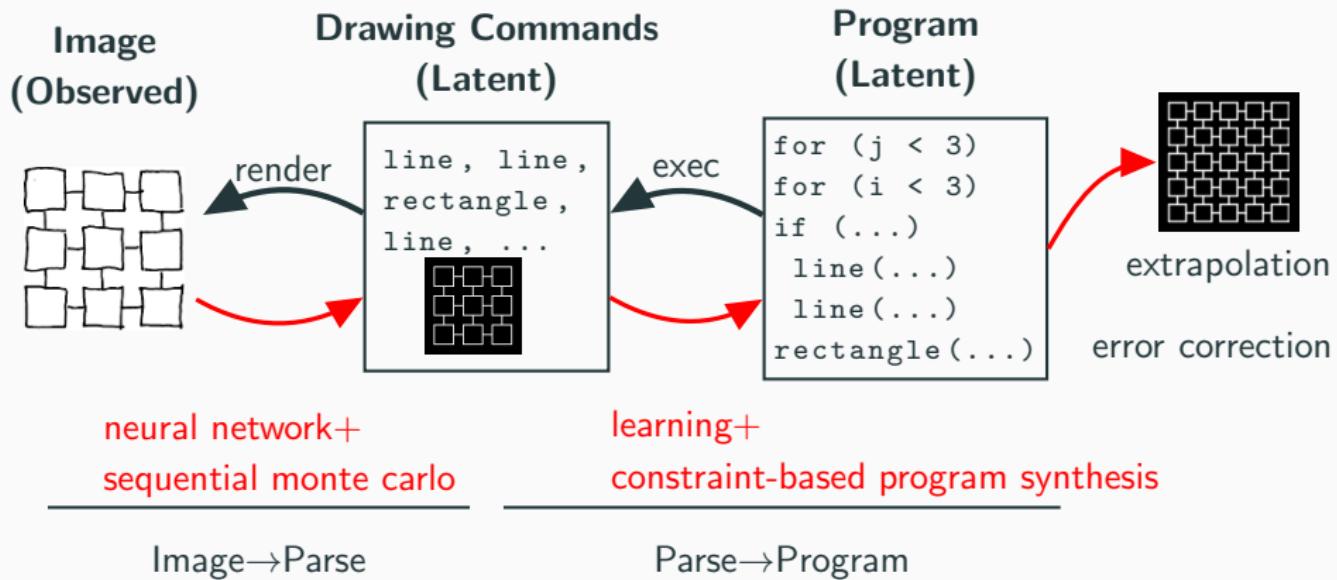
How to infer graphics programs from hand-drawn images



How to infer graphics programs from hand-drawn images

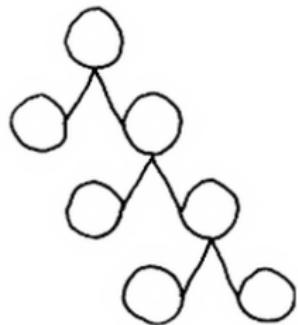


How to infer graphics programs from hand-drawn images



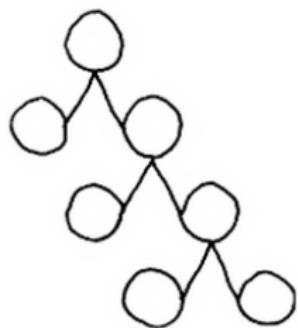
Top-down influences on perception

drawing

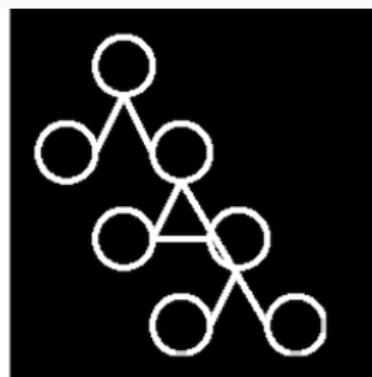


Top-down influences on perception

drawing

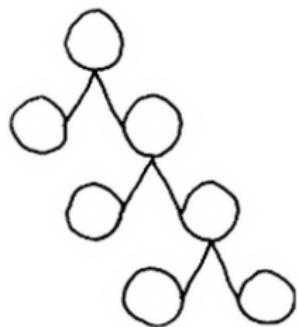


bottom-up neural net

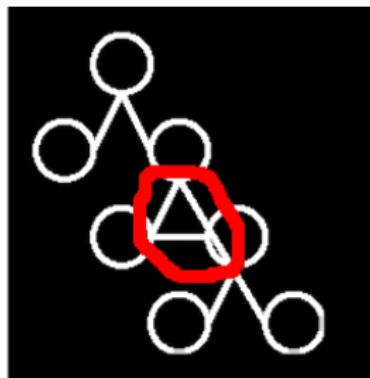


Top-down influences on perception

drawing

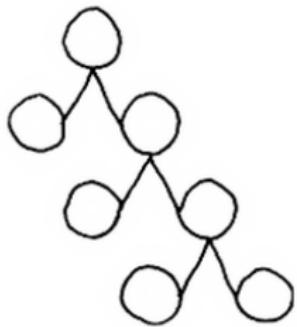


bottom-up neural net

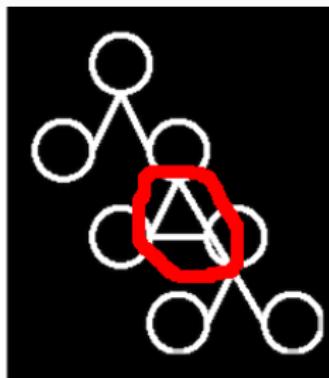


Top-down influences on perception

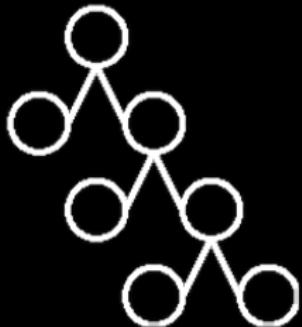
drawing



bottom-up neural net

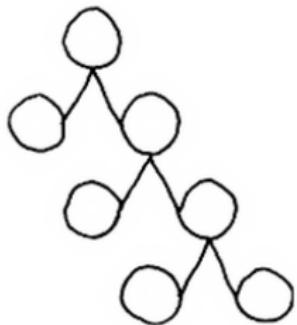


w/ top-down program bias

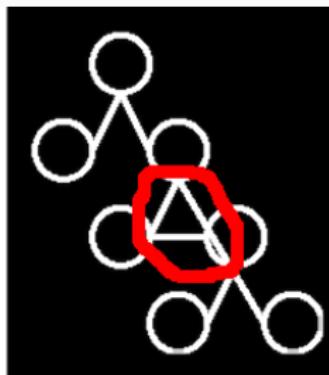


Top-down influences on perception

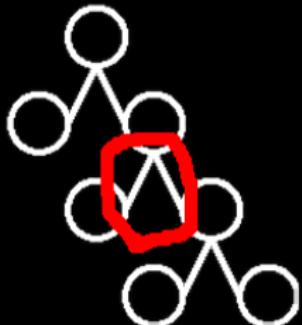
drawing



bottom-up neural net

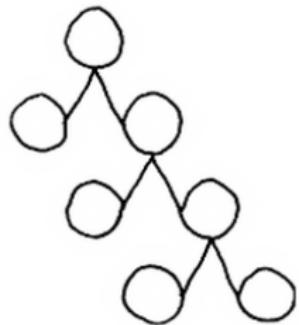


w/ top-down program bias

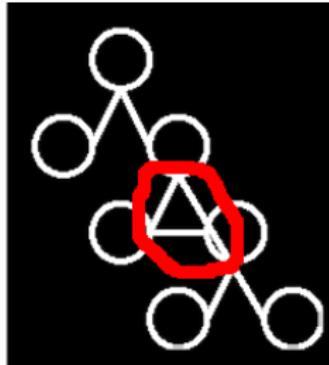


Top-down influences on perception

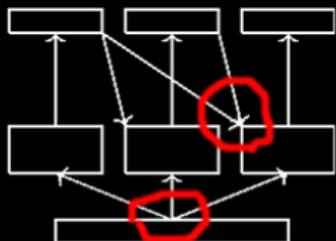
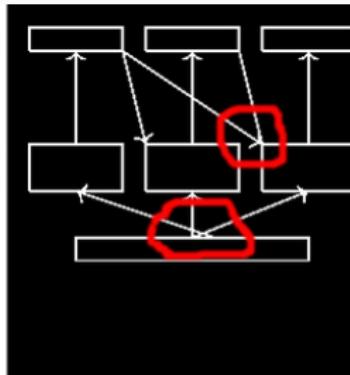
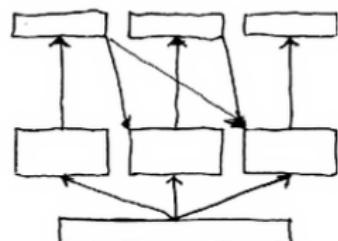
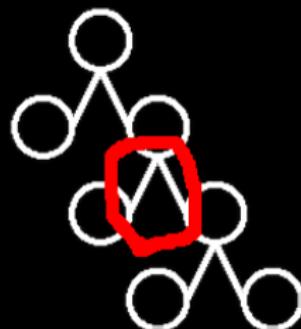
drawing



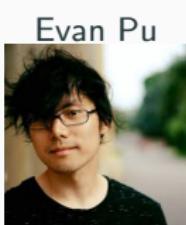
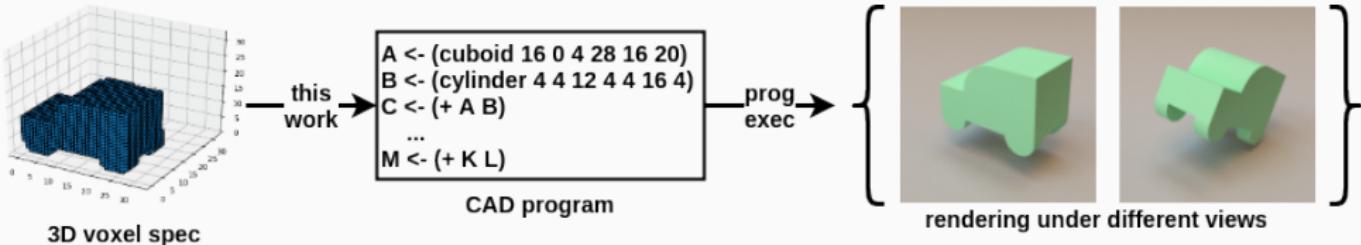
bottom-up neural net



w/ top-down program bias



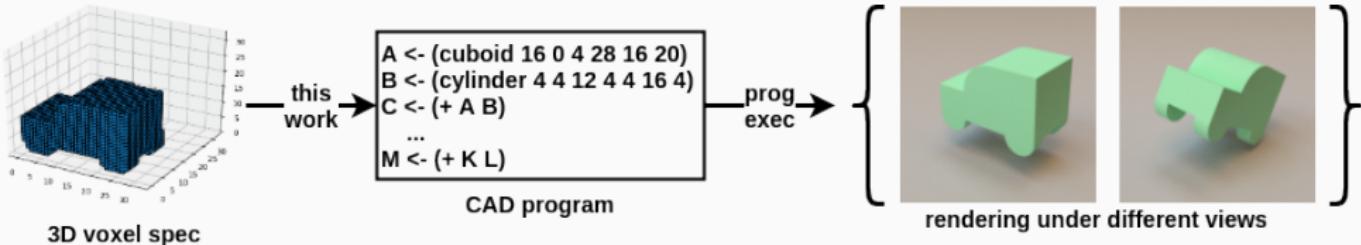
3D program induction



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

*equal contribution

3D program induction



Challenge: combinatorial search!

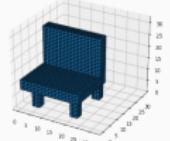
Branching factor: > 1.3 million per line of code, ≈ 20 lines of code
search space size: $(1.3 \text{ million})^{20} \approx 10^{122}$ programs

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

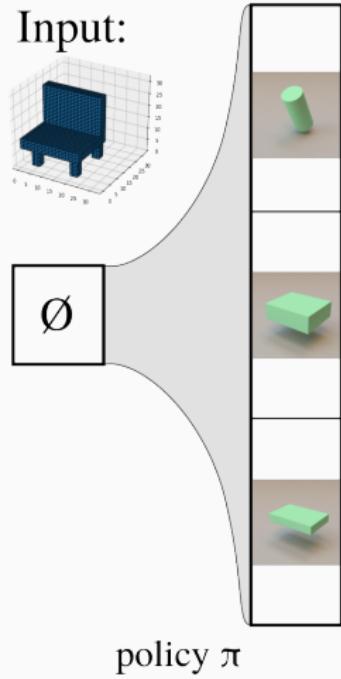
*equal contribution

Solution: stochastic **tree search** + learn **policy** that writes code
+ learn **value** function that assesses execution of program so far;
analogous to **AlphaGo** [Silver et al. 2016]

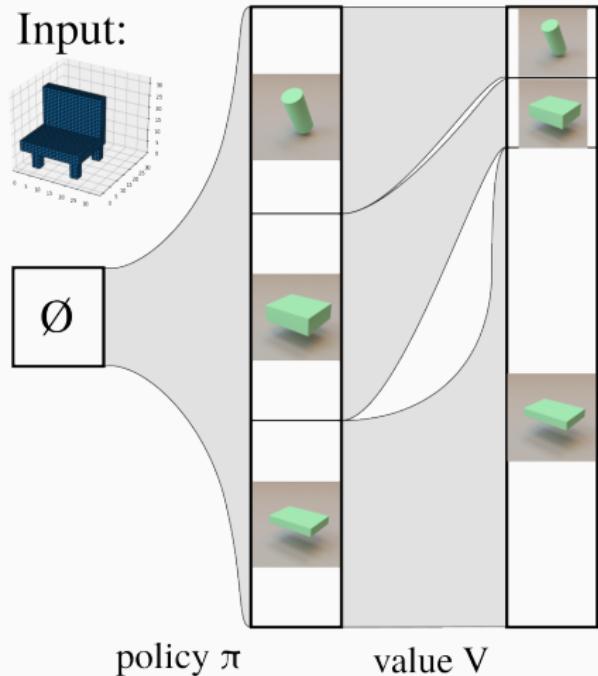
Input:



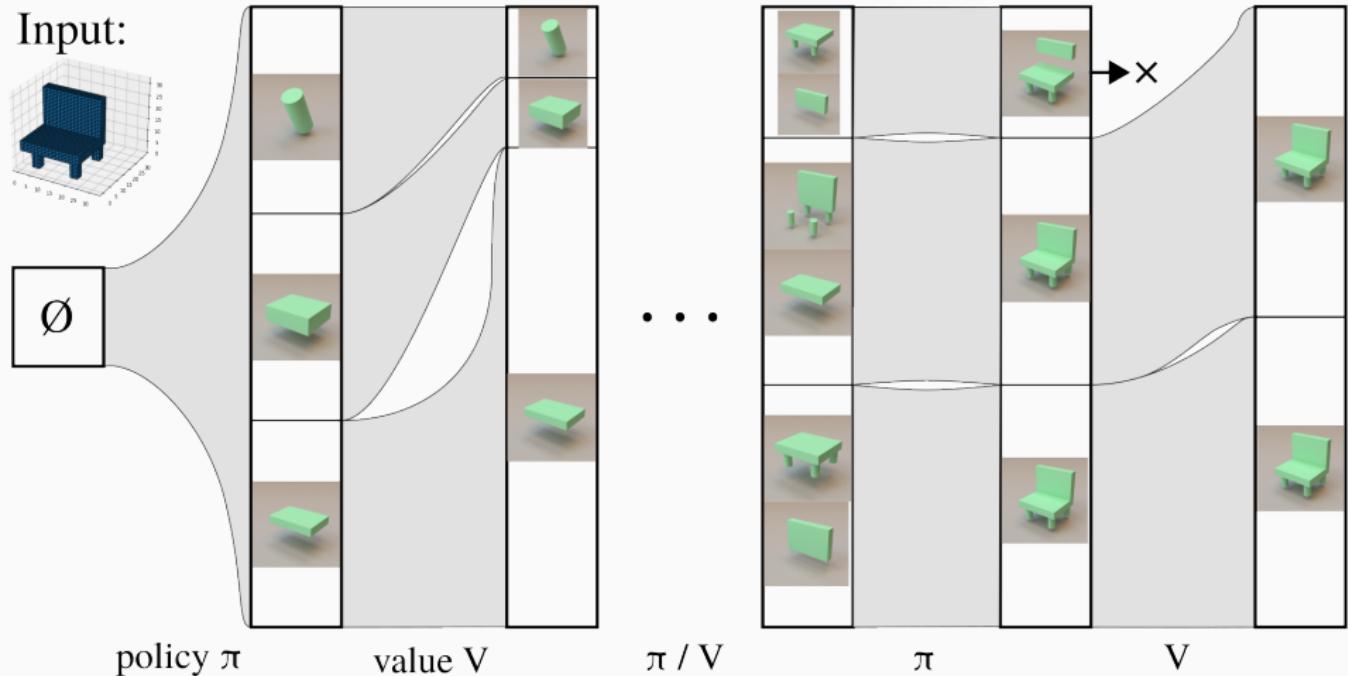
Solution: stochastic **tree search** + learn **policy** that writes code
+ learn **value** function that assesses execution of program so far;
analogous to **AlphaGo** [Silver et al. 2016]



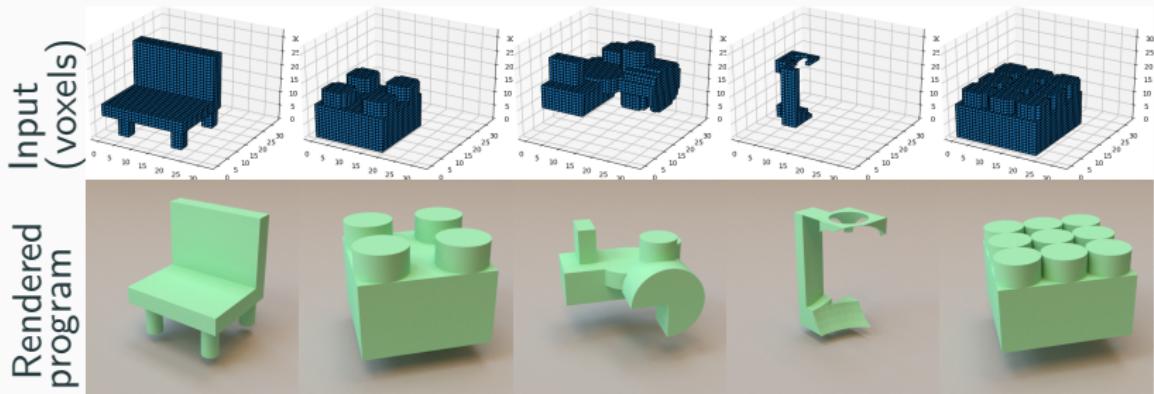
Solution: stochastic **tree search** + learn **policy** that writes code
+ learn **value** function that assesses execution of program so far;
analogous to **AlphaGo** [Silver et al. 2016]



Solution: stochastic **tree search** + learn **policy** that writes code
+ learn **value** function that assesses execution of program so far;
analogous to **AlphaGo** [Silver et al. 2016]



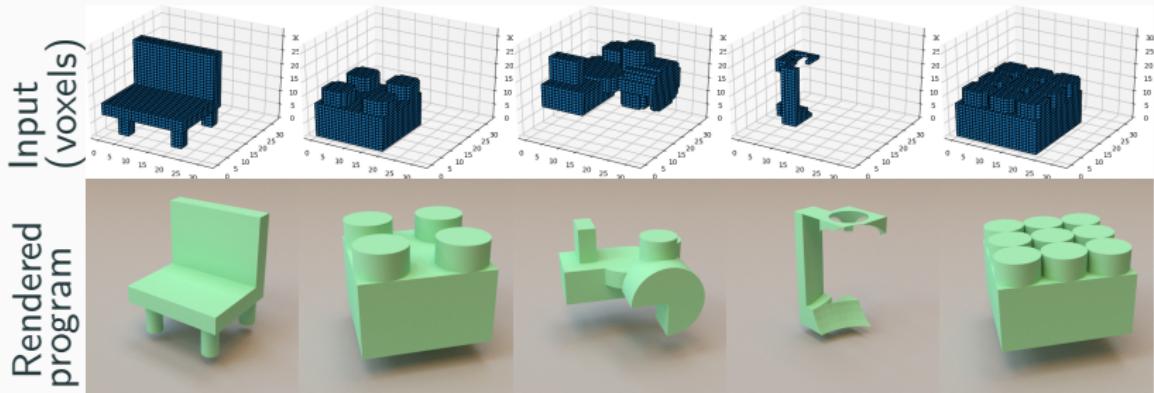
3D program induction



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

*equal contribution

3D program induction



**same architecture learns to synthesize text editing
programs (FlashFill, Gulwani 2012)**

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

*equal contribution

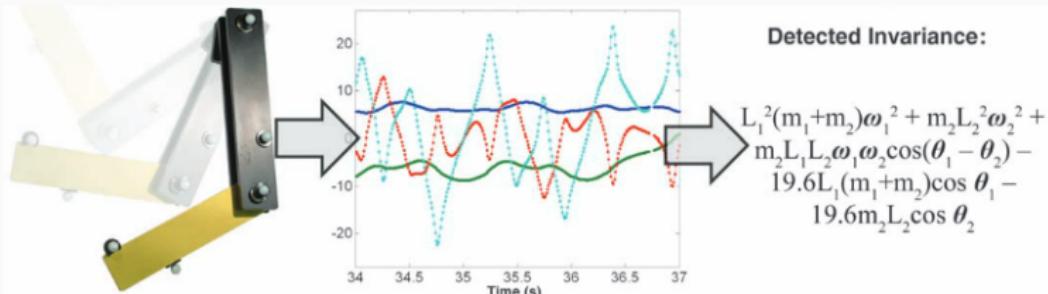
Lessons

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

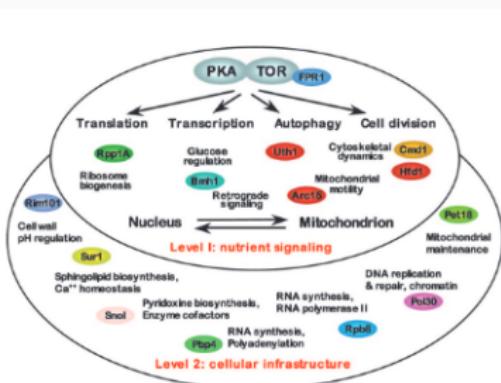
Combine the best of different techniques: neural nets for perception and pattern recognition, symbols for reasoning, Bayesian methods for uncertainty

Program Induction and perception
model discovery
learning to learn

Scientific discovery

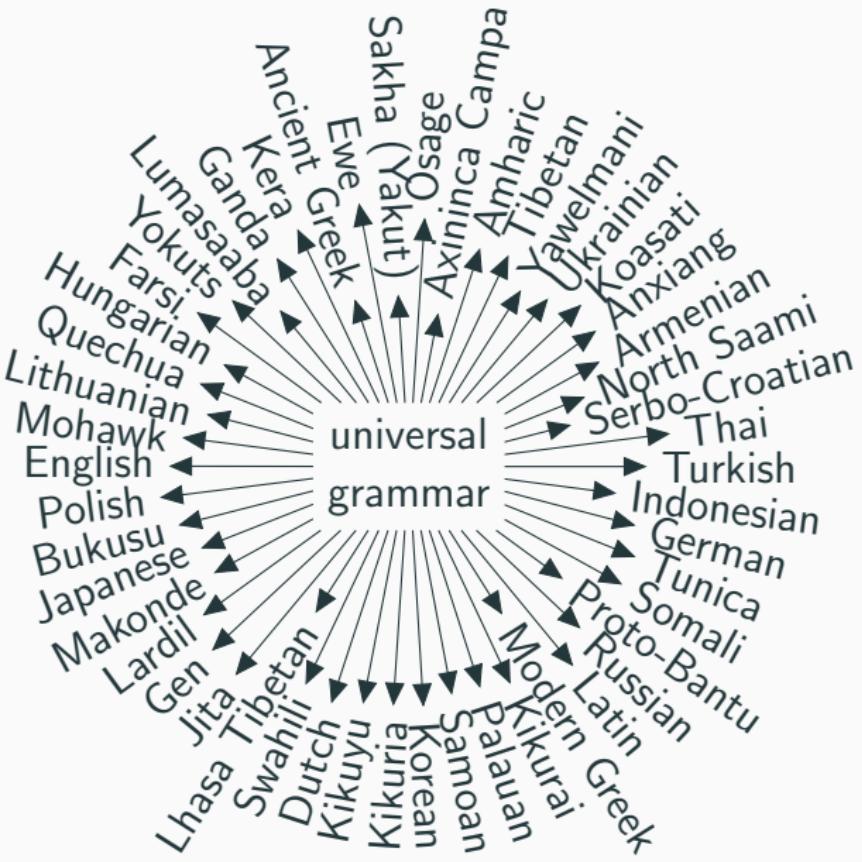


Schmidt & Lipson: "Distilling Free-Form Natural Laws from Experimental Data"



Lezon et al. 2006
inferring genetic interaction networks

Discovering human-understandable models of language



Tim O'Donnell



Few-shot language learning experiment

Mandarin:

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

Few-shot language learning experiment

Mandarin:

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

Few-shot language learning experiment

Mandarin:

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

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

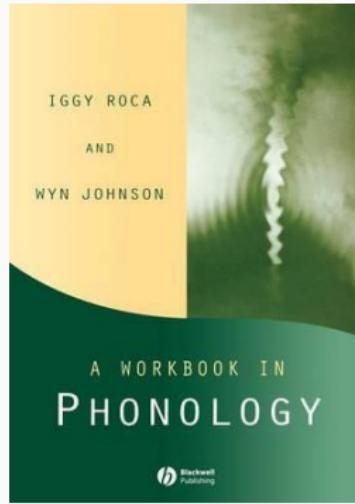
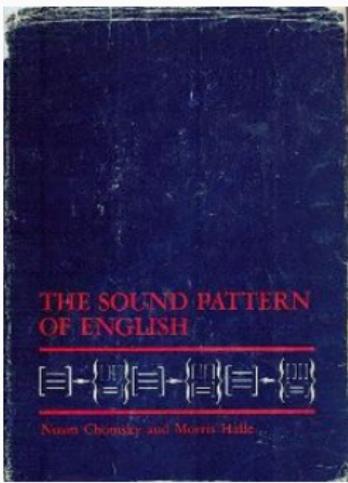
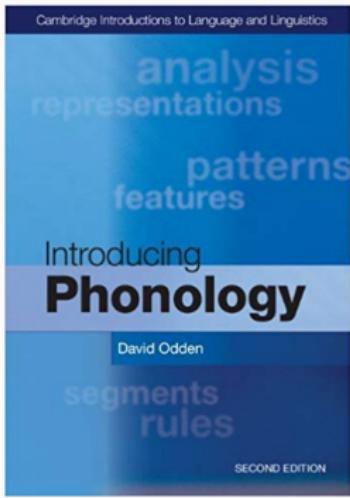
Serbo-Croatian:

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

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	üçügeyde	'good person'		
paarta	paartalar	paartaliin	'school desk'	ejiy	ejiyde	'elder sister'		
tia	tialar	tialiin	'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'	xatiij	xatignar	'birch'		
bie	bieler	bieliin	'mare'	aan	aannar	'doo'		
oyo	oyolor	oyoluun	'child'	tiij	tiigner	'squirrel'		
xopto	xoptolor	xoptoluun	'gull'	sordoj	sordognor	'pike'		
börö	börölör	böröliün	'wolf'	olom	olomnor	'ford'		
tial	tiallar	tialiin	'wind'	oron	oronnor	'bed'		
ial	iallar	ialliin	'neighbor'	bödöj	bödögñör	'strong one'		
kuul	kuullar	kuulluun	'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'	xopto	xoptoto	xoptotooyor	xoptotton	'gull'
sorox	soroxtor	soroxtuun	'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'
ötöx	ötöxtör	ötöxtüün	'abandoned farm'	kuul	kuulla	kuullaaýar	kuultan	'sack'
ubay	ubaydar	ubaydiin	'elder brother'	moxsoyol	moxsoyollo	moxsoyollooyor	moxsyoitolon	'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'
				oyus	oyusta	oyustaayar	oyustan	'bul'
				kus	kusta	kustaayar	kustan	'duck'
				tünnük	tünnükte	tünnükteeyer	tünnükten	'window'

Turkic Sakha (Yakut)

observed data

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	is̥kaap	is̥kaaptar

138 total examples

Turkic Sakha (Yakut)

grammar (unobserved)

SINGULAR→stem
PLURAL→stem+lar

observed data

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	iskaap	iskaaptar

138 total examples

Turkic Sakha (Yakut)

constraint-based
synthesis
[Solar-Lezama 2008]
test-driven synthesis
[Perelman et al. 2016]

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”

observed data

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	iskaap	iskaaptar

138 total examples

Turkic Sakha (Yakut)

constraint-based
synthesis
[Solar-Lezama 2008]
test-driven synthesis
[Perelman et al. 2016]

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 : ̄skaap

observed data

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	̄skaap	̄skaaptar

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)

observed data

BED : oron

BEDS

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)

observed data

BED : oron

BEDS→oron+lar→oronlar

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)

observed data

BED : oron

BEDS → oron+lar → oronlar $\xrightarrow{r_1}$ orondar

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)

observed data

BED : oron

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

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"

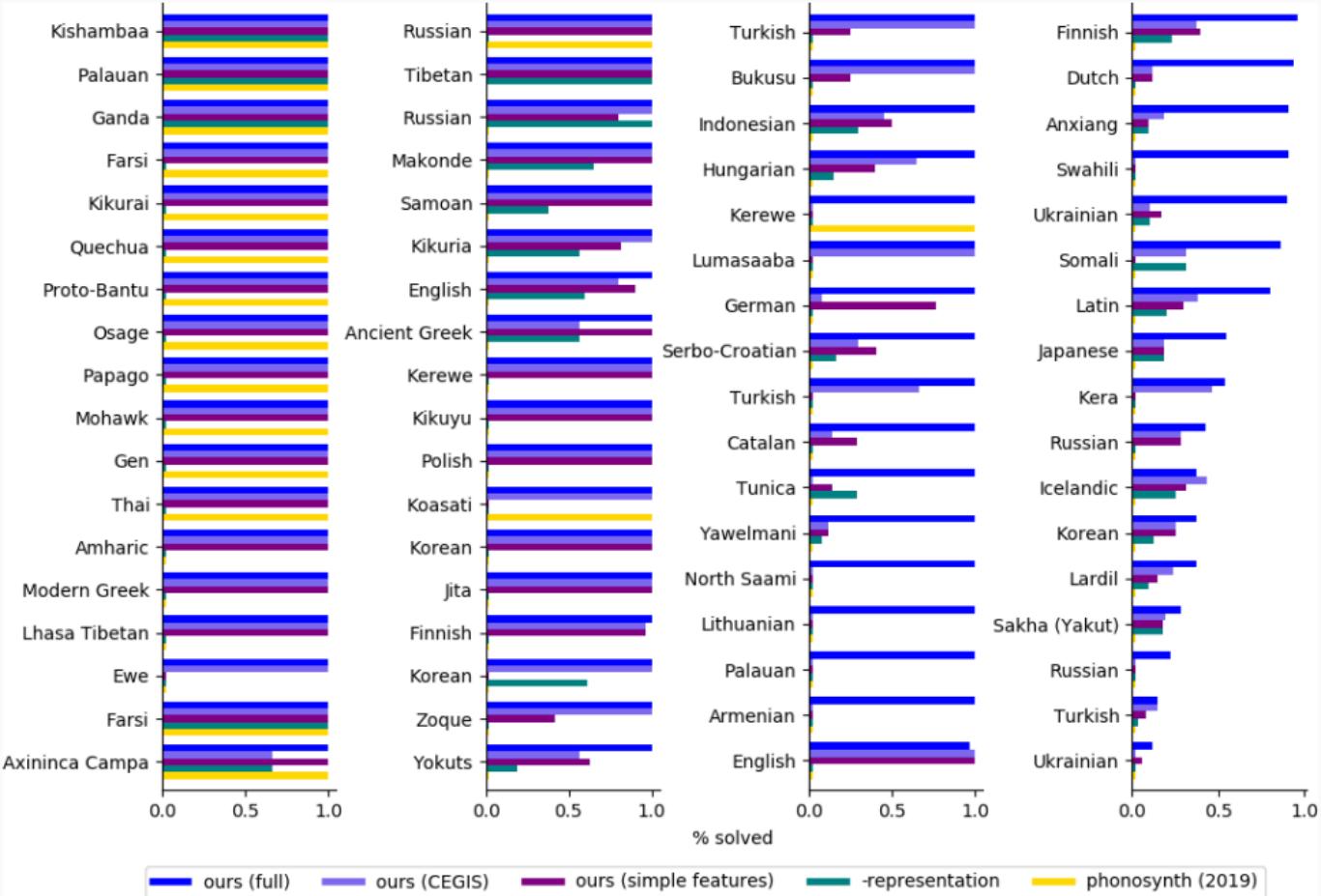
stems
(unobserved)

observed data

BED : oron

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

Kishambaa	Russian	Turkish	Finnish
Palauan	Tibetan	Bukusu	Dutch
Ganda	Russian	Indonesian	Anxiang
Farsi	Makonde	Hungarian	Swahili
Kikurai	Samoan	Kerewe	Ukrainian
Quechua	Kikuria	Lumasaaba	Somali
Proto-Bantu	English	German	Latin
Osage	Ancient Greek	Serbo-Croatian	Japanese
Papago	Kerewe	Turkish	Kera
Mohawk	Kikuyu	Catalan	Russian
Gen	Polish	Tunica	Icelandic
Thai	Koasati	Yawelmani	Korean
Amharic	Korean	North Saami	Lardil
Modern Greek	Jita	Lithuanian	Sakha (Yakut)
Lhasa Tibetan	Finnish	Palauan	Russian
Ewe	Korean	Armenian	Turkish
Farsi	Zoque	English	Ukrainian
Axininca Campa	Yokuts		



Distilling higher-level knowledge

Ewe
data

Gen
data

Jita
data

Thai
data

Kera
data

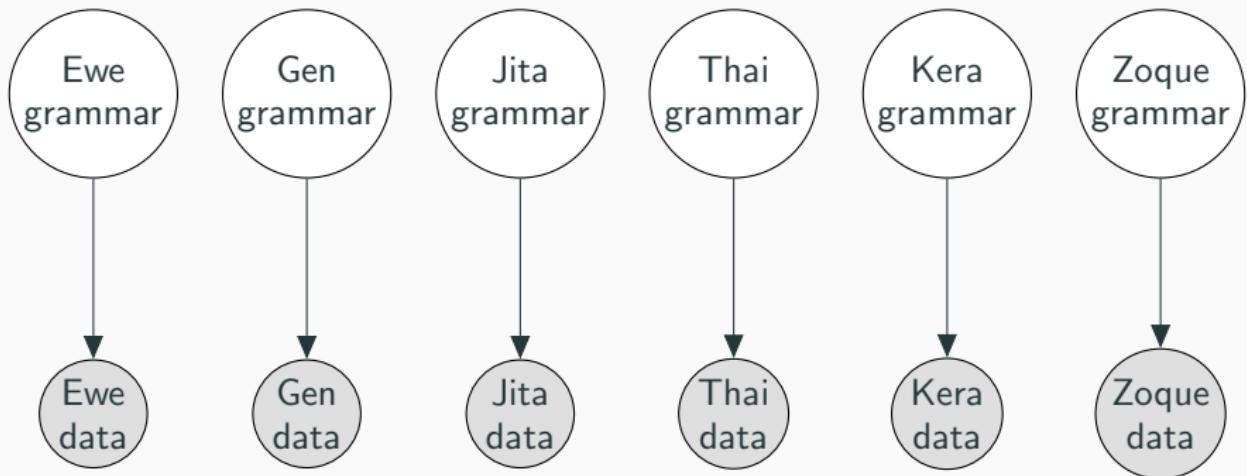
Zoque
data

Distilling higher-level knowledge

dark: we know it

light: we wanna know it

A → B: A makes B

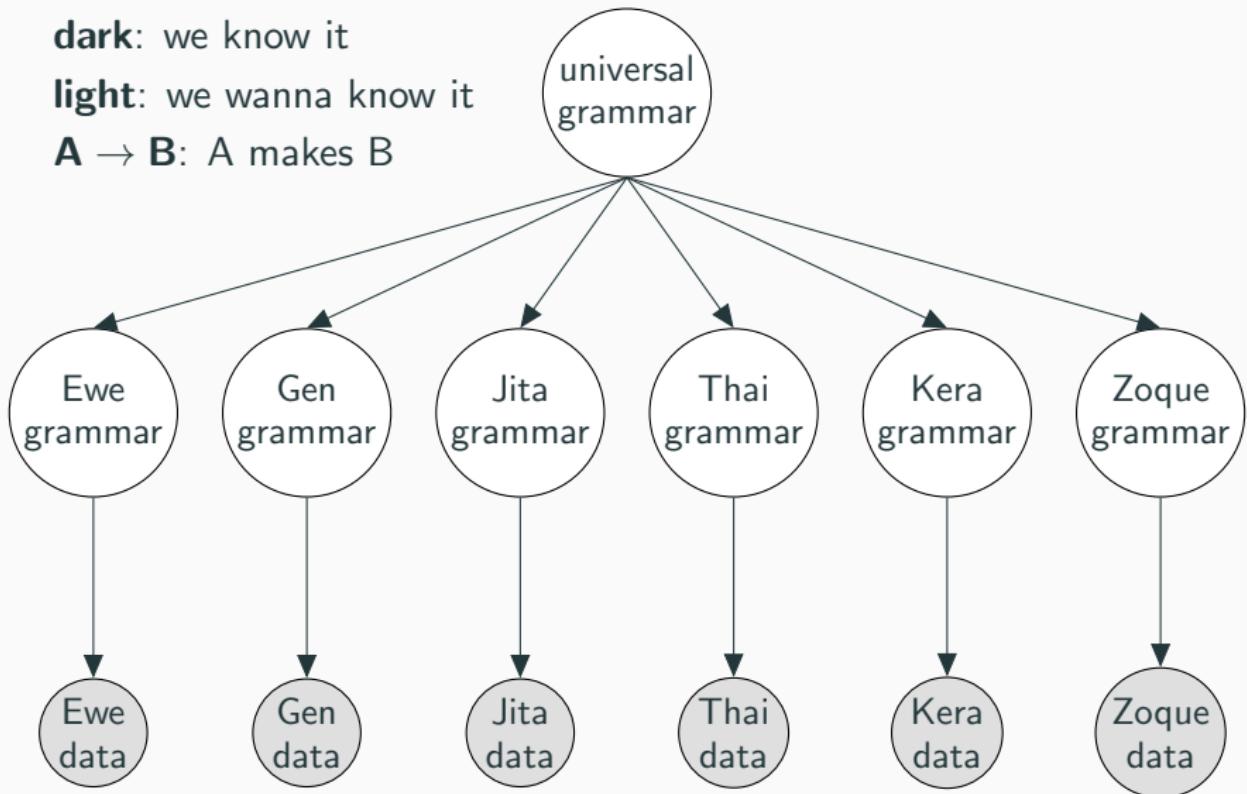


Distilling higher-level knowledge

dark: we know it

light: we wanna know it

A → B: A makes B

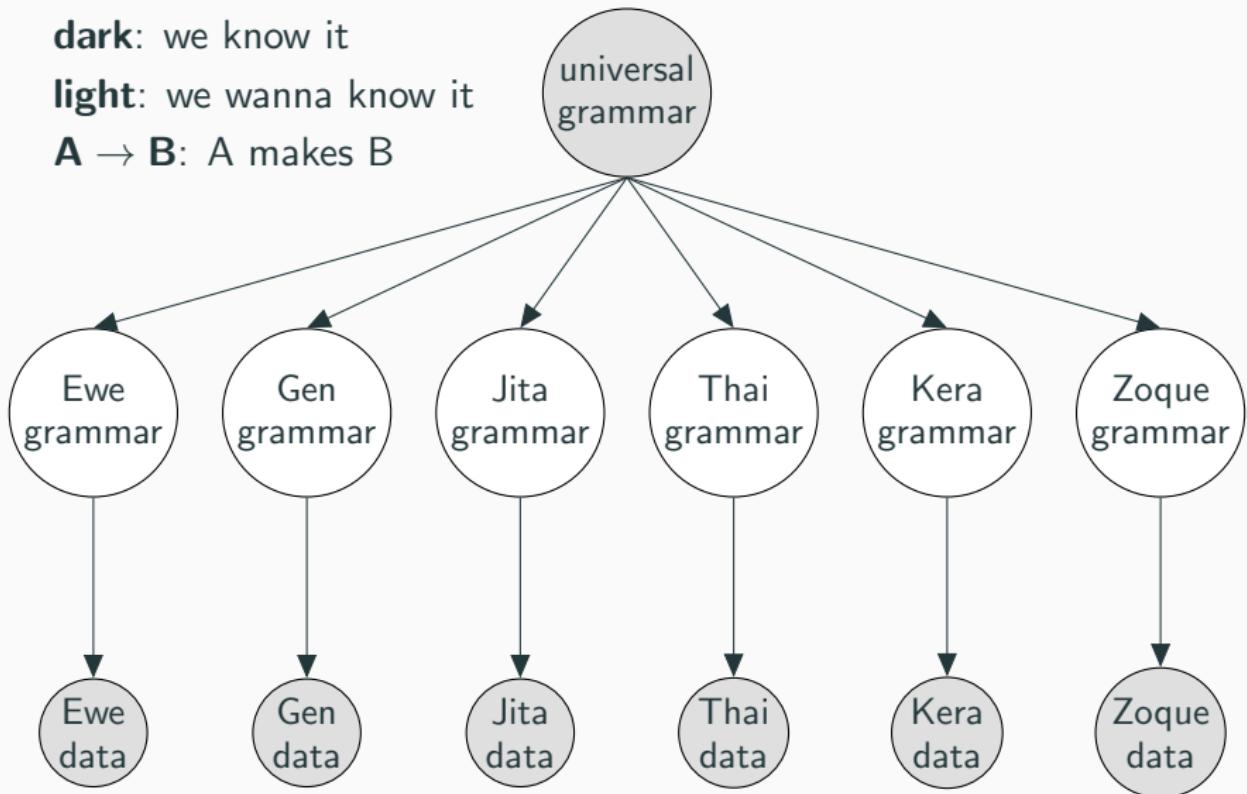


Distilling higher-level knowledge

dark: we know it

light: we wanna know it

A → B: A makes B

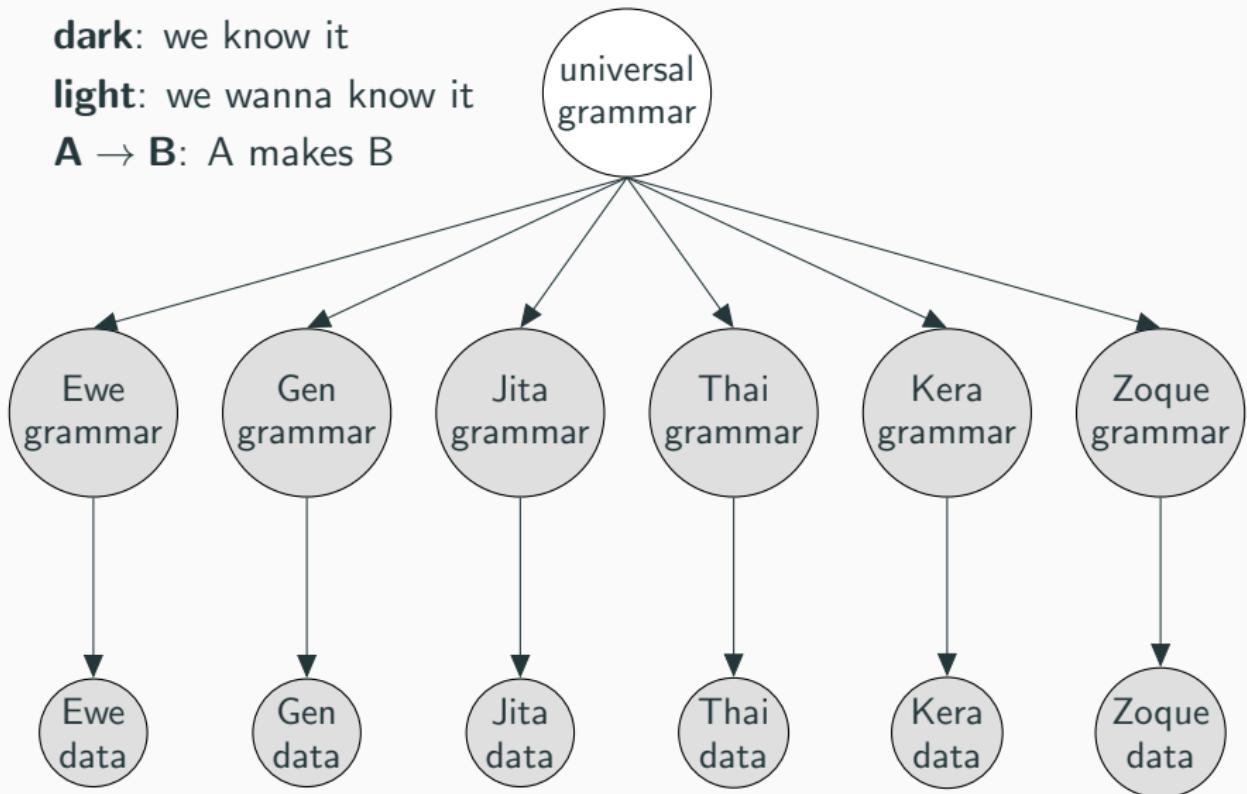


Distilling higher-level knowledge

dark: we know it

light: we wanna know it

A → B: A makes B

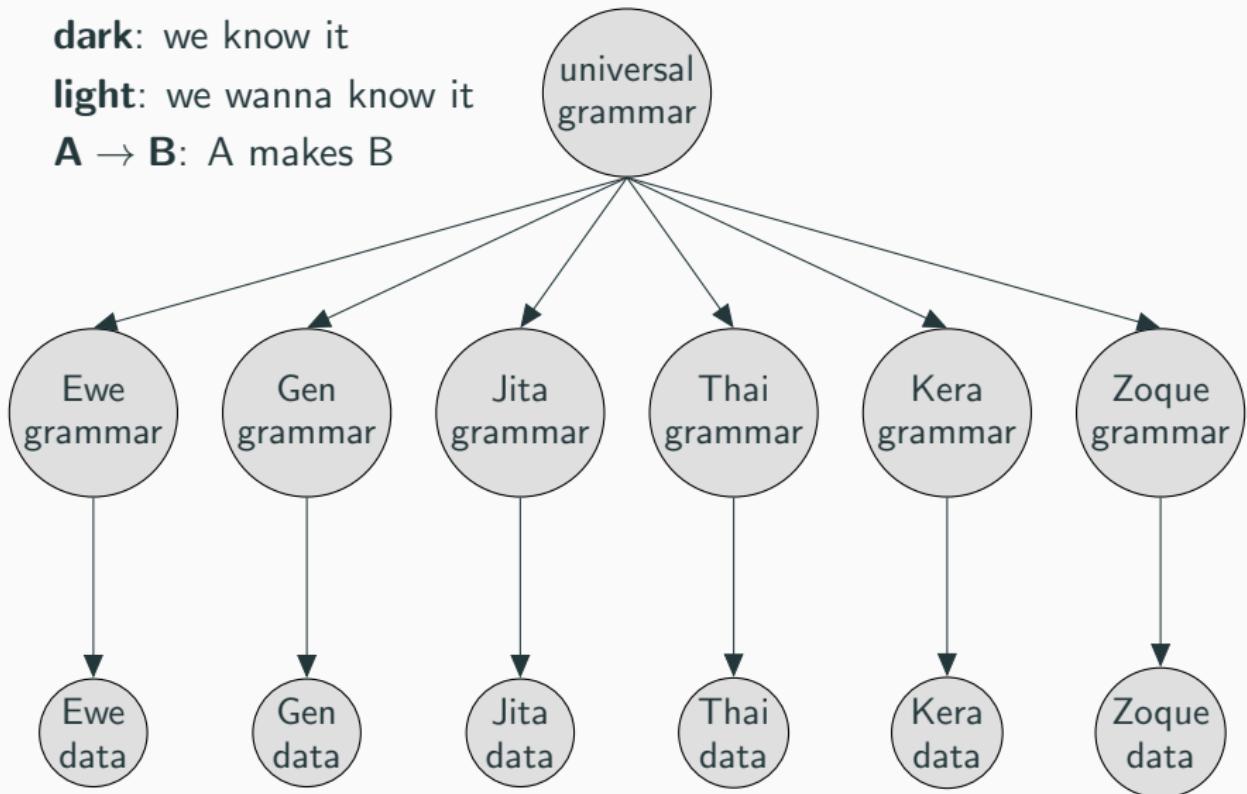


Distilling higher-level knowledge

dark: we know it

light: we wanna know it

A → B: A makes B



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)

Cathy Wong



Max Nye



Mathias
Sable-Meyer



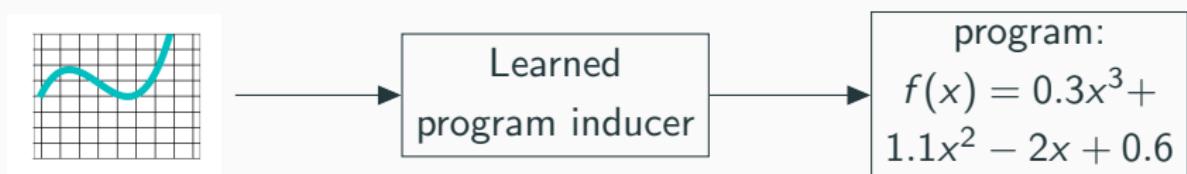
Lucas Morales



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

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]

...

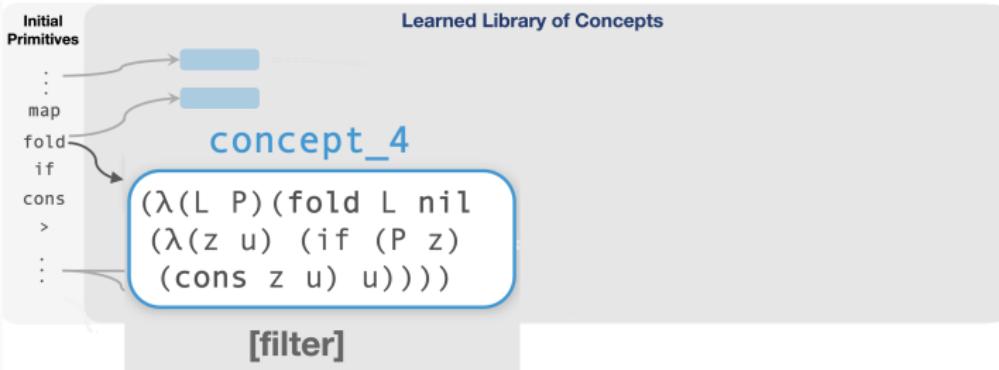
Library learning



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] → [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]
...

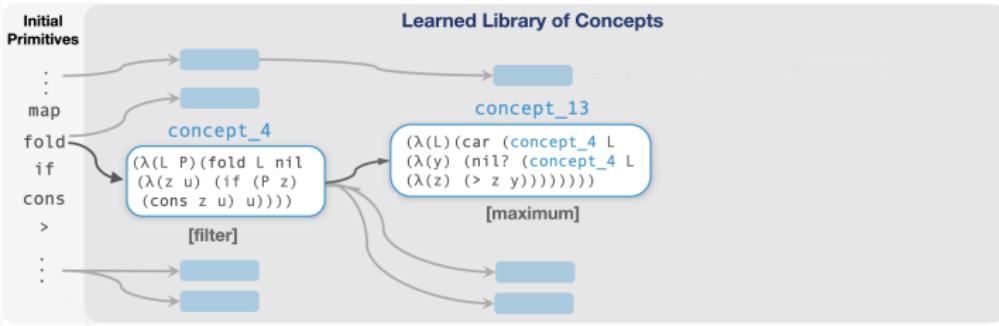
Library learning



Sample Problem: sort list

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

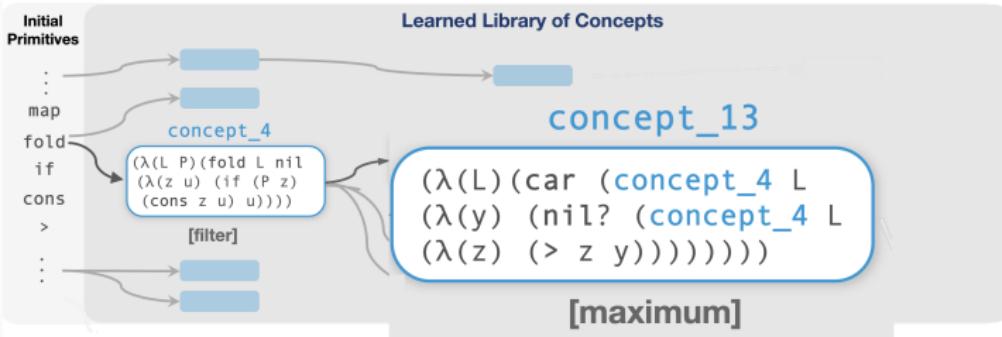
Library learning



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]
...

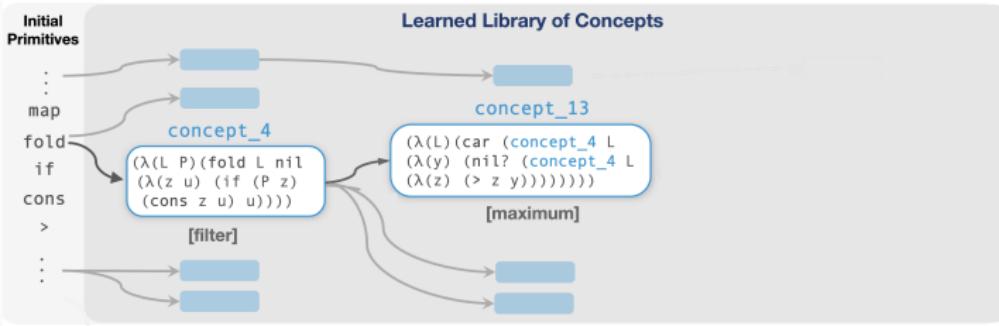
Library learning



Sample Problem: sort list

[9 2 7 1] → [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]
...

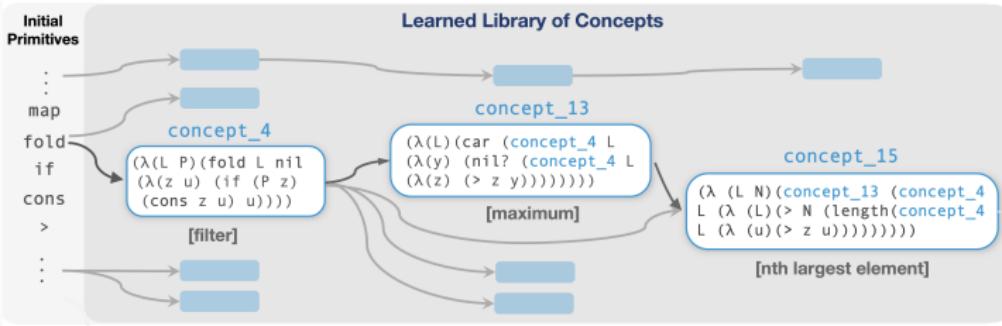
Library learning



Sample Problem: sort list

[9 2 7 1] → [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]
...

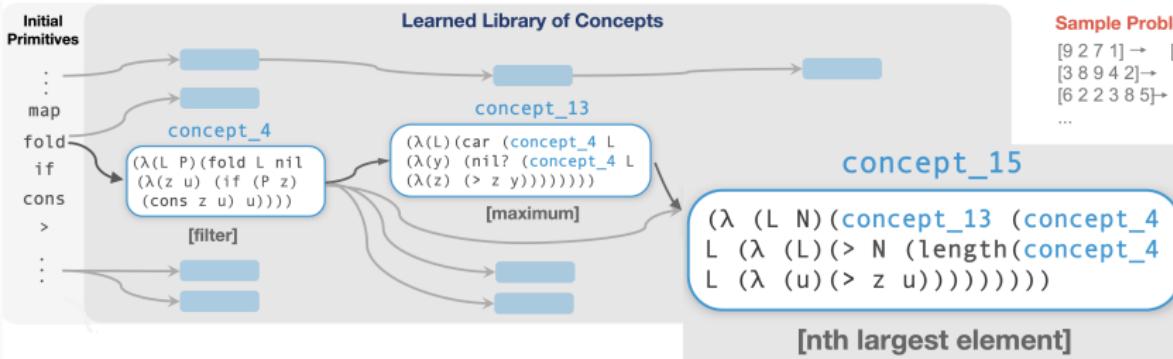
Library learning



Sample Problem: sort list

[9 2 7 1] → [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]
...

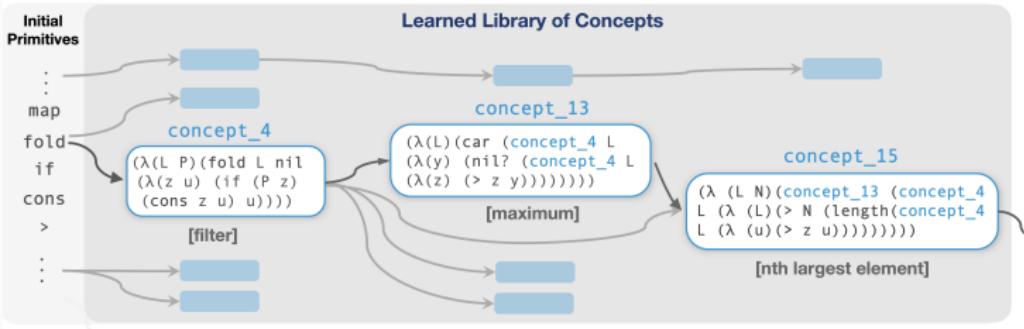
Library learning



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



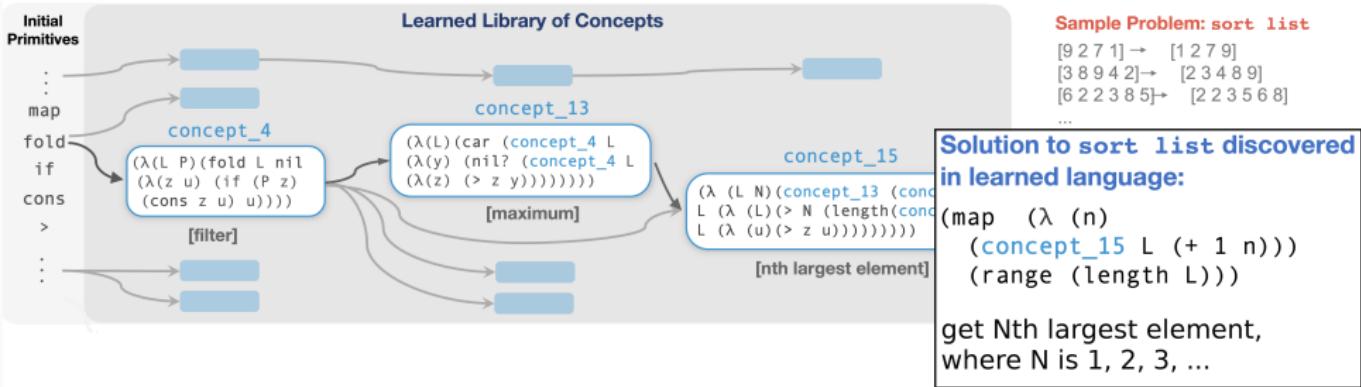
Sample Problem: sort list

[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]
...

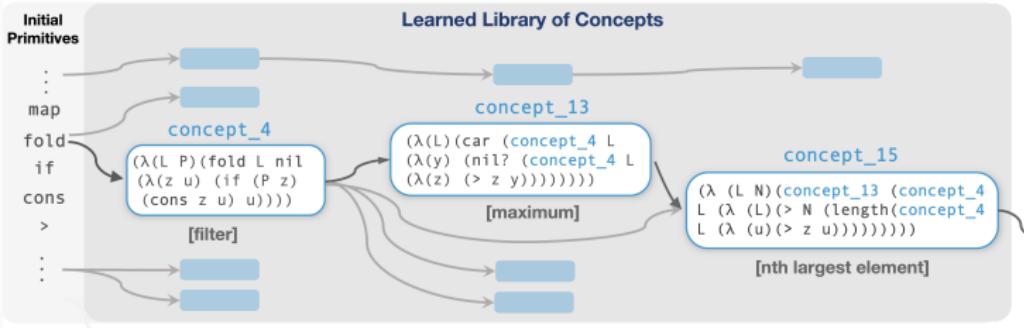
Solution to sort list discovered in learned language:

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

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 8 5] → [2 2 3 5 6 8]
...

Solution to sort list discovered in learned language:

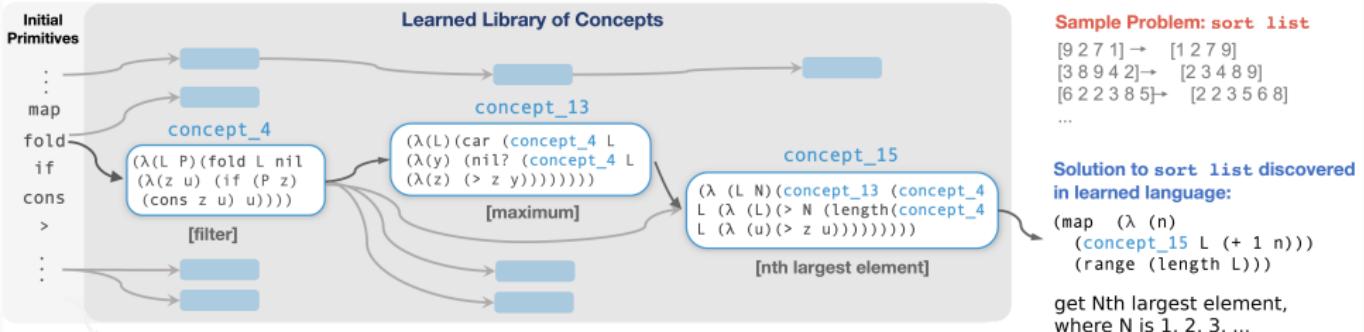
```
(map (λ (n)
  (concept_15 L (+ 1 n)))
  (range (length L)))
```

get Nth largest element,
where N is 1, 2, 3, ...

Solution rewritten in initial primitives:

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

Library learning



Solution rewritten in initial primitives:

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

induced sort program found in $\leq 10\text{min}$. Brute-force search without learned library would take $\approx 10^{73}$ years

DreamCoder

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



cf. Helmholtz machine, wake/sleep neural network training algorithms

DreamCoder

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



List Processing

Sum List

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

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

Double

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

$[4\ 5\ 1] \rightarrow [8\ 10\ 2]$

Text Editing

Abbreviate

Allen Newell → A.N.

Herb Simon → H.S.

Drop Last Three

shrdlu → shr

shakey → sha

Regexes

Phone numbers

(555) 867-5309

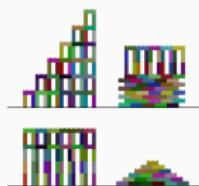
(650) 555-2368

Currency

\$100.25

\$4.50

Block Towers



Symbolic Regression



$$y = f(x)$$

Recursive Programming

Filter Red

[■■■■■■] → [■■■■■]

[■■■■■■■■] → [■■■■■■■]

[■■■■■■■■■] → [■■■■■■■■]

Physical Laws

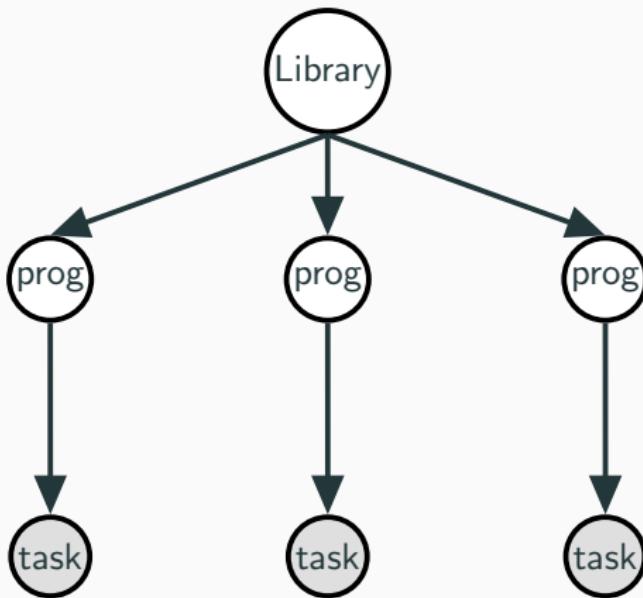
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$



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

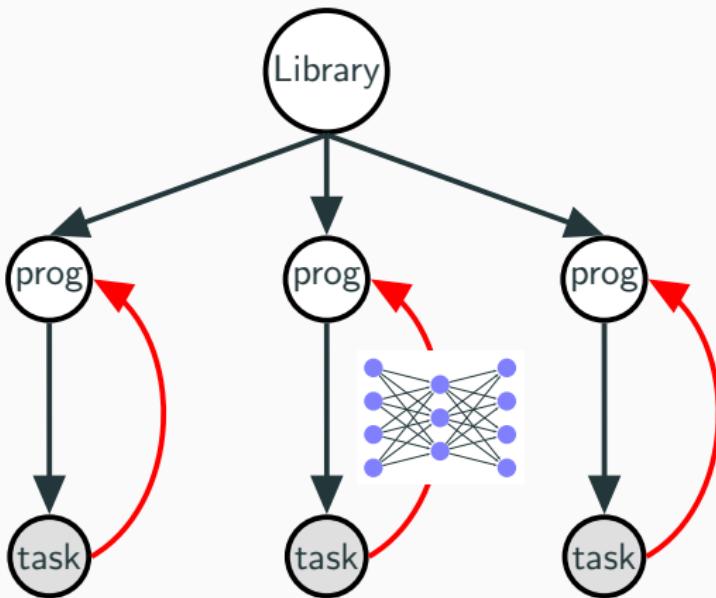
cf. Helmholtz machine, wake/sleep neural network training algorithms

Library learning as Bayesian inference

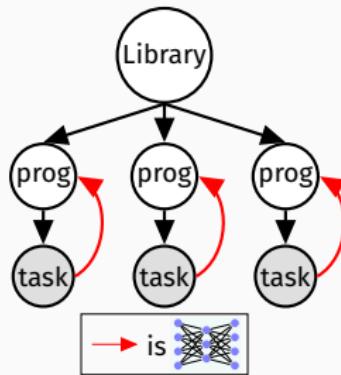


[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

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])



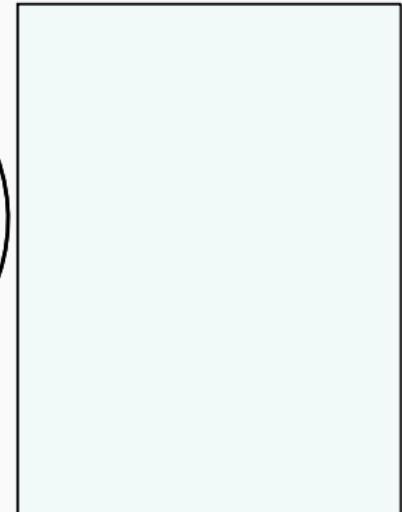
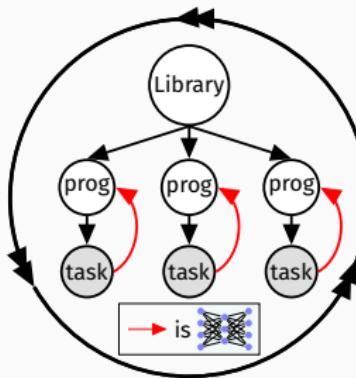
WAKE

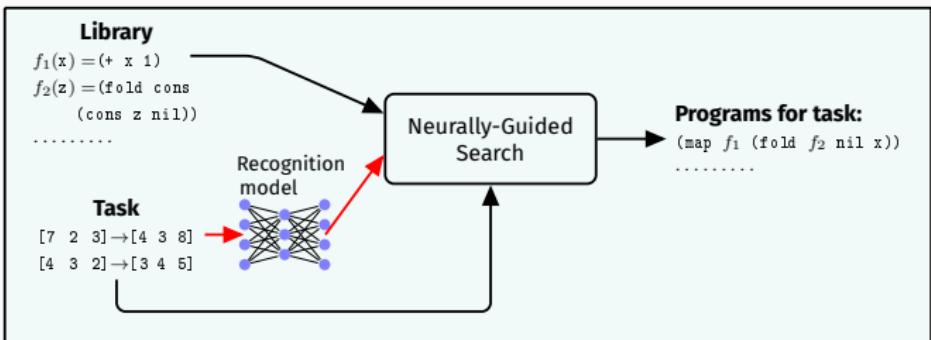
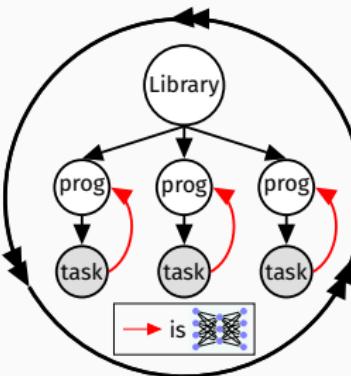


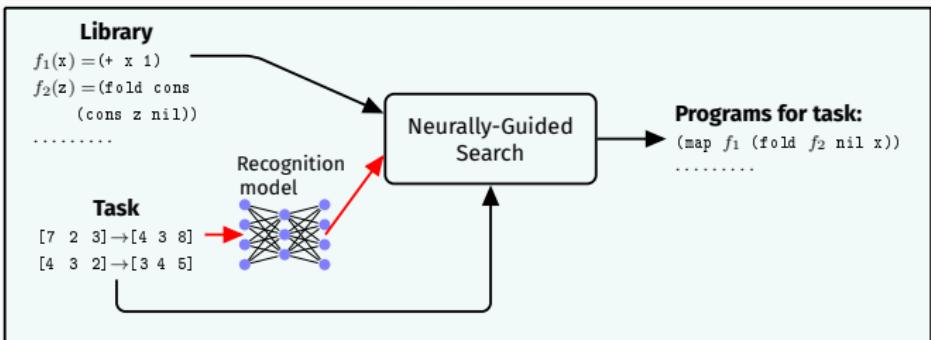
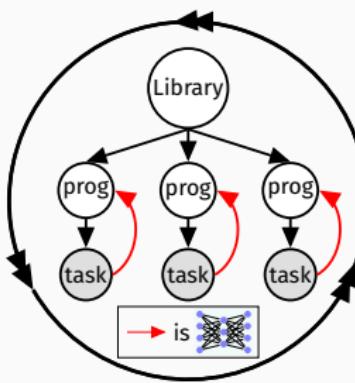
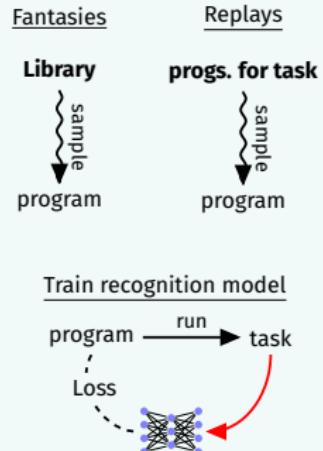
SLEEP: ABSTRACTION

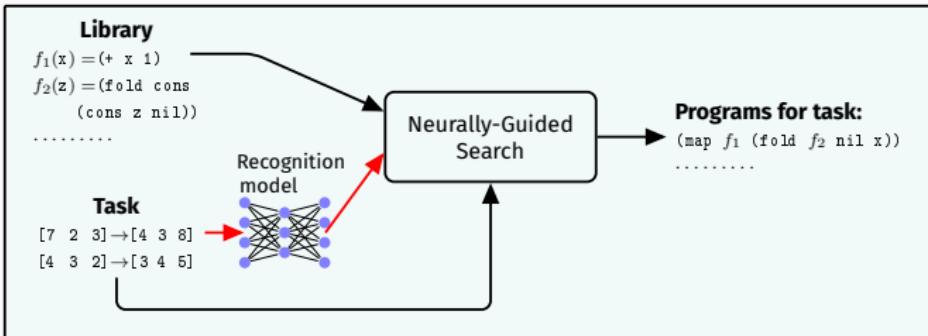
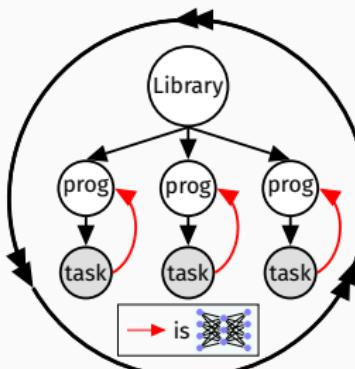
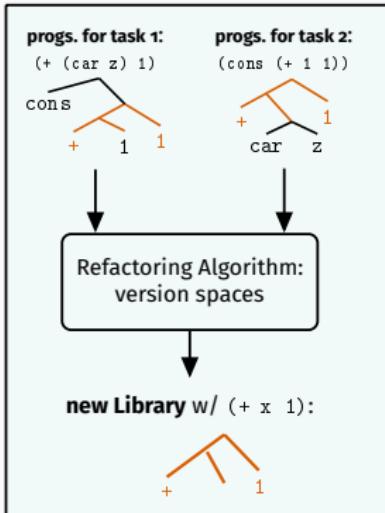
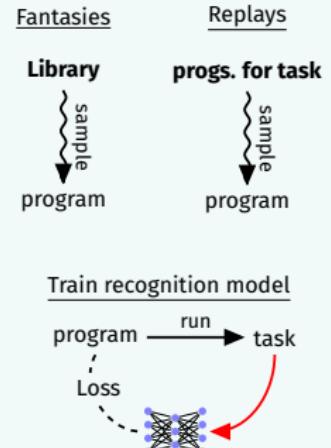


SLEEP: DREAMING

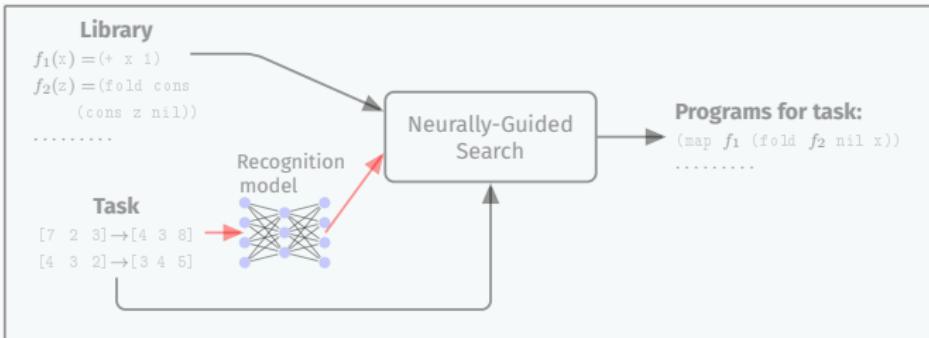


WAKE**SLEEP: ABSTRACTION****SLEEP: DREAMING**

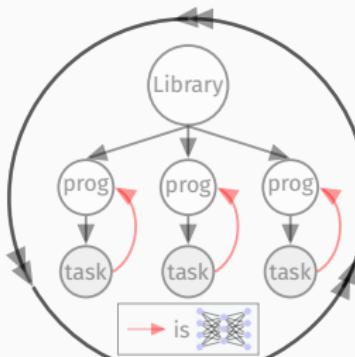
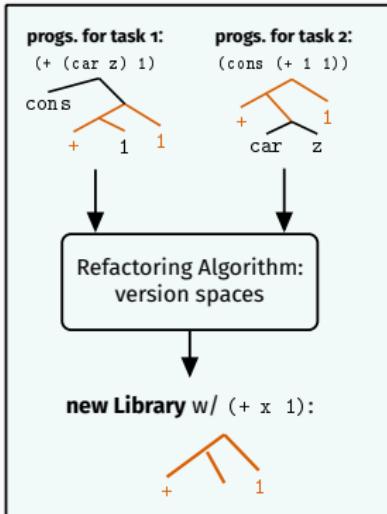
WAKE**SLEEP: ABSTRACTION****SLEEP: DREAMING**

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

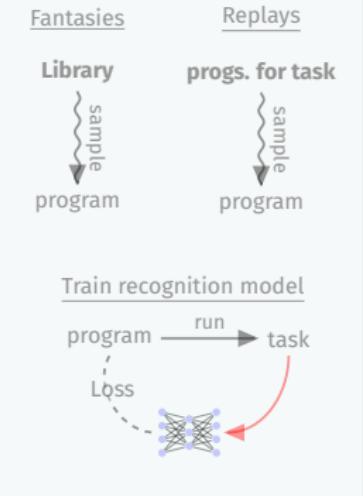
WAKE



SLEEP: ABSTRACTION



SLEEP: DREAMING



Abstraction Sleep: Growing the library via refactoring

$$5 + 5$$

Abstraction Sleep: Growing the library via refactoring

$5 + 5$

(+ 5 5)

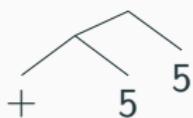
Abstraction Sleep: Growing the library via refactoring

$5 + 5$

$(+ 5 5)$



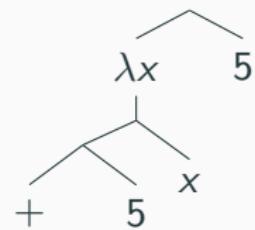
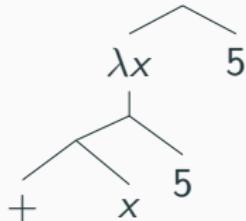
Abstraction Sleep: Growing the library via refactoring



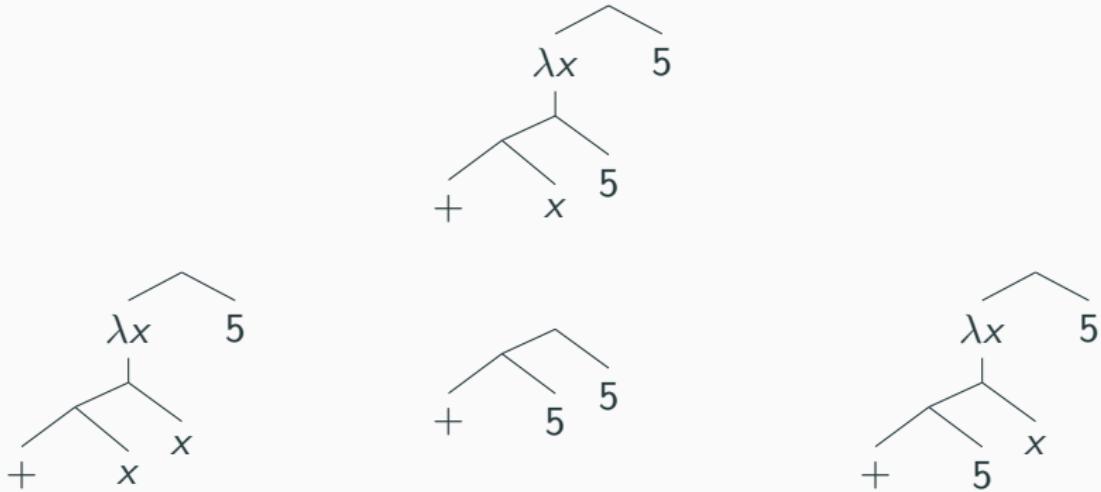
Abstraction Sleep: Growing the library via refactoring



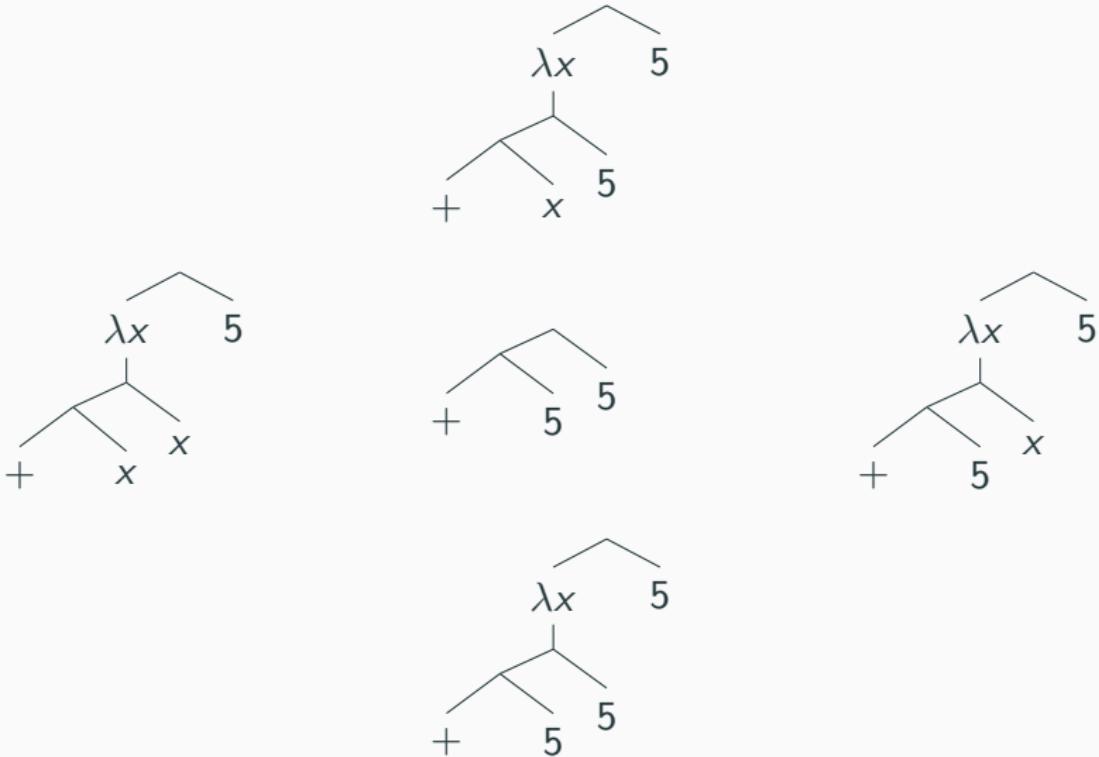
Abstraction Sleep: Growing the library via refactoring



Abstraction Sleep: Growing the library via refactoring

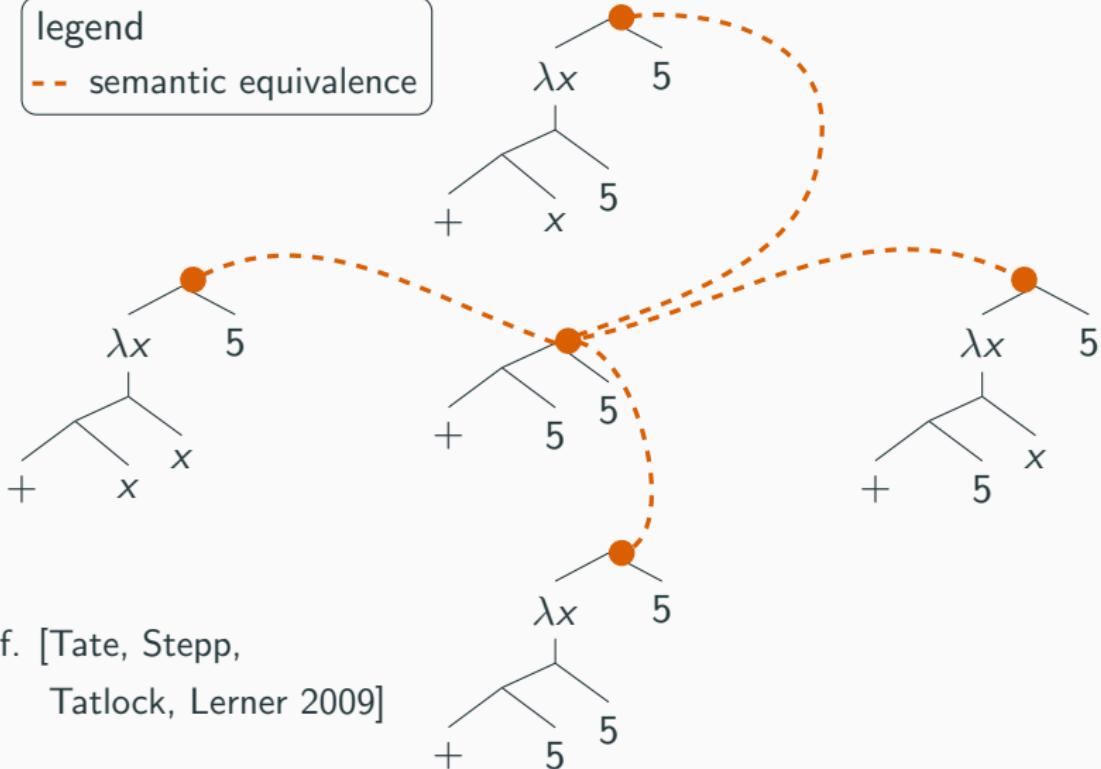


Abstraction Sleep: Growing the library via refactoring



Abstraction Sleep: Growing the library via refactoring

legend
--- semantic equivalence



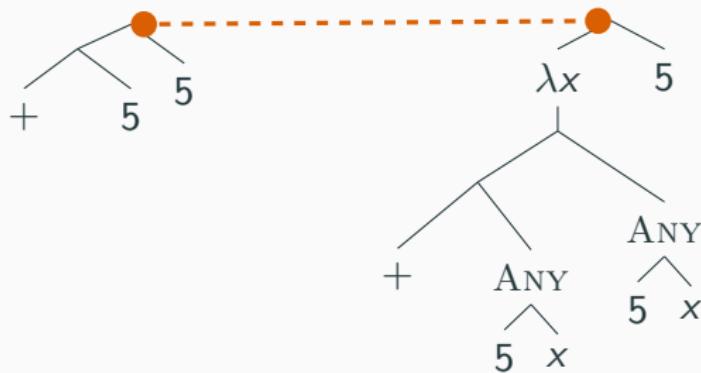
cf. [Tate, Stepp,
Tatlock, Lerner 2009]

Abstraction Sleep: Growing the library via refactoring

legend

dashed line semantic equivalence

ANY nondeterministic choice

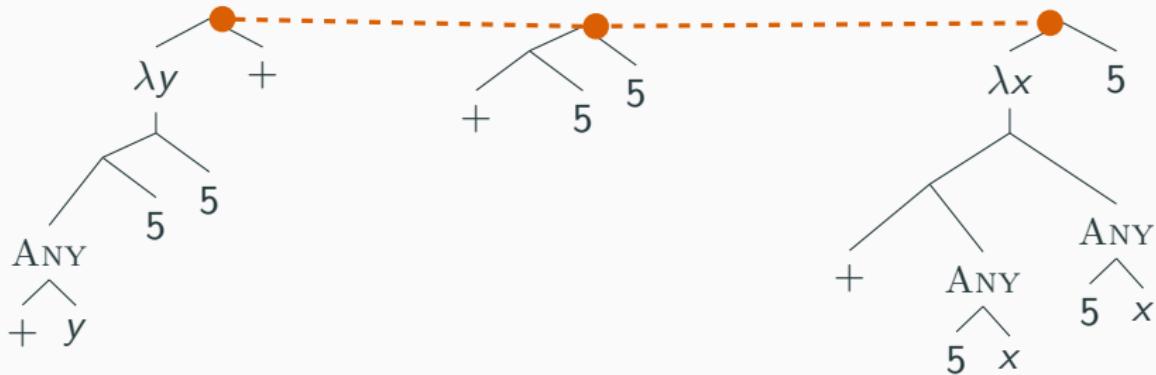


cf. [Lau, Wolfman,
Domingos, Weld '02]

Abstraction Sleep: Growing the library via refactoring

legend

semantic equivalence
ANY nondeterministic choice

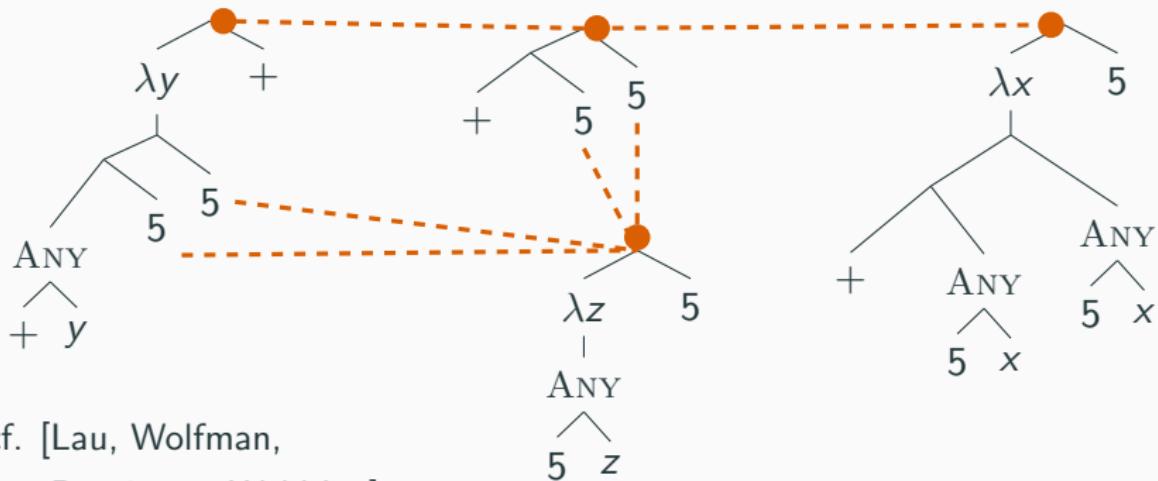


cf. [Lau, Wolfman,
Domingos, Weld '02]

Abstraction Sleep: Growing the library via refactoring

legend

- semantic equivalence
- ANY nondeterministic choice

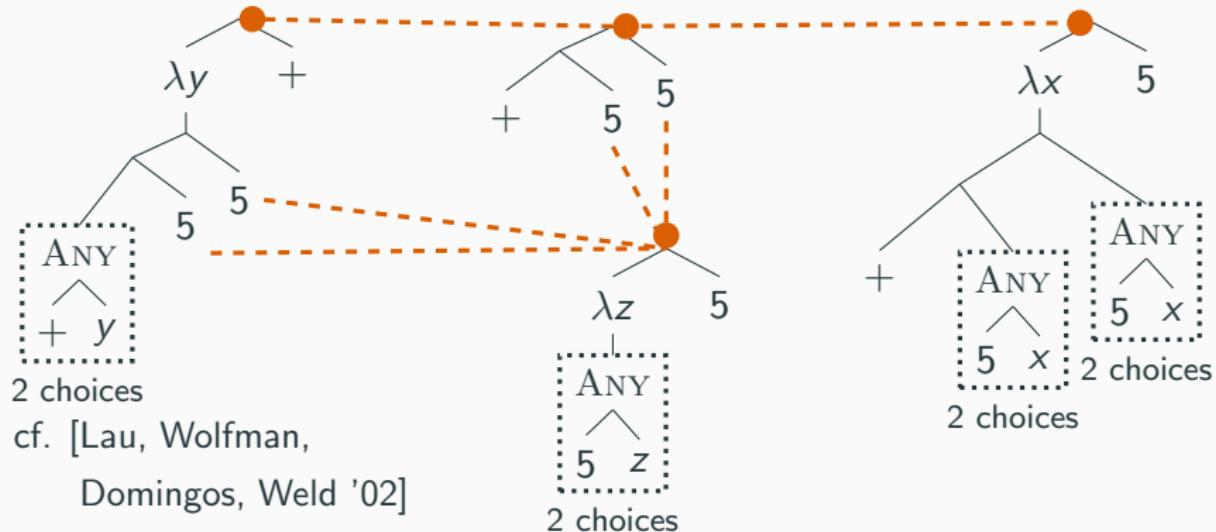


cf. [Lau, Wolfman,
Domingos, Weld '02]

Abstraction Sleep: Growing the library via refactoring

legend

dashed orange line semantic equivalence
ANY nondeterministic choice



cf. [Lau, Wolfman,

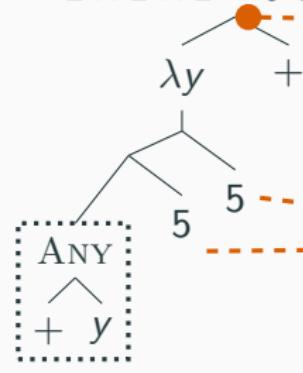
Domingos, Weld '02]

Abstraction Sleep: Growing the library via refactoring

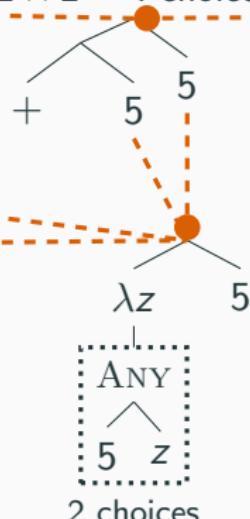
legend

dashed orange line semantic equivalence
ANY nondeterministic choice

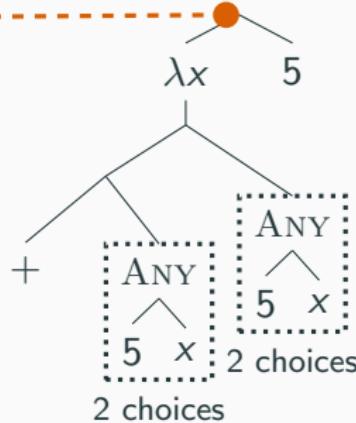
$2 \times 2 \times 2 = 8$ choices



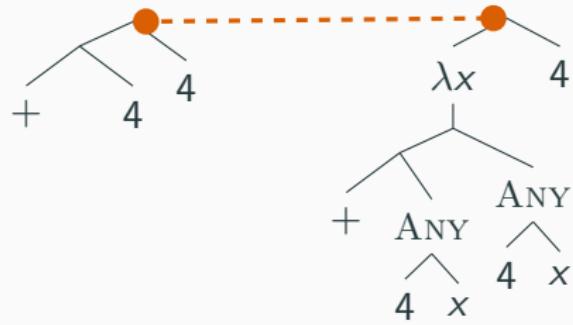
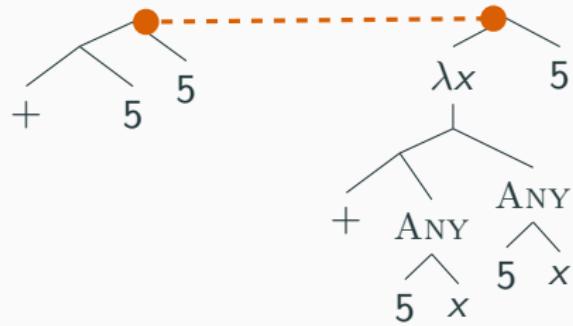
$2 \times 2 = 4$ choices



$2 \times 2 = 4$ choices



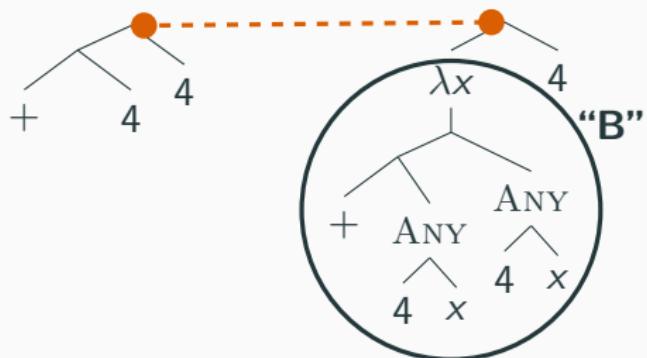
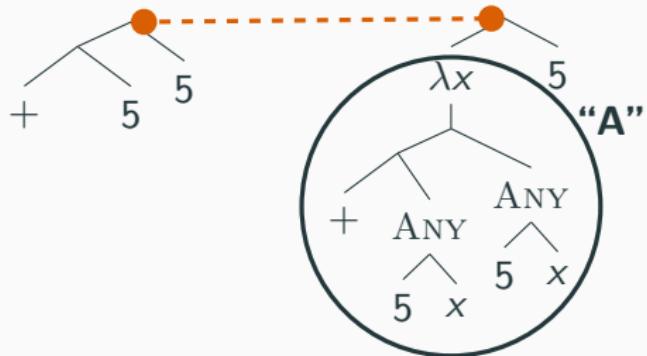
cf. [Lau, Wolfman,
Domingos, Weld '02]



legend

— semantic equivalence

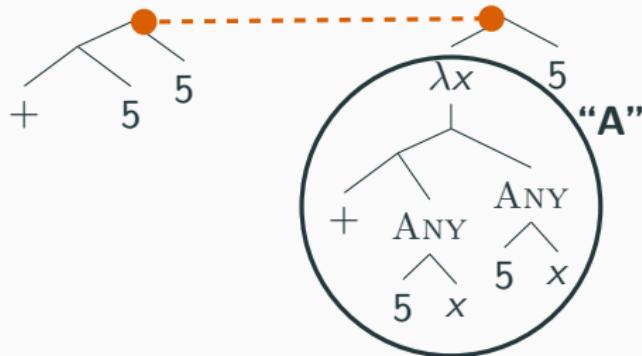
ANY nondeterministic choice



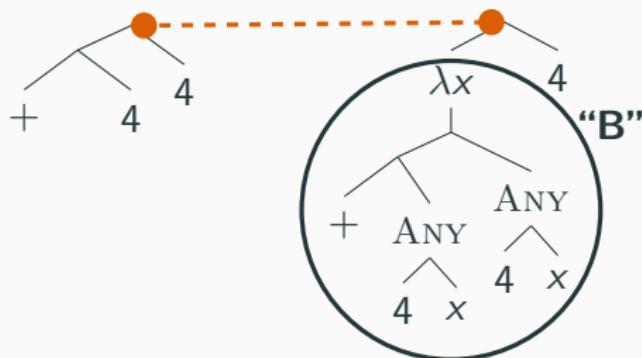
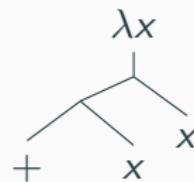
legend

— semantic equivalence

ANY nondeterministic choice



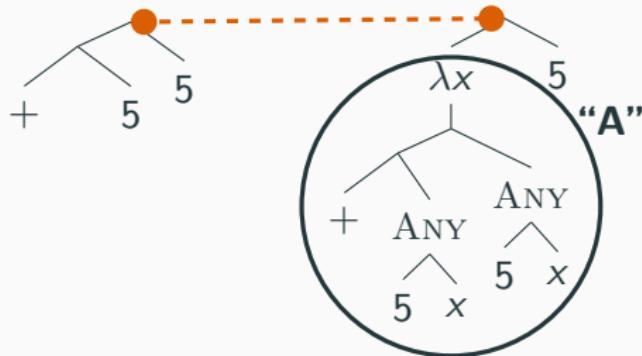
A intersect B:



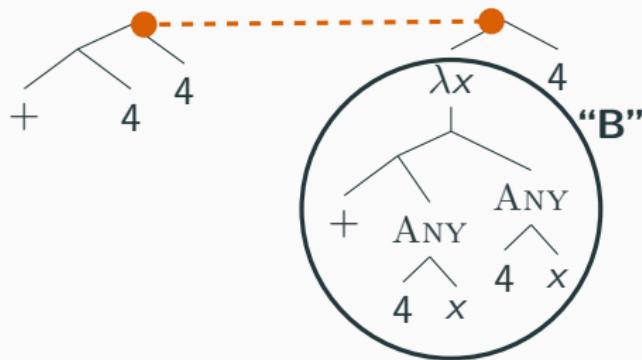
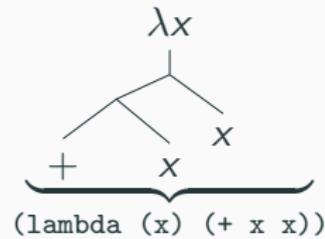
legend

— semantic equivalence

ANY nondeterministic choice



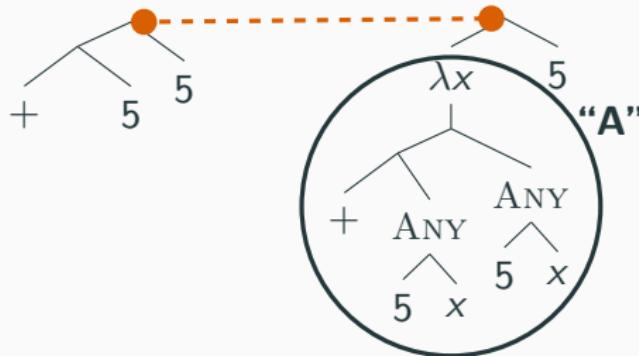
A intersect B:



legend

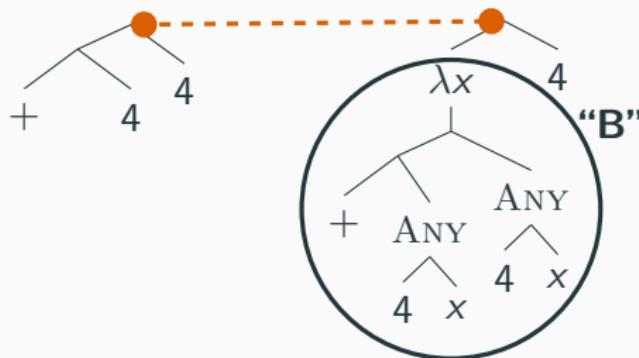
— semantic equivalence

ANY nondeterministic choice



A intersect B:

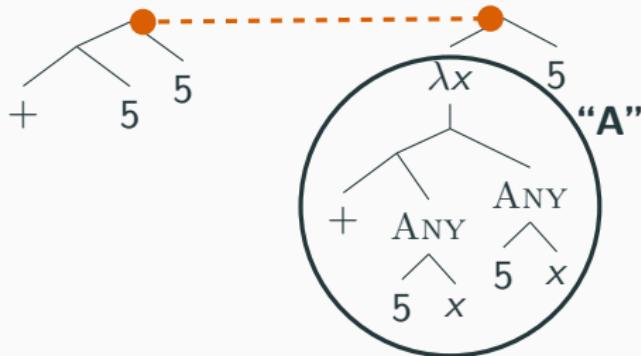
λx
 + x
 (lambda (x) (+ x x))
 = double



legend

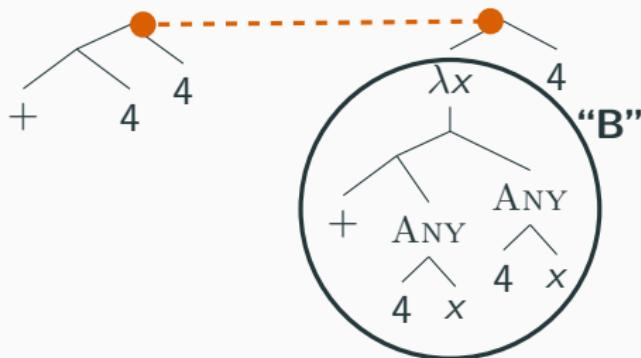
— dashed orange line semantic equivalence

ANY nondeterministic choice



A intersect B:

$$\begin{array}{c}
 \lambda x \\
 / \quad \backslash \\
 + \quad x \\
 \underbrace{\quad}_{(\text{lambda } (x) \ (\text{+ } x \ x))} \\
 = \text{double}
 \end{array}$$



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

legend

— semantic equivalence

ANY nondeterministic choice

Abstraction Sleep: Growing the library via refactoring

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

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)

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

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 1)))
```

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)))))))
```

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)))
```

Compress (MDL/Bayes objective)

```
([MAP] (λ (z) (+ z z))) ([MAP] (λ (z) (- z 1)))  
[MAP] = (λ (f) (Y (λ (r 1) (if (nil? 1) nil  
                           (cons (f (car 1))  
                                 (r (cdr 1)))))))
```

Abstraction Sleep: Growing the library via refactoring



these 10^{14} refactorings are represented in DreamCoder's exponentially more efficient refactoring data structure using 10^6 nodes, calculated in under 5min

```
(r (cdr 1))))))  
(λ (z) (+ z z)))
```

```
(r (cdr 1)))))))  
(λ (z) (- z 1)))
```

Compress (MDL/Bayes objective)

```


$$(\text{MAP } (\lambda (z) (+ z z))) \quad (\text{MAP } (\lambda (z) (- z 1)))$$


$$\text{MAP} = (\lambda (f) (\text{Y } (\lambda (r 1) (\text{if } (\text{nil? } 1) \text{ nil} \\
\qquad \qquad \qquad (\text{cons } (f \text{ (car } 1)) \\
\qquad \qquad \qquad (\text{r } (\text{cdr } 1)))))))$$


```

DreamCoder Domains

List Processing

Sum List

[1 2 3] → 6

[4 6 8 1] → 17

Double

[1 2 3] → [2 4 6]

[4 5 1] → [8 10 2]

Text Editing

Abbreviate

Allen Newell → A.N.

Herb Simon → H.S.

Drop Last Three

shrdlu → shr

shakey → sha

Regexes

Phone numbers

(555) 867-5309

(650) 555-2368

Currency

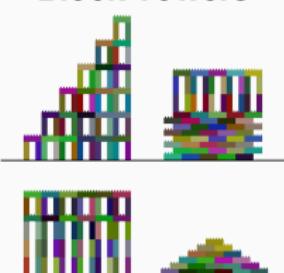
\$100.25

\$4.50

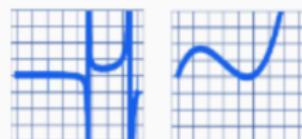
LOGO Graphics



Block Towers



Symbolic Regression



$$y = f(x)$$

Recursive Programming

Filter Red

[■■■■■■■■] → [■■■■]

[■■■■■■■■■■] → [■■■■■■■■]

[■■■■■■■■■■■] → [■■■■■■■■]

Physical Laws

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

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

DreamCoder Domains

List Processing

Sum List

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

Double

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

Text Editing

Abbreviate

Allen Newell → A.N.
Herb Simon → H.S.

Drop Last Three

shrdlu → shr
shakey → sha

Regexes

Phone numbers

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

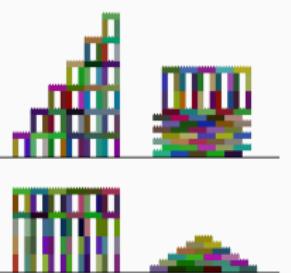
Currency

\$100.25
\$4.50

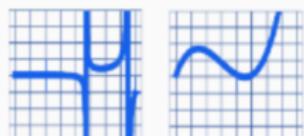
LOGO Graphics



Block Towers



Symbolic Regression



$$y = f(x)$$

Recursive Programming

Filter Red

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

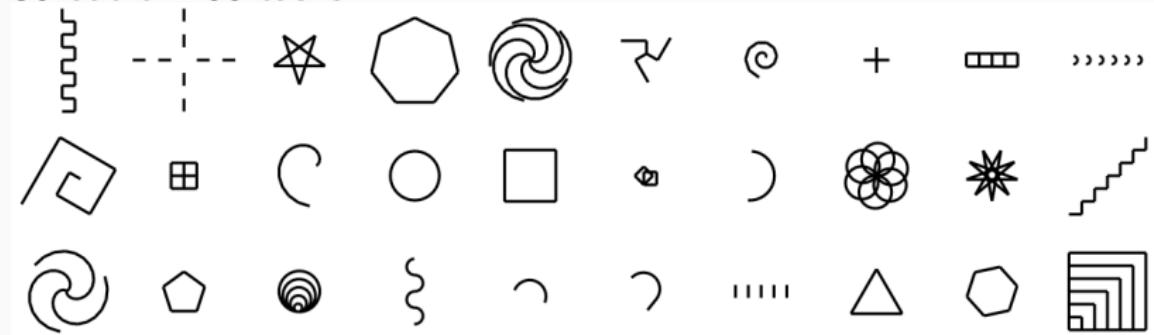
Physical Laws

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

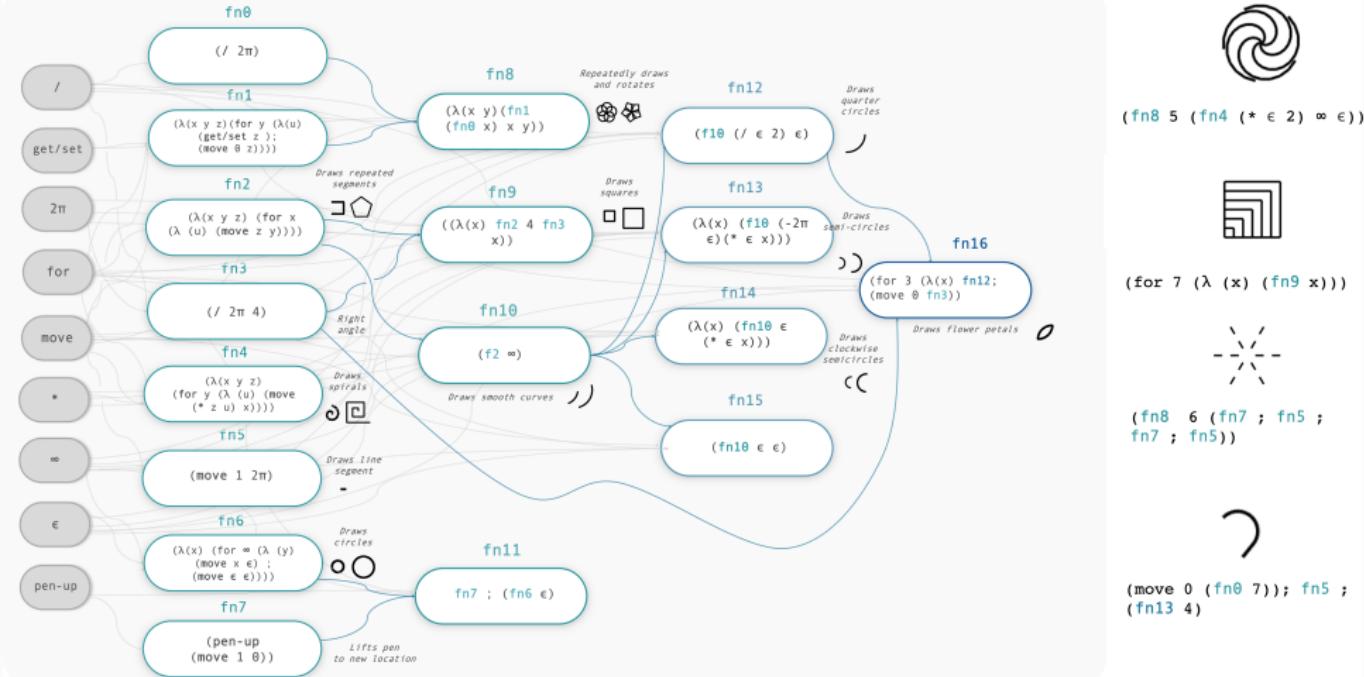
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}|^2} \hat{r}$$

LOGO Turtle Graphics

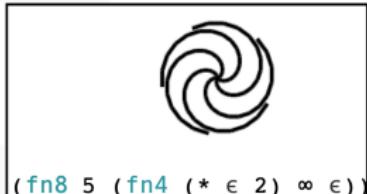
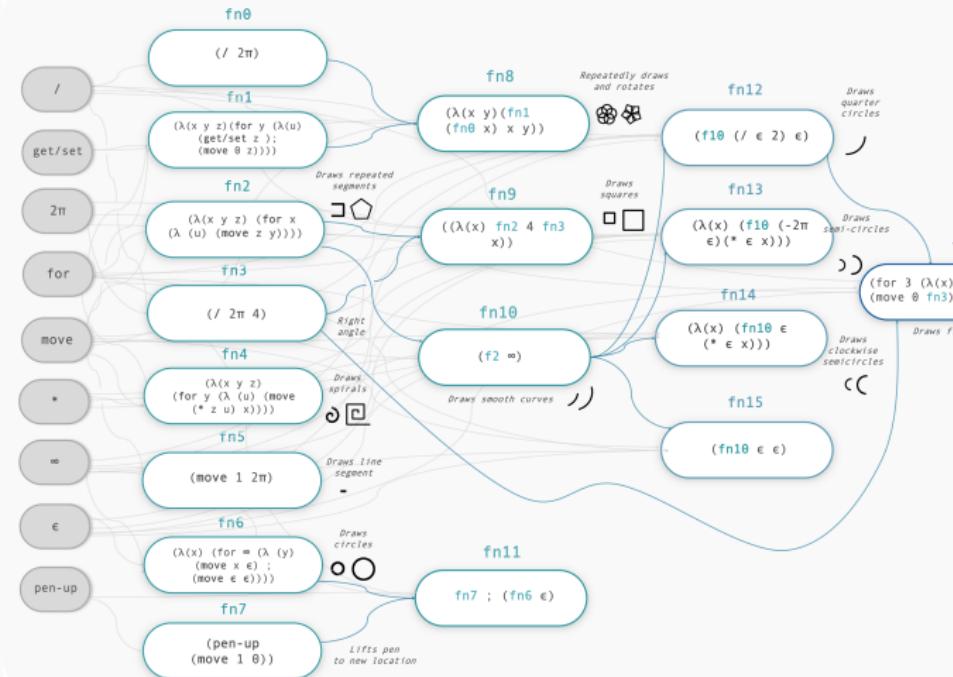
30 out of 160 tasks



LOGO Turtle Graphics – learning an interpretable library



LOGO Turtle Graphics – learning an interpretable library

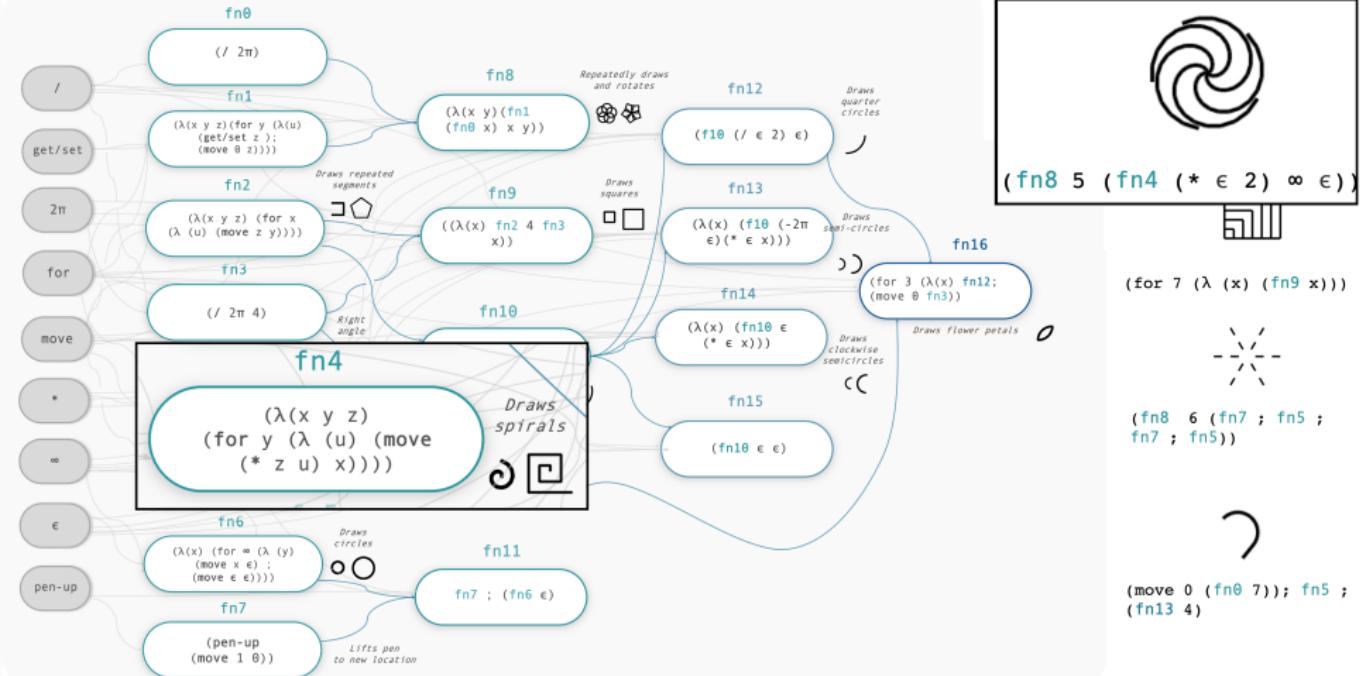


```
(for 7 (\lambda (x) (fn9 x)))
  (fn8 6 (fn7 ; fn5 ;
    fn7 ; fn5))
```

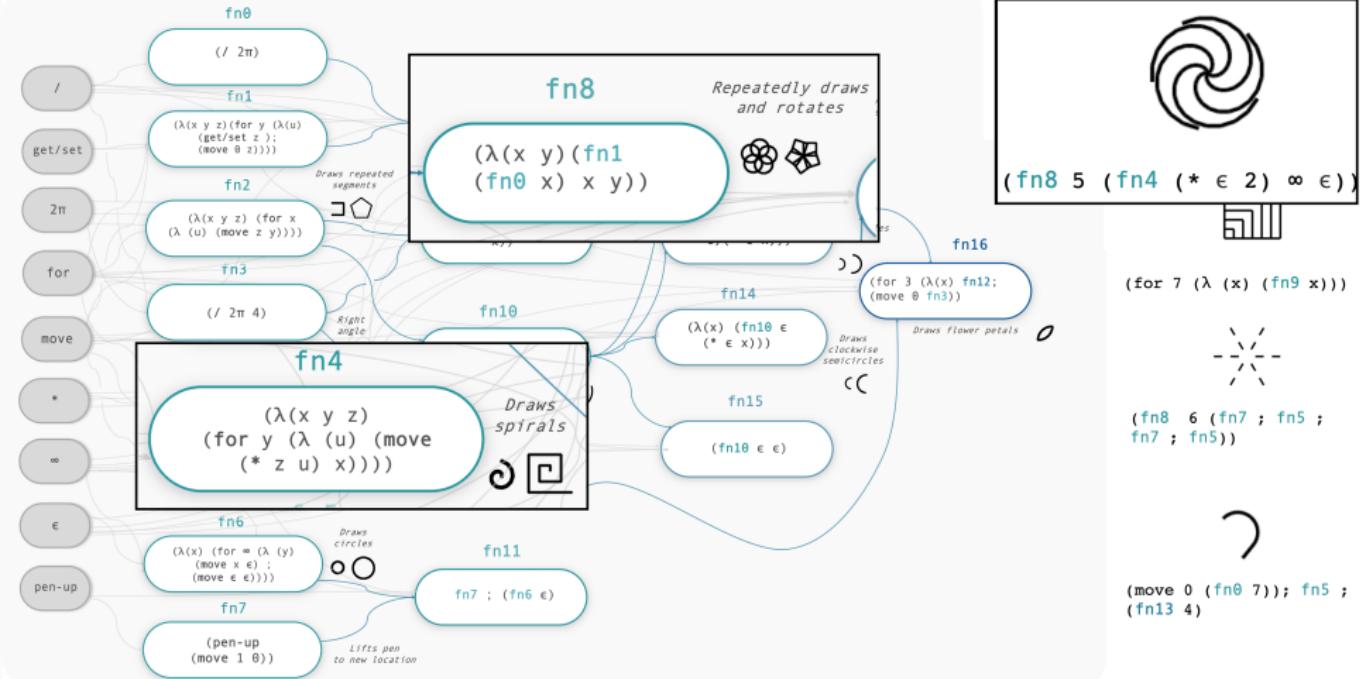


```
(move 0 (fn0 7)); fn5 ;
  (fn13 4))
```

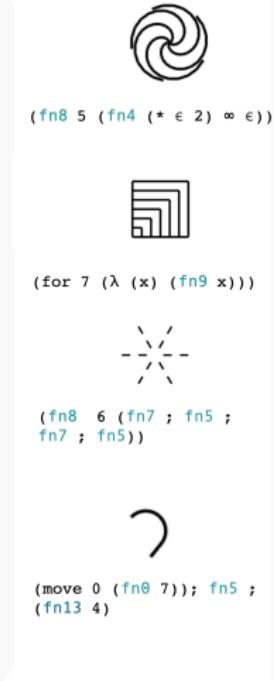
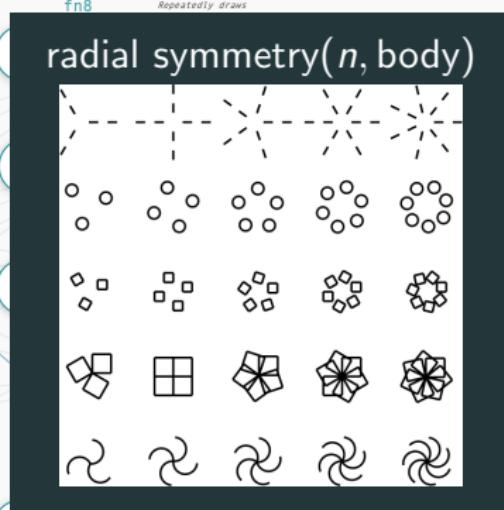
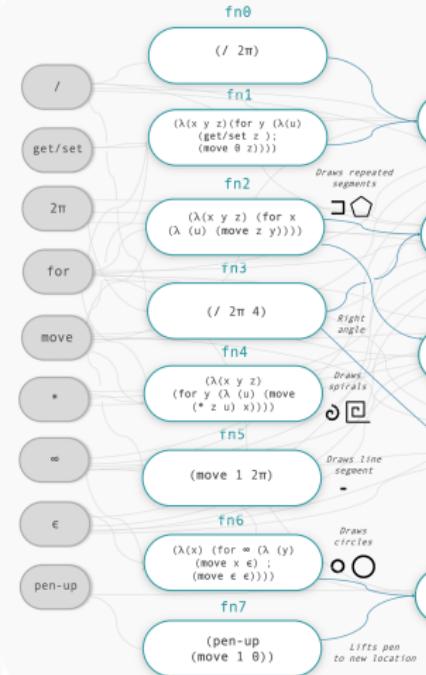
LOGO Turtle Graphics – learning an interpretable library



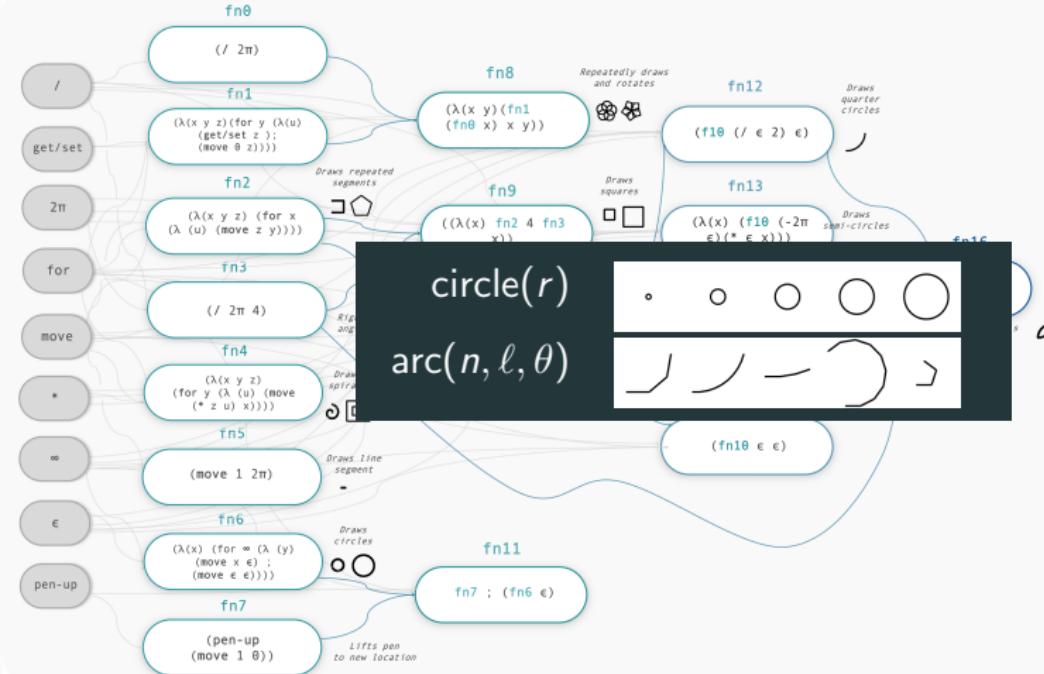
LOGO Turtle Graphics – learning an interpretable library



LOGO Turtle Graphics – learning an interpretable library



LOGO Turtle Graphics – learning an interpretable library



```
(fn8 5 (fn4 (* \epsilon 2) \infty))
```



```
(for 7 (\lambda (x) (fn9 x)))
```

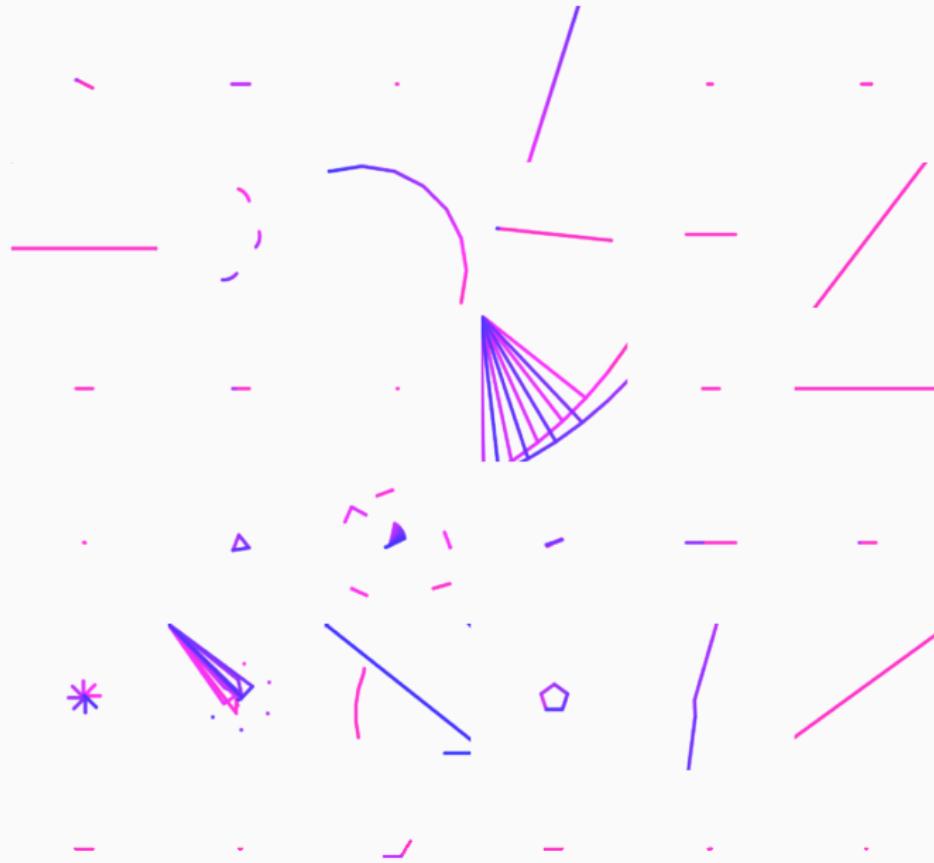


```
(fn8 6 (fn7 ; fn5 ; fn7 ; fn5))
```

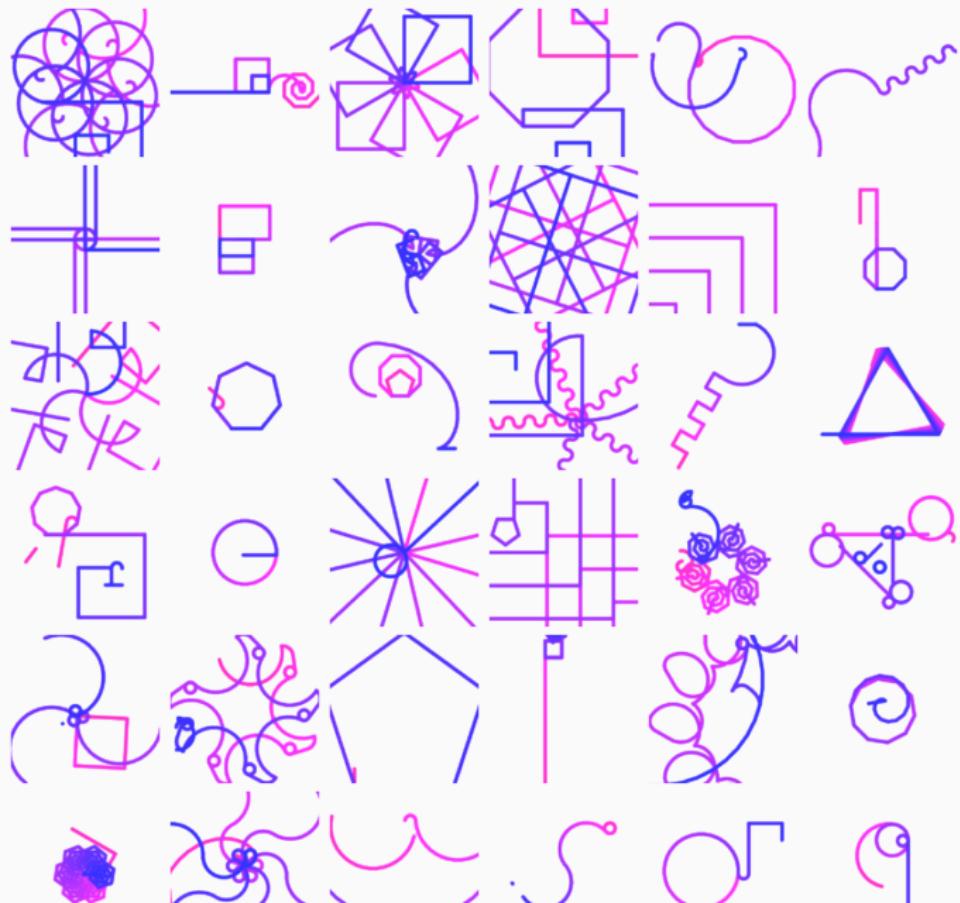


```
(move 0 (fn0 7)); fn5 ; (fn13 4)
```

What does DreamCoder dream of? (before learning)

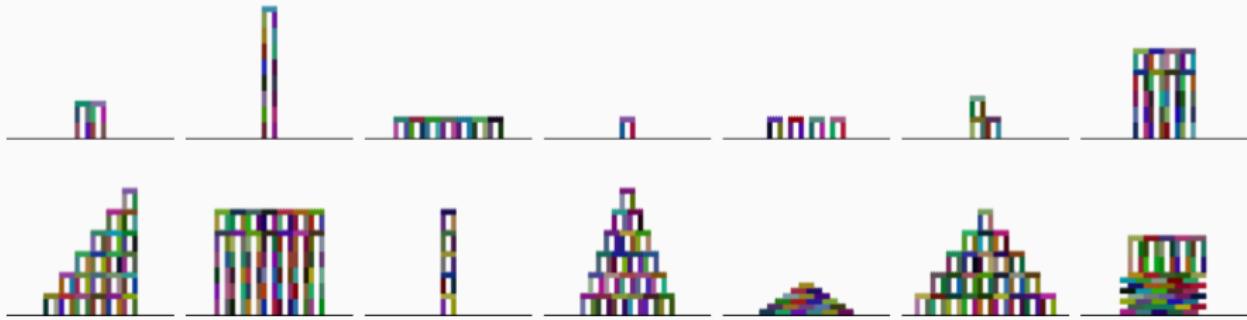


What does DreamCoder dream of? (after learning)



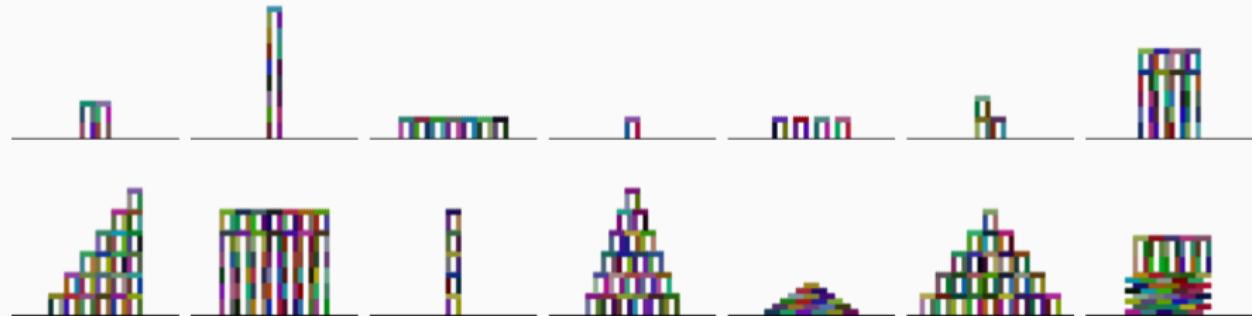
Planning to build towers

example tasks (112 total)



Planning to build towers

example tasks (112 total)



learned library routines (≈ 20 total)

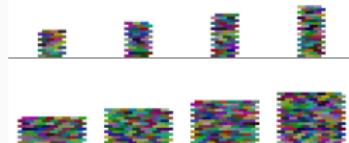
$\text{arch}(h)$



$\text{pyramid}(h)$



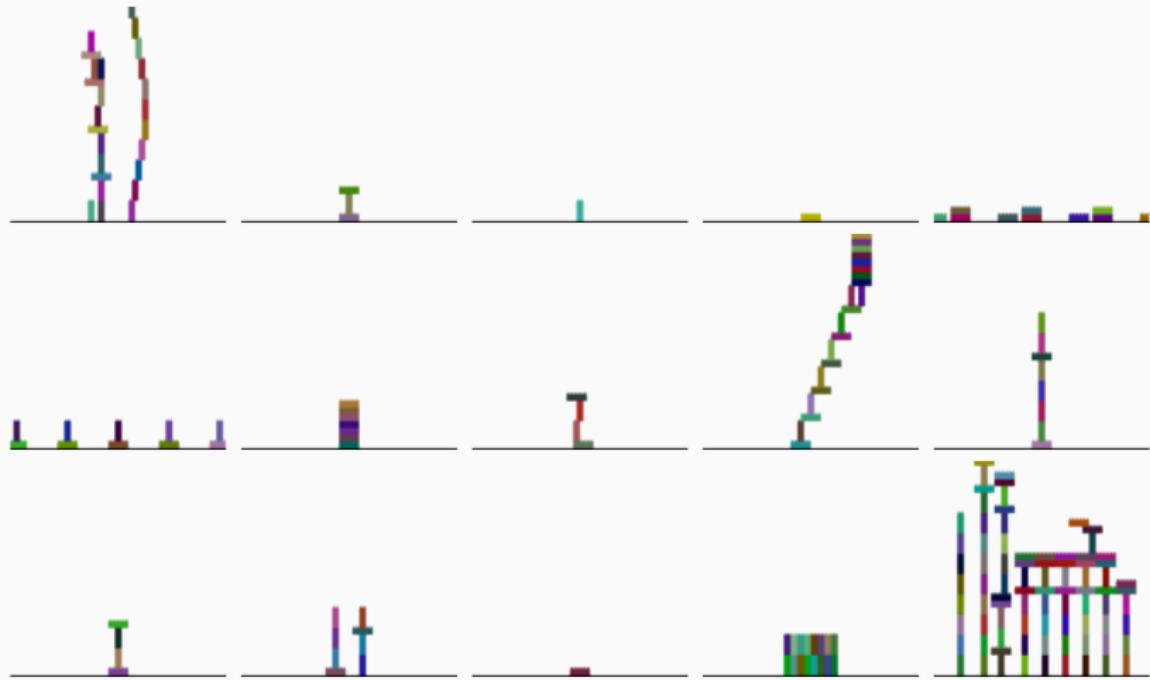
$\text{wall}(w, h)$



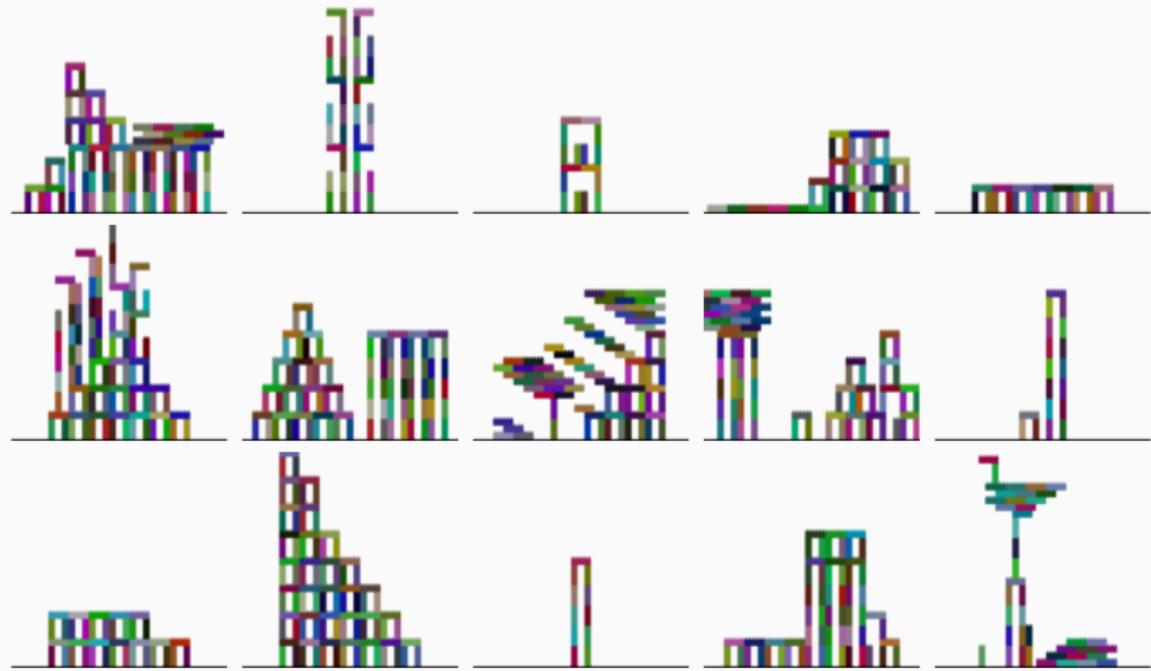
$\text{bridge}(w, h)$



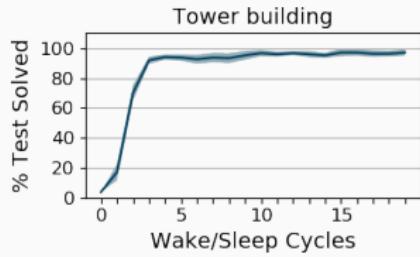
Dreams before learning



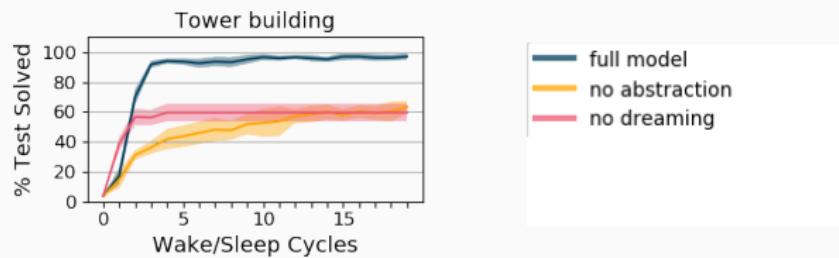
Dreams after learning



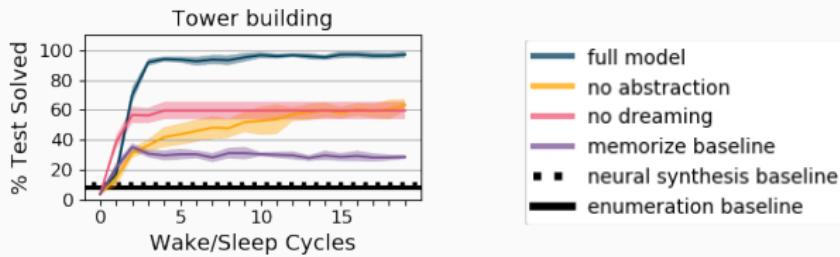
Learning dynamics



Learning dynamics

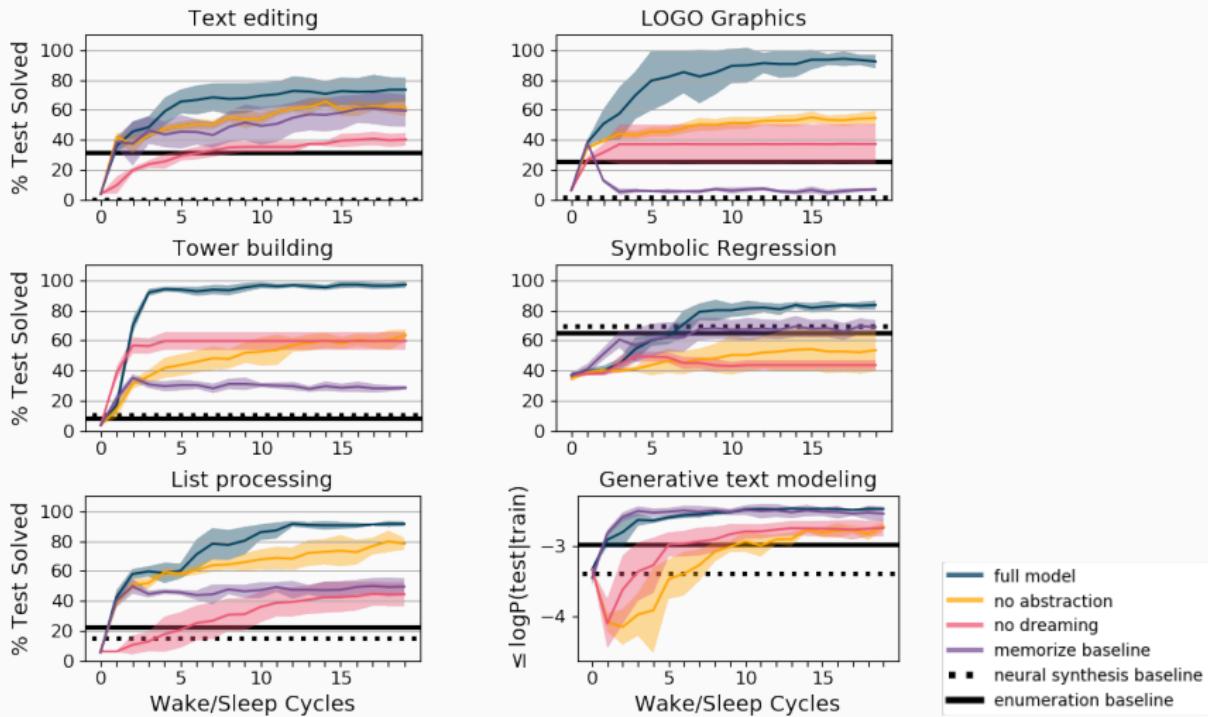


Learning dynamics



baselines: memorize programs rather than refactor them [Cropper 2019]
neural program synthesis, RobustFill [Devlin et al. 2017]
24 hours of brute-force enumeration

Learning dynamics



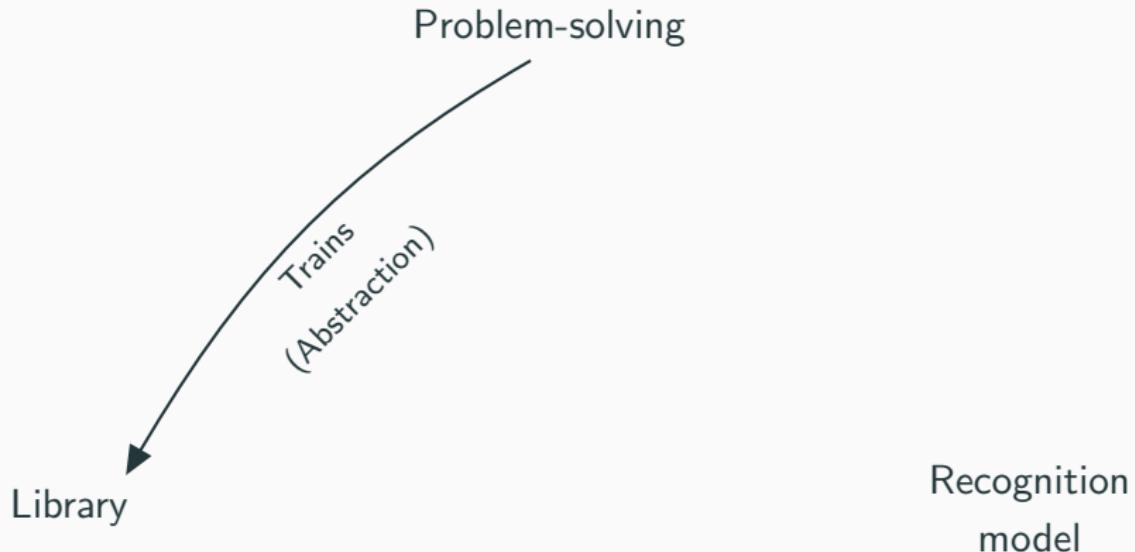
Synergy between dreaming and library learning

Problem-solving

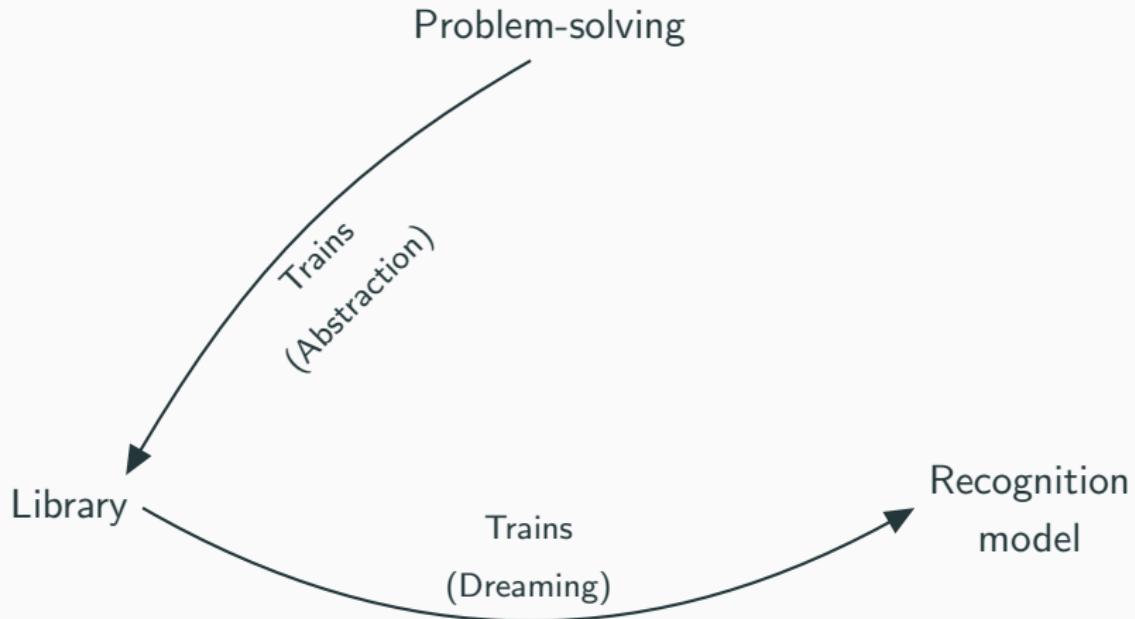
Library

Recognition
model

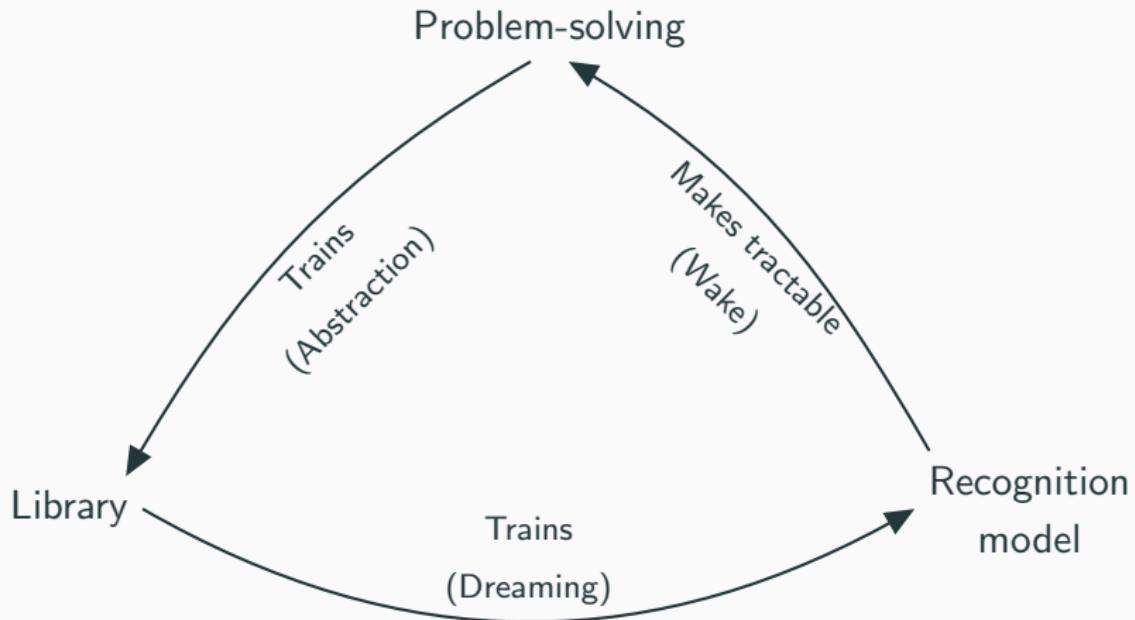
Synergy between dreaming and library learning



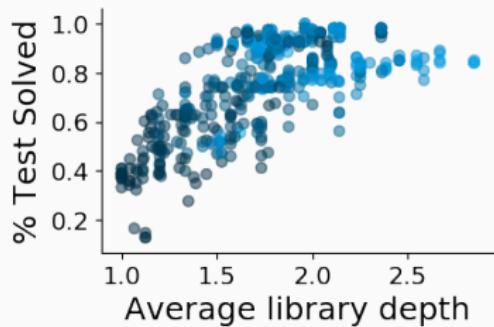
Synergy between dreaming and library learning



Synergy between dreaming and library learning



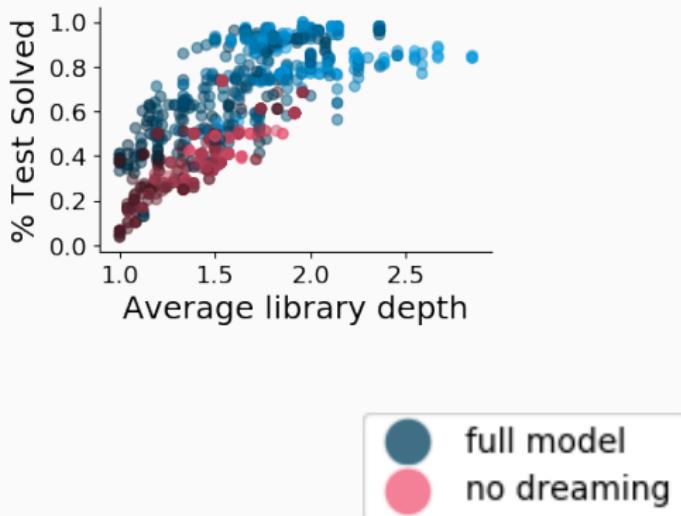
Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Brighter: Later in learning

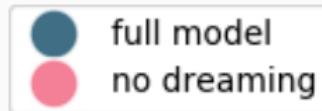
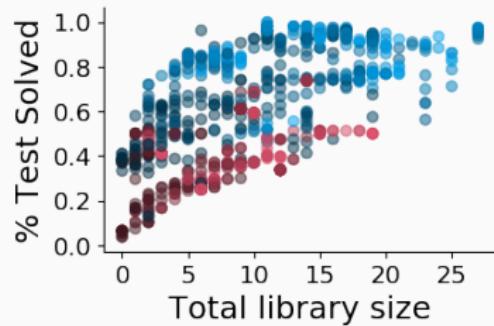
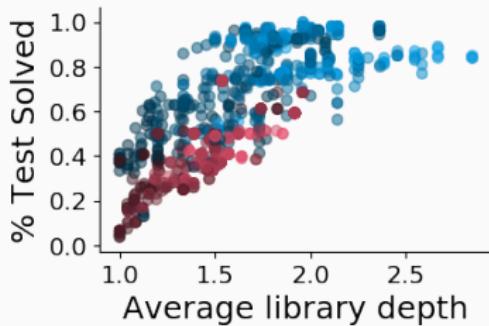
Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Brighter: Later in learning

Evidence for dreaming bootstrapping better libraries



Darker: Early in learning

Brighter: Later in learning

From learning libraries,
to learning languages

From learning libraries, to learning languages

modern functional programming → physics

From learning libraries,
to learning languages

1950's Lisp → modern functional programming → physics

Physics Formula Sheet

Mechanics

$x = x_0 + v_{x0}t + \frac{1}{2}a_xt^2$	$a_t = \frac{v^2}{r}$	$ \vec{F}_{\text{spring}} = k \vec{x} $
$v = v_0 + at$	$\theta = \theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2$	$\text{PE}_{\text{spring}} = \frac{1}{2}kx^2$
$v_s^2 - v_{s0}^2 = 2a(x - x_0)$	$\omega = \omega_0 + \alpha t$	$T_{\text{spring}} = 2\pi \sqrt{\frac{m}{k}}$
$\bar{a} = \frac{\sum \vec{F}}{m} = \frac{\vec{F}_{\text{net}}}{m}$	$T = \frac{2\pi}{\omega} = \frac{1}{f}$	$T_{\text{pendulum}} = 2\pi \sqrt{\frac{l}{g}}$
$ \vec{F}_{\text{friction}} \leq \mu \vec{F}_{\text{Normal}} $	$v = f\lambda$	
$\bar{p} = m\bar{v}$	$x = A\cos(2\pi ft)$	$ \vec{F}_{\text{gravity}} = G \frac{m_1 m_2}{r^2}$
$\Delta \bar{p} = \vec{F} \Delta t$	$\bar{a} = \frac{\sum \vec{F}}{l} = \frac{\vec{F}_{\text{net}}}{l}$	$ \vec{F}_{\text{gravity}} = m\bar{g}$
$KE = \frac{1}{2}mv^2$	$\vec{r} = r \times F$	$\text{PE}_{\text{gravity}} = -G \frac{m_1 m_2}{r}$
$\Delta PE = mg\Delta y$	$L = I\omega$	$p = \frac{m}{V}$
$\Delta E = W = Fd\cos\theta$	$\Delta L = \tau \Delta t$	$KE = \frac{1}{2}I\omega^2$

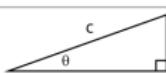
Electricity

$ \vec{F}_E = k \left \frac{q_1 q_2}{r^2} \right $	$\Delta V = IR$	$R = \frac{\rho l}{A}$
$I = \frac{\Delta q}{\Delta t}$		$P = I\Delta V$
$R_{\text{series}} = R_1 + R_2 + \dots + R_n$	$\frac{1}{R_{\text{parallel}}} = \frac{1}{R_1} + \frac{1}{R_2} + \dots + \frac{1}{R_n}$	

Geometry

Rectangle	$A = bh$	Rectangular Solid	$V = lwh$	Triangle	$A = \frac{1}{2}bh$
Circle	$A = \pi r^2$	Cylinder	$V = \pi r^2 l$	Sphere	$V = \frac{4}{3}\pi r^3$
	$C = 2\pi r$		$S = 2\pi rl + 2\pi r^2$		$S = 4\pi r^2$

Trigonometry

	$c^2 = a^2 + b^2$	$\sin\theta = \frac{a}{c}$	$\cos\theta = \frac{b}{c}$	$\tan\theta = \frac{a}{b}$
--	-------------------	----------------------------	----------------------------	----------------------------

Variables

a = acceleration
 A = amplitude
 A = Area
 b = base length
 C = circumference
 d = distance
 E = energy
 f = frequency
 F = force
 h = height
 I = current
 I = rotational inertia
 KE = kinetic energy
 k = spring constant
 L = angular momentum
 l = length
 m = mass
 P = power
 p = momentum
 q = charge
 r = radius
 R = resistance
 S = surface area
 T = period
 t = time
 PE = potential energy
 V = electric potential
 V = volume
 v = velocity
 w = width
 W = work
 x = position
 y = height
 α = angular acceleration
 λ = wavelength
 μ = coefficient of friction

Growing languages for vector algebra and physics

Initial Primitives

map

zip

cons

empty

cdr

power

fold

car

+

-

*

/

0

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

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

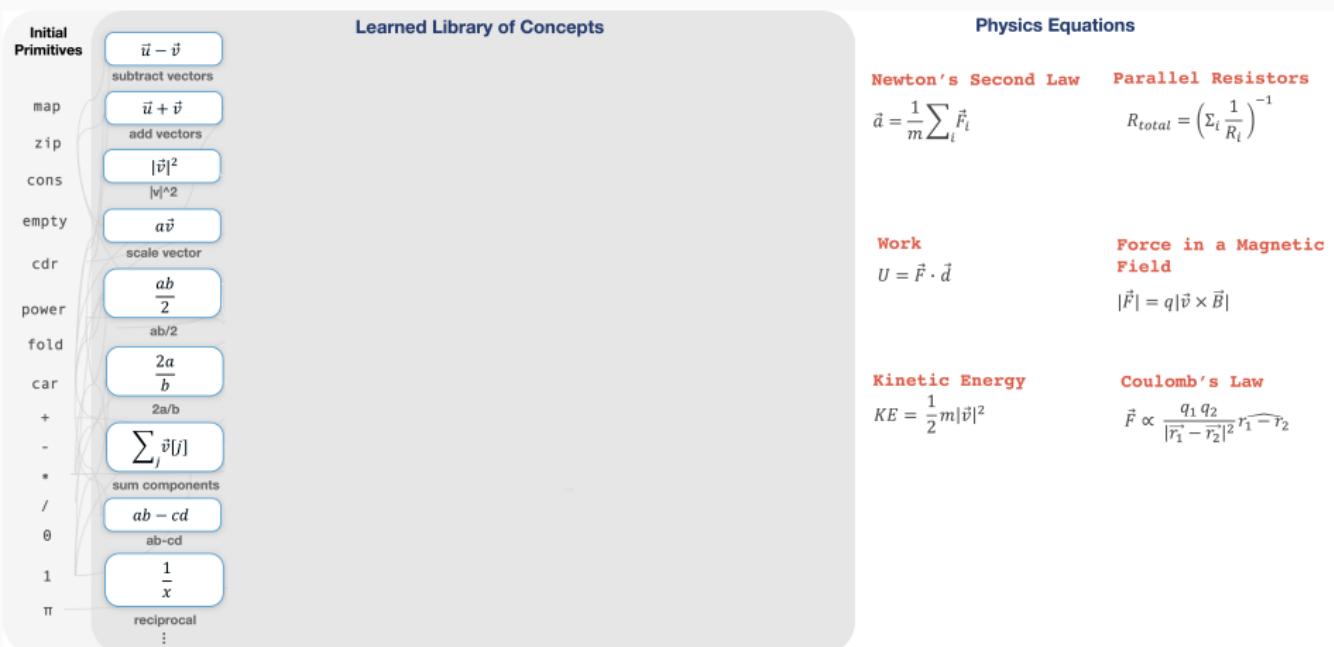
Kinetic Energy

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

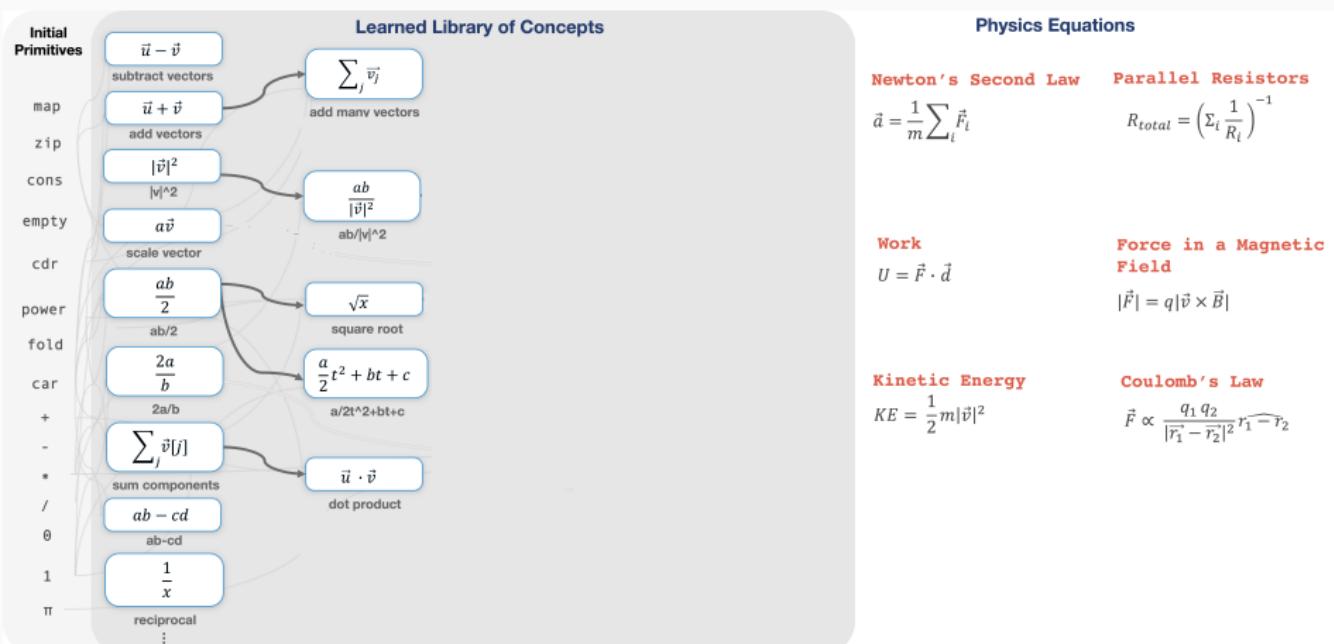
Coulomb's Law

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

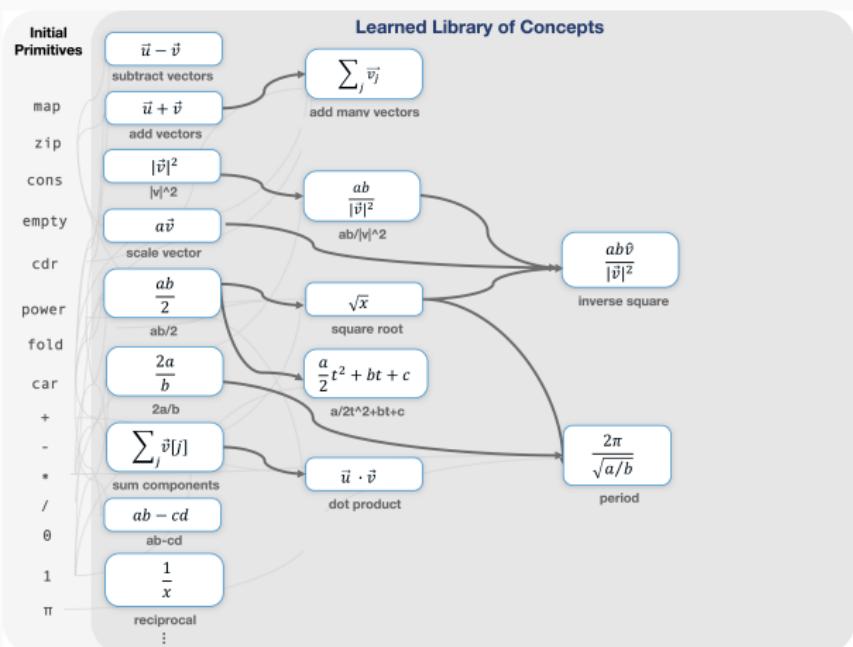
Growing languages for vector algebra and physics



Growing languages for vector algebra and physics



Growing languages for vector algebra and physics



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

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

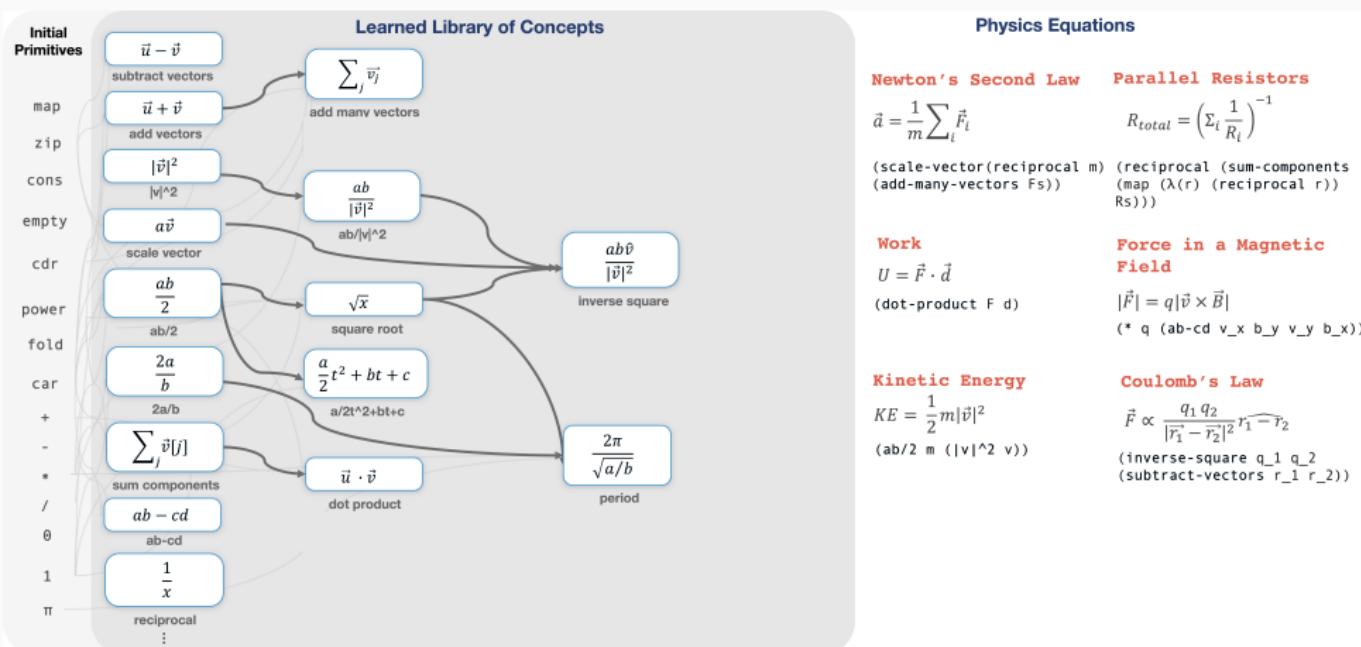
Kinetic Energy

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

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

Initial
Primitives

map

zip

cons

empty

cdr

power

fold

car

+

-

*

/

θ

1

π

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

subtract vectors

$\vec{u} + \vec{v}$

add vectors

$|\vec{v}|^2$

$|v|^2$

$a\vec{v}$

scale vector

$\frac{ab}{2}$

$ab/2$

$\frac{2a}{b}$

$2a/b$

$\sum_j \vec{v}[j]$

sum components

$ab - cd$

$ab - cd$

$\frac{1}{x}$

reciprocal

⋮

Learned Library of Concepts

$\sum_j \vec{v}_j$

add many vectors

$\frac{ab}{|\vec{v}|^2}$

$ab/|v|^2$

\sqrt{x}

square root

$\frac{a}{2}t^2 + bt + c$

$a/2t^2 + bt + c$

$\vec{u} \cdot \vec{v}$

dot product

$\frac{ab\vec{v}}{|\vec{v}|^2}$

inverse square

$\frac{2\pi}{\sqrt{a/b}}$

period

Physics Equations

Newton's Second Law

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

(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)

Parallel Resistors

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

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

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

(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

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

[] → []
[] → []

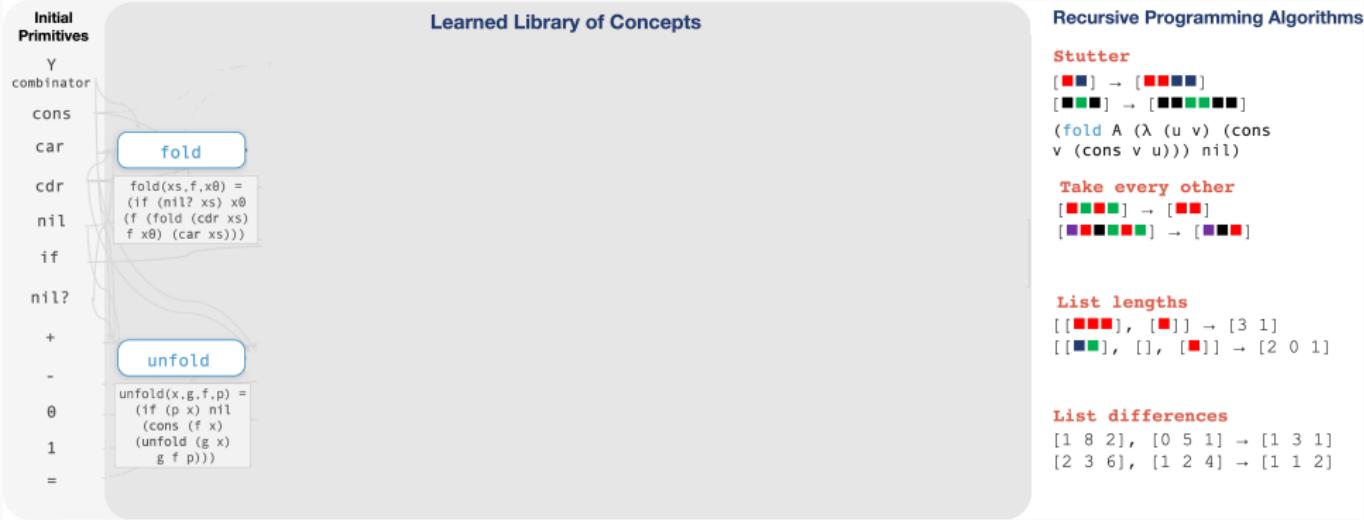
List lengths

[, []] → [3 1]
[[], [], []] → [2 0 1]

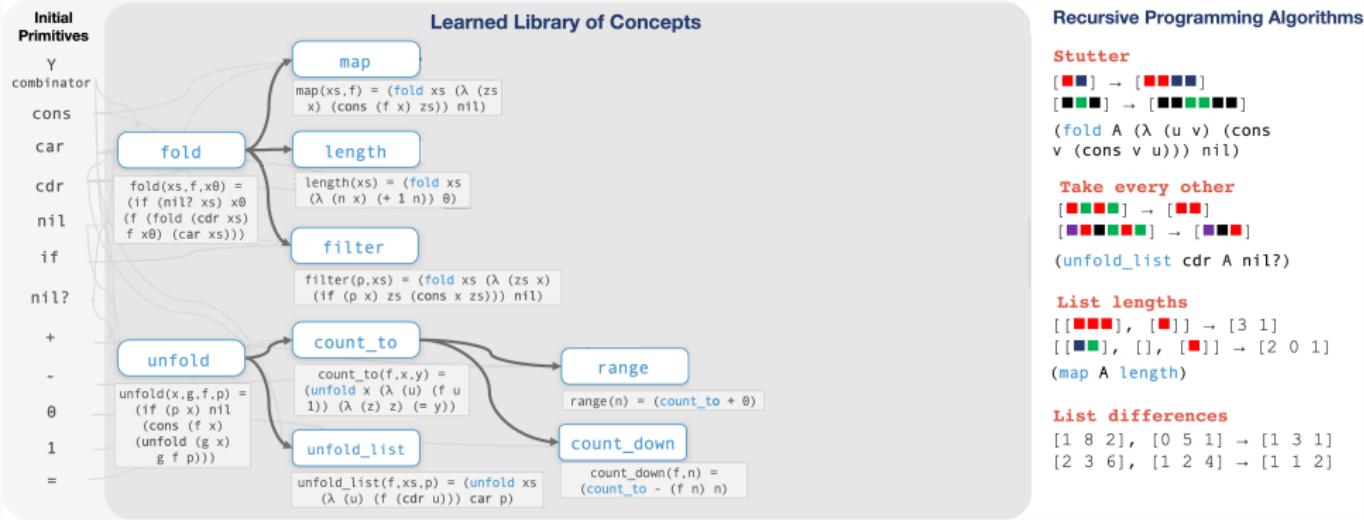
List differences

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

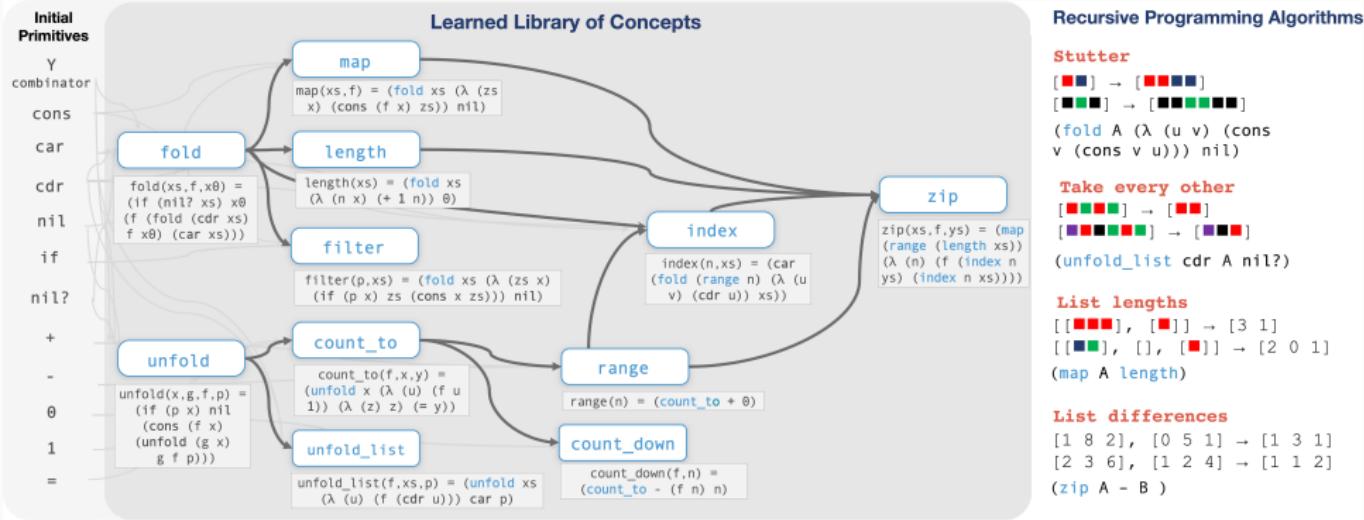
Growing a language for recursive programming



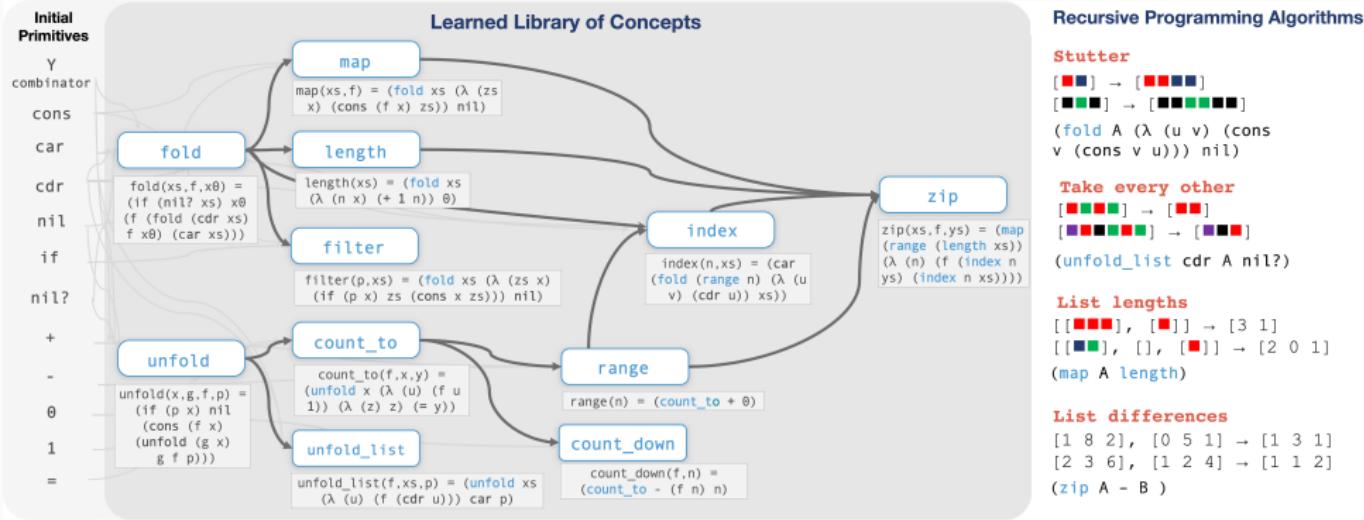
Growing a language for recursive programming



Growing a language for recursive programming

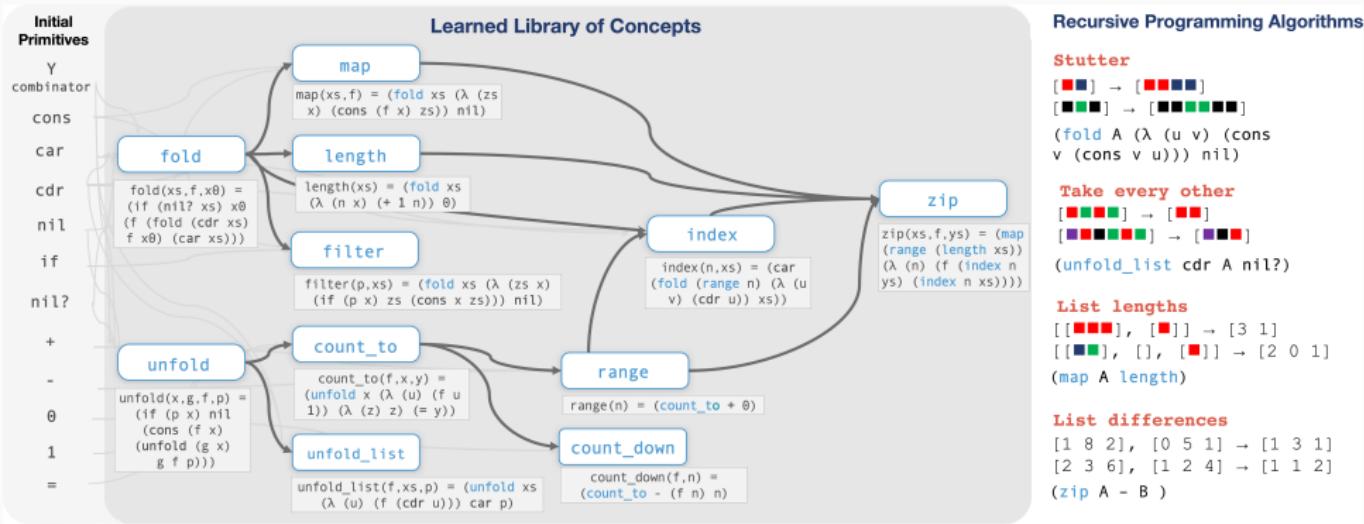


Growing a language for recursive programming



Origami Programming: Jeremy Gibbons, 2003

Growing a language for recursive programming



1 year of compute. 5 days on 64 CPUs.



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. But program induction makes it convenient to build in what we know how to build in, and then learn and adapt on top of that

Program Induction and perception
learning to learn
model discovery
the future

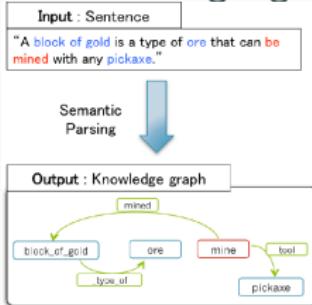
What we've got now:

a toolkit for program induction,
addressing combinatorial program search via learning, integrating
techniques for machine learning, AI, and programming languages

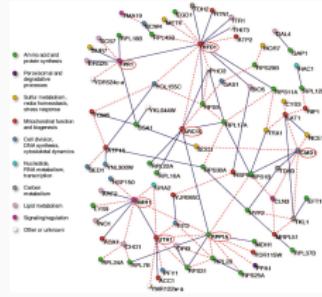
Where will this toolkit take us?

What's in reach

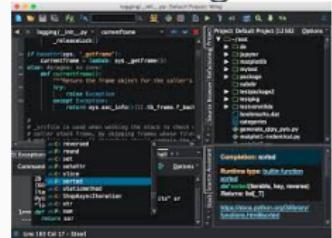
Library learning+ Natural language



Computer-Aided Science

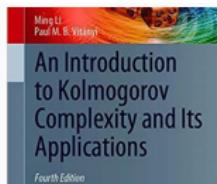


Synthesis for software engineers



Theory for program induction

$$P_M(x) = \sum_{i=1}^{\infty} 2^{-|s_i(x)|}$$



Modeling the physical world

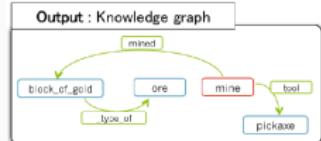


What's in reach

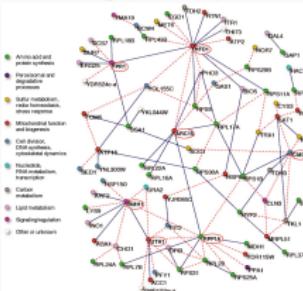
Library learning + Natural language

Input : Sentence
"A block of gold is a type of ore that can be mined with any pickaxe."

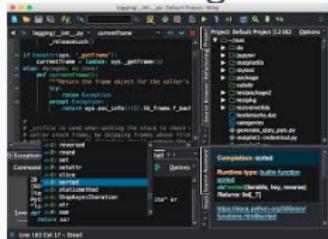
Semantic Parsing



Computer-Aided Science

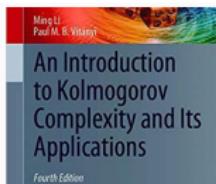


Synthesis for software engineers



Theory for program induction

$$P_M(x) = \sum_{i=1}^{\infty} 2^{-|s_i(x)|}$$

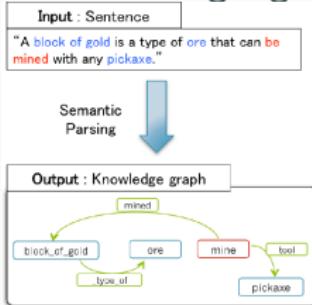


Modeling the physical world

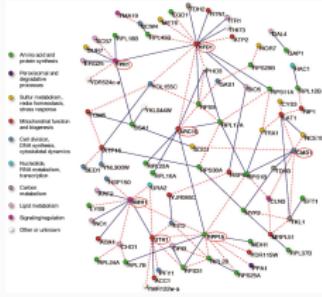


What's in reach

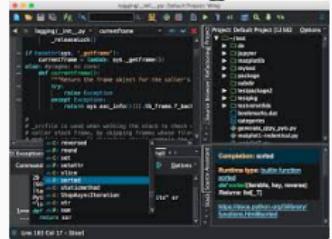
Library learning + Natural language



Computer-Aided Science

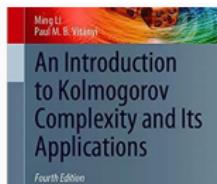


Synthesis for software engineers

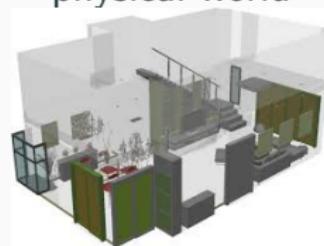


Theory for program induction

$$P_M(x) = \sum_{i=1}^{\infty} 2^{-|s_i(x)|}$$

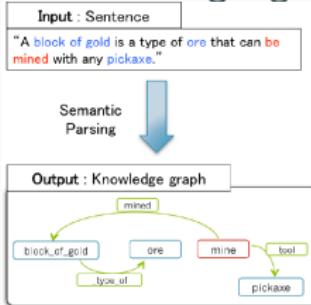


Modeling the physical world

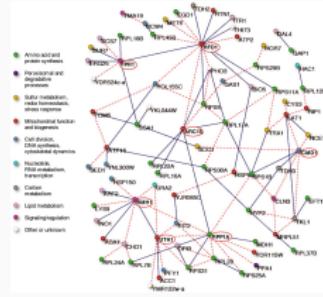


What's in reach

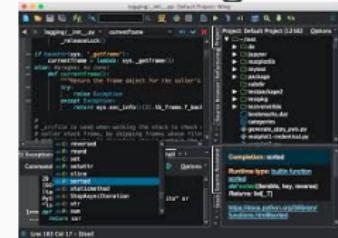
Library learning+ Natural language



Computer-Aided Science

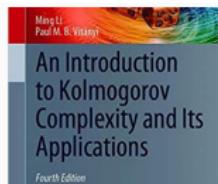


Synthesis for software engineers



Theory for program induction

$$P_M(x) = \sum_{i=1}^{\infty} 2^{-|s_i(x)|}$$

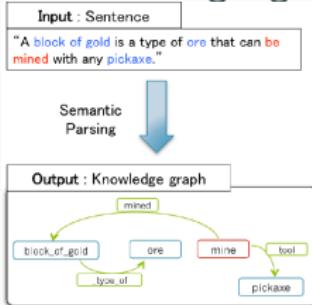


Modeling the physical world

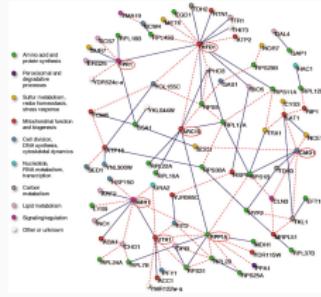


What's in reach

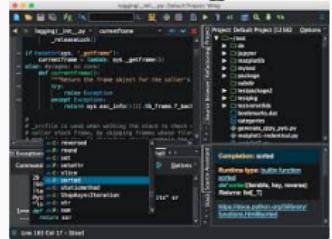
Library learning+ Natural language



Computer-Aided Science

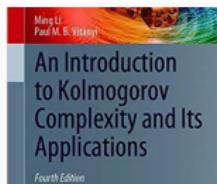


Synthesis for software engineers

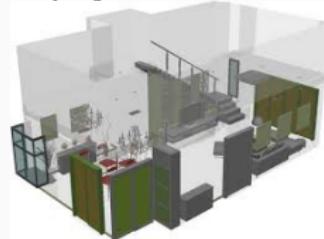


Theory for program induction

$$P_M(x) = \sum_{i=1}^{\infty} 2^{-|s_i(x)|}$$

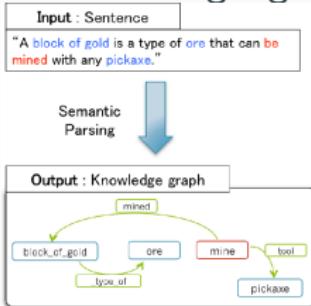


Modeling the physical world

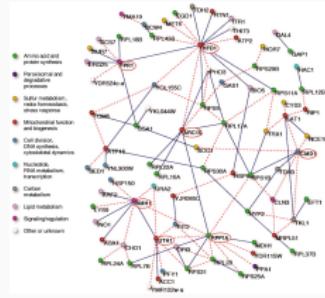


What's in reach

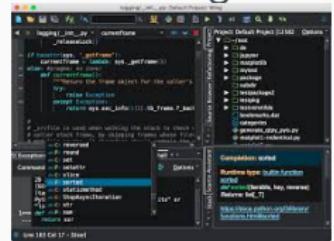
Library learning + Natural language



Computer-Aided Science

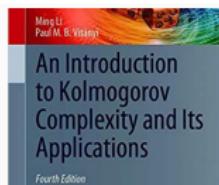


Synthesis for software engineers



Theory for program induction

$$P_M(x) = \sum_{i=1}^{\infty} 2^{-|s_i(x)|}$$



Modeling the physical world



These next steps are in reach.

More broadly: what's in the future of machine learning?

What we want for the future of machine learning

Strong generalization

What we want for the future of machine learning

Strong generalization

Bootstrapping, learning-to-learn, representation learning

What we want for the future of machine learning

Strong generalization

Bootstrapping, learning-to-learn, representation learning

Discovering knowledge that humans can understand and build on

Josh Tenenbaum



Armando Solar-Lezama



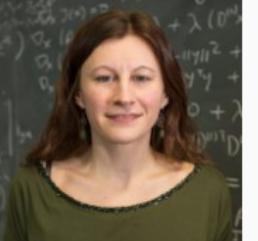
Evan Pu



Lucas Tian



Marta Kryven



Chris Yang



Ronald Alvarez



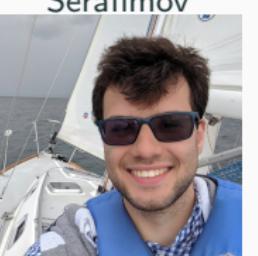
Max Nye



Cathy Wong



Kliment Serafimov



Eyal Dechter



Mathias Sable-Meyer



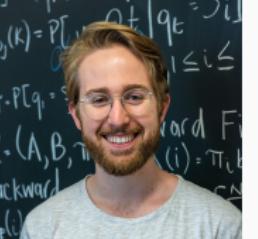
Lucas Morales



Tao Du



Felix Sosa



Sam Tenka

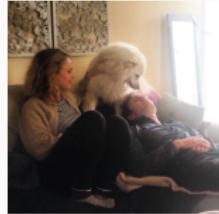


Julio Cortazar



thank you...

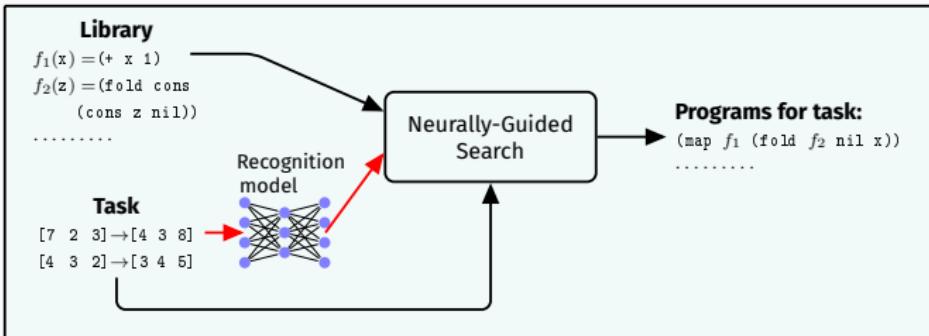
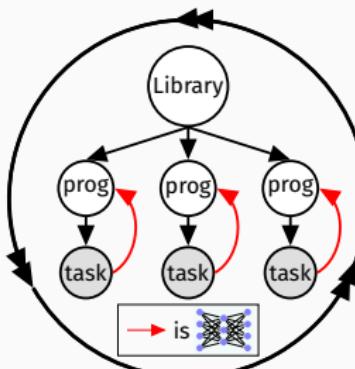
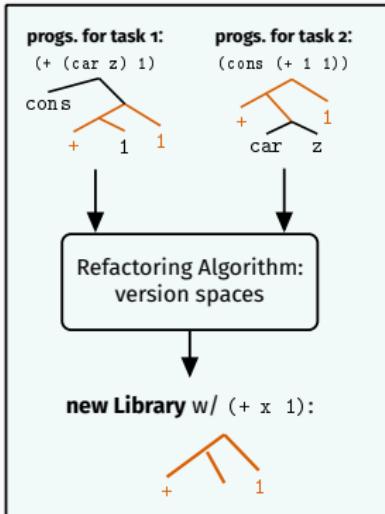
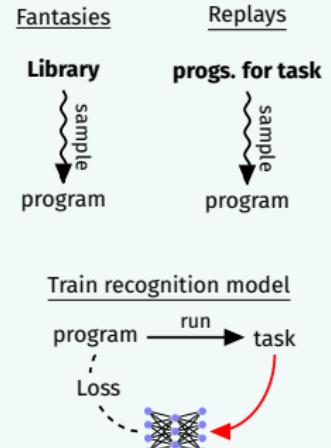
family



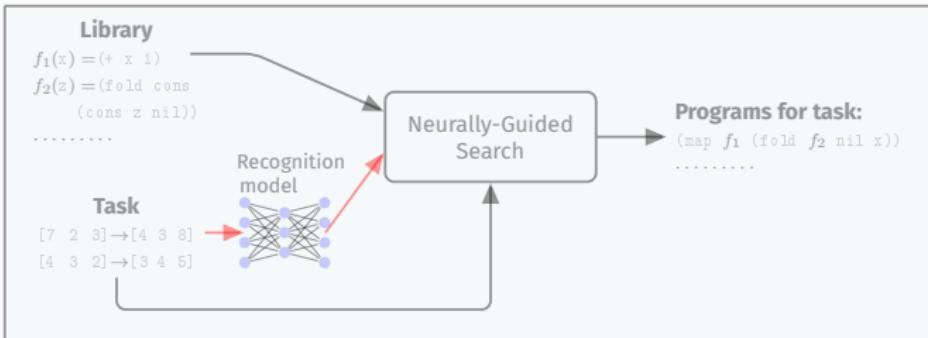
MIT Brain and Cognitive Sciences

twitch viewers

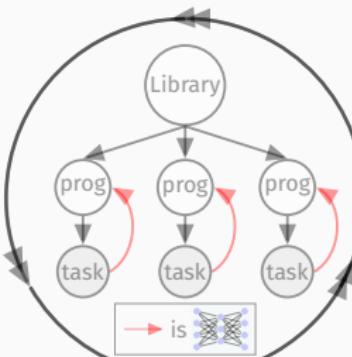
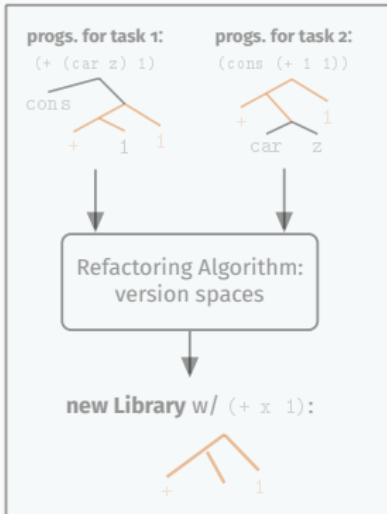
the end.

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

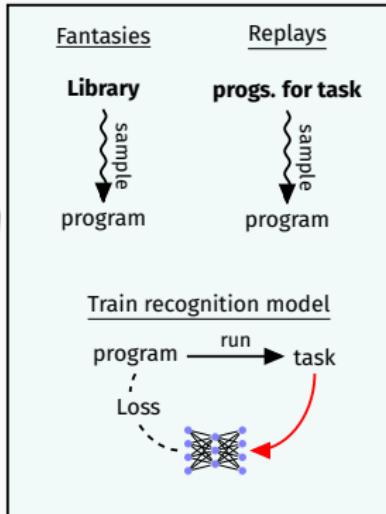
WAKE



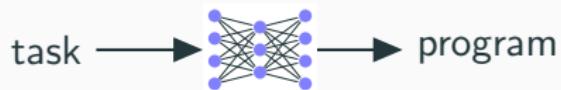
SLEEP: ABSTRACTION



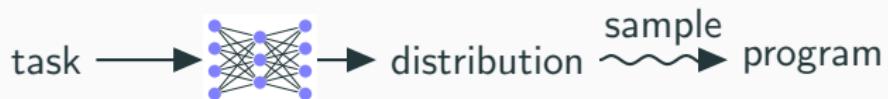
SLEEP: DREAMING



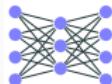
Neural recognition model guides search



Neural recognition model guides search



Neural recognition model guides search

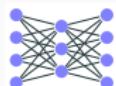


is a...

recurrent network (Devlin et al 2017)

unigram model (Menon et al 2013; Balog et al 2016)

Neural recognition model guides search

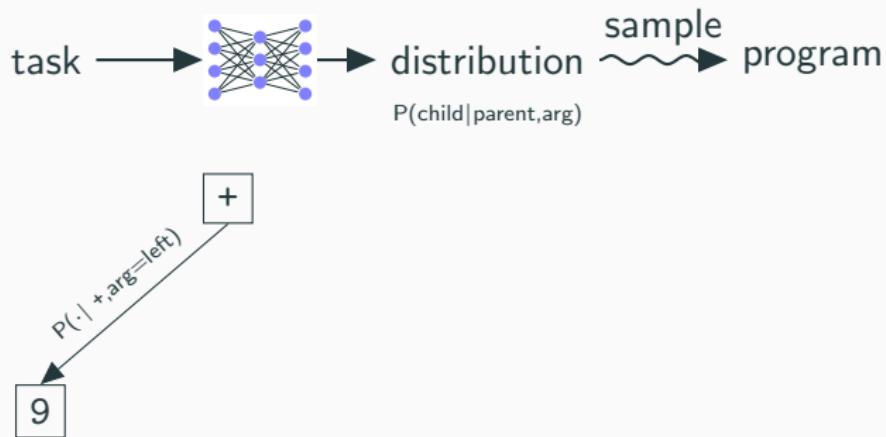


is a “**bigram**” model over syntax trees

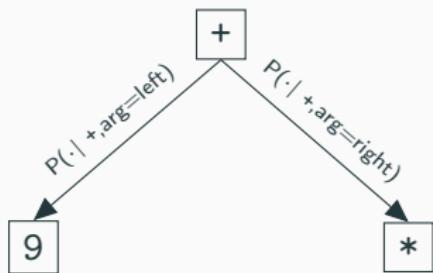
Neural recognition model guides search



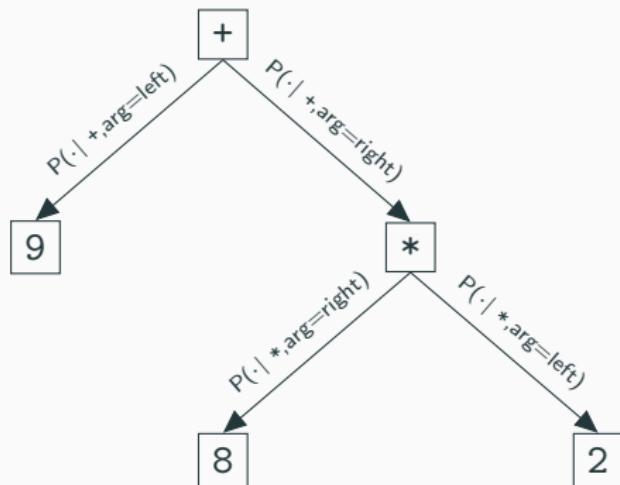
Neural recognition model guides search



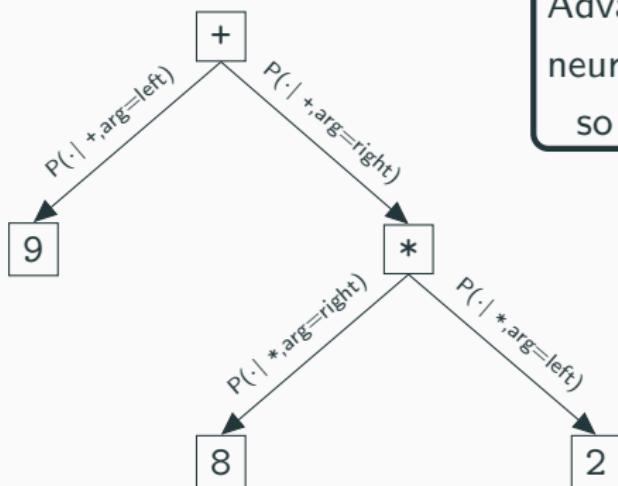
Neural recognition model guides search



Neural recognition model guides search

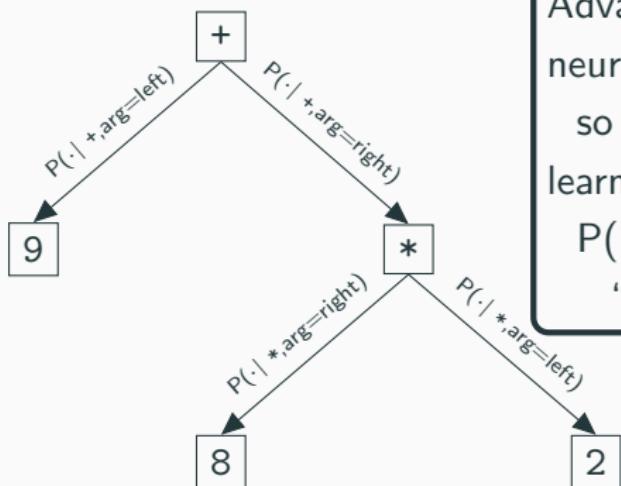


Neural recognition model guides search



Advantages:
neural net runs once per task,
so CPU bottlenecks instead of GPU

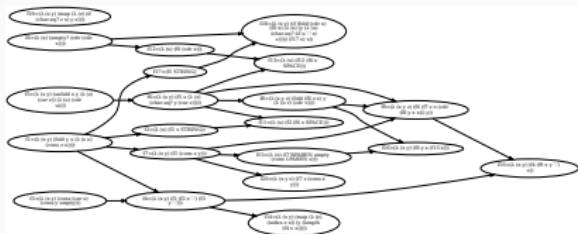
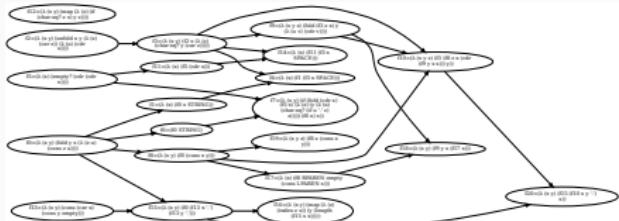
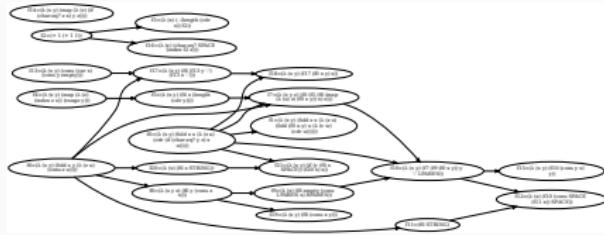
Neural recognition model guides search



Advantages:
neural net runs once per task,
so CPU bottlenecks instead of GPU
learns to break syntactic symmetries:
 $P(1|*,\text{arg}=left)=0.0$
“do not multiply by one”

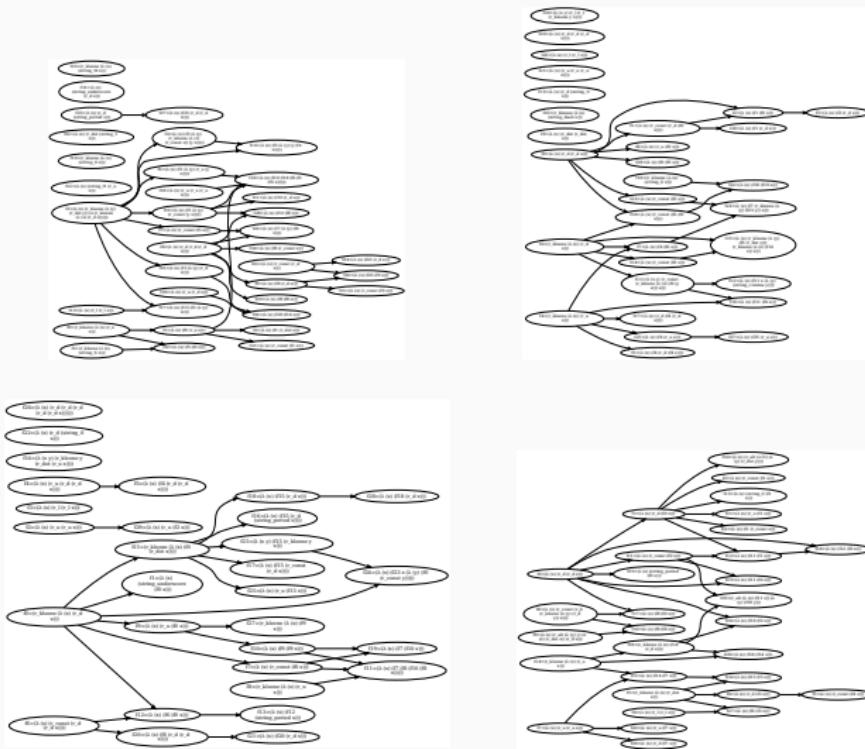
Library structure: Text Editing

DreamCoder learns libraries for FlashFill-style text editing [Gulwani 2012]

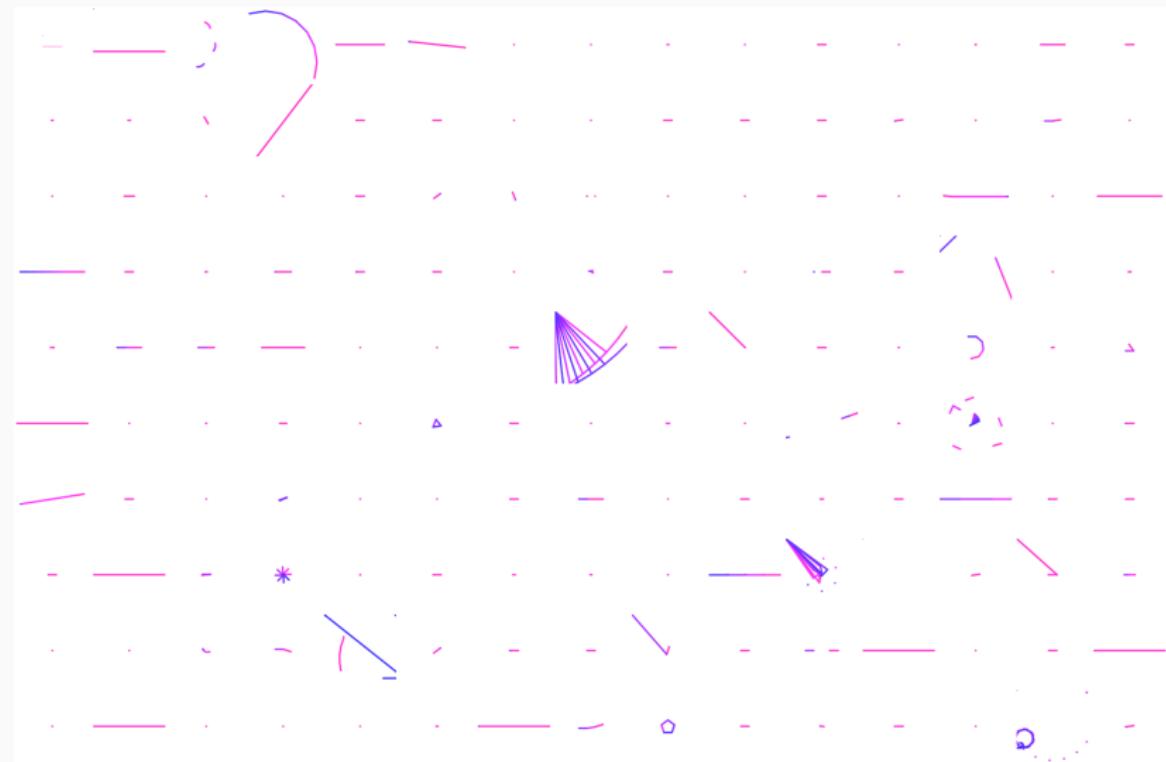


Library structure: Generating Text

Libraries for probabilistic generative models over text:
data from crawling web for CSV files



150 random dreams before learning



150 random dreams after learning

