Building Machines that Discover Generalizable, Interpretable Knowledge

Kevin Ellis

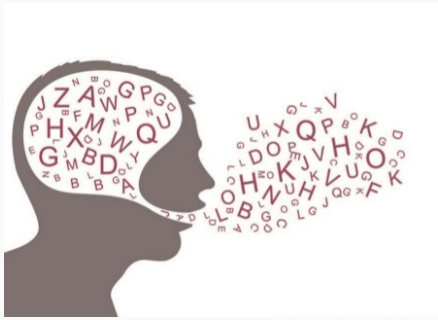
2020

MIT

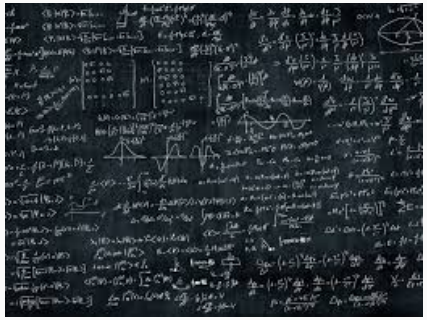
What computational problems are solved by intelligence?

an endless range of problems

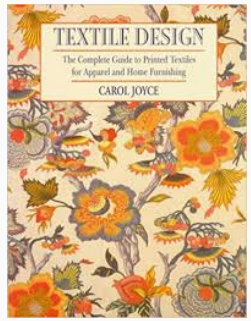
language



science

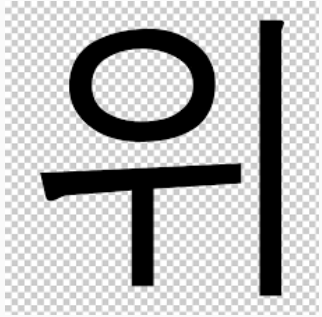


design



using new devices

writing new characters



coding

```
(MEMBER
 (LAMBDA (X L)
  (COND  ((NULL L) NIL)
         ((EQ X (FIRST L)) T)
         (T (MEMBER X (REST L))))))
```

Allen, Anatomy of Lisp, 1975

engineering

play

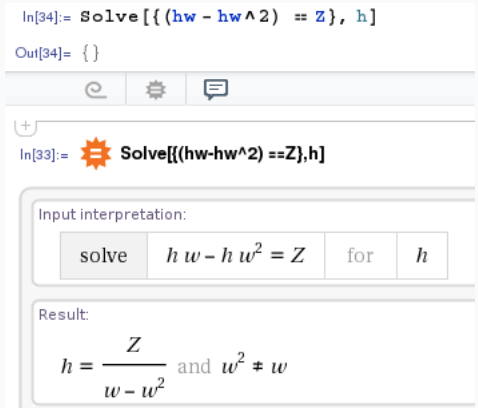
What computational frameworks can contribute to this picture?

Three AI traditions

What computational frameworks can contribute to this picture?

Three AI traditions

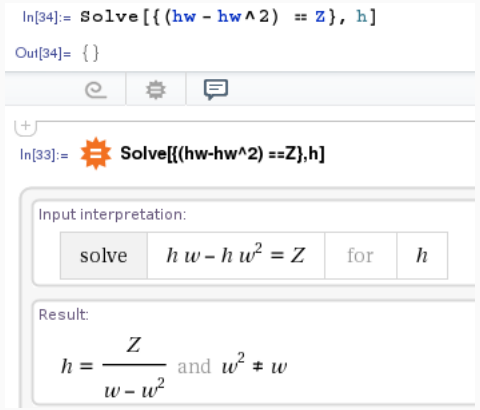
Symbolic



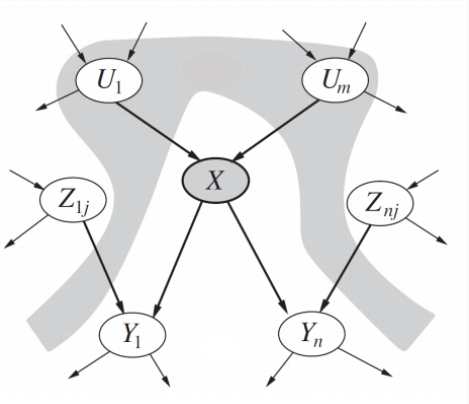
What computational frameworks can contribute to this picture?

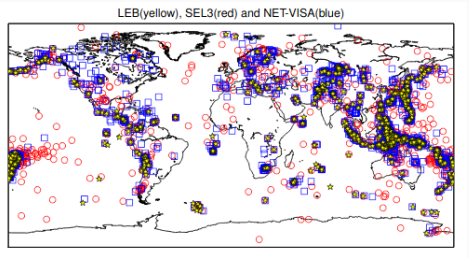
Three AI traditions

Symbolic



Probabilistic





What computational frameworks can contribute to this picture?

Three AI traditions

Symbolic


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

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

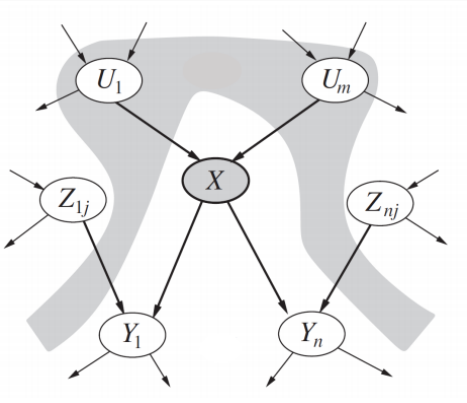
Input interpretation:

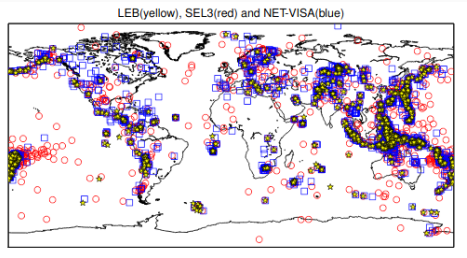
solve | $h\,w - h\,w^2 = Z$ | for | h

Result:

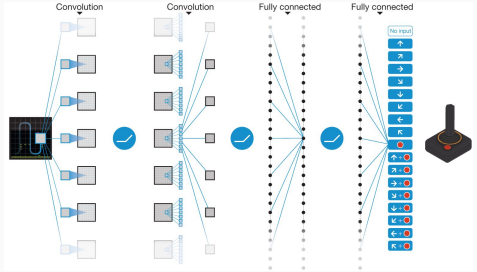
$$h = \frac{Z}{w - w^2} \quad \text{and} \quad w^2 \neq w$$

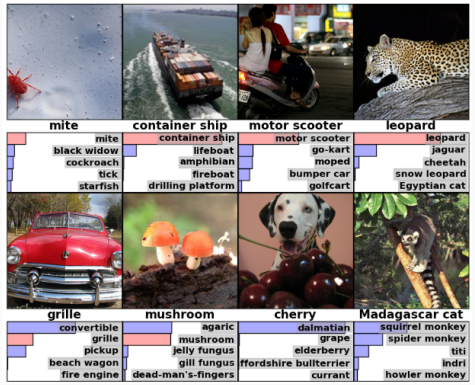
Probabilistic





Neural

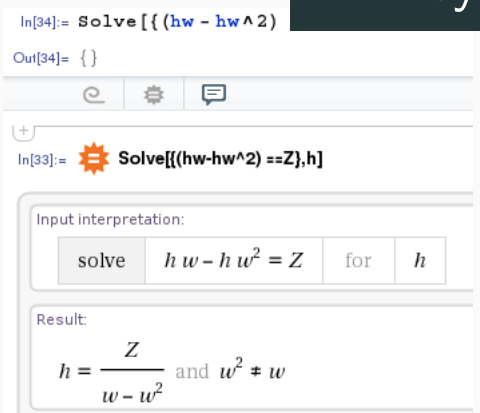




What computational frameworks can contribute to this picture?

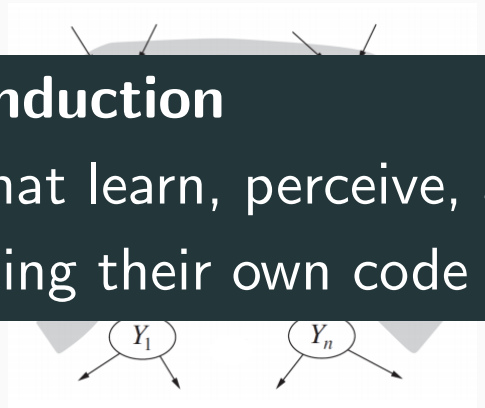
Three AI traditions

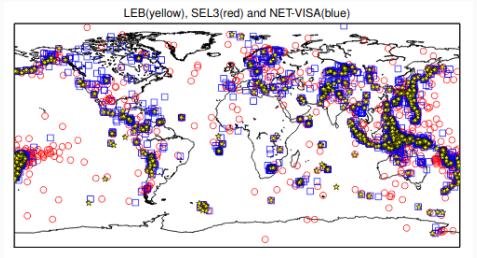
Symbolic | Probabilistic | Neural



Program induction

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



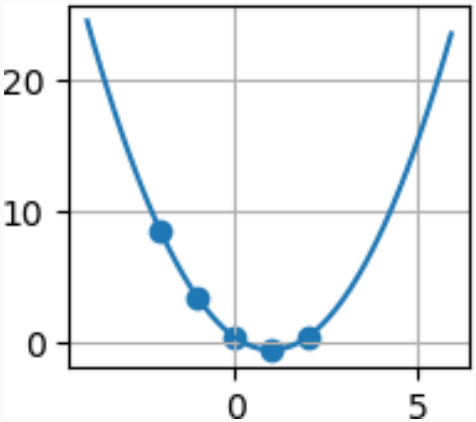


Why program induction?

Why program induction?

strong generalization

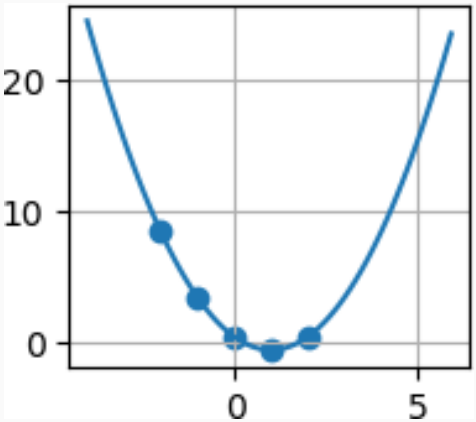
+data efficiency




```
f(x)=(x-1)**2 - 0.5
```

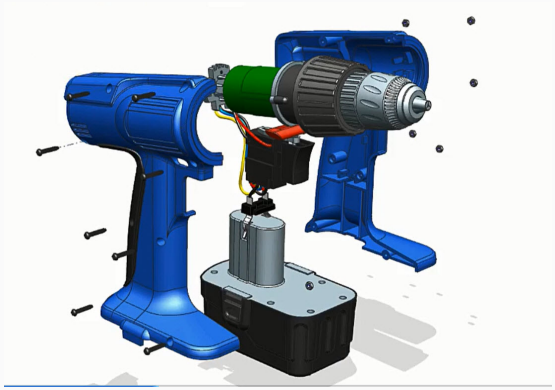
Why program induction?

strong generalization
+data efficiency

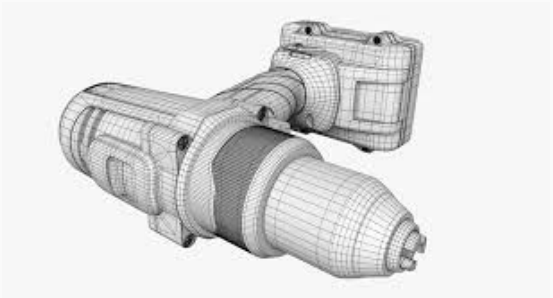



```
f(x)=(x-1)**2 - 0.5
```

interpretability

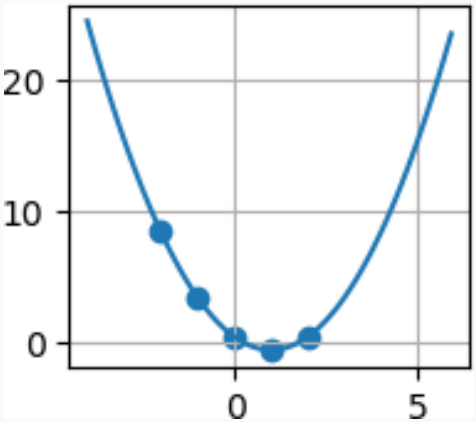


vs



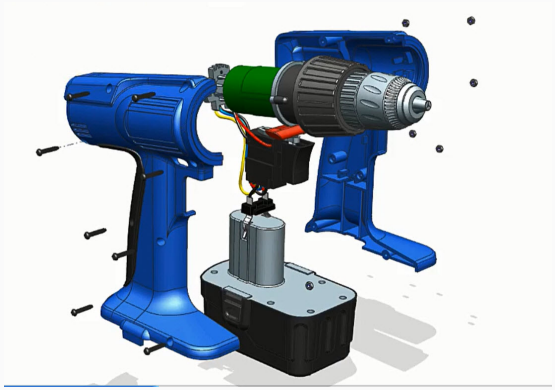
Why program induction?

strong generalization
+data efficiency

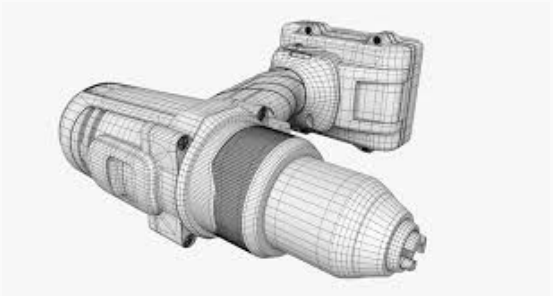



```
f(x)=(x-1)**2 - 0.5
```

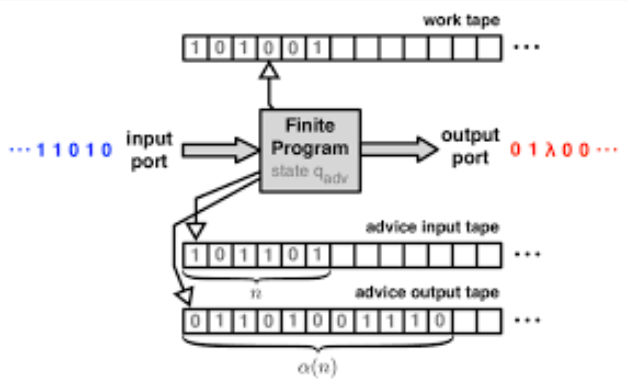
interpretability



vs



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 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]

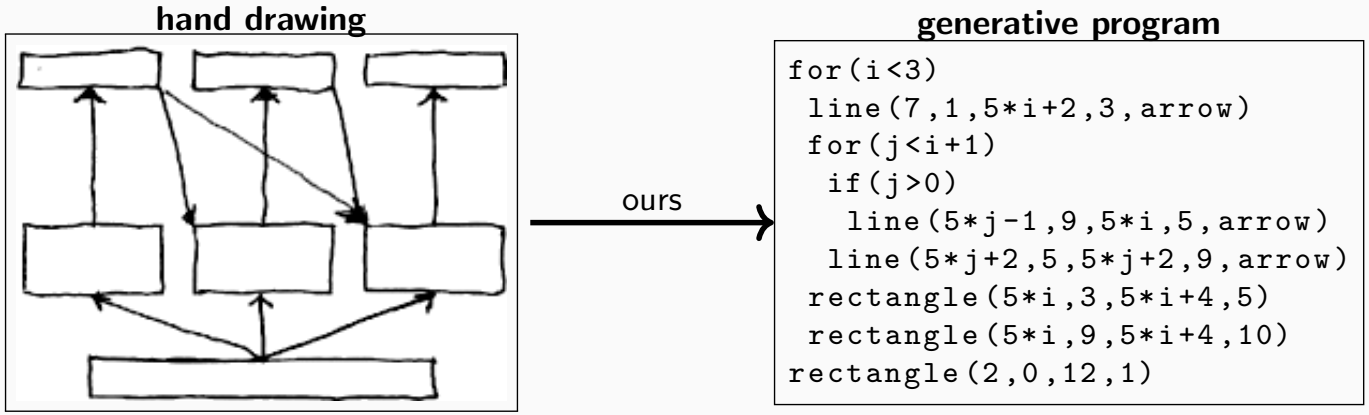
programming by examples [Gulwani 2011]

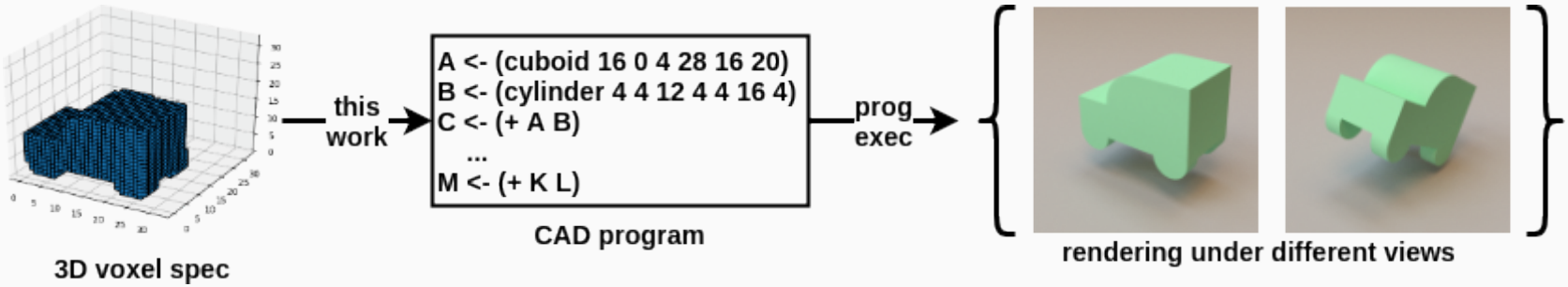
computer-aided-programming [Solar-Lezama 2008]

inverse procedural modeling [Kulkarni et al. 2015]

Perception, Synthesizing models, Learning-to-Learn

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

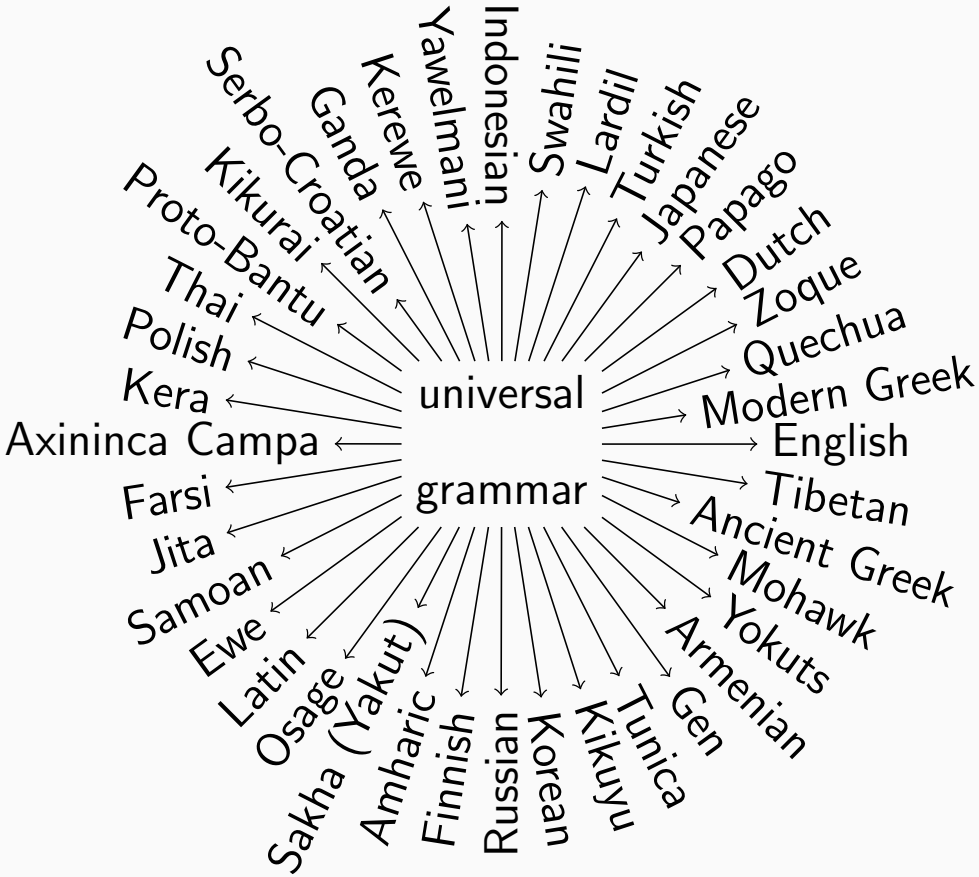




Perception, Synthesizing models, Learning-to-Learn

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] → 6
[4 6 8 1] → 17
```

Double

```
[1 2 3 4] → [2 4 6 8]
[6 5 1] → [12 10 2]
```

Check Evens

```
[0 2 3] → [T T F]
[2 4 9 6] → [T T F T]
```

Text Editing

Abbreviate

```
Allen Newell → A.N.
Herb Simon → H.S.
```

Drop Last Characters

```
jabberwocky → jabberw
copycat → cop
```

Extract

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

Regexes

Phone Numbers

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

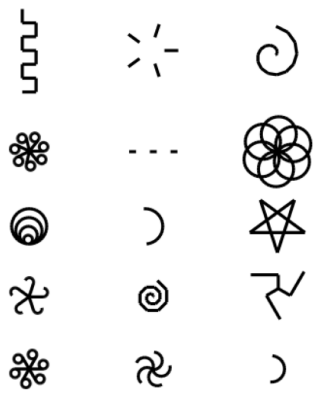
Currency

```
$100.25
$4.50
```

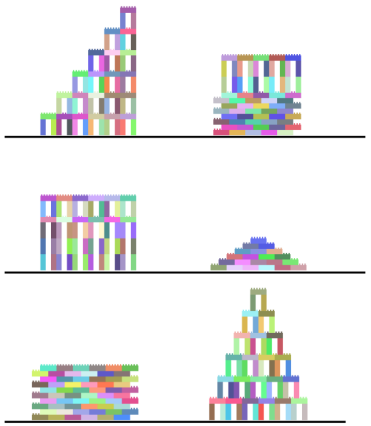
Dates

```
Y1775/0704
Y2000/0101
```

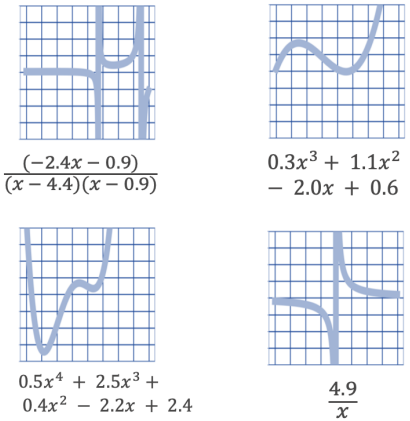
LOGO Graphics



Block Towers

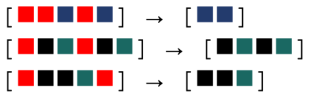


Symbolic Regression



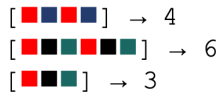
Recursive Programming

Filter

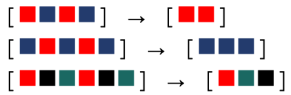


Index List

Length



Every Other



Physics

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

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

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1} - \vec{r_2}|^2}\widehat{r_1 - r_2}$$

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

Program Induction and perception

model discovery

learning to learn

Vision is more than knowing what is where

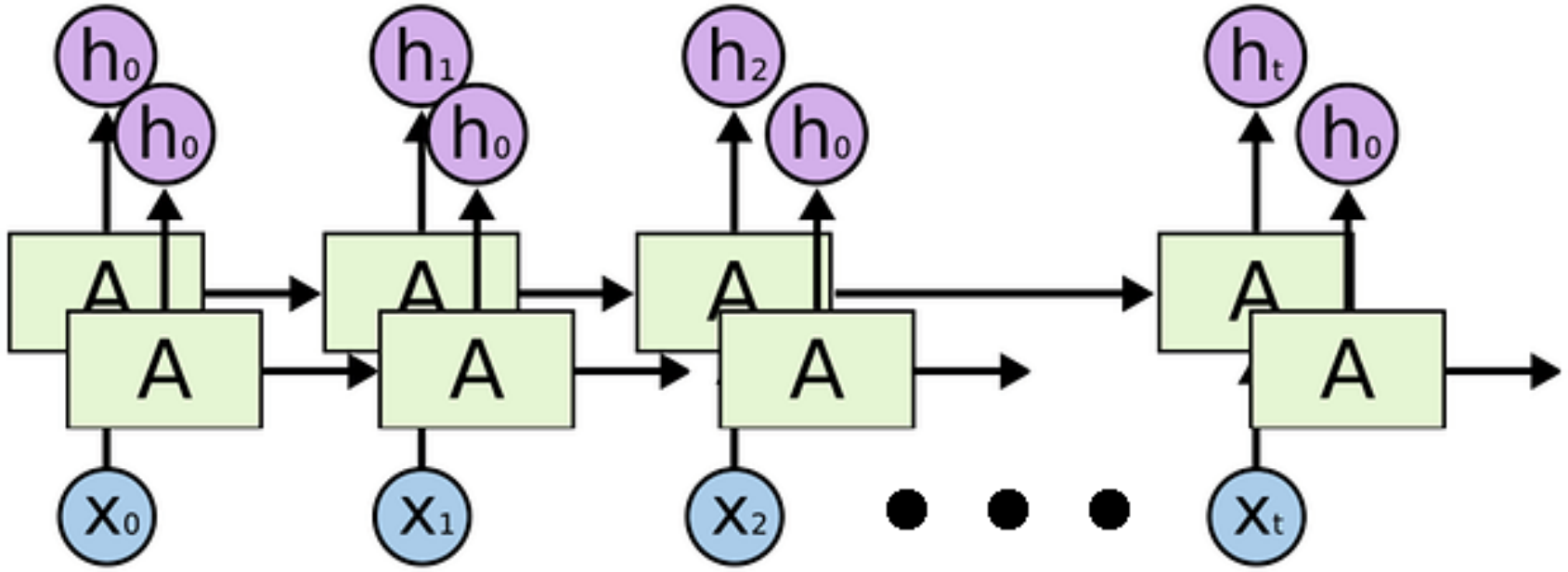
Can you visually extrapolate this aqueduct?

Vision is more than knowing what is where

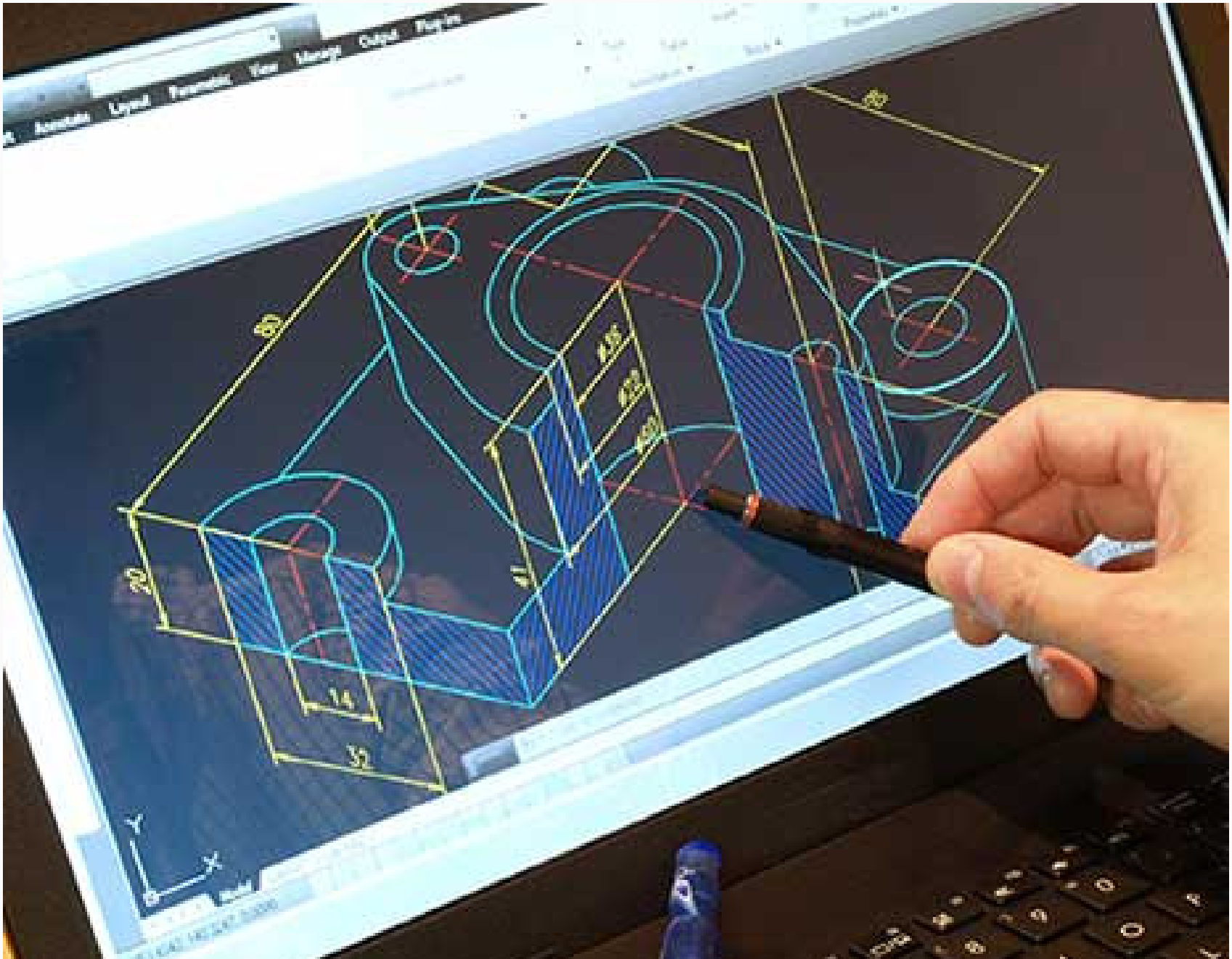
Can you visually extrapolate this aqueduct?

Vision is more than knowing what is where

Can you infer what goes in the ellipses?

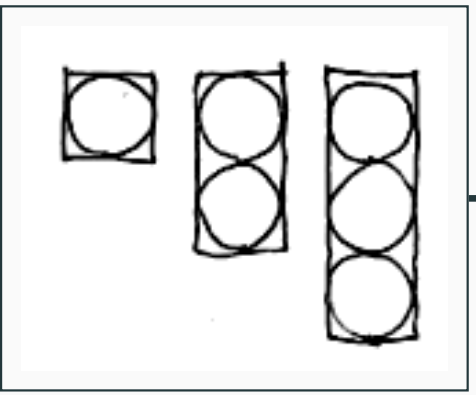


Vision is more than knowing what is where



Learning to infer graphics programs from hand-drawn images

hand drawing

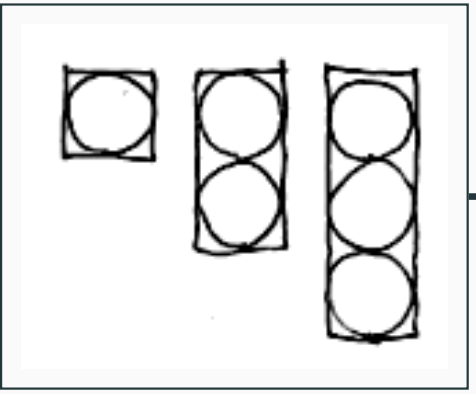


machine-generated program

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

Learning to infer graphics programs from hand-drawn images

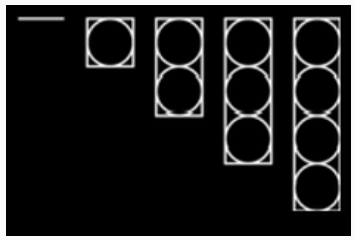
hand drawing



machine-generated program

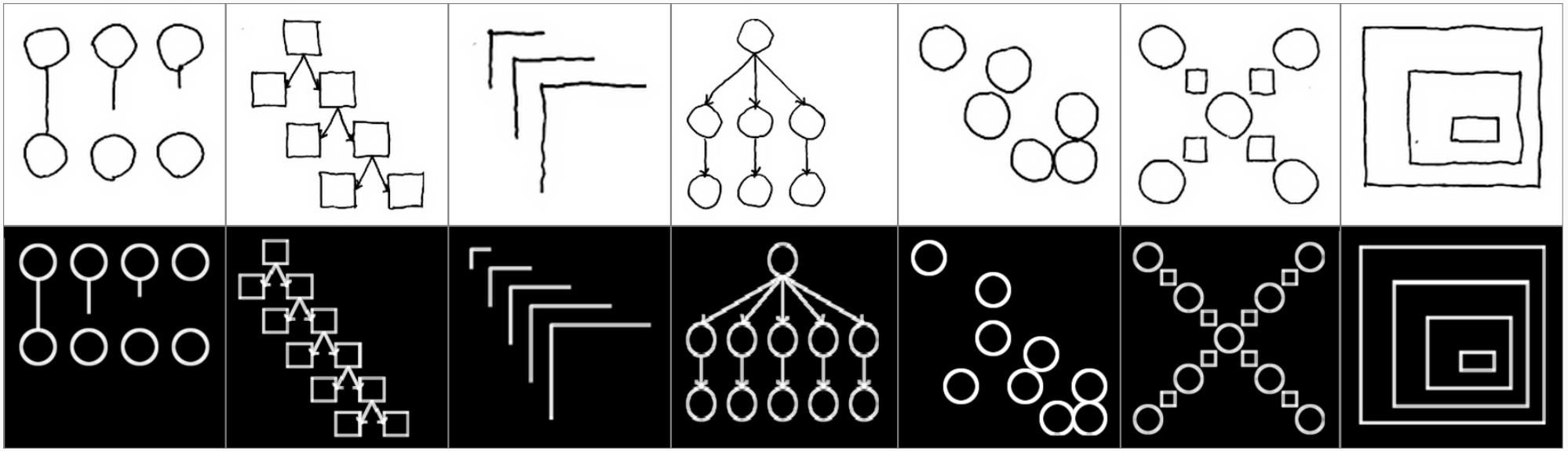
```
for (0 <= i < 3 + 1)
 rectangle(3*i,-2*i+4,
           3*i+2,6)
 for (0 <= j < i + 1)
  circle(3*i+1,-2*j+5)
```

autogenerated extrapolation

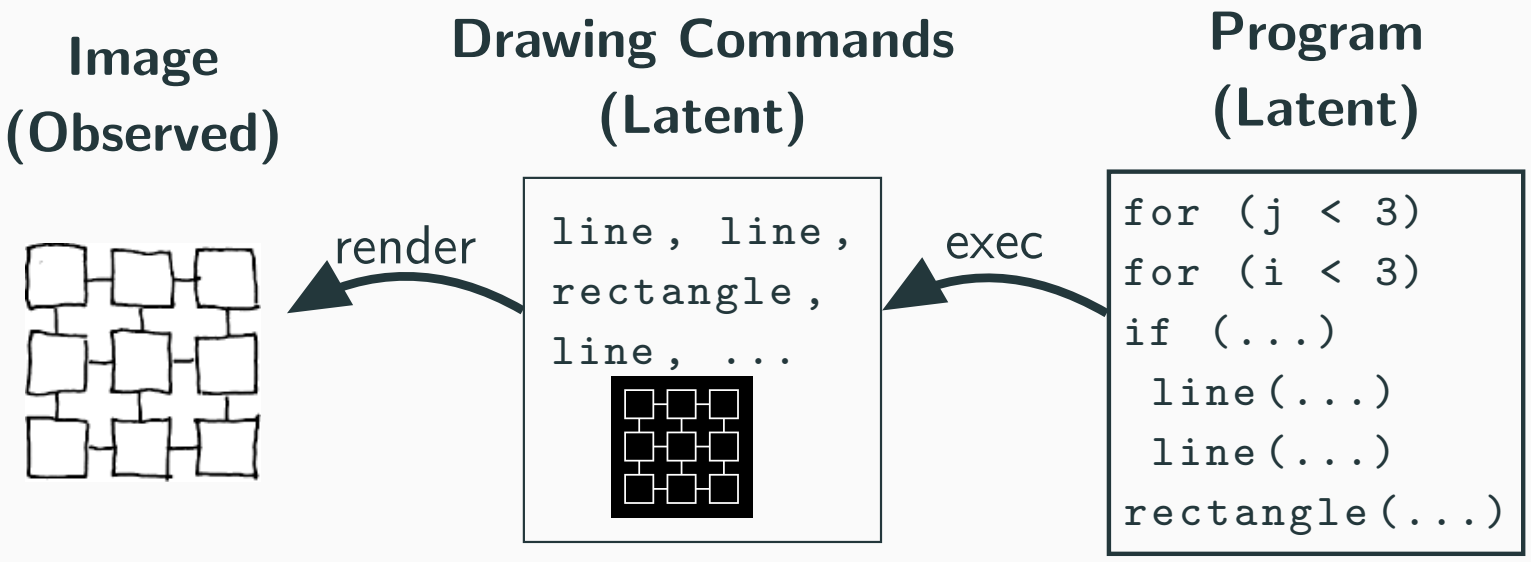


zero-shot generalization / extrapolation

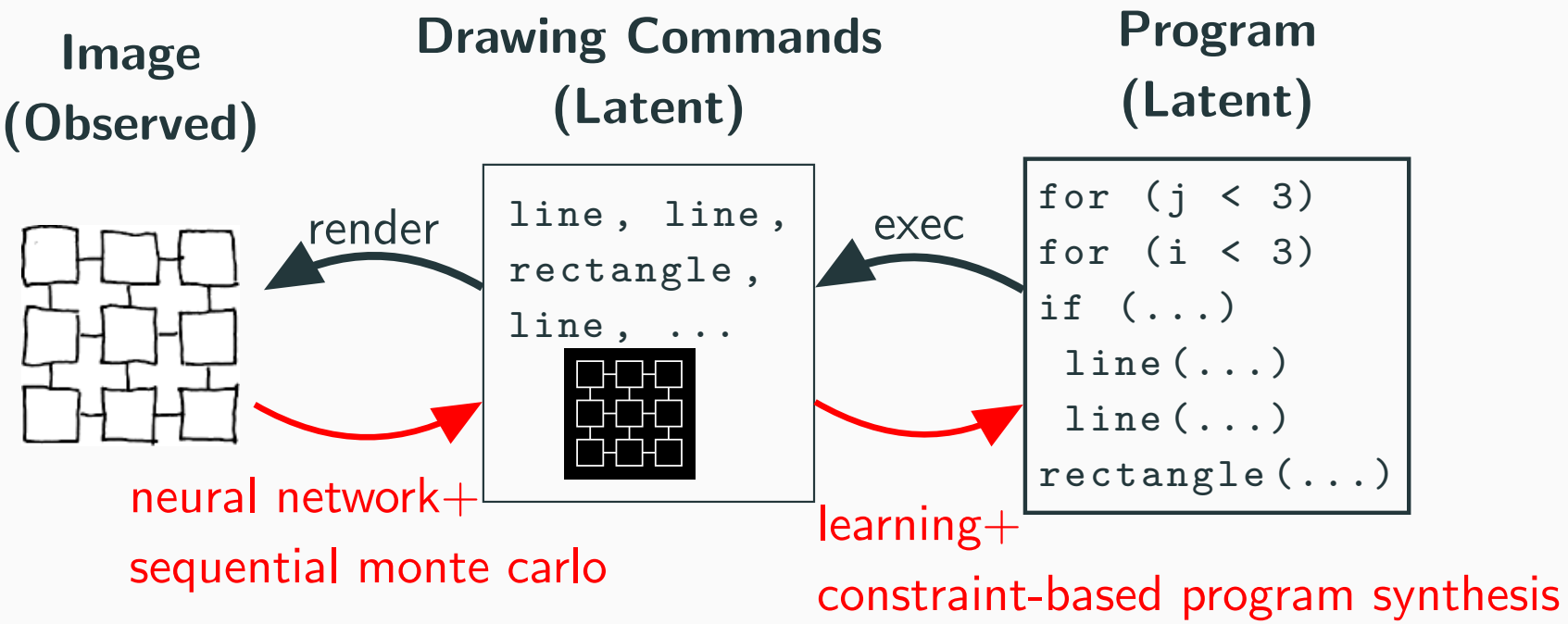
Extrapolation from a single image



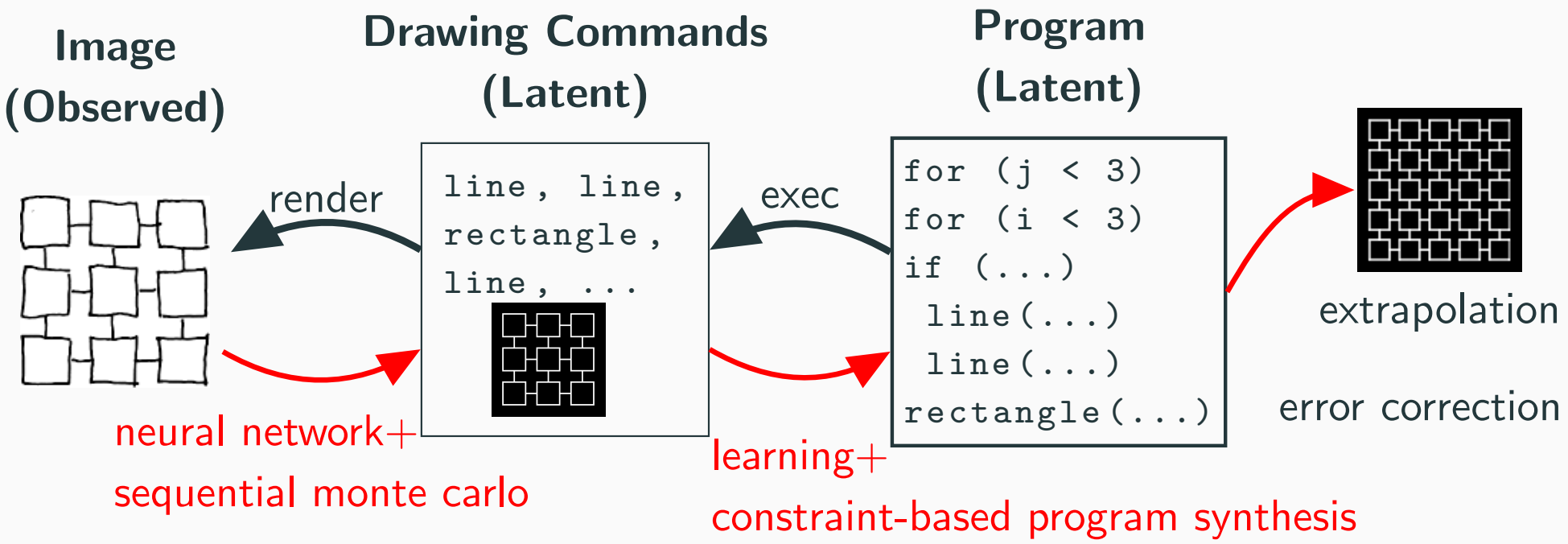
How to infer graphics programs from hand-drawn images



How to infer graphics programs from hand-drawn images



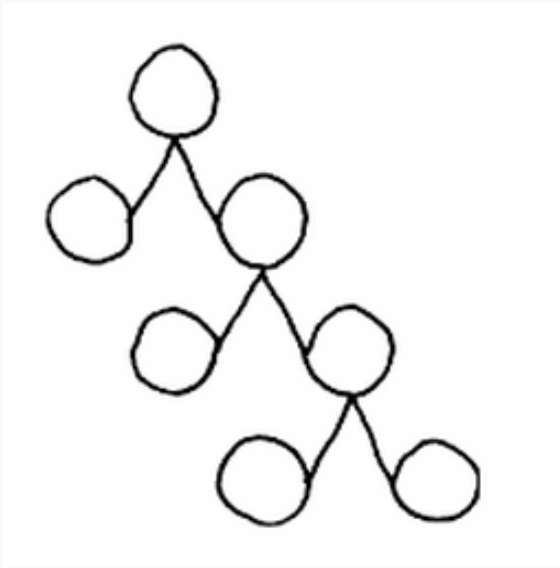
How to infer graphics programs from hand-drawn images



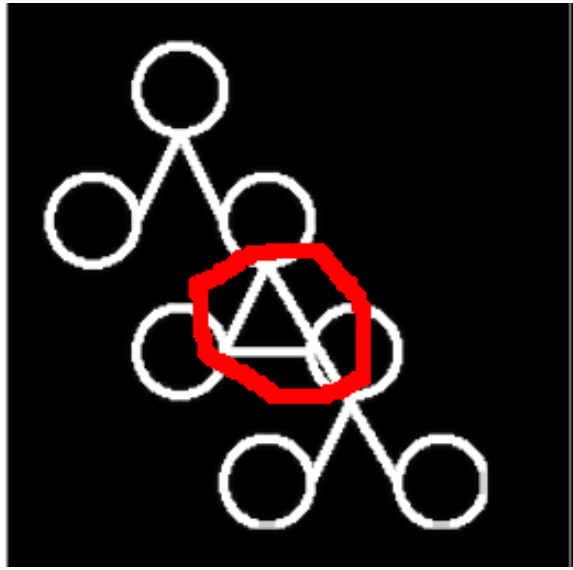
Image→Parse | Parse→Program | Applications

Top-down influences on perception

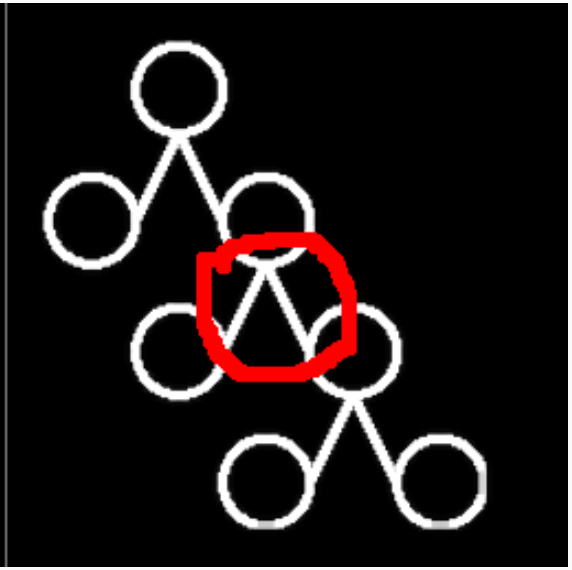
drawing



bottom-up neural net

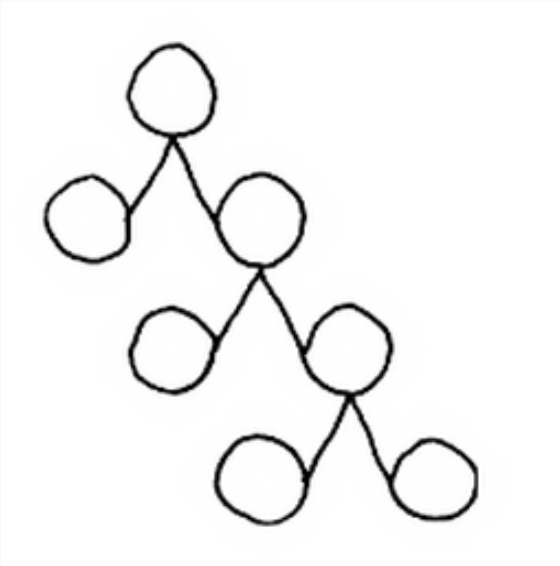


w/ top-down program bias

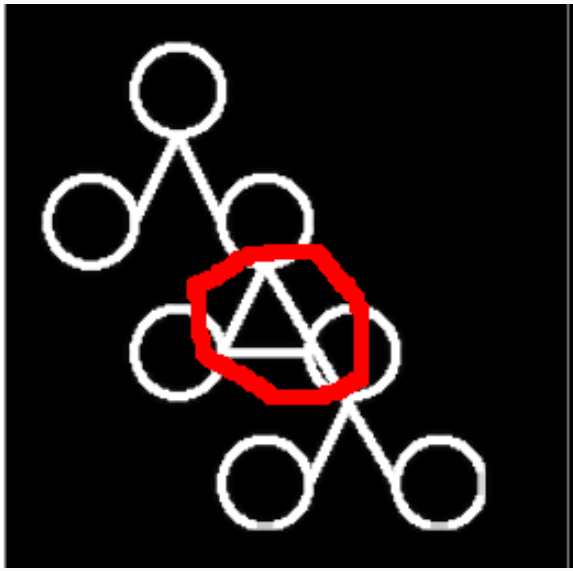


Top-down influences on perception

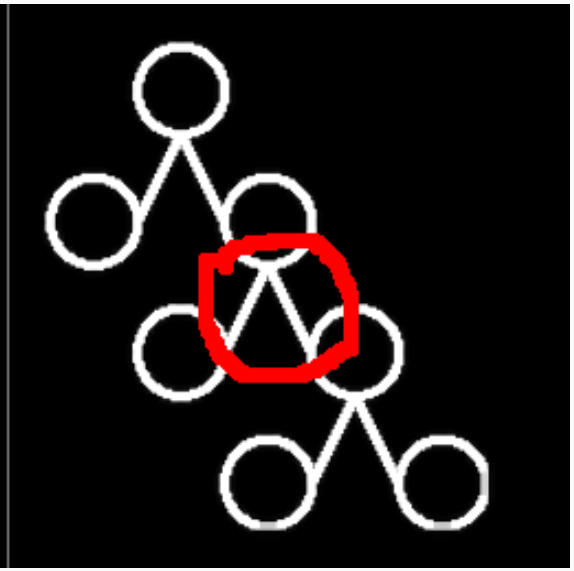
drawing

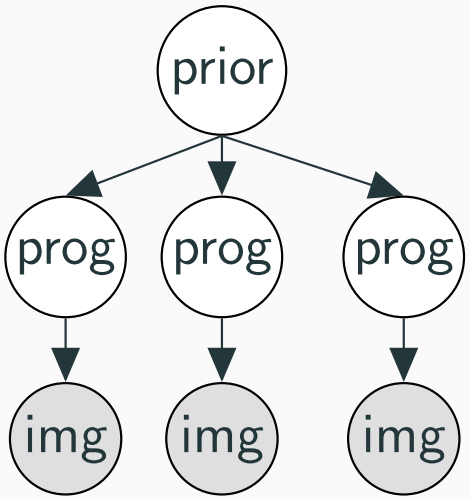


bottom-up neural net



w/ top-down program bias



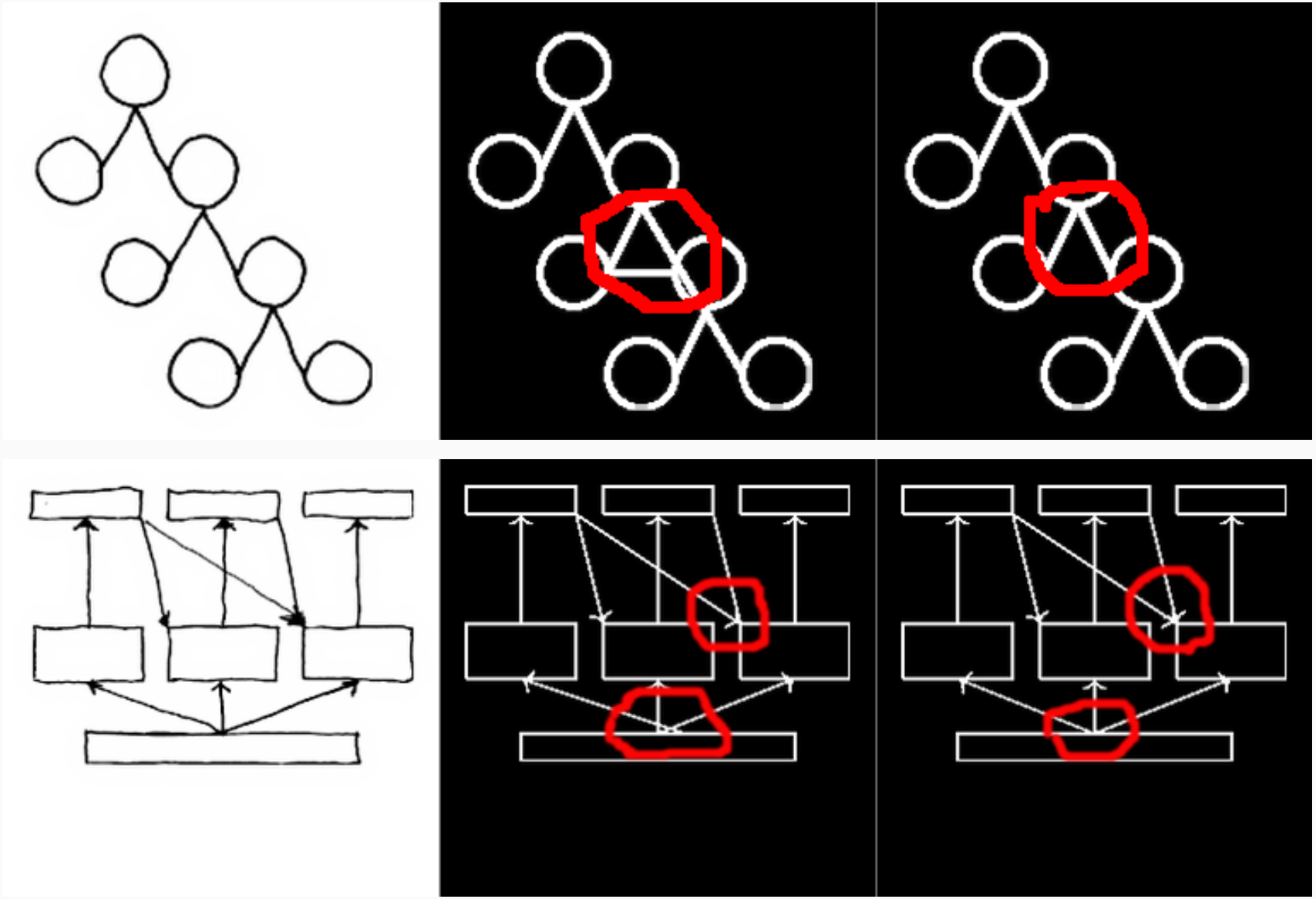


predicted program $=$

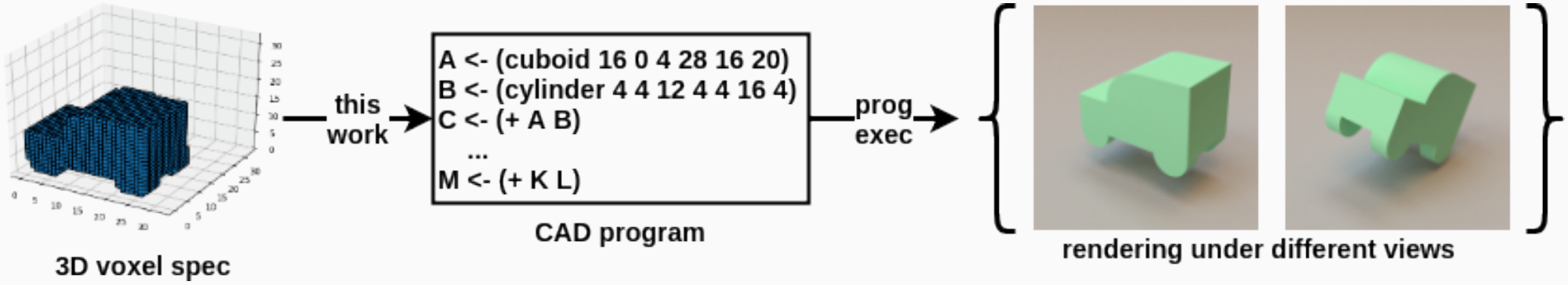
$\arg\max_{\text{progs}} \mathbb{P}\left[\text{img}|\text{prog}\right] \mathbb{P}\left[\text{prog}|\text{prior}\right]$

Top-down influences on perception

drawing | bottom-up neural net | w/ top-down program bias

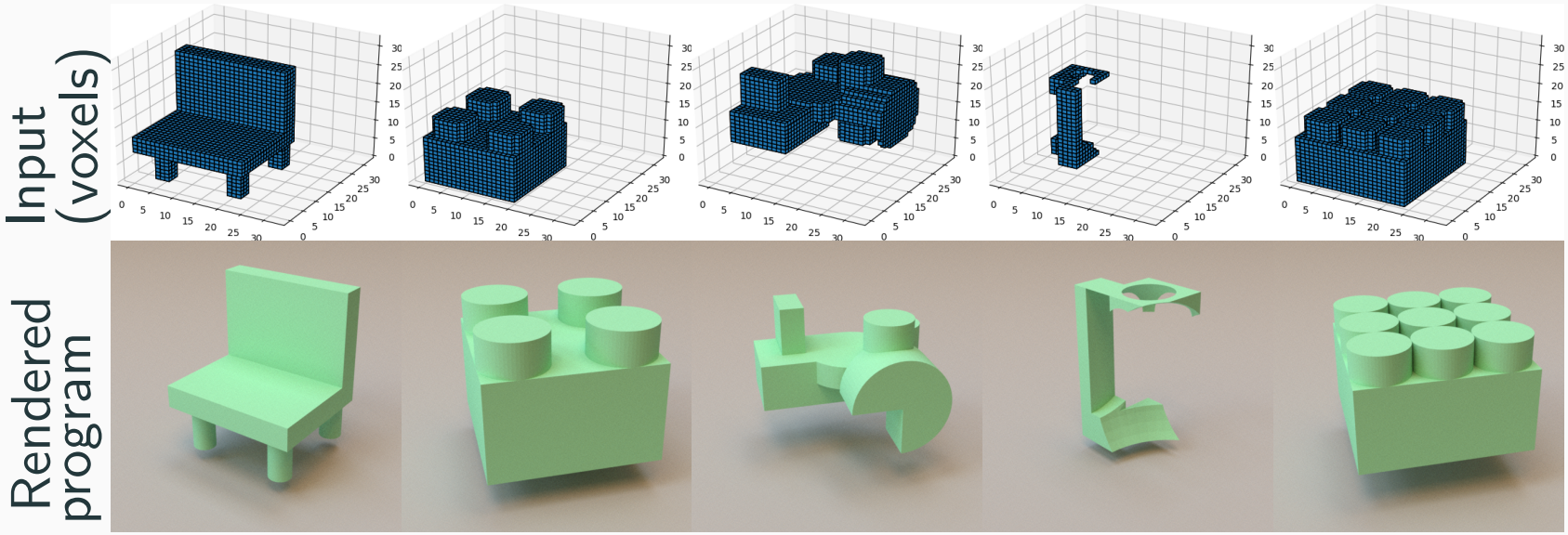


3D program induction



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

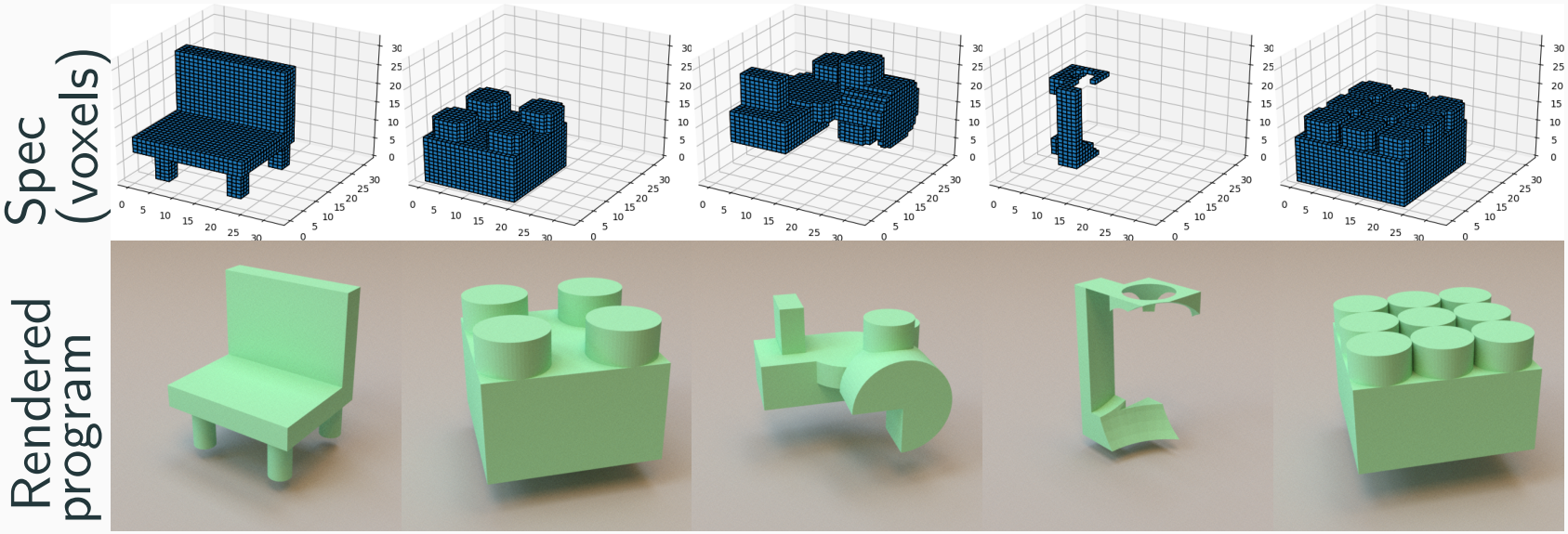
3D program induction



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

*equal contribution

3D program induction



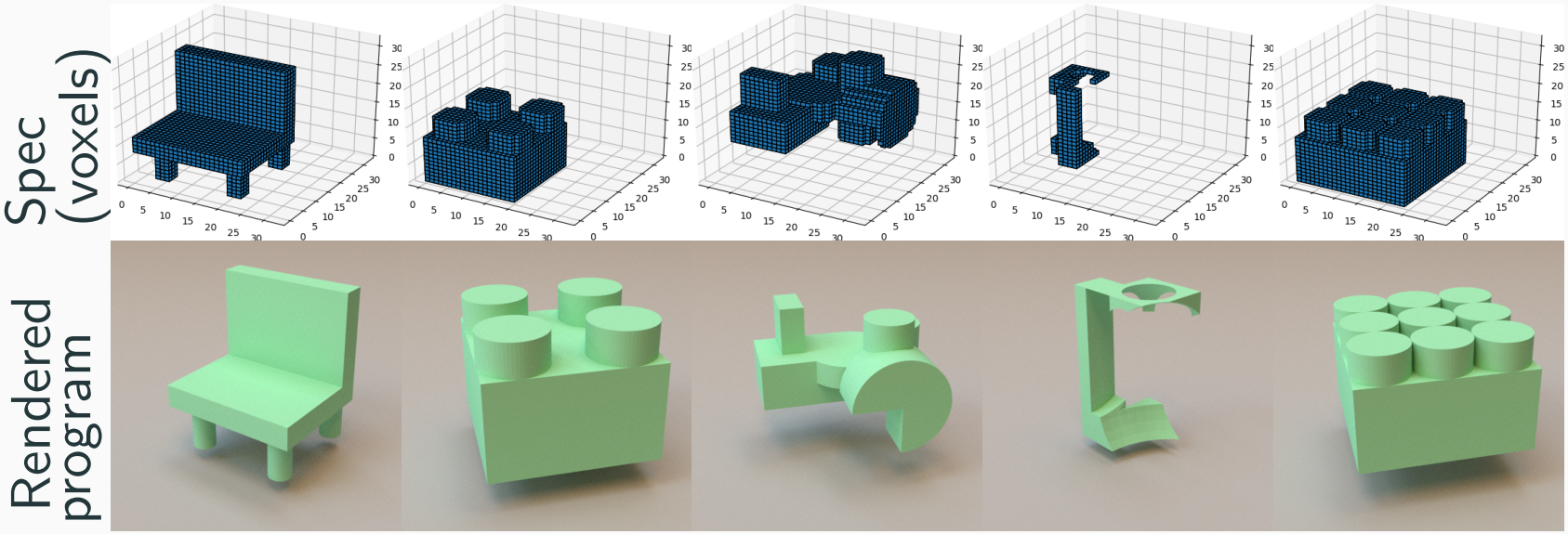
Challenge: combinatorics!

Branching factor: > 1.3 million per line of code, ≈ 20 lines of code

search space size: $(1.3 \text{ million})^{20} \approx 10^{122}$ programs

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

3D program induction



Challenge: combinatorics!

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

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

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

Lessons

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

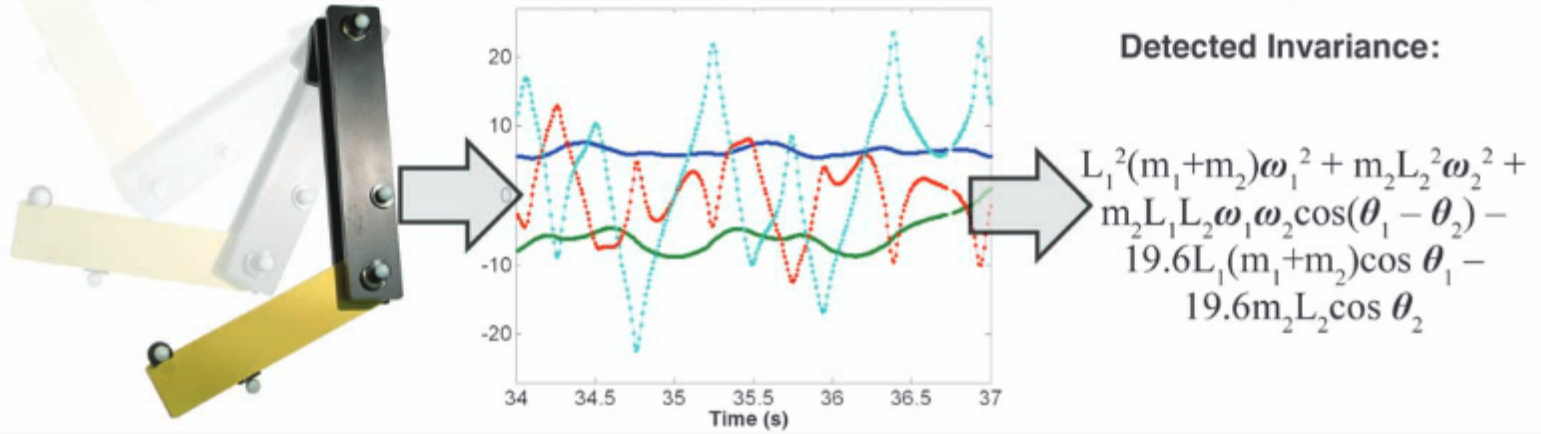
Combine the best of different techniques: neural nets for perception and pattern recognition, symbols for reasoning, Bayesian methods for uncertainty

Program Induction and perception

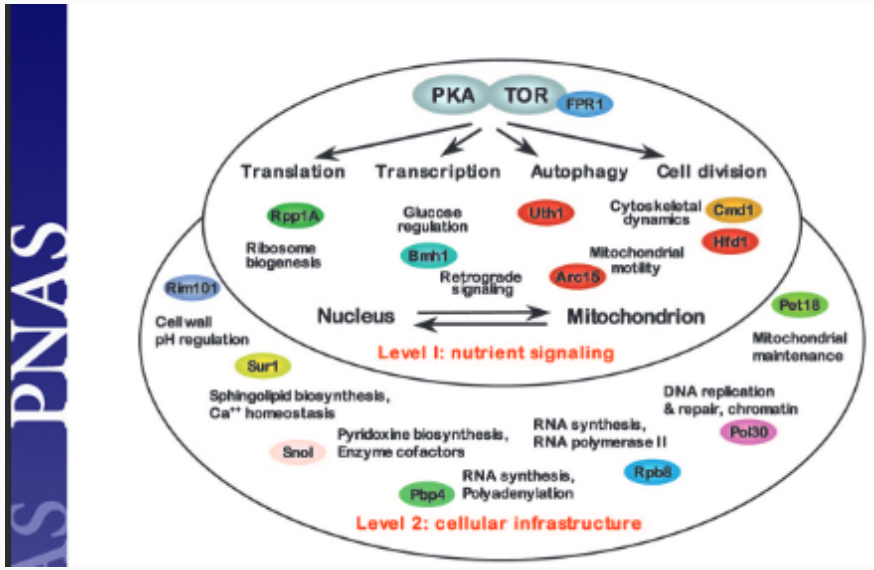
model discovery

learning to learn

Scientific discovery



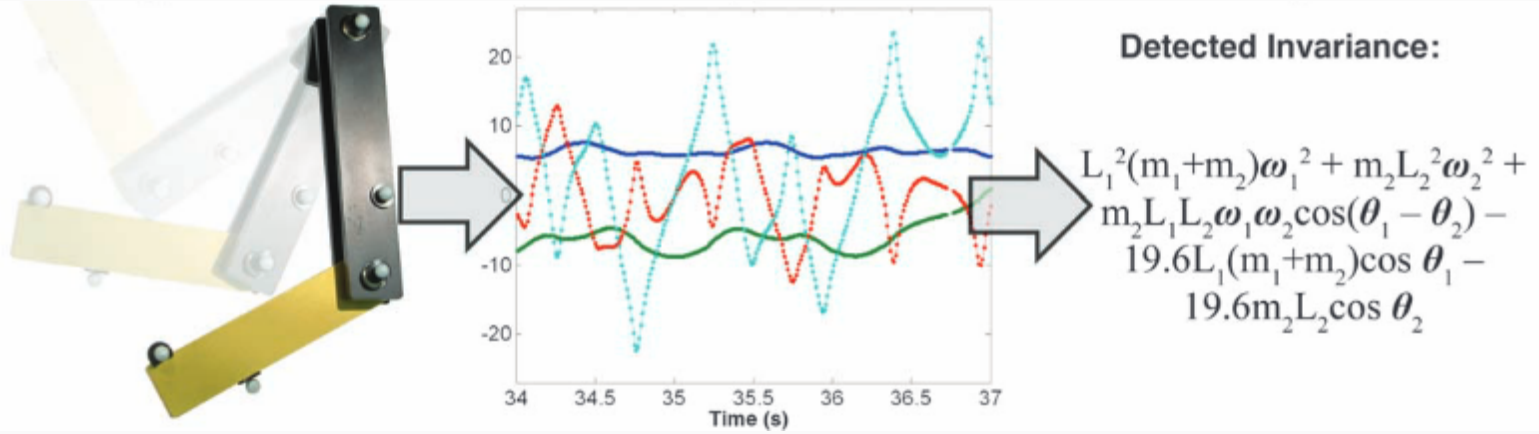
Schmidt & Lipson: "Distilling Free-Form Natural Laws from Experimental Data"



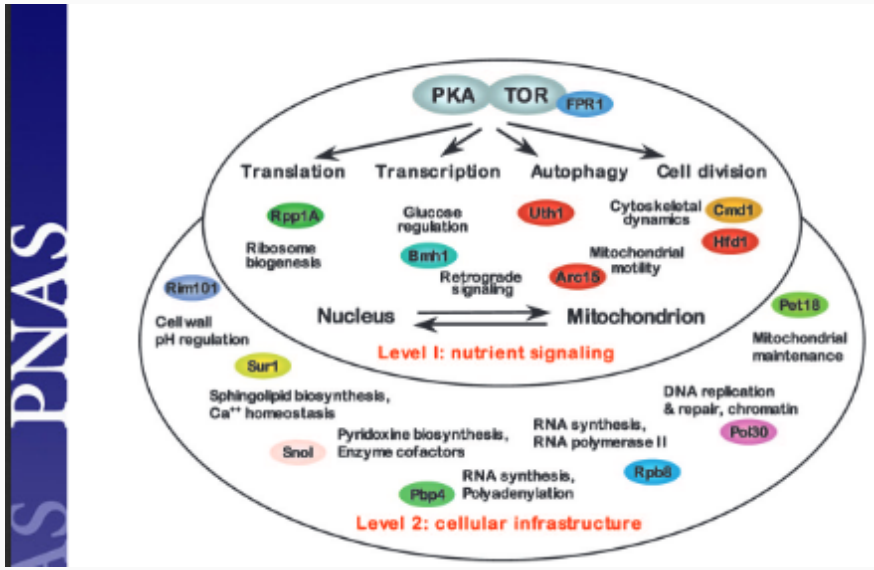
Lezon et al. 2006

inferring genetic interaction networks

Scientific discovery



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



THE APPLICATION OF ALGORITHMIC PROBABILITY TO PROBLEMS IN ARTIFICIAL INTELLIGENCE

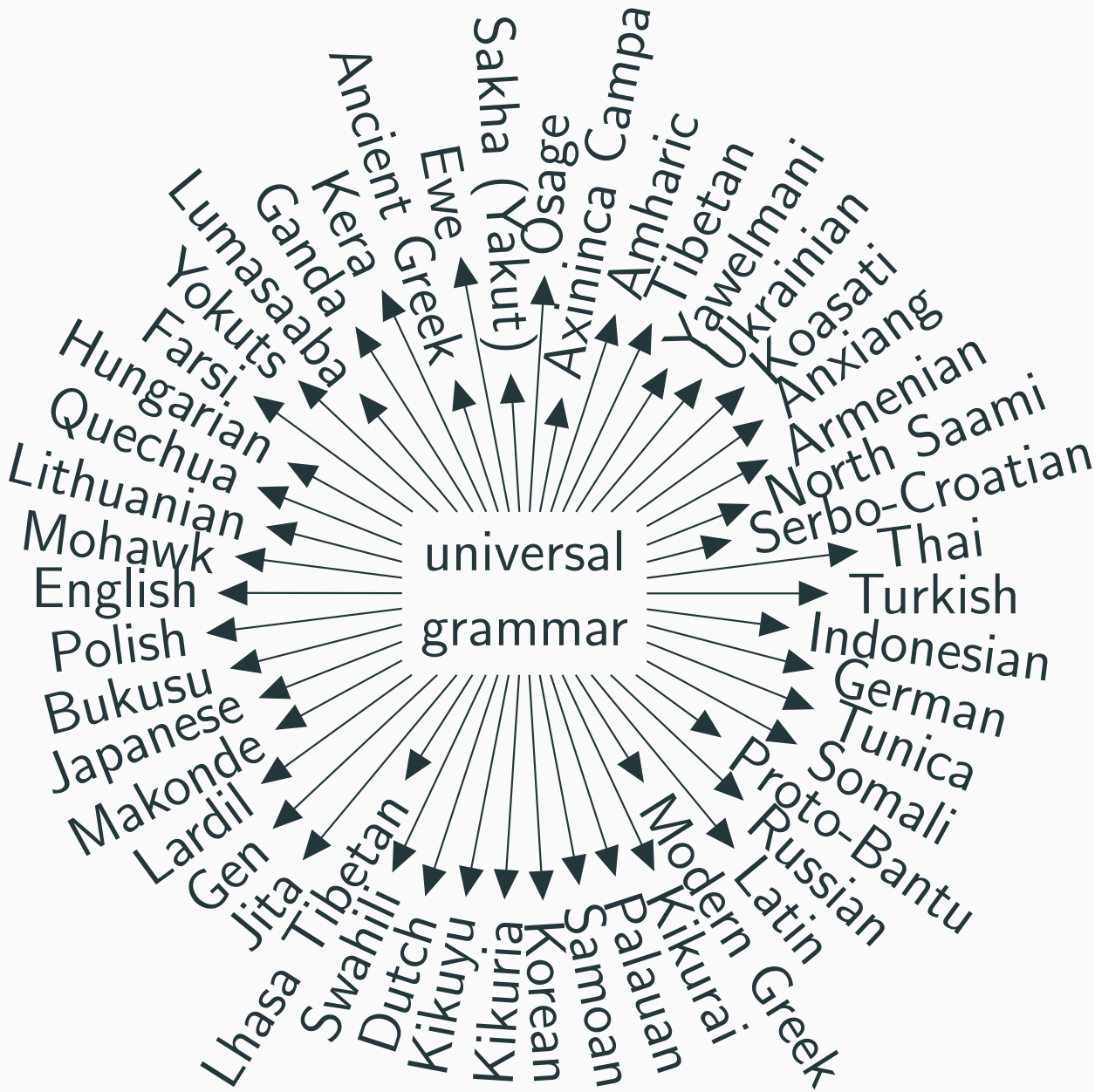
Ray J. Solomonoff 1986

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

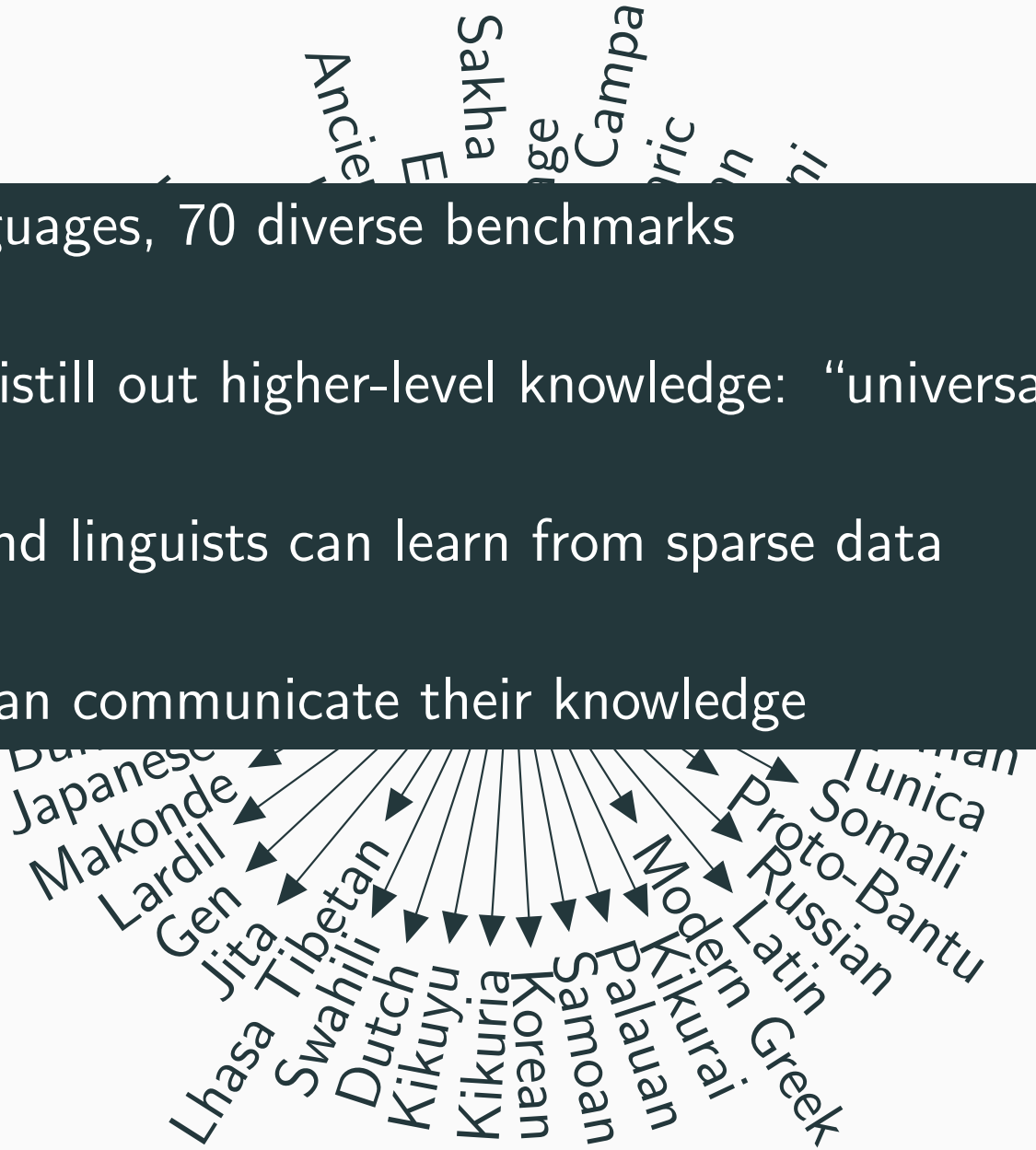
Lezon et al. 2006

inferring genetic interaction networks

Discovering human-understandable models of language



Discovering human-understandable models of language



many languages, 70 diverse benchmarks

linguists distill out higher-level knowledge: "universal grammar"

children and linguists can learn from sparse data

linguists can communicate their knowledge

Few-shot language learning experiment

Mandarin:

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

Few-shot language learning experiment

Mandarin:

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

Few-shot language learning experiment

Mandarin:

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

stem+stem+də

Few-shot language learning experiment

Serbo-Croatian:

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

Few-shot language learning experiment

Serbo-Croatian:

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

Few-shot language learning experiment

Serbo-Croatian:

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

add “a” to stem to make feminine

Few-shot language learning experiment

Serbo-Croatian:

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

add “a” to stem to make feminine

Few-shot language learning experiment

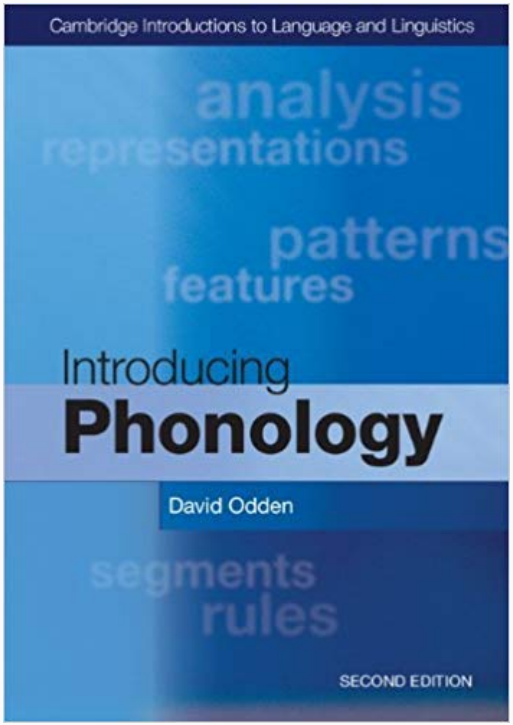
Serbo-Croatian:

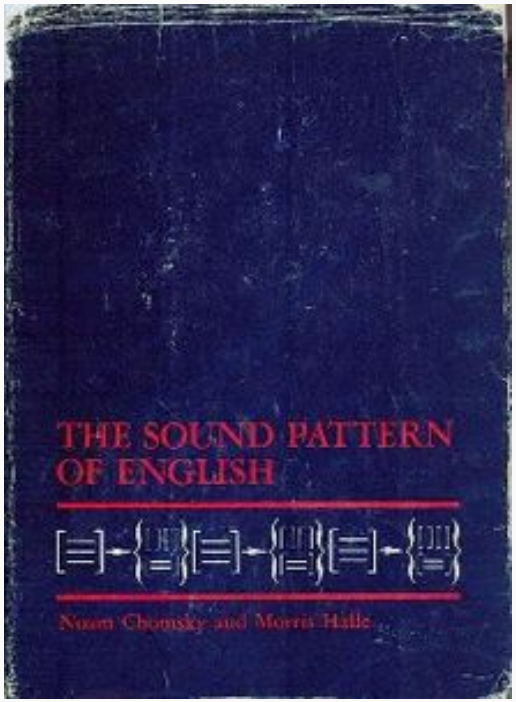
	masculine	feminine	stem (unobserved)
"rich"	bogat	bogata	bogat
"mild"	blag	blaga	blag
"green"	zelen	zelena	zelen
"clear"	**yasan**	yasna	yasn

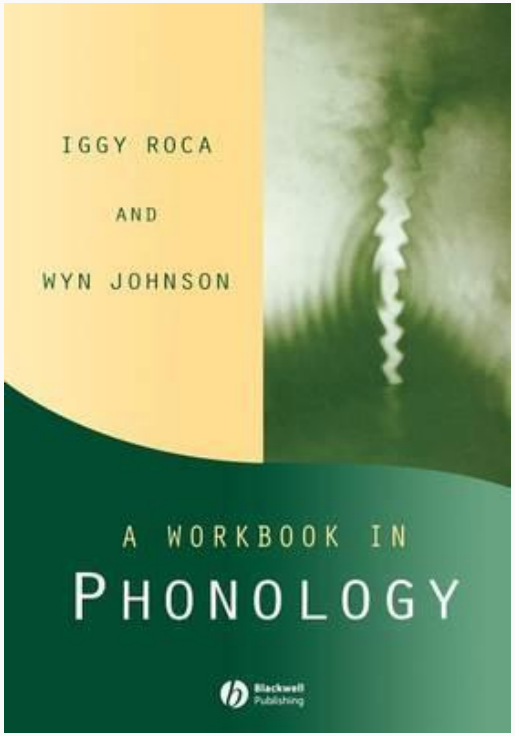
add "a" to stem to make feminine

insert "a" between two word-final consonants

$\varnothing \rightarrow$ a / C_C#


Cambridge Introductions to Language and Linguistics
analysis
representations
patterns
features
Introducing
Phonology
David Odden
segments
rules
SECOND EDITION


THE SOUND PATTERN
OF ENGLISH
Noam Chomsky and Morris Halle


IGGY ROCA
AND
WYN JOHNSON
A WORKBOOK IN
PHONOLOGY
Blackwell
Publishing

10 Sakha (Yakut)

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

Noun	*Plural*	*Associative*	
aya	ayalar	ayalɨɨn	'father'
paarta	paartalar	paartalɨɨn	'school desk'
tɨa	tɨalar	tɨalɨɨn	'forest'
kinige	kinigeler	kinigeliin	'book'
ǰie	ǰieler	ǰieliin	'house'
iye	iyeler	iyeliin	'mother'
kini	kiniler	kiniliin	'3rd person'
bie	bieler	bieliin	'mare'
oyo	oyolor	oyoluun	'child'
χopto	χoptolor	χoptoluun	'gull'
börö	börölör	börölüün	'wolf'
tɨal	tɨallar	tɨalɨɨn	'wind'
ɨal	ɨallar	ɨalɨɨn	'neighbor'
kuul	kuullar	kuulluun	'sack'
at	attar	attɨɨn	'horse'
balɨk	balɨktar	balɨktɨɨn	'fish'
ɨskaap	ɨskaaptar	ɨskaaptɨɨn	'cabinet'
oyus	oyustar	oyustuun	'bull'
kus	kustar	kustuun	'duck'
tünnük	tünnükter	tünnüktüün	'window'
sep	septer	septiin	'tool'
et	etter	ettiin	'meat'
örüs	örüster	örüstüün	'river'
tiis	tiister	tiistiin	'tooth'
soroχ	soroχtor	soroχtuun	'some person'
oχ	oχtor	oχtuun	'arrow'
oloppos	oloppostor	oloppostuun	'chair'
ötöχ	ötöχtör	ötöχtüün	'abandoned farm'
ubay	ubaydar	ubaydɨɨn	'elder brother'
saray	saraydar	saraydɨɨn	'barn'
tɨy	tɨydar	tɨydɨɨn	'foal'
atɨɨr	atɨɨrdar	atɨɨrdɨɨn	'stallion'
oyuur	oyuurdar	oyuurduun	'forest'
üčügey	üčügeyder	üčügeydiin	'good person'
eǰiy	eǰiyder	eǰiydiin	'elder sister'
tomtor	tomtordor	tomtorduun	'knob'
moyotoy	moyotoydor	moyotoyduun	'chipmunk'
kötör	kötördör	kötördüün	'bird'
bölköy	bölköydör	bölköydüün	'islet'
χatɨŋ	χatɨŋnar	χatɨŋnɨɨn	'birch'
aan	aannar	aannɨɨn	'door'
tɨŋ	tɨŋner	tɨŋnɨɨn	'squirrel'
sordoŋ	sordoŋnor	sordoŋnuun	'pike'
olom	olomnor	olomnuun	'ford'
oron	oronnor	oronnuun	'bed'
bödöŋ	bödöŋnör	bödöŋnüün	'strong one'

Noun	*Partitive*	*Comparative*	*Ablative*	
aya	ayata	ayataayar	ayattan	'father'
paarta	paartata	paartataayar	paartattan	'school desk'
tɨa	tɨata	tɨataayar	tɨattan	'forest'
kinige	kinigete	kinigeteeyer	kinigetten	'book'
ǰie	ǰiete	ǰieteeyer	ǰietten	'house'
iye	iyete	iyeteeyer	iyetten	'mother'
kini	kinite	kiniteeyer	kinitten	'3rd person'
bie	biete	bieteeyer	bietten	'mare'
oyo	oyoto	oyotooyor	oyotton	'child'
χopto	χoptoto	χoptotooyor	χoptotton	'gull'
börö	börötö	börötööyör	böröttön	'wolf'
tɨal	tɨalla	tɨallaayar	tɨaltan	'wind'
ɨal	ɨalla	ɨallaayar	ɨaltan	'neighbor'
kuul	kuulla	kuullaayar	kuultan	'sack'
moχsoyol	moχsoyollo	moχsoyollooyor	moχsoyolton	'falcon'
at	atta	attaayar	attan	'horse'
balɨk	balɨkta	balɨktaayar	balɨktan	'fish'
ɨskaap	ɨskaapta	ɨskaaptaayar	ɨskaaptan	'cabinet'
oyus	oyusta	oyustaayar	oyustan	'bull'
kus	kusta	kustaayar	kustan	'duck'
tünnük	tünnükte	tünnükteeyer	tünnükten	'window'

Turkic Sakha (Yakut)

observed data

	Singular	Plural
Bed	oron	oronnor
Mare	bie	bieler
Cabinet	ɨskaap	ɨskaaptar

138 total examples

Turkic Sakha (Yakut)

grammar (unobserved)

SINGULAR→stem
PLURAL→stem+lar

observed data

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	ɨskaap	ɨskaaptar

138 total examples

Turkic Sakha (Yakut)

grammar (unobserved)

SINGULAR→stem
PLURAL→stem+lar

r_1: l → d / [-lateral -tense]_
"l" becomes "d" next to "r", "t", but not "l"

r_2: C → [-voice] / [-voice]_
do not voice next to voiceless

r_3: V → [+rounded] / [+rounded] [-low]$_0$_
r_4: [+continuant -high] → [-rounded] / u C$_0$_
"harmonize" round vowels like "u", "o"

r_5: V → [-back -low] / [-back +vowel] [ ]$_0$_
"harmonize" vowels to be not at back of mouth

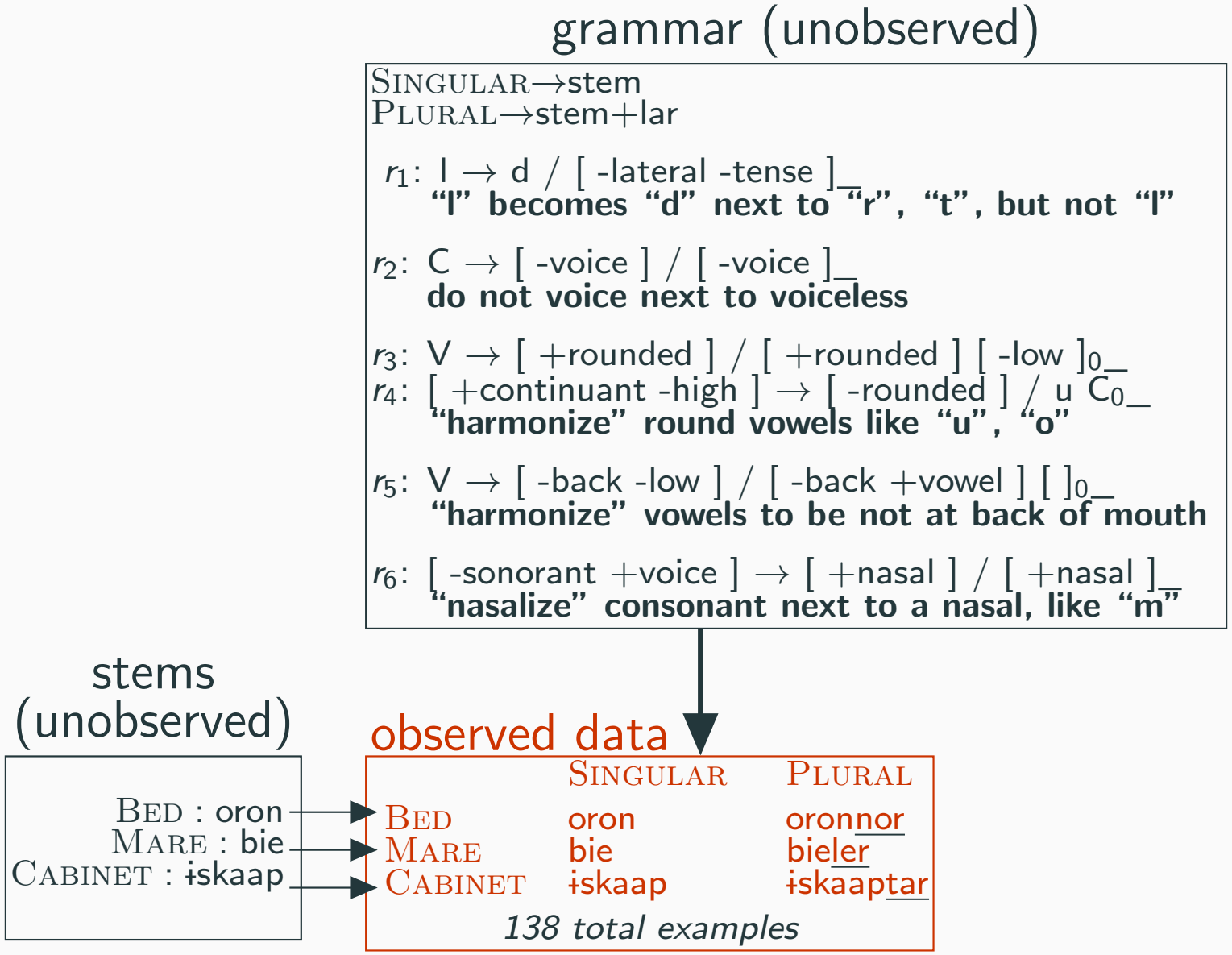
r_6: [-sonorant +voice] → [+nasal] / [+nasal]_
"nasalize" consonant next to a nasal, like "m"

observed data

	SINGULAR	PLURAL
BED	oron	oronnor
MARE	bie	bieler
CABINET	ɨskaap	ɨskaaptar

138 total examples

Turkic Sakha (Yakut)



Turkic Sakha (Yakut)

grammar (unobserved)

SINGULAR→stem
PLURAL→stem+lar

r_1: l → d / [-lateral -tense]_
"l" becomes "d" next to "r", "t", but not "l"

r_2: C → [-voice] / [-voice]_
do not voice next to voiceless

r_3: V → [+rounded] / [+rounded] [-low]$_0$_
r_4: [+continuant -high] → [-rounded] / u C$_0$_
"harmonize" round vowels like "u", "o"

r_5: V → [-back -low] / [-back +vowel] [ ]$_0$_
"harmonize" vowels to be not at back of mouth

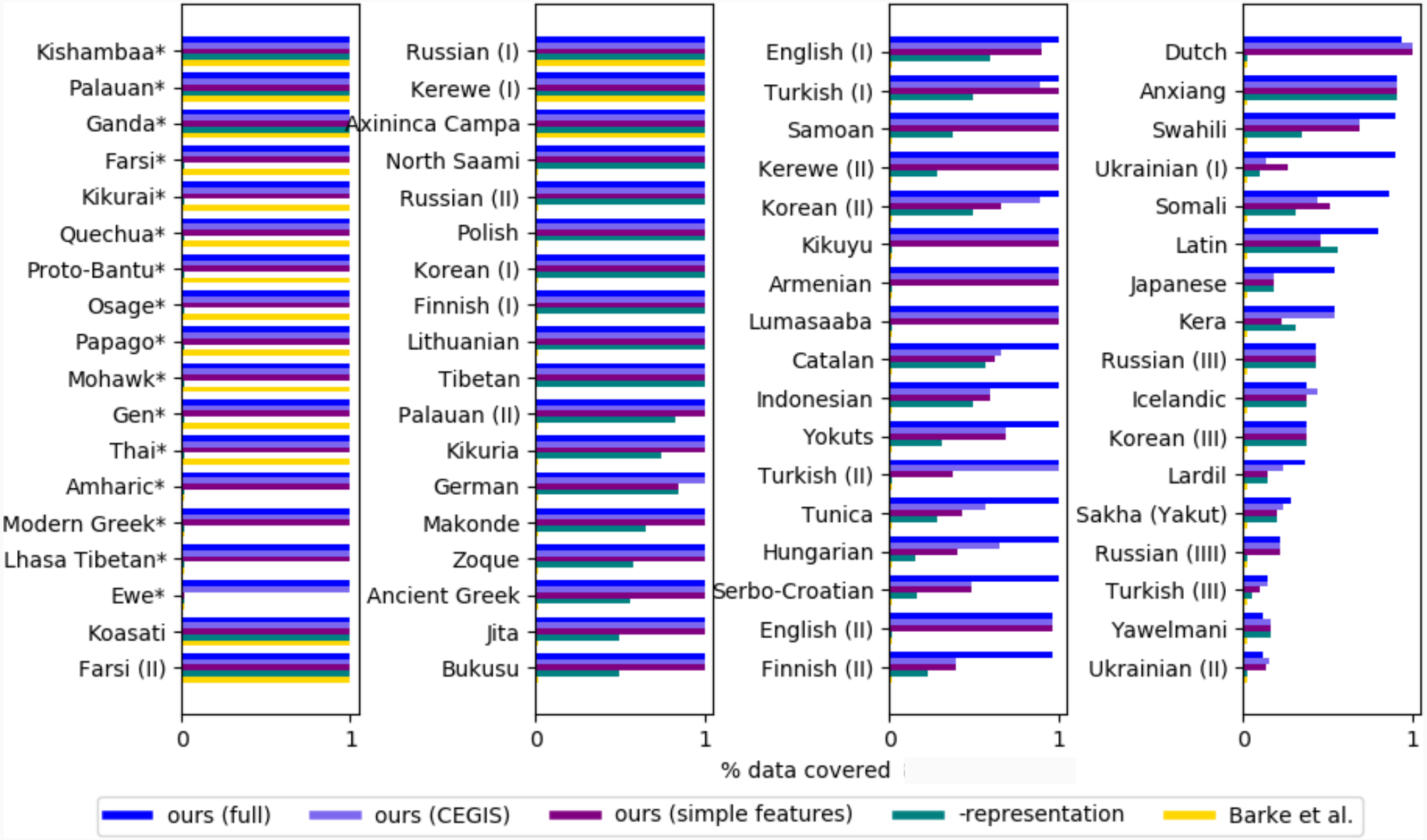
r_6: [-sonorant +voice] → [+nasal] / [+nasal]_
"nasalize" consonant next to a nasal, like "m"

stems (unobserved)

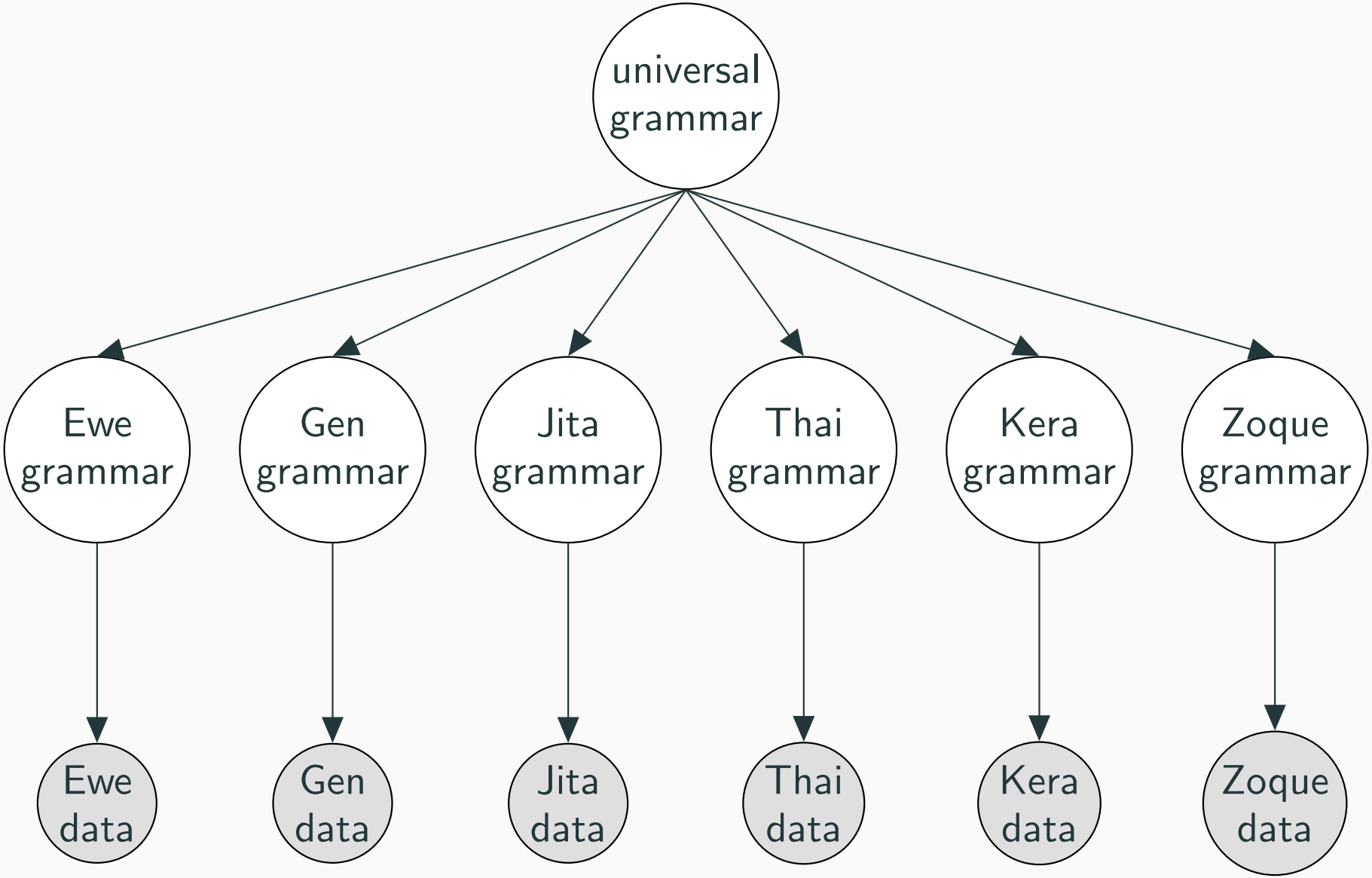
CABINET : ɨskaap
BED : oron
MARE : bie
RIVER : örus

observed data

CABINETS→ɨskaap+lar→ɨskaaplar$\xrightarrow{r_1}$ɨskaapdar$\xrightarrow{r_2}$ɨskaaptar
BEDS→oron+lar→oronlar$\xrightarrow{r_1}$orondar$\xrightarrow{r_3}$orondor$\xrightarrow{r_6}$oronnor
MARES→bie+lar→bielar$\xrightarrow{r_5}$bieler
RIVER (ASSOC)→örus+lɨɨn→örusl ɨɨn$\xrightarrow{r_1}$örusdɨɨn$\xrightarrow{r_2}$
örustɨɨn$\xrightarrow{r_3}$örustuun$\xrightarrow{r_5}$[örüstüün]


Kishambaa*
Palauan*
Ganda*
Farsi*
Kikurai*
Quechua*
Proto-Bantu*
Osage*
Papago*
Mohawk*
Gen*
Thai*
Amharic*
Modern Greek*
Lhasa Tibetan*
Ewe*
Koasati
Farsi (II)
Russian (I)
Kerewe (I)
Axininca Campa
North Saami
Russian (II)
Polish
Korean (I)
Finnish (I)
Lithuanian
Tibetan
Palauan (II)
Kikuria
German
Makonde
Zoque
Ancient Greek
Jita
Bukusu
English (I)
Turkish (I)
Samoan
Kerewe (II)
Korean (II)
Kikuyu
Armenian
Lumasaaba
Catalan
Indonesian
Yokuts
Turkish (II)
Tunica
Hungarian
Serbo-Croatian
English (II)
Finnish (II)
Dutch
Anxiang
Swahili
Ukrainian (I)
Somali
Latin
Japanese
Kera
Russian (III)
Icelandic
Korean (III)
Lardil
Sakha (Yakut)
Russian (IIII)
Turkish (III)
Yawelmani
Ukrainian (II)
0
1
% data covered
ours (full)
ours (CEGIS)
ours (simple features)
-representation
Barke et al.

Distilling higher-level knowledge



Lessons

Higher-level knowledge matters ("universal grammar"). Get the basics of the representation correct

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

Program Induction and perception

model discovery

learning to learn

Learning to write code

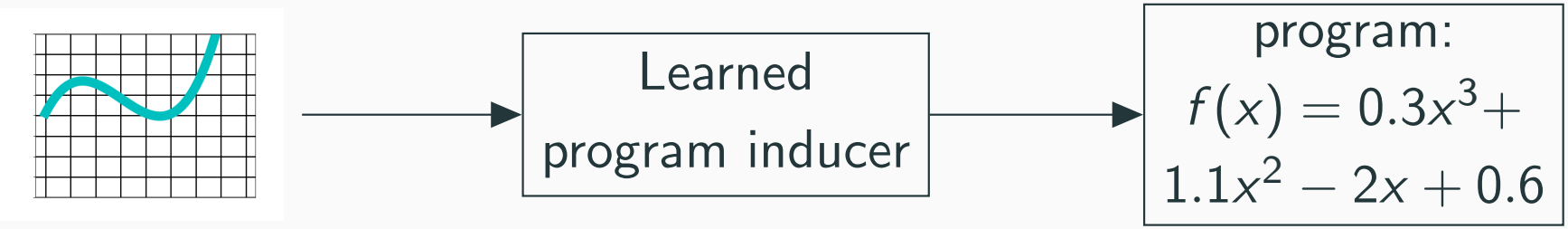
Goal: acquire domain-specific knowledge needed to induce a class of programs

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

Learning to write code

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

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



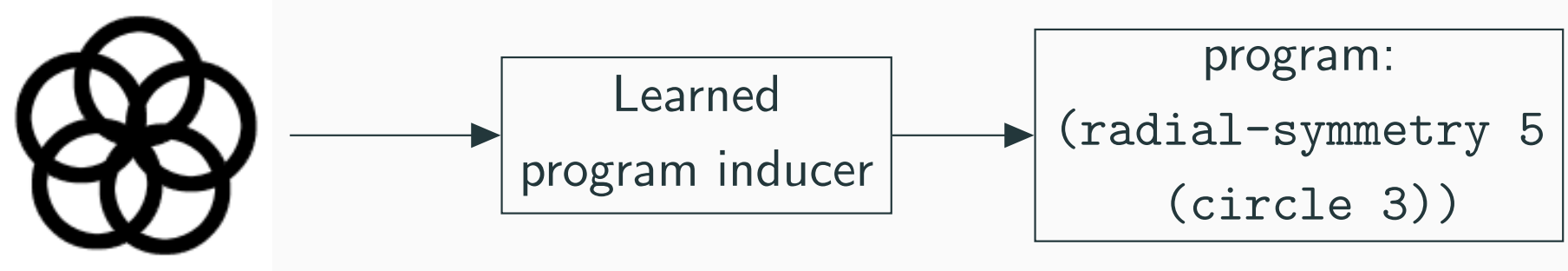
Concepts: x^3, $\alpha x + \beta$, etc

Inference strategy: neurosymbolic search for programs

Learning to write code

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

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



Concepts: `circle`, `radial-symmetry`, etc

Inference strategy: neurosymbolic search for programs

Library learning

Initial Primitives

⋮

`map`

`fold`

`if`

`cons`

`>`

⋮

Sample Problem: `sort list`

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

Library learning

Initial Primitives

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

Sample Problem: `sort list`

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

Library learning

Initial Primitives

```
⋮
map
fold
if
cons
>
⋮
```

Sample Problem: `sort list`

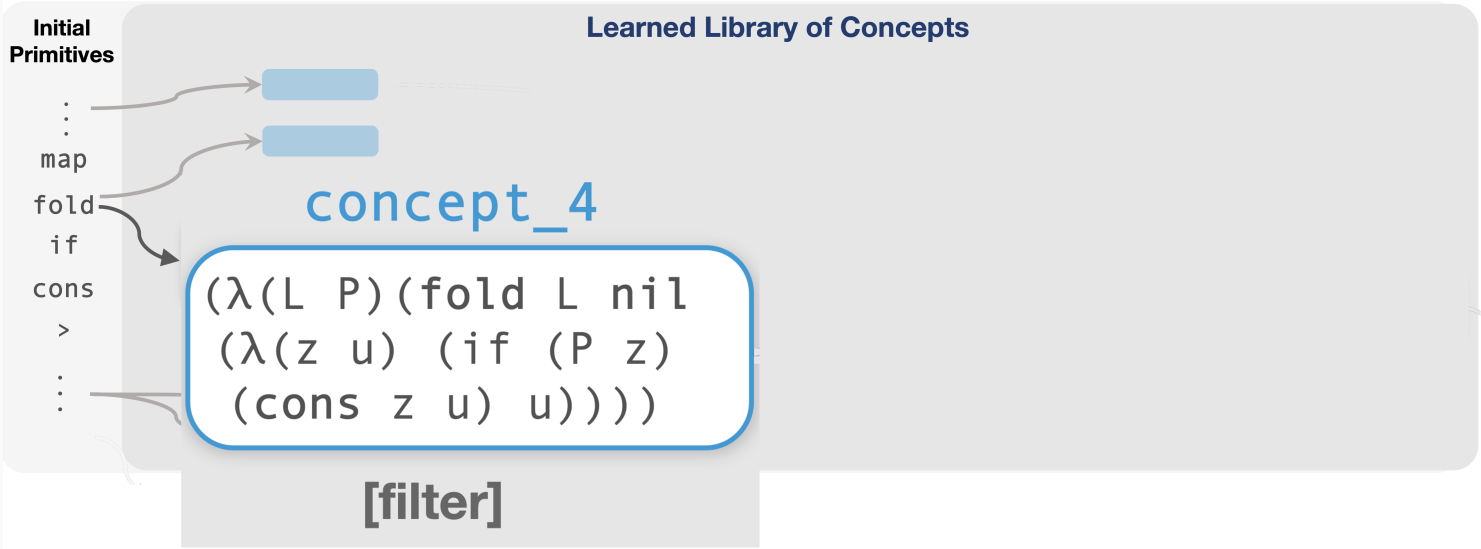
[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
...

Library learning

Sample Problem: `sort list`

[9 2 7 1] → [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`

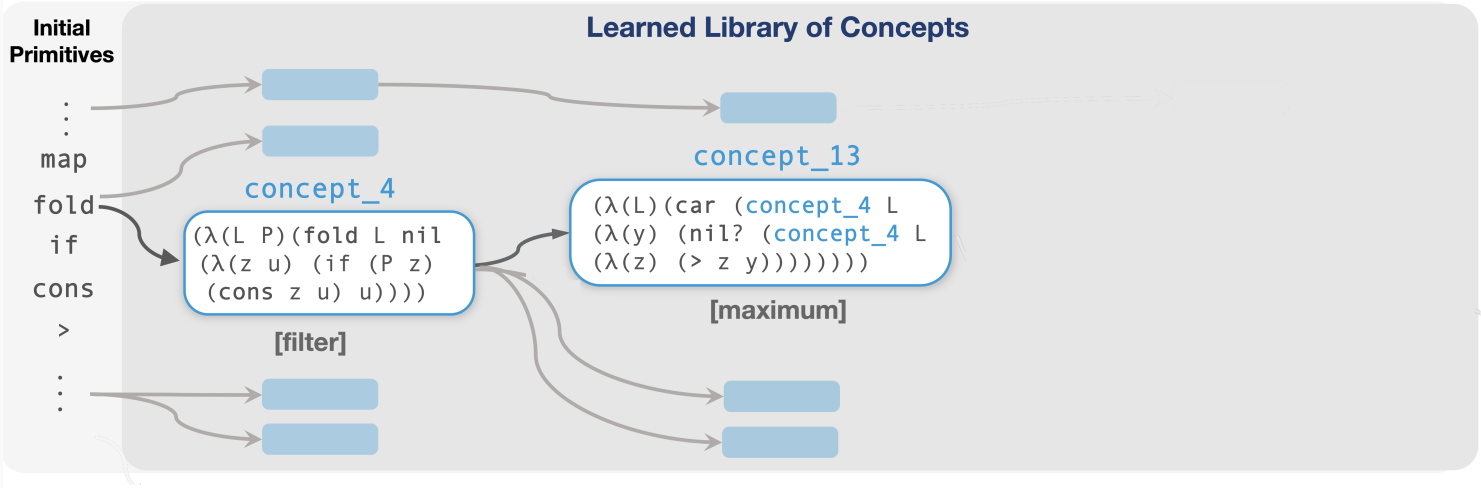
[9 2 7 1] → [1 2 7 9]
[3 8 9 4 2] → [2 3 4 8 9]
[6 2 2 3 8 5] → [2 2 3 5 6 8]
...

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]
...

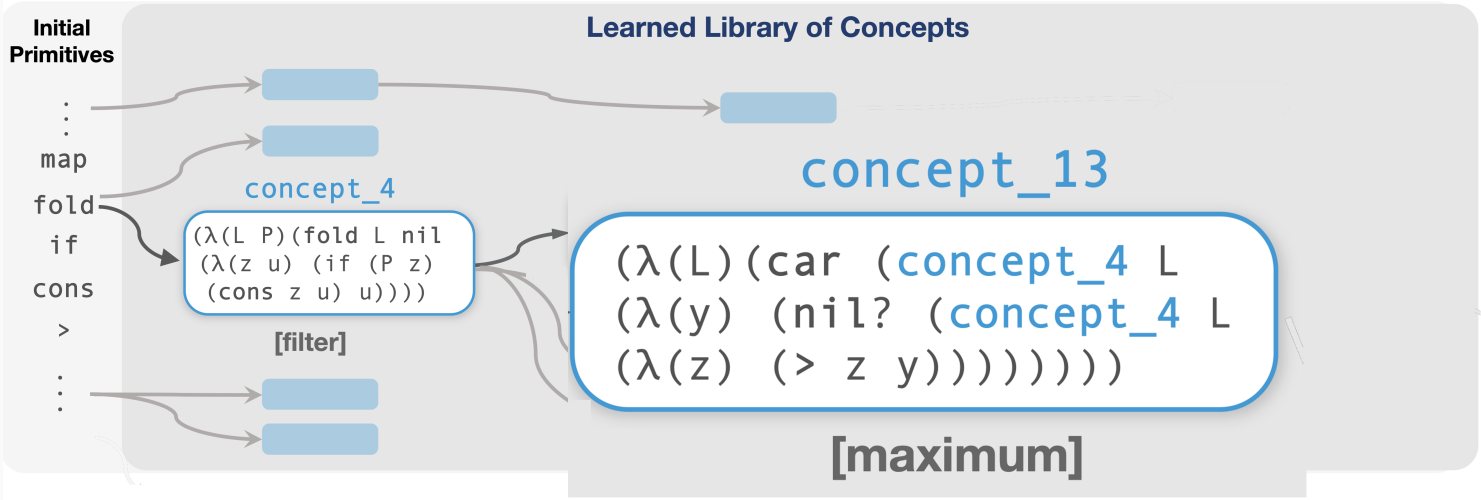
Library learning



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]
...

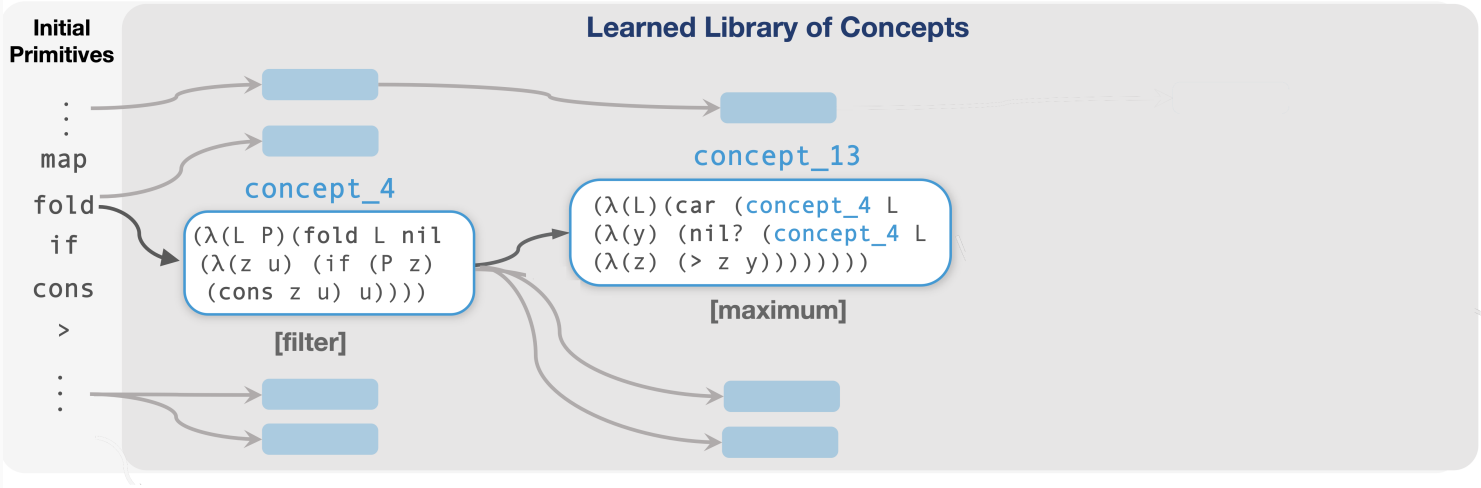
Library learning



Sample Problem: `sort list`

[9 2 7 1] → [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]
...

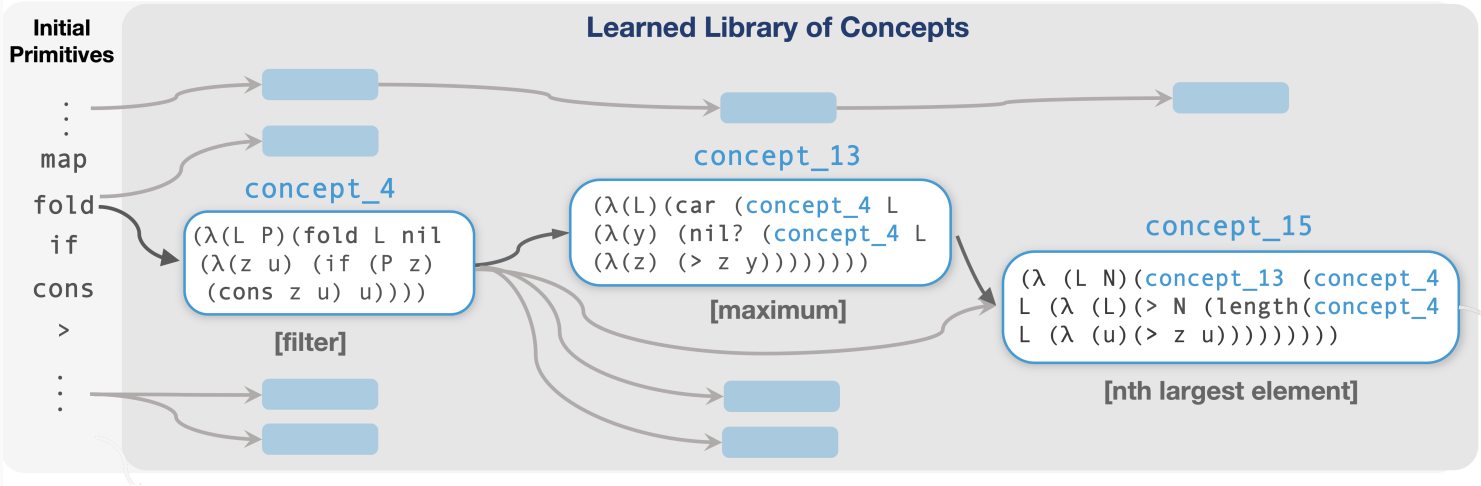
Library learning



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]
...

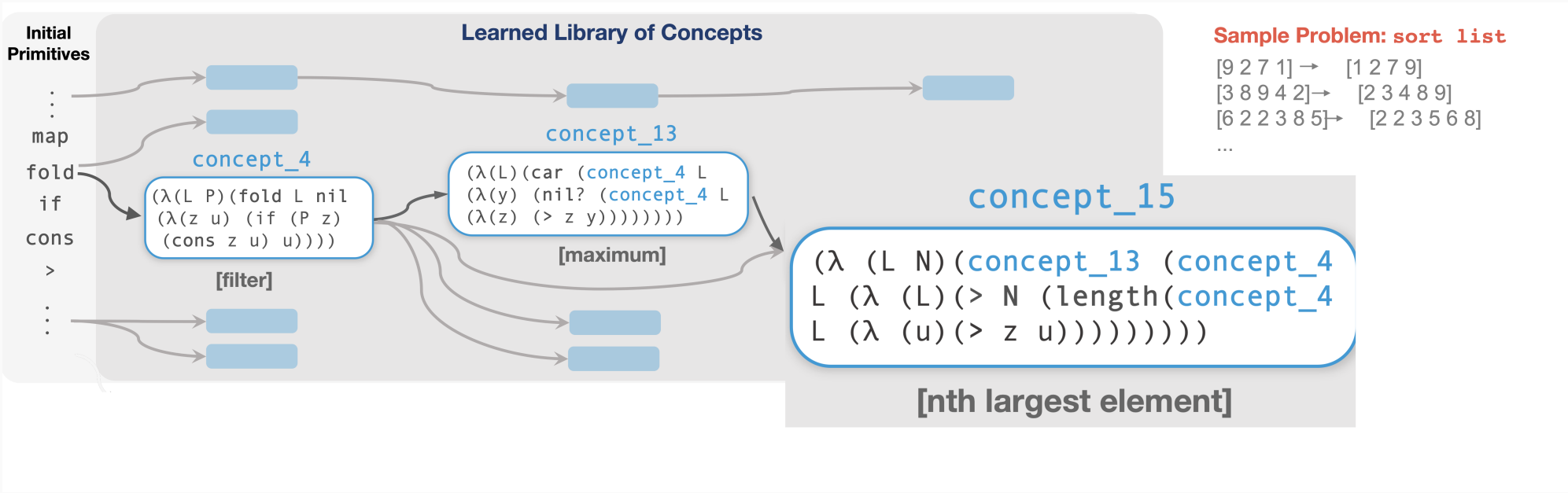
Library learning



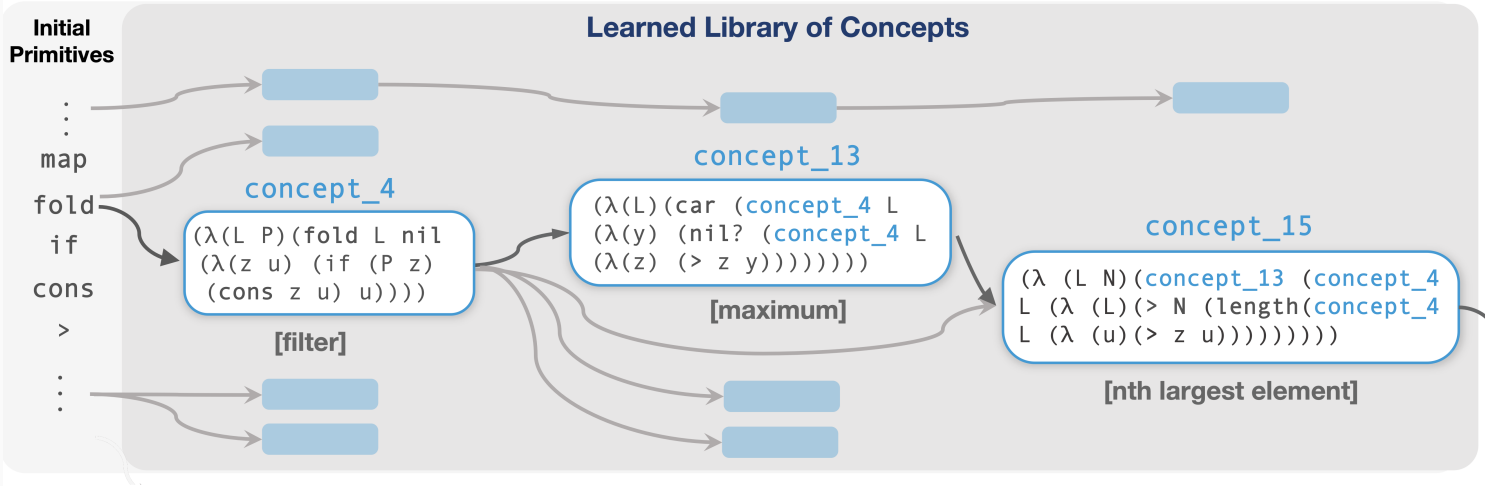
Sample Problem: `sort list`

[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



Library learning



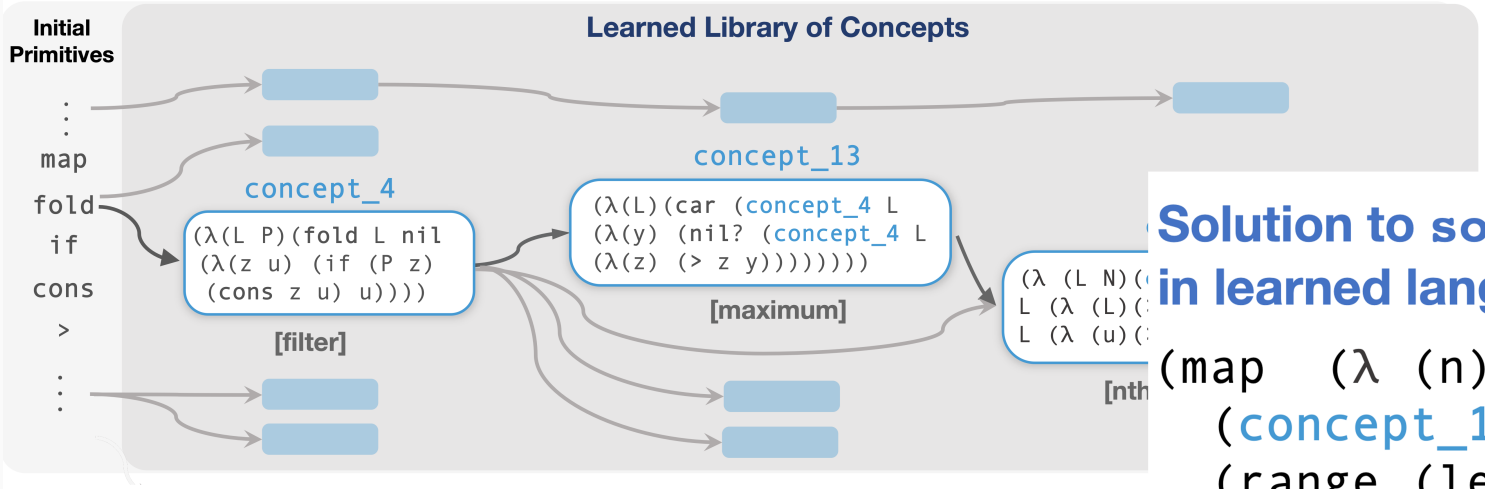
Sample Problem: `sort list`

[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



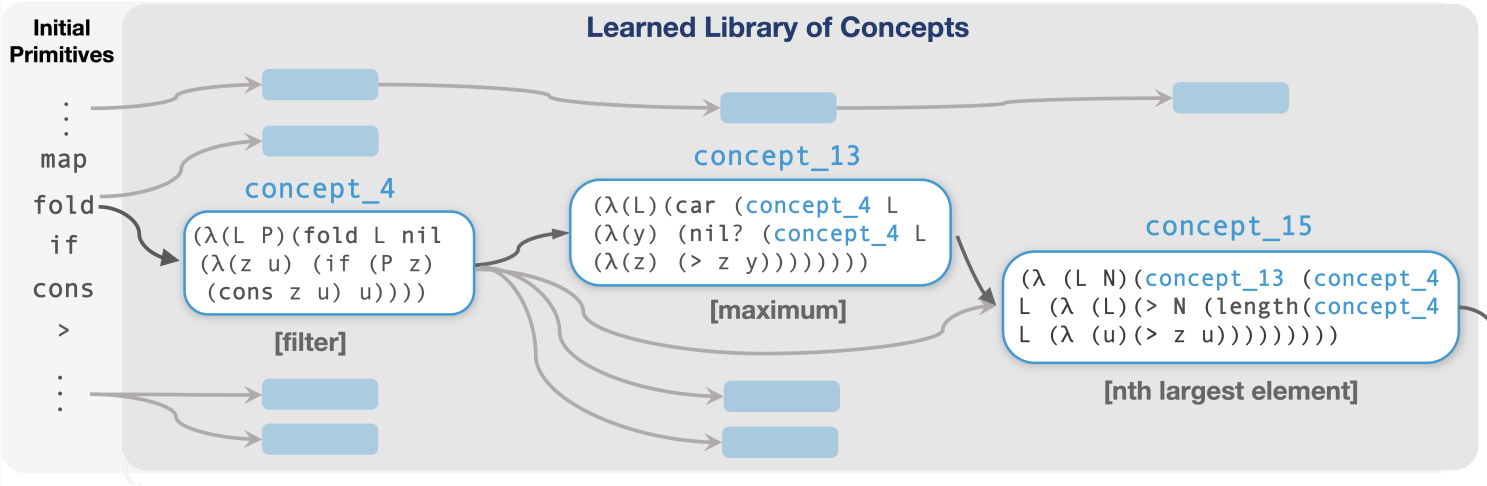
Sample Problem: `sort list`

[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



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]
...

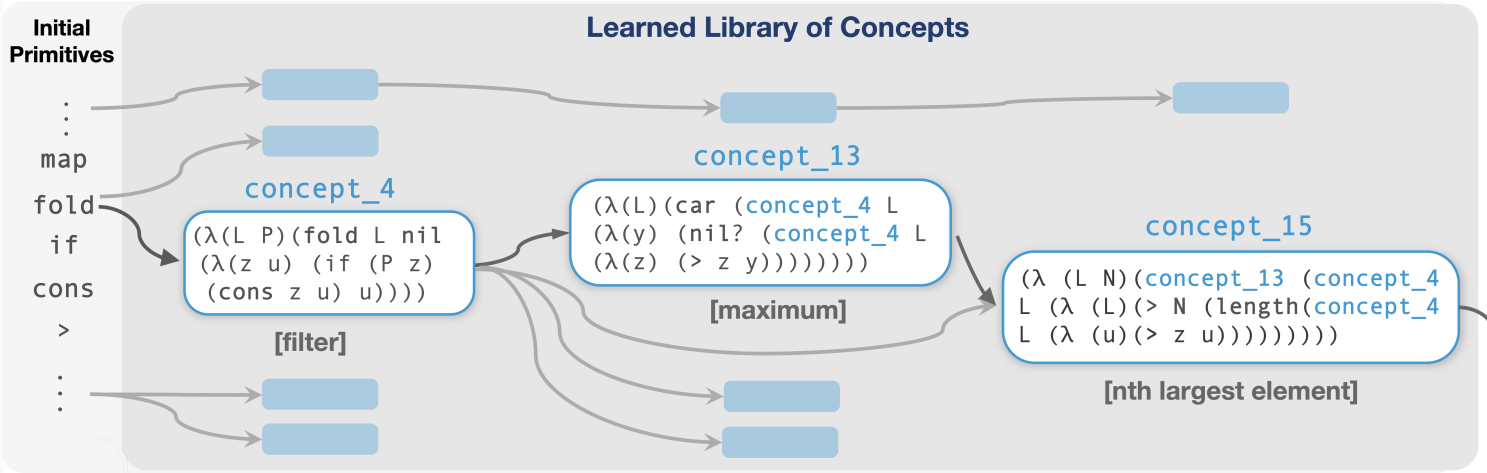
Solution to `sort list` discovered in learned language:

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

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



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)))
```

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 $\leq$ 10min. Brute-force search without learned library would take $\approx 10^{78}$ years

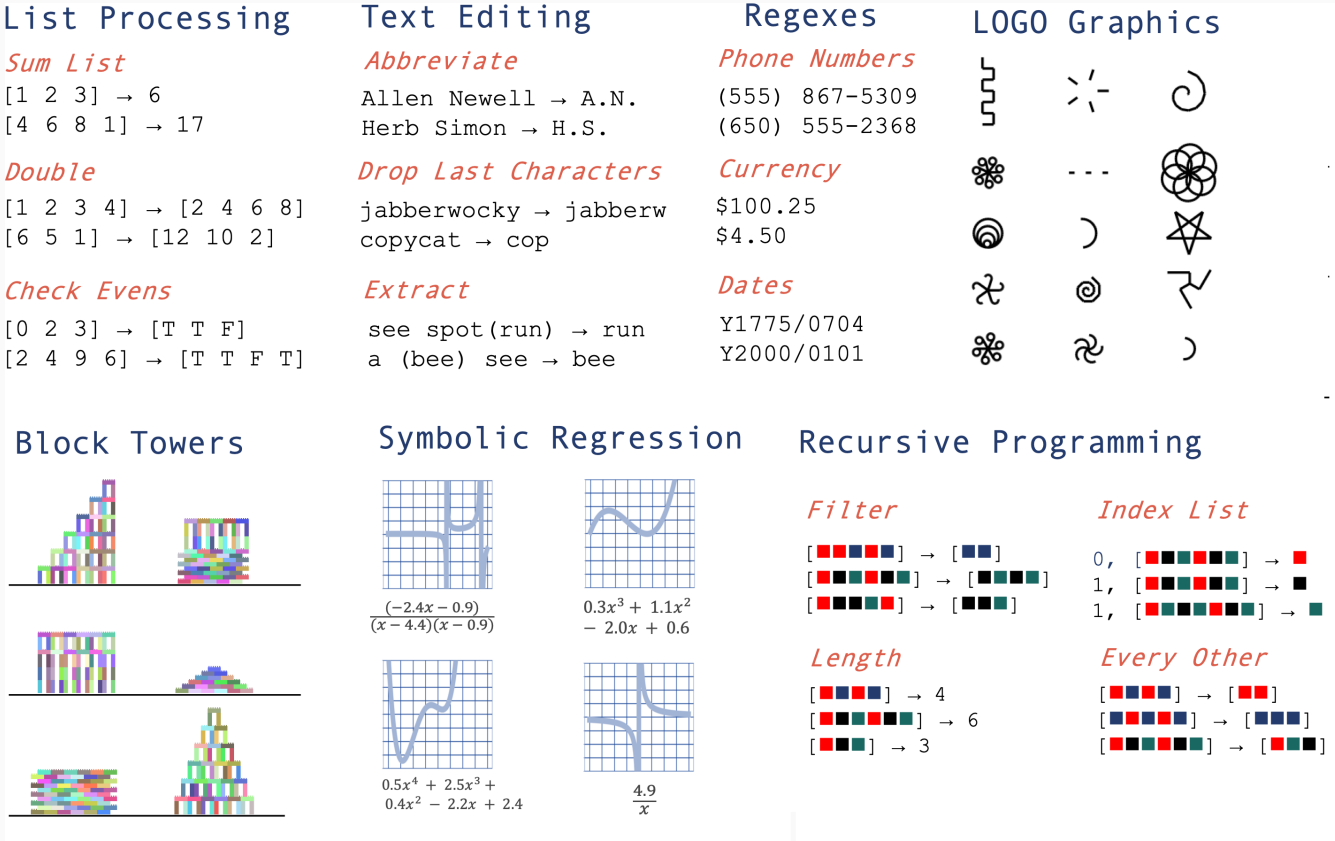
DreamCoder

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

cf. Helmholtz machine, wake/sleep neural network training

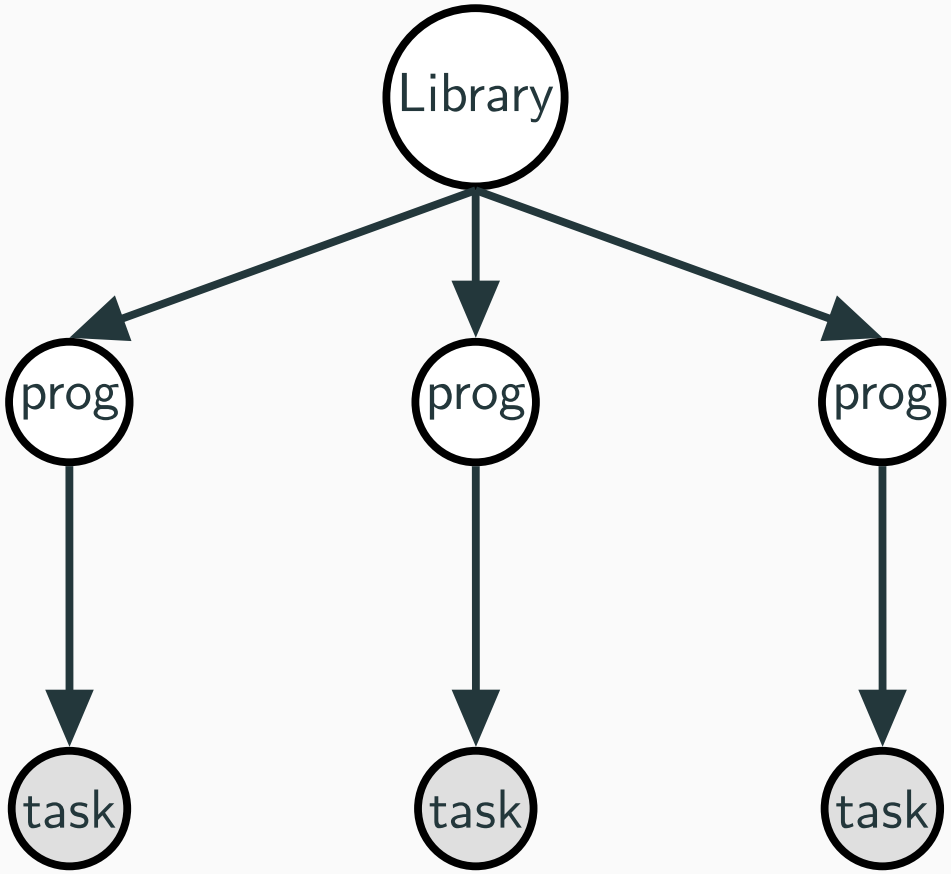
DreamCoder

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



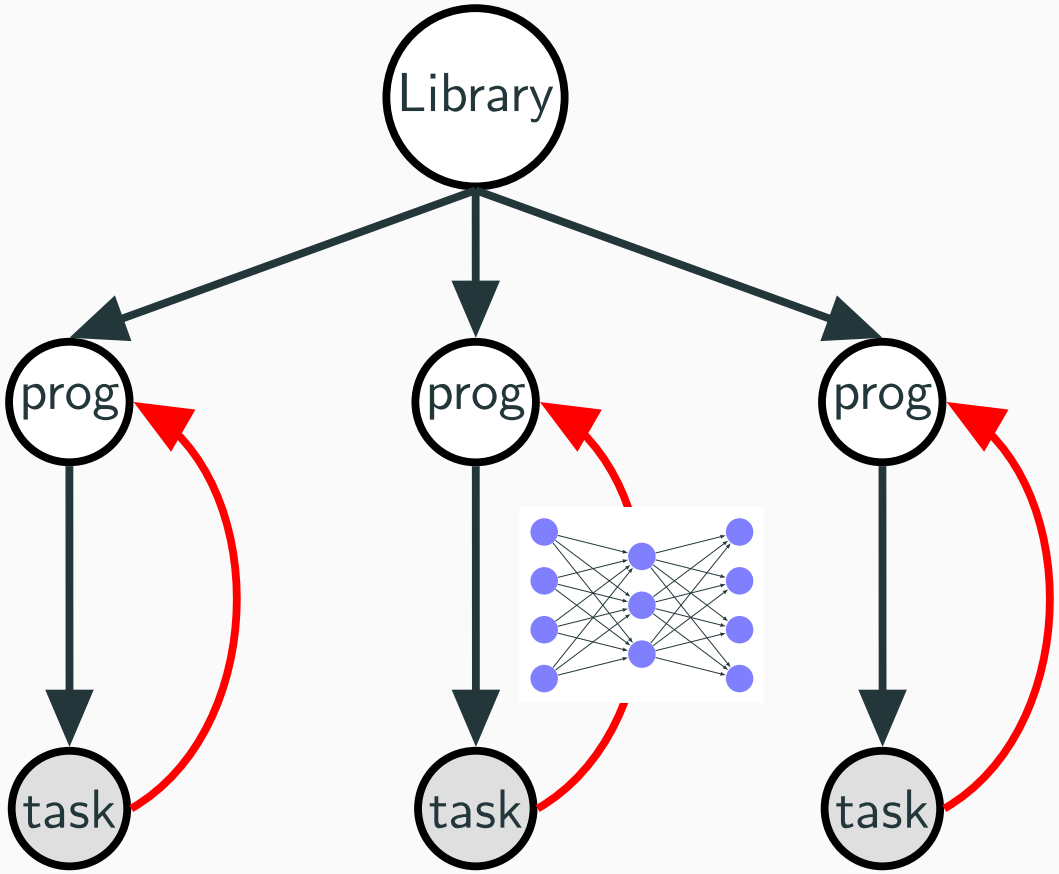
cf. Helmholtz machine, wake/sleep neural network training

Library learning as Bayesian inference



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

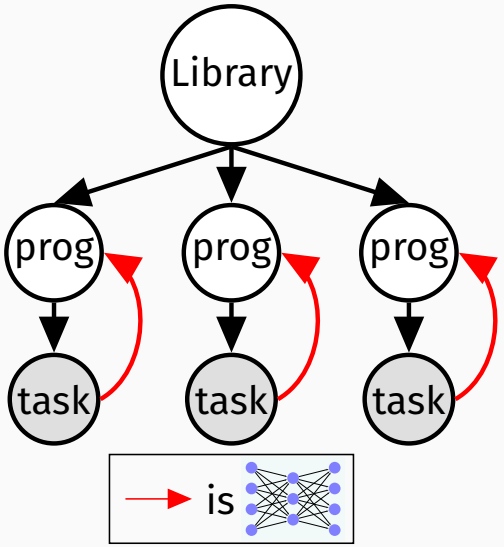
Library learning as neurally-guided Bayesian inference



library learning via program analysis +

new neural inference network for program synthesis +

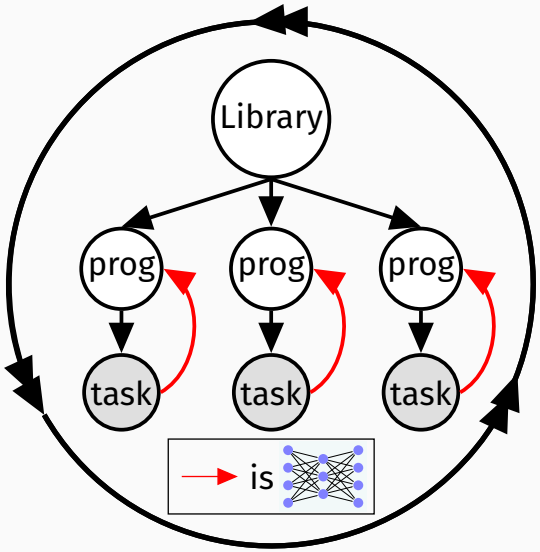
better program representation (Lisp+polymorphic types [Milner 1978])


Library
prog
prog
prog
task
task
task
is

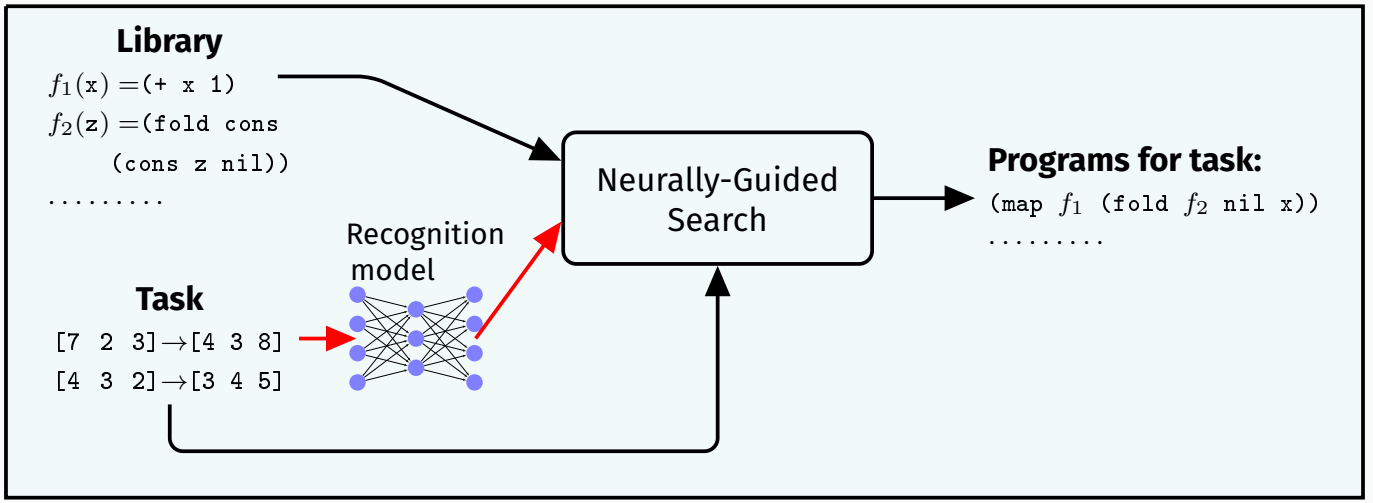
Wake

Sleep: Abstraction

Sleep: Dreaming

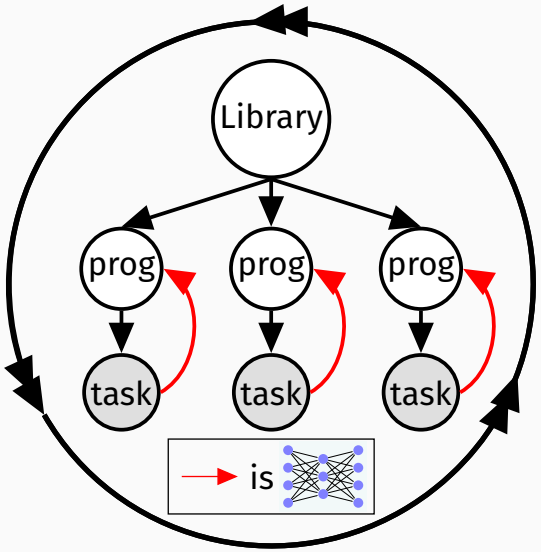


WAKE

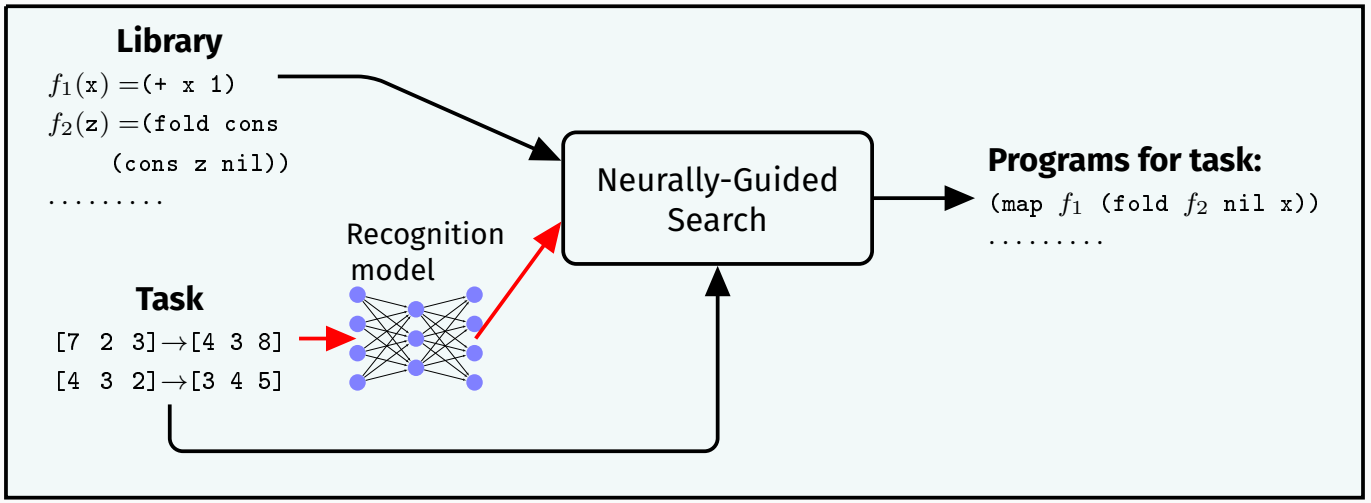


SLEEP: ABSTRACTION

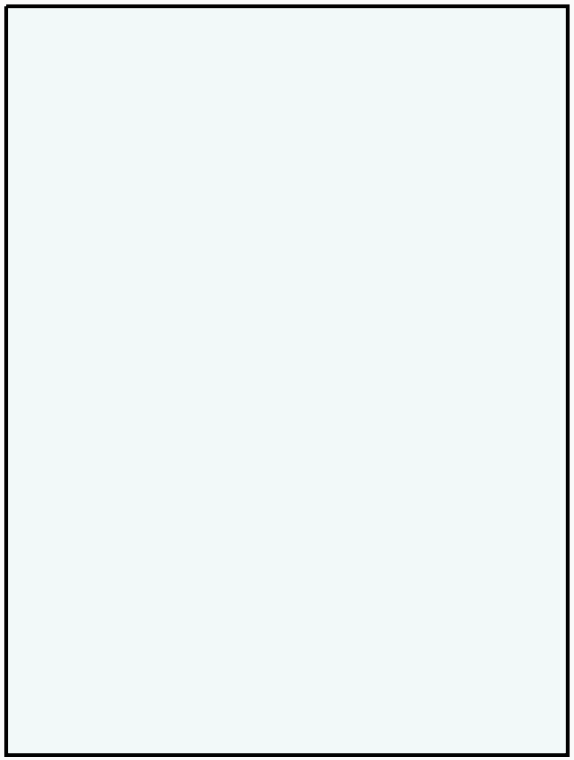
SLEEP: DREAMING

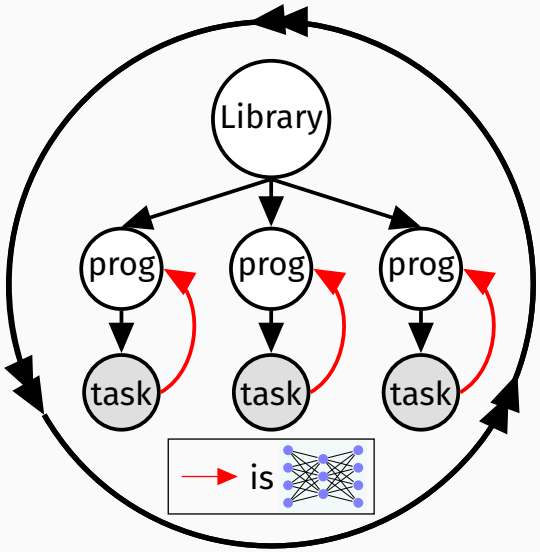


WAKE

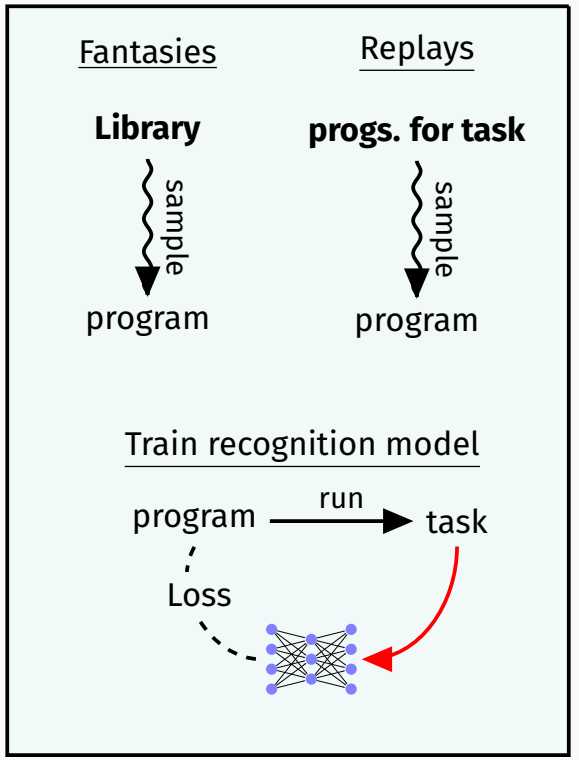


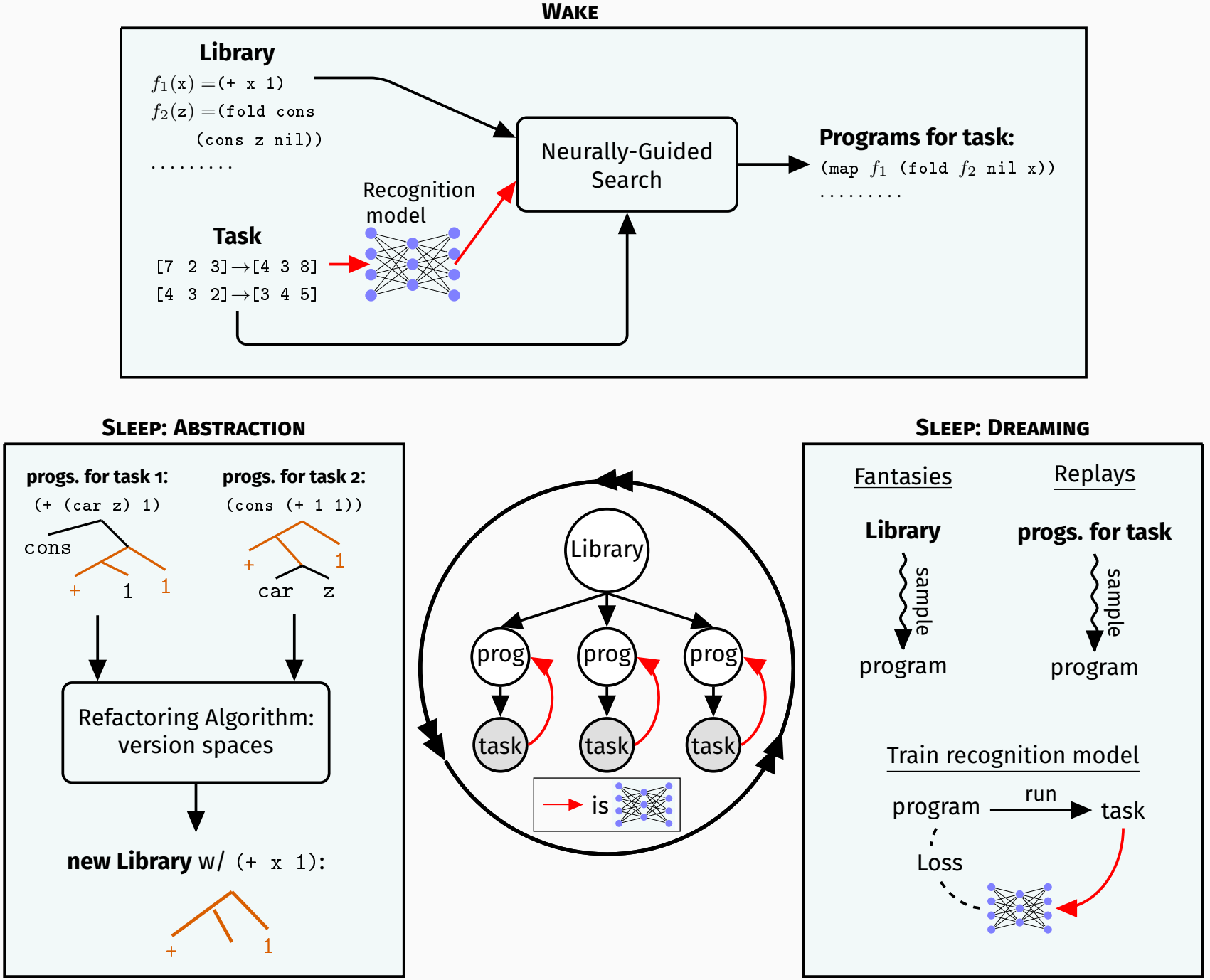
SLEEP: ABSTRACTION





SLEEP: DREAMING




WAKE
Library
f_1(x) =(+ x 1)
f_2(z) =(fold cons
(cons z nil))
.....
Task
[7 2 3]→[4 3 8]
[4 3 2]→[3 4 5]
Recognition model
Neurally-Guided Search
Programs for task:
(map f_1 (fold f_2 nil x))
.....
SLEEP: ABSTRACTION
progs. for task 1:
(+ (car z) 1)
cons
+
1
1
progs. for task 2:
(cons (+ 1 1))
+
1
car
z
Refactoring Algorithm: version spaces
new Library w/ (+ x 1):
+
1
Library
prog
prog
prog
task
task
task
is
SLEEP: DREAMING
Fantasies
Replays
Library
progs. for task
sample
sample
program
program
Train recognition model
program
run
task
Loss

Abstraction Sleep: Growing the library via refactoring

$5 + 5$

Abstraction Sleep: Growing the library via refactoring

$5 + 5$

```
(+ 5 5)
```

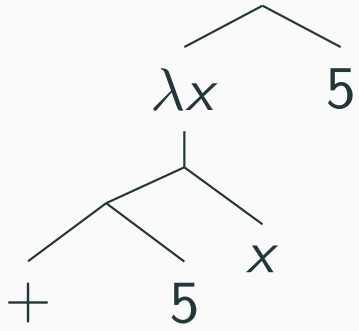
Abstraction Sleep: Growing the library via refactoring

$5 + 5$

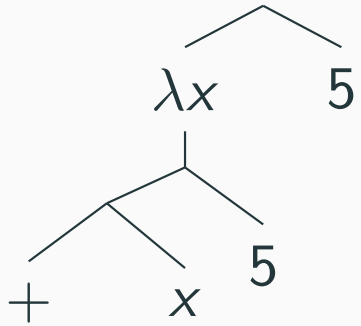
```
(+ 5 5)
```

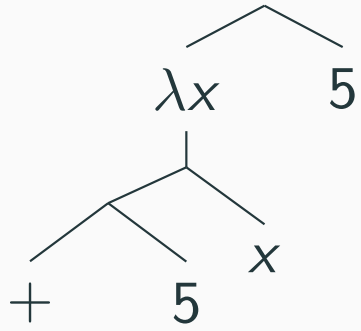
Abstraction Sleep: Growing the library via refactoring

Abstraction Sleep: Growing the library via refactoring

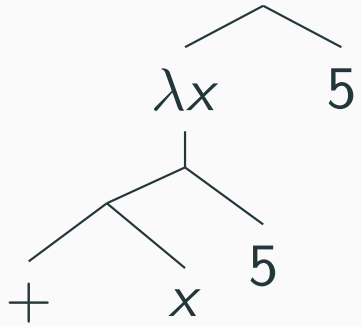


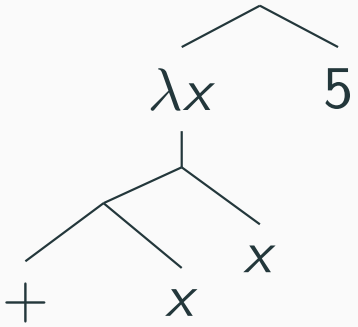
Abstraction Sleep: Growing the library via refactoring

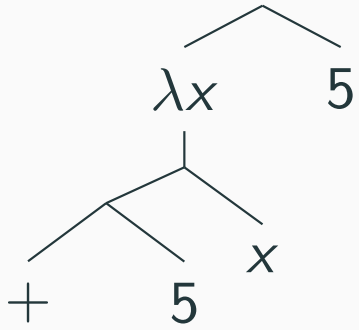




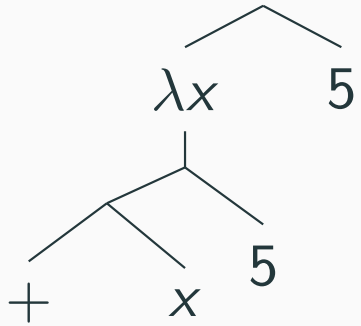
Abstraction Sleep: Growing the library via refactoring

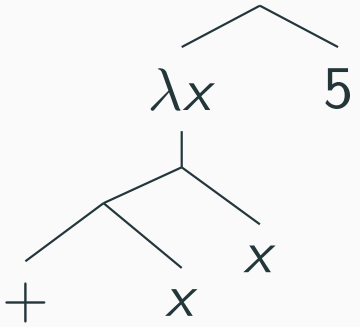


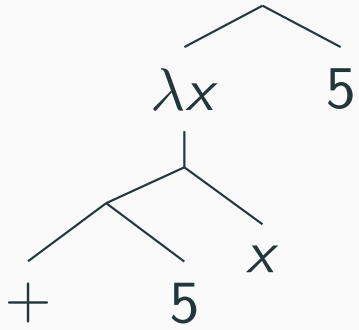


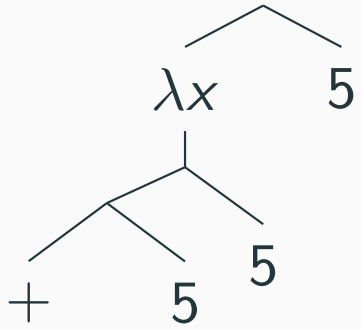


Abstraction Sleep: Growing the library via refactoring

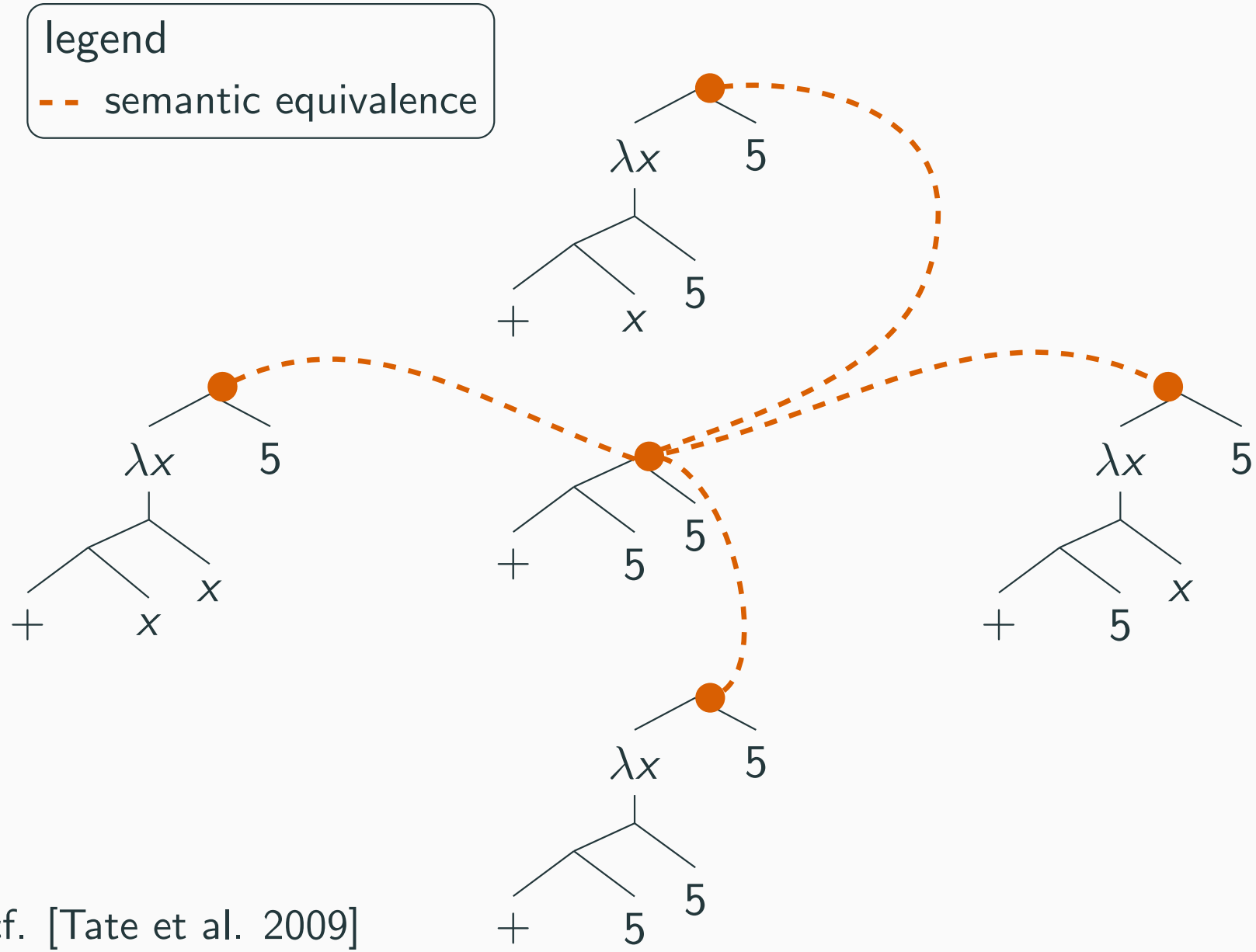








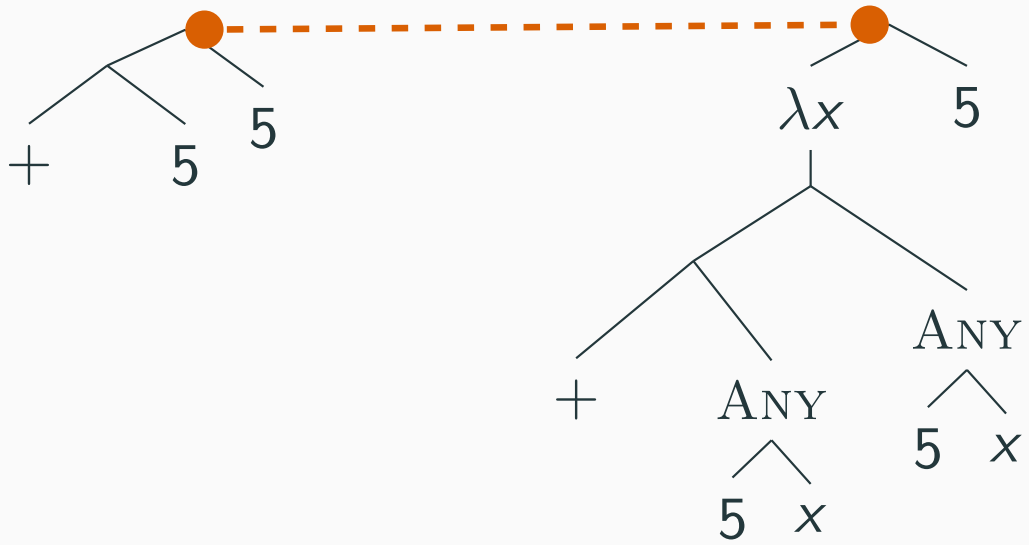
Abstraction Sleep: Growing the library via refactoring



cf. [Tate et al. 2009]

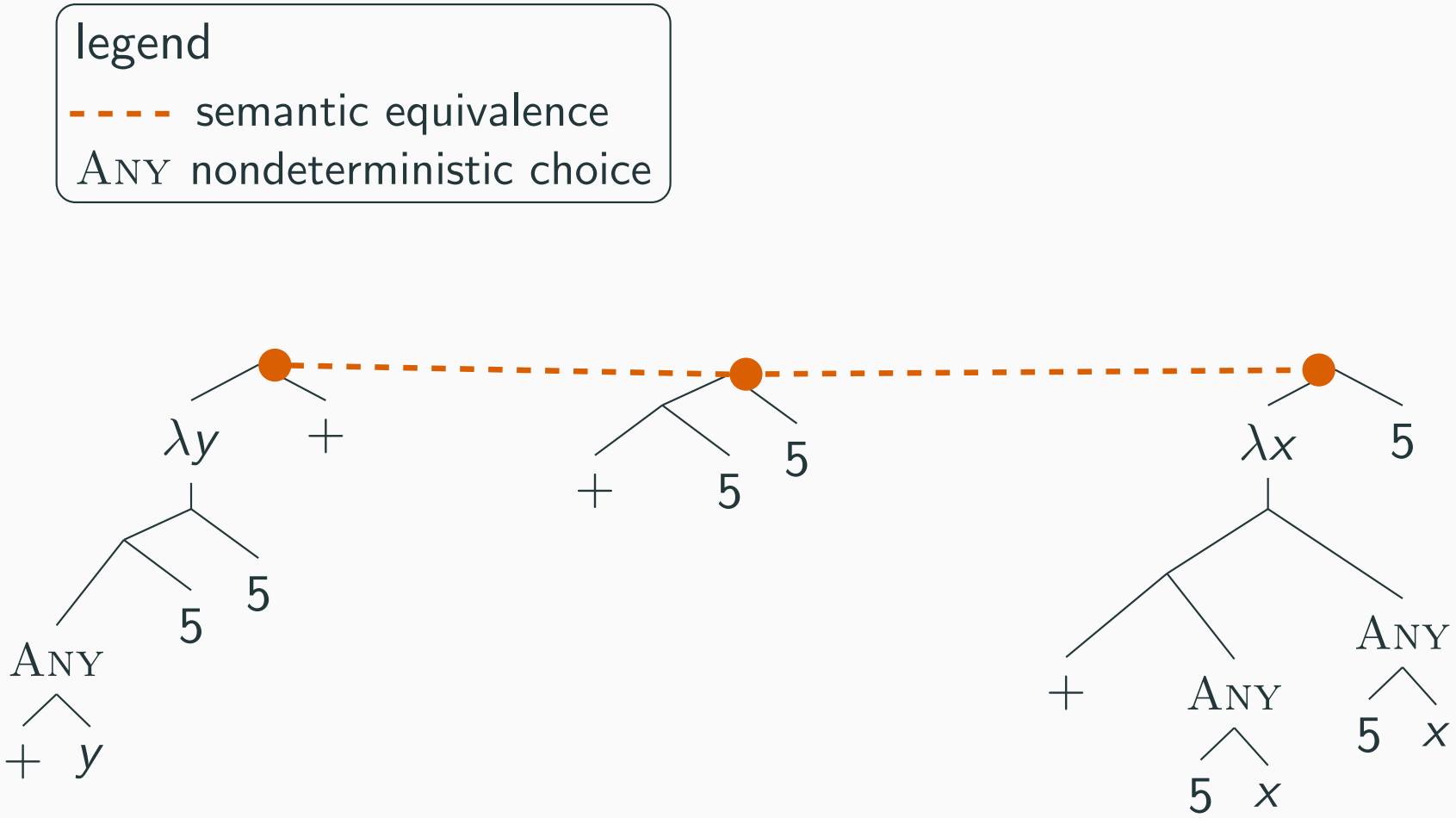
Abstraction Sleep: Growing the library via refactoring

legend
- - - - semantic equivalence
ANY nondeterministic choice



cf. Tate et al. 2009
Gulwani 2012
Liang et al. 2010

Abstraction Sleep: Growing the library via refactoring

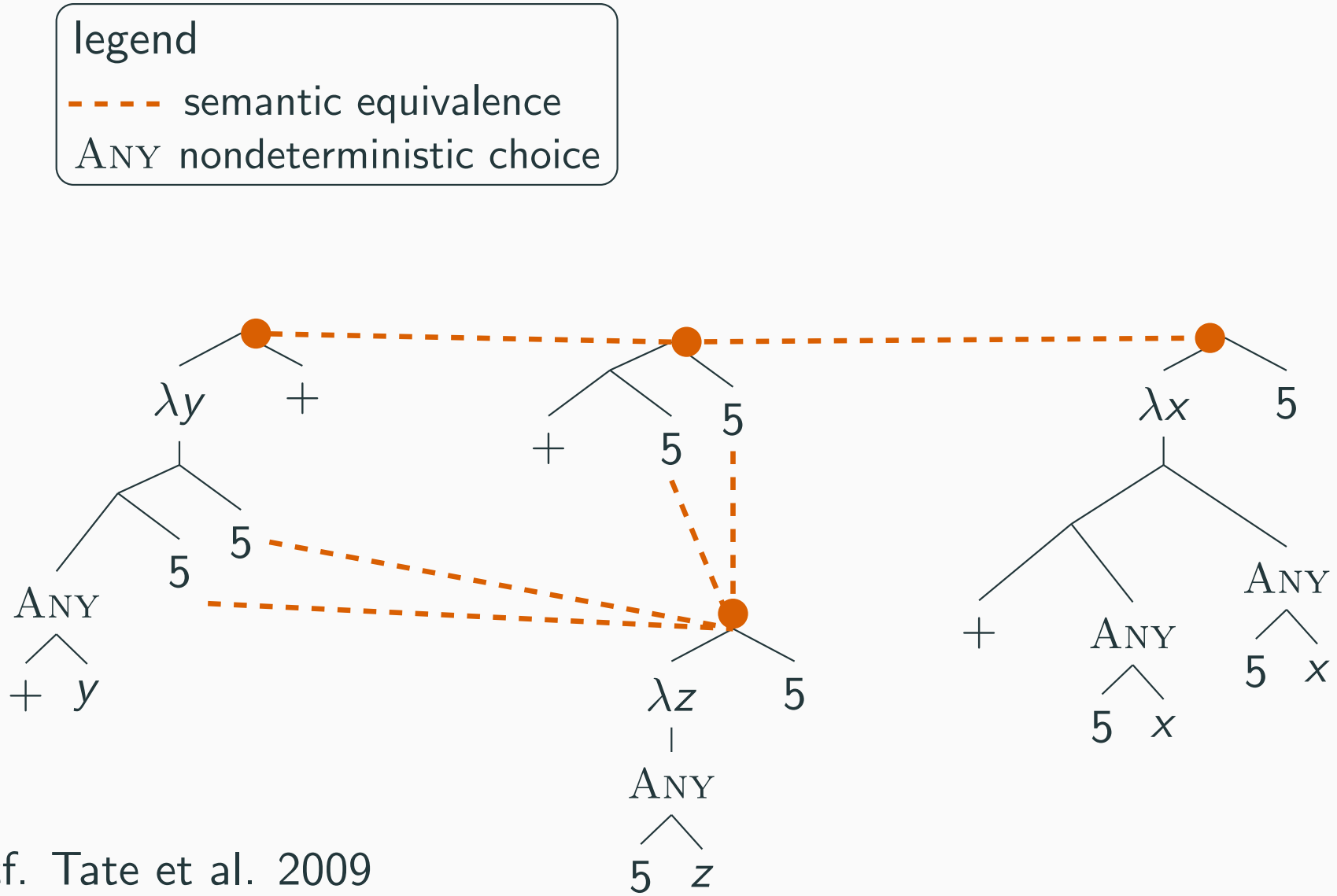


cf. Tate et al. 2009

Gulwani 2012

Liang et al. 2010

Abstraction Sleep: Growing the library via refactoring

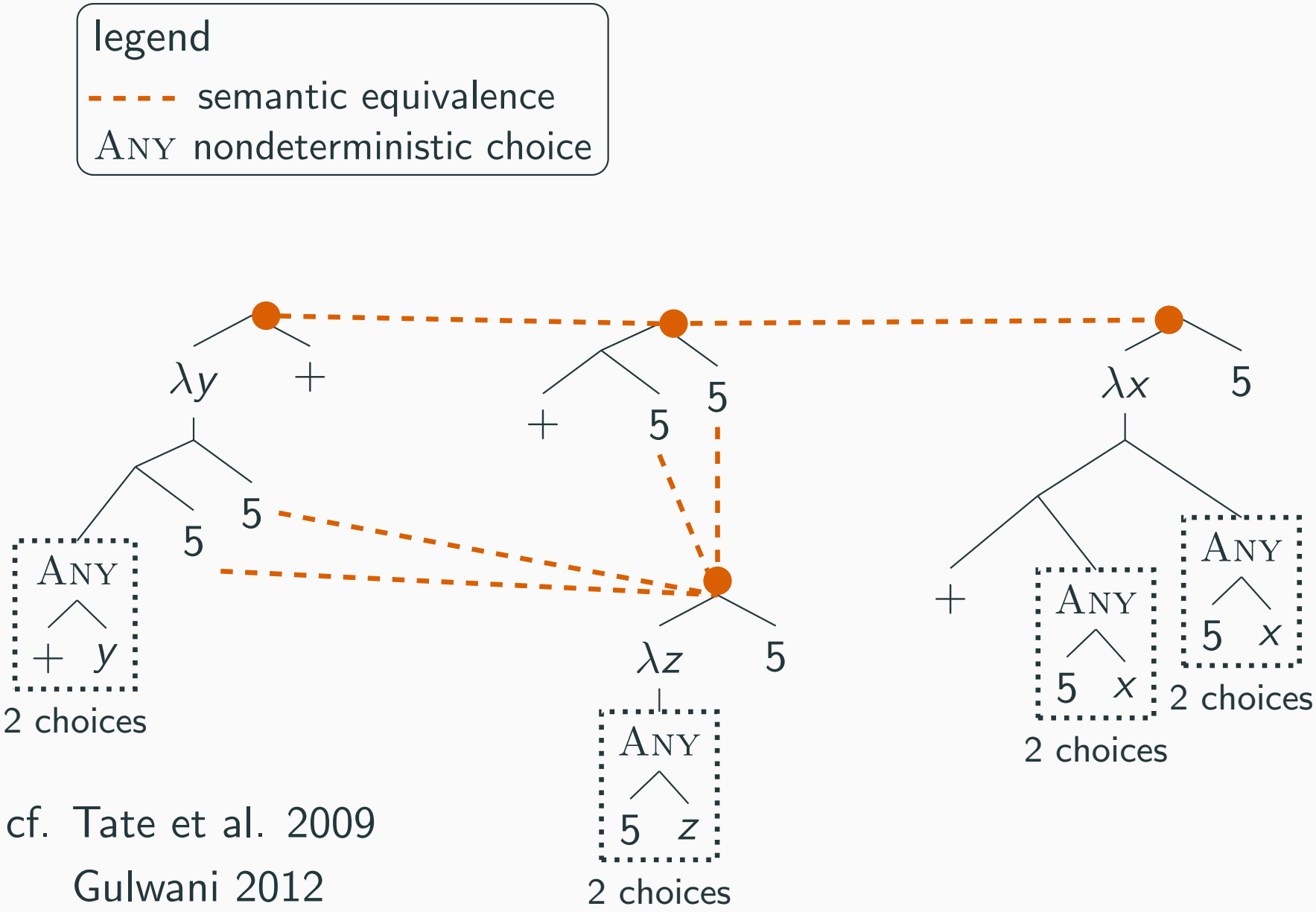


cf. Tate et al. 2009

Gulwani 2012

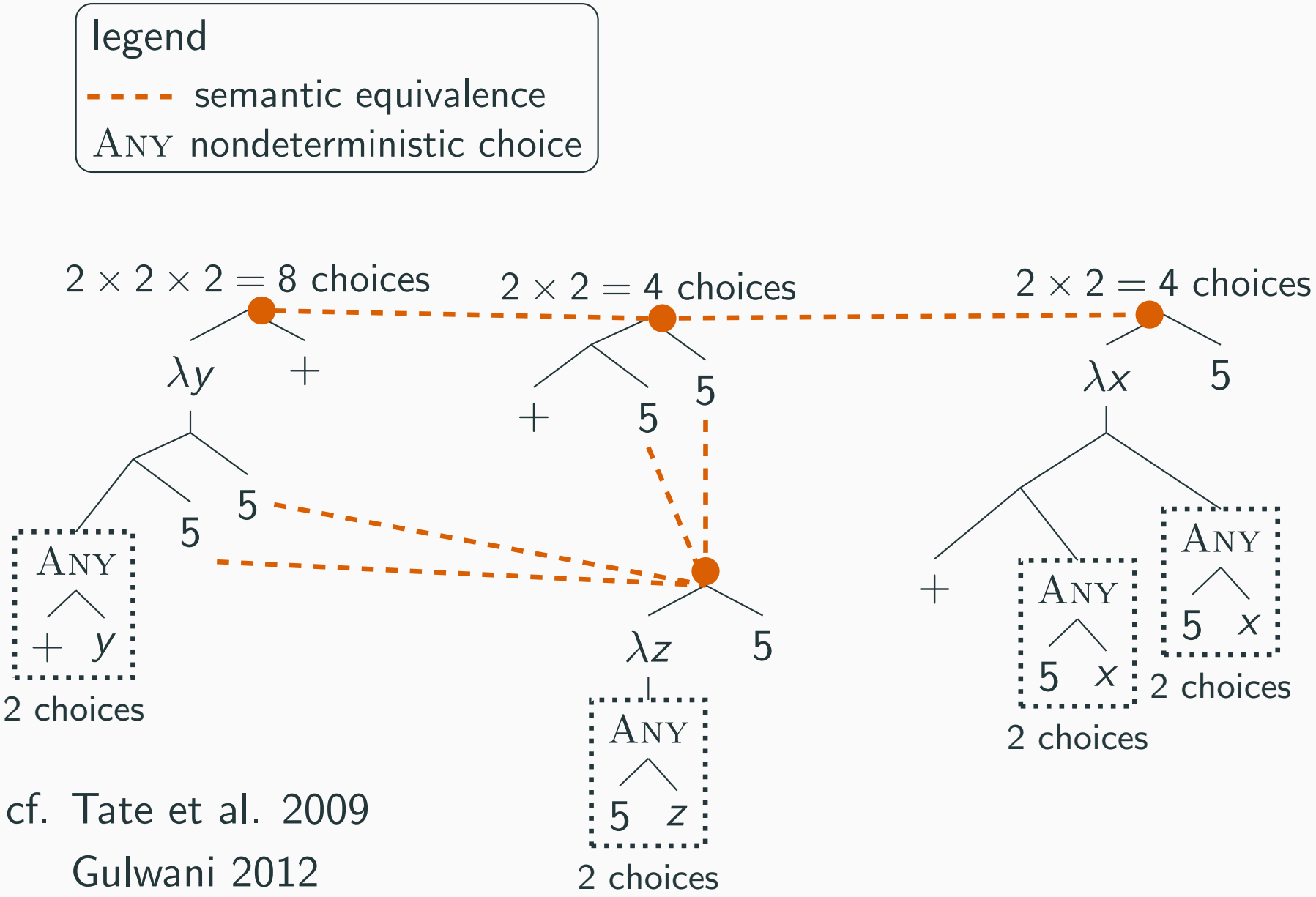
Liang et al. 2010

Abstraction Sleep: Growing the library via refactoring

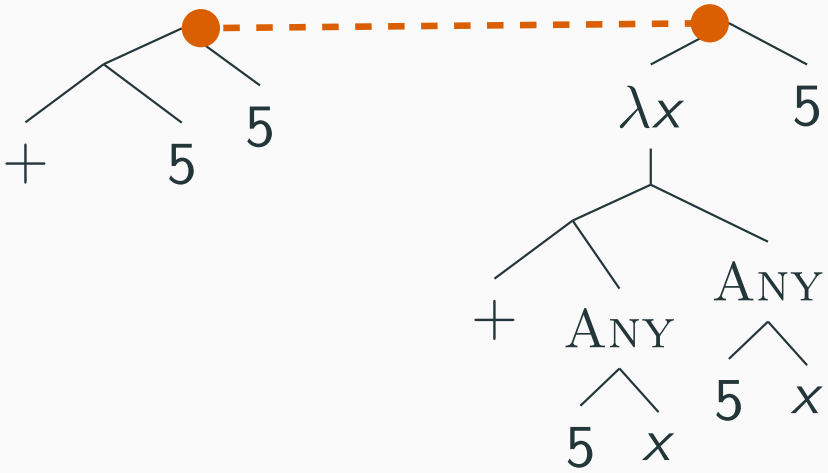


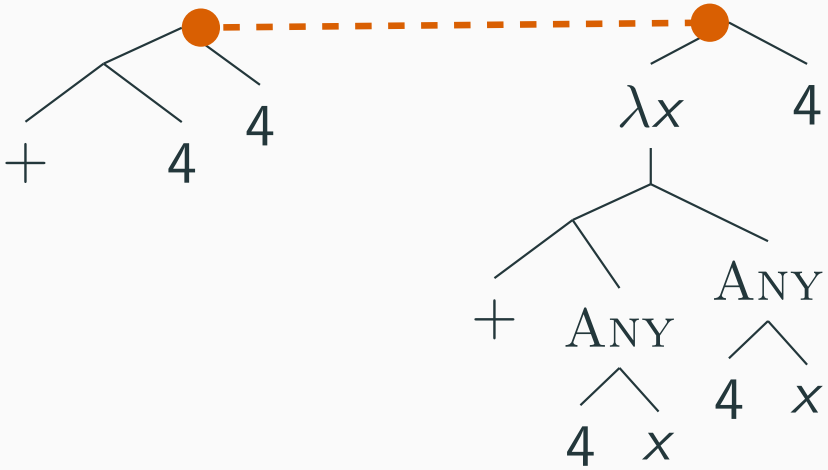
cf. Tate et al. 2009
Gulwani 2012
Liang et al. 2010

Abstraction Sleep: Growing the library via refactoring



cf. Tate et al. 2009
Gulwani 2012
Liang et al. 2010

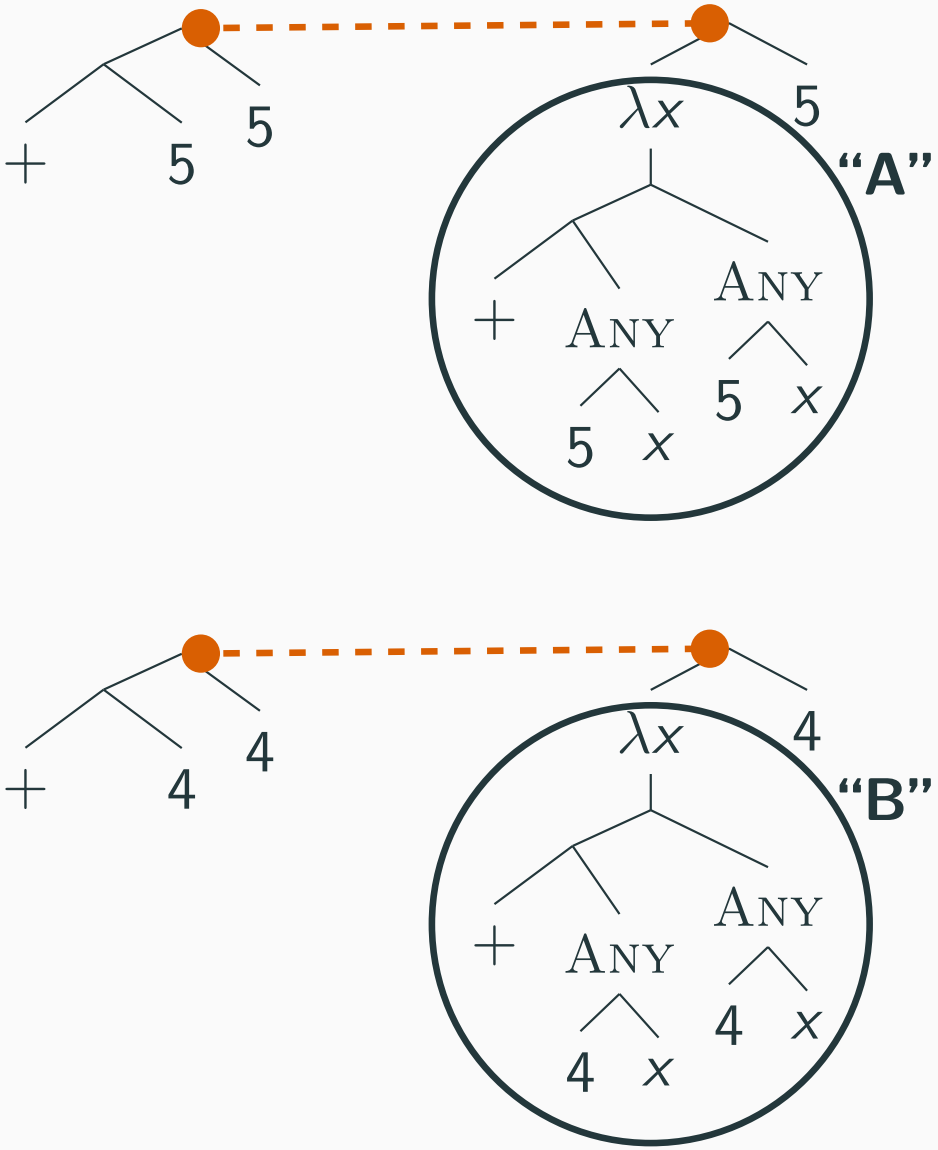


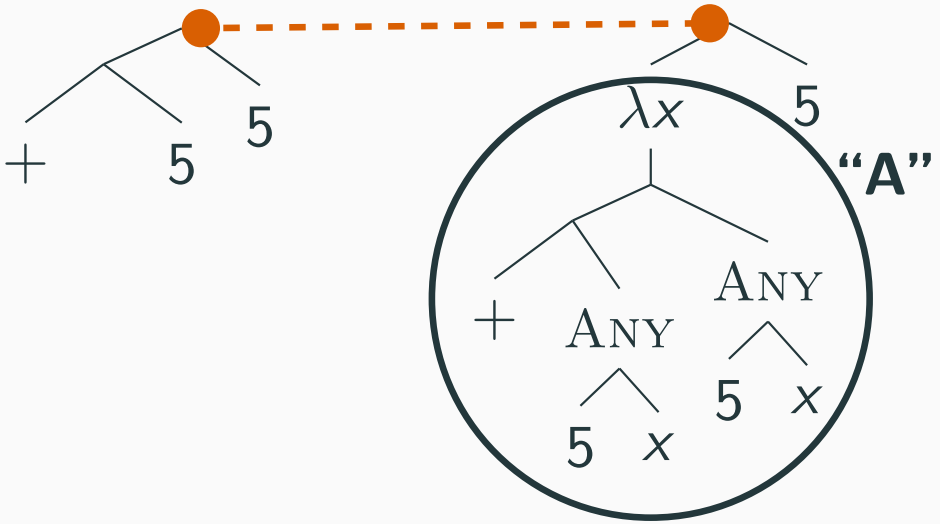


legend

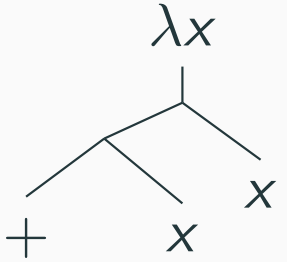
semantic equivalence

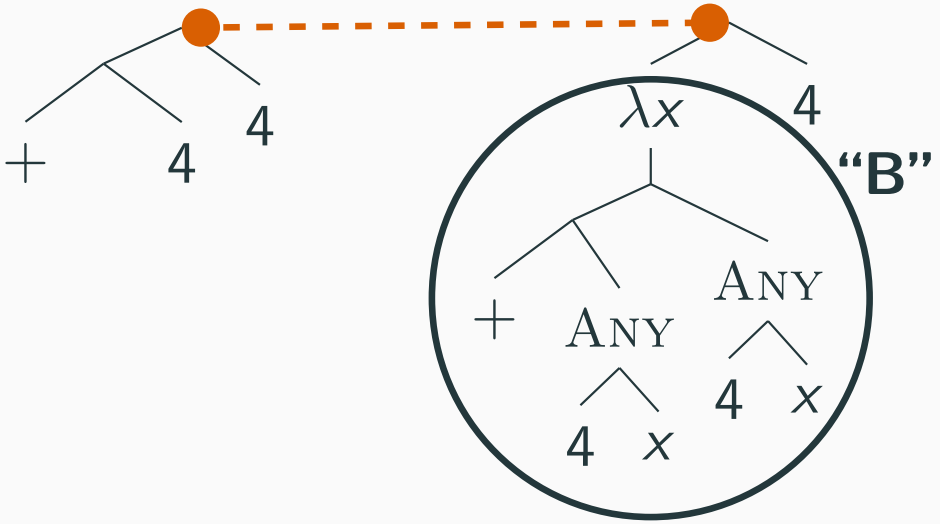
ANY nondeterministic choice


+
5
5
λx
5
“A”
+
ANY
ANY
5
x
5
x
+
4
4
λx
4
“B”
+
ANY
ANY
4
x
4
x
legend
semantic equivalence
ANY nondeterministic choice



A intersect B:

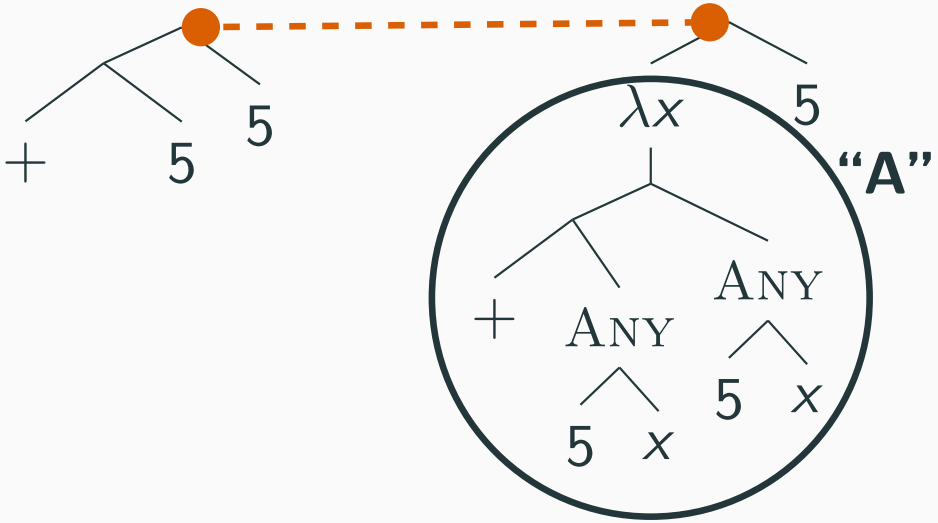




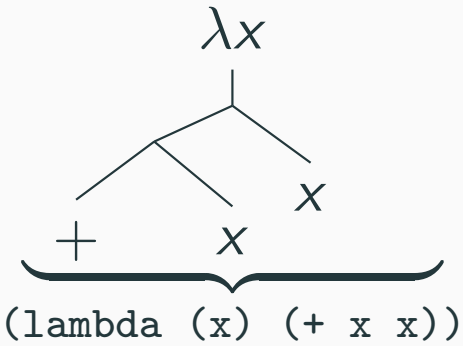
legend

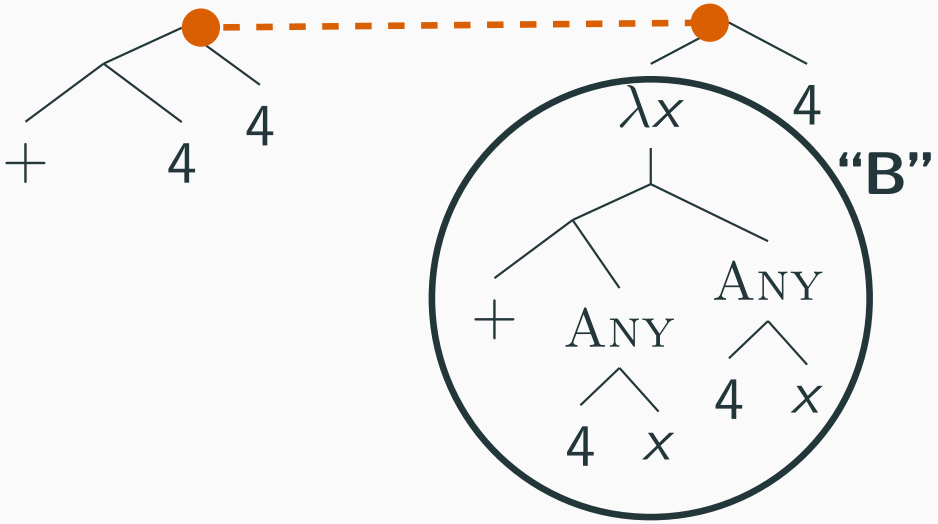
- - - - semantic equivalence

ANY nondeterministic choice



A intersect B:

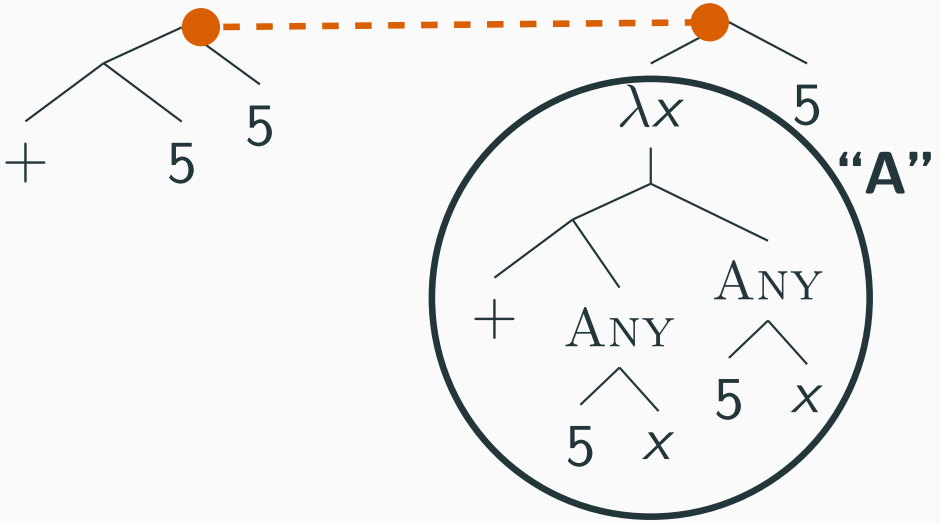




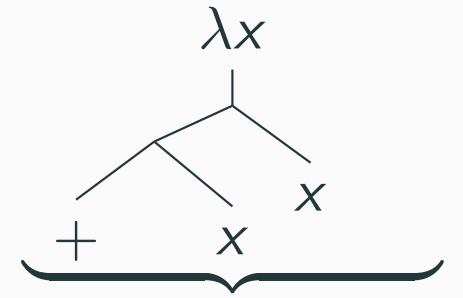
legend

- - - - semantic equivalence

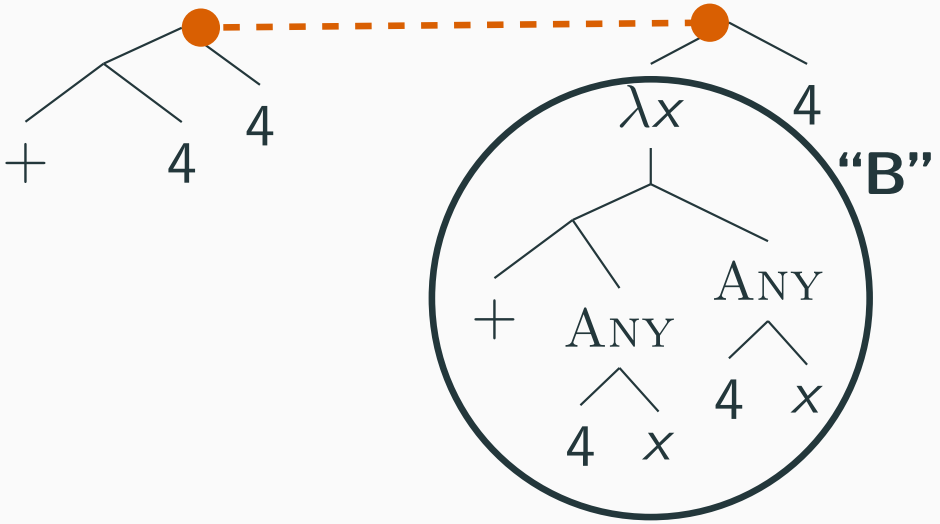
ANY nondeterministic choice



A intersect B:



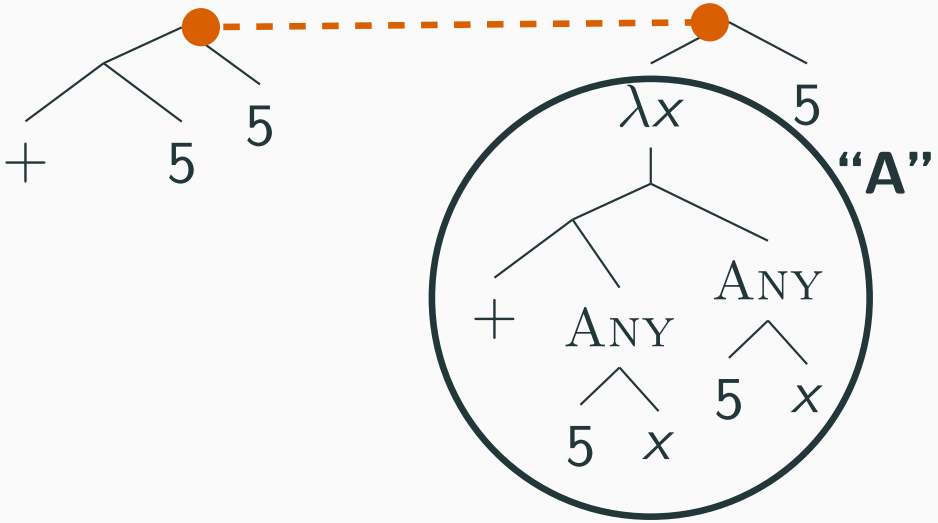
=double



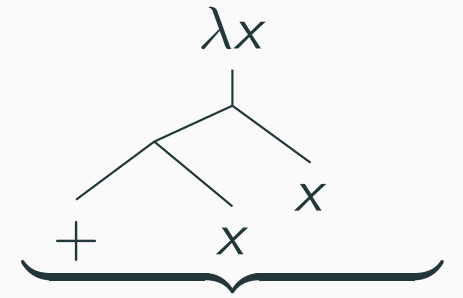
legend

- - - - semantic equivalence

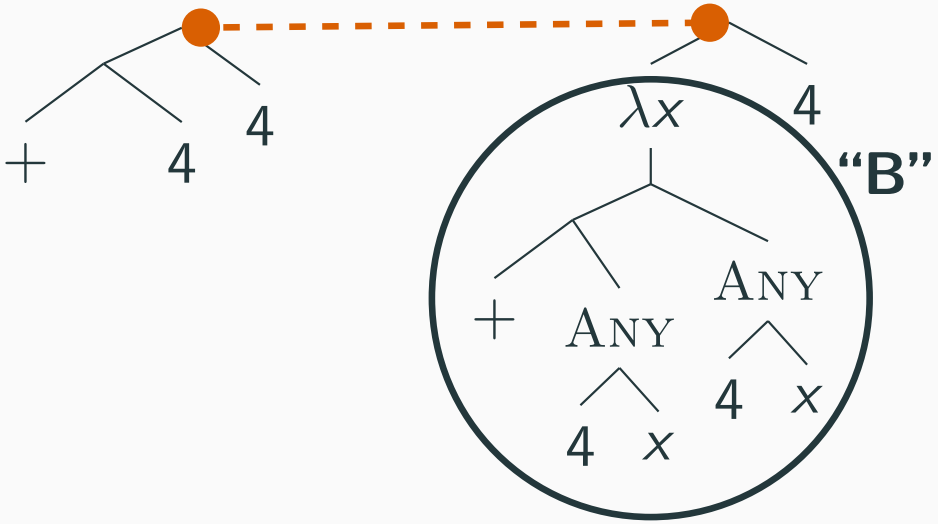
ANY nondeterministic choice



A intersect B:



=double



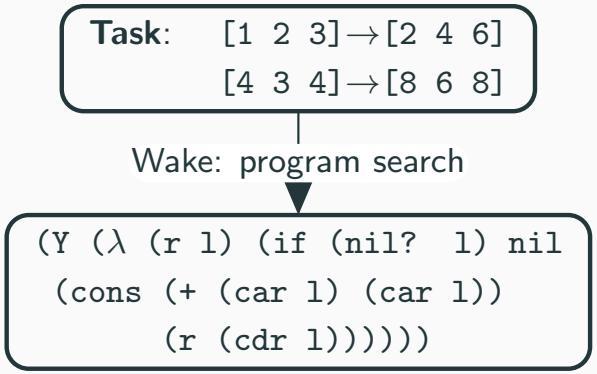
w/o double	w/ double
(+ 5 5)	(double 5)
(+ 4 4)	(double 4)
(+ 3 3)	(double 3)
	...

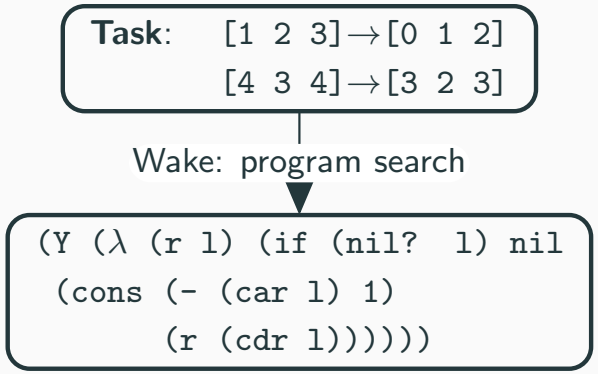
legend

- - - - semantic equivalence

ANY nondeterministic choice

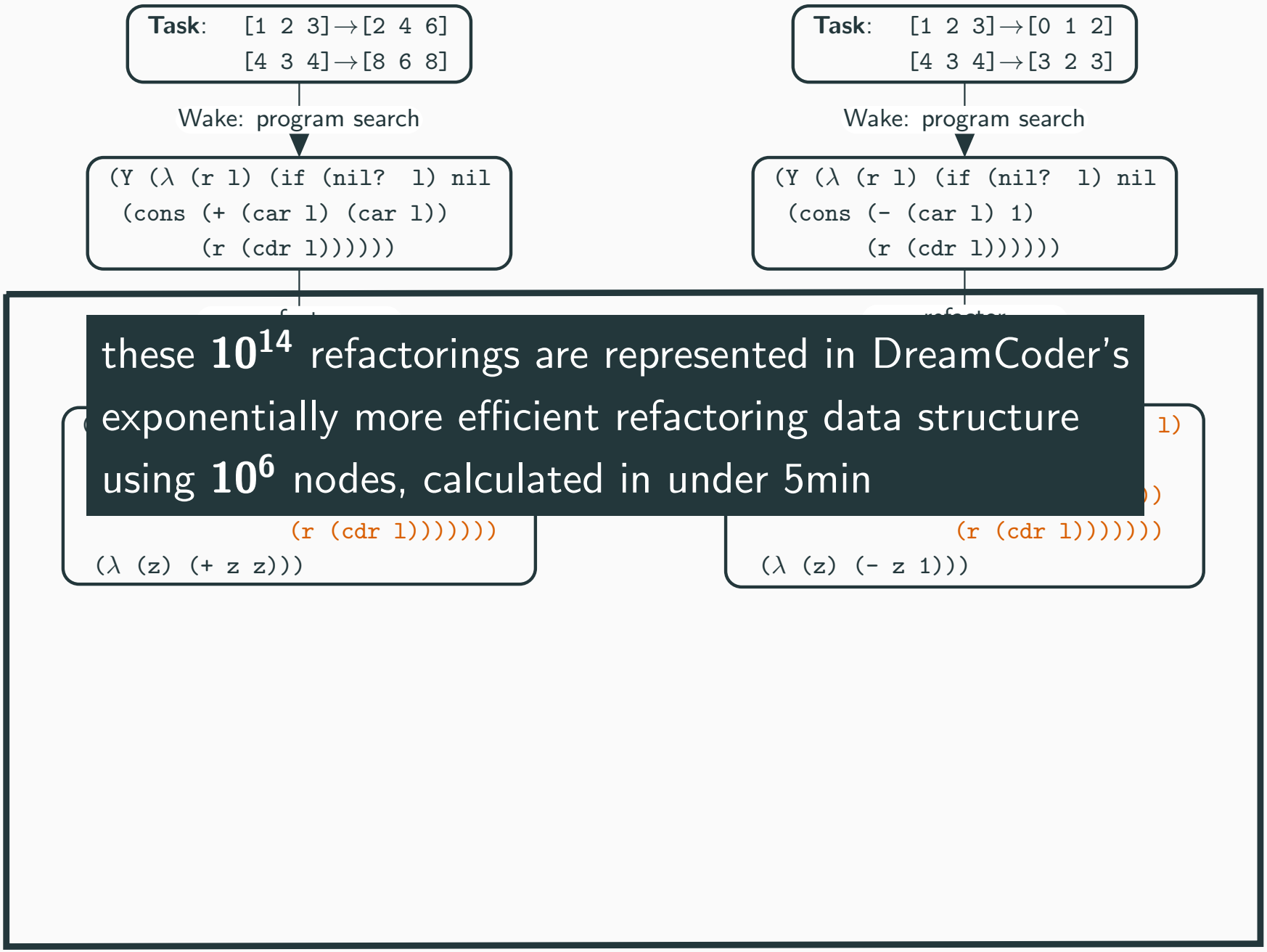
Abstraction Sleep: Growing the library via refactoring



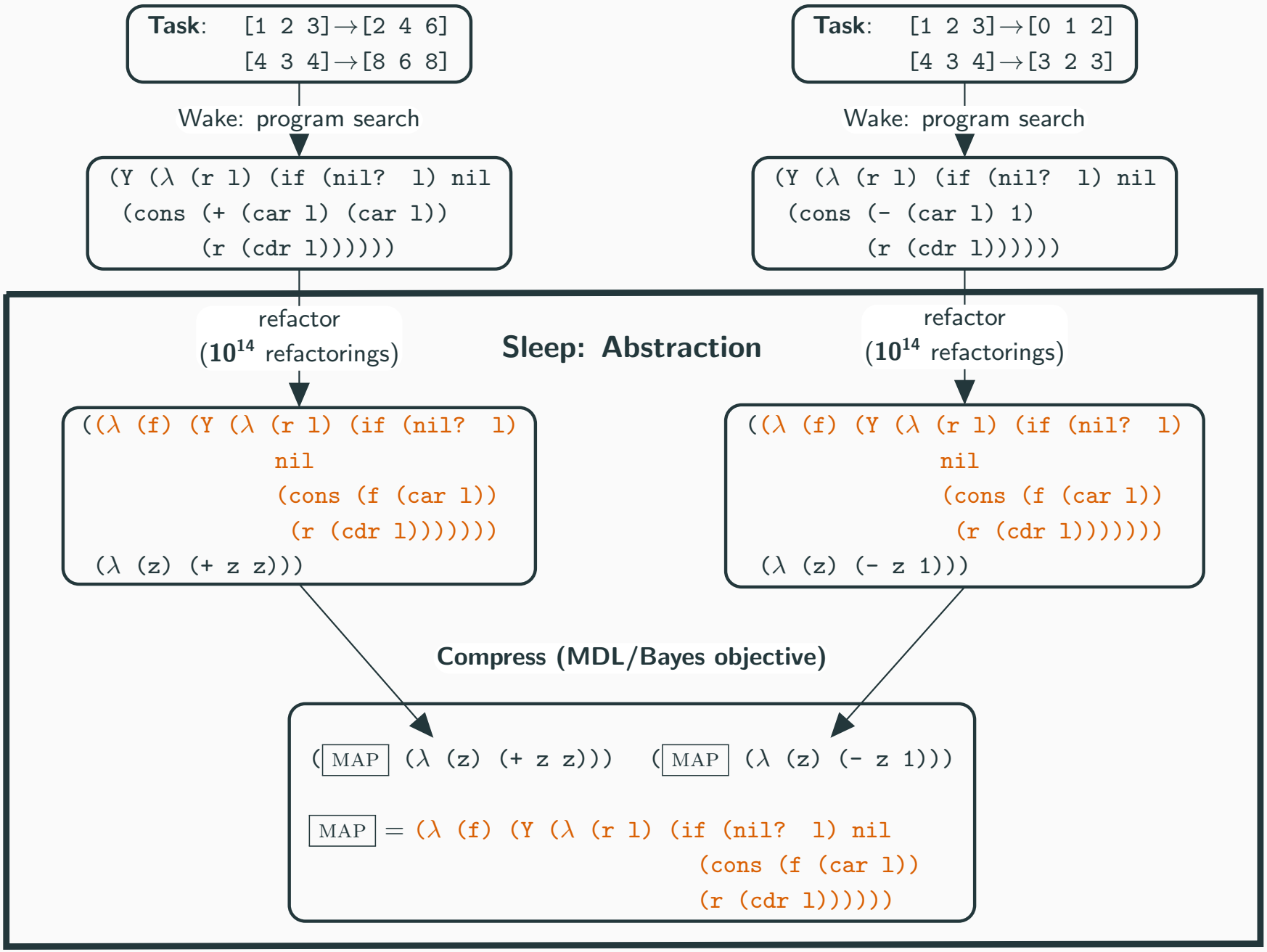


these $\mathbf{10^{14}}$ refactorings are represented in DreamCoder's exponentially more efficient refactoring data structure using $\mathbf{10^{6}}$ nodes, calculated in under 5min

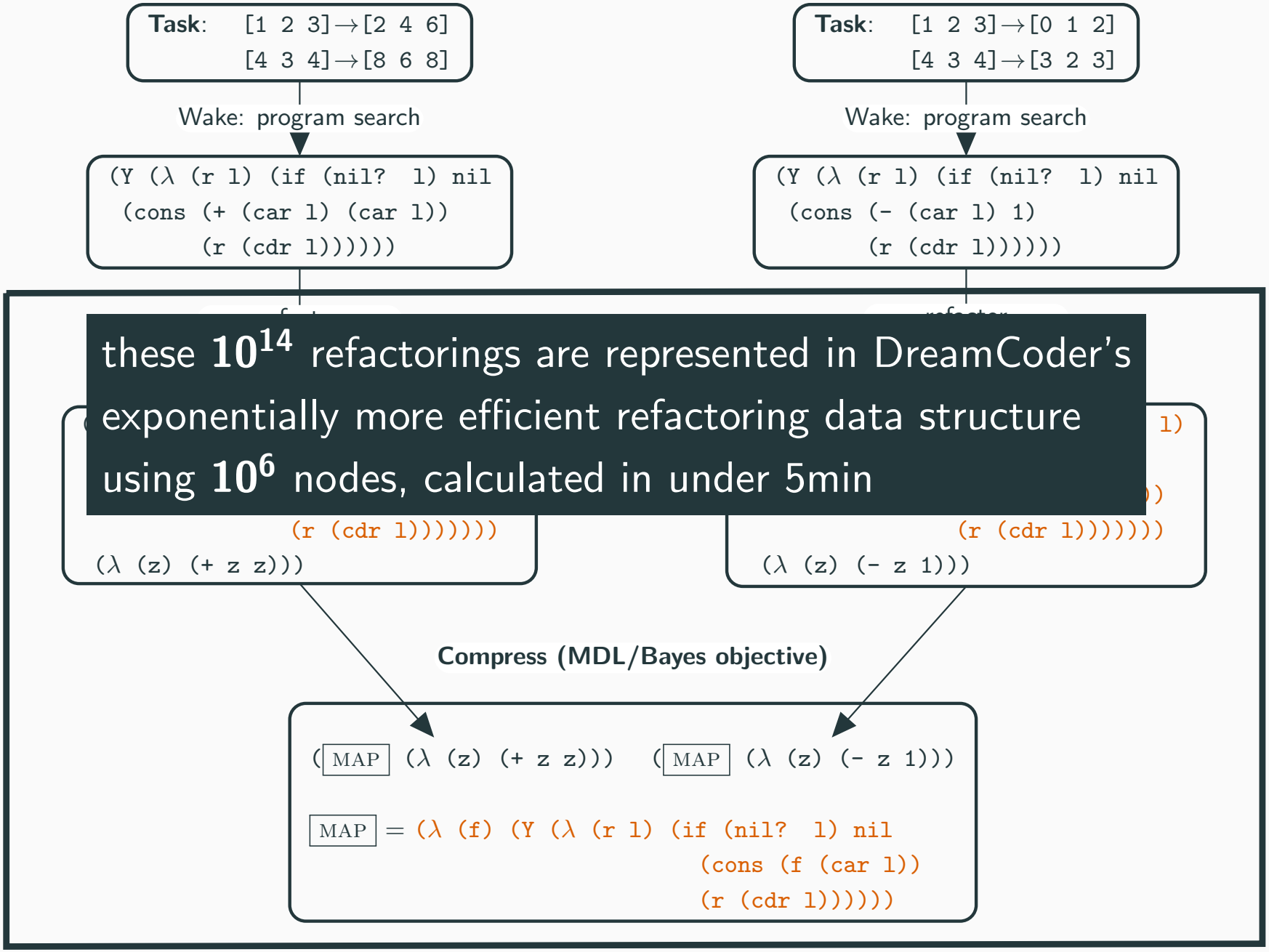
Abstraction Sleep: Growing the library via refactoring



Abstraction Sleep: Growing the library via refactoring



Abstraction Sleep: Growing the library via refactoring



DreamCoder Domains

List Processing

Sum List

```
[1 2 3] → 6
[4 6 8 1] → 17
```

Double

```
[1 2 3 4] → [2 4 6 8]
[6 5 1] → [12 10 2]
```

Check Evens

```
[0 2 3] → [T T F]
[2 4 9 6] → [T T F T]
```

Text Editing

Abbreviate

```
Allen Newell → A.N.
Herb Simon → H.S.
```

Drop Last Characters

```
jabberwocky → jabberw
copycat → cop
```

Extract

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

Regexes

Phone Numbers

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

Currency

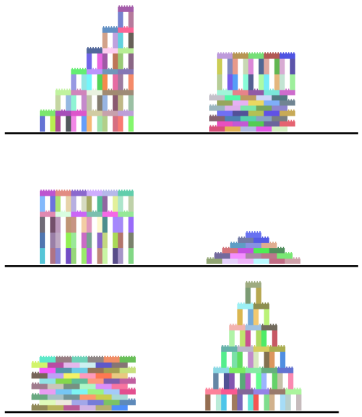
```
$100.25
$4.50
```

Dates

```
Y1775/0704
Y2000/0101
```

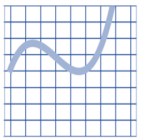
LOGO Graphics

Block Towers

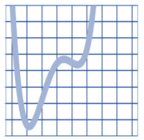


Symbolic Regression

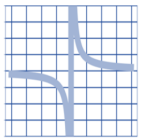
$\frac{(-2.4x - 0.9)}{(x - 4.4)(x - 0.9)}$



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



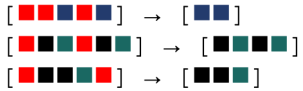
$0.5x^4 + 2.5x^3 + 0.4x^2 - 2.2x + 2.4$



$\frac{4.9}{x}$

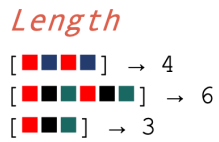
Recursive Programming

Filter

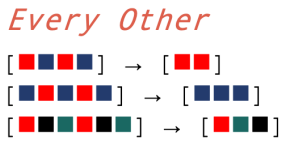


Index List

Length



Every Other



Physics

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

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

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1} - \vec{r_2}|^2}\widehat{r_1 - r_2}$$

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

DreamCoder Domains

List Processing

Sum List

```
[1 2 3] → 6
[4 6 8 1] → 17
```

Double

```
[1 2 3 4] → [2 4 6 8]
[6 5 1] → [12 10 2]
```

Check Evens

```
[0 2 3] → [T T F]
[2 4 9 6] → [T T F T]
```

Text Editing

Abbreviate

```
Allen Newell → A.N.
Herb Simon → H.S.
```

Drop Last Characters

```
jabberwocky → jabberw
copycat → cop
```

Extract

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

Regexes

Phone Numbers

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

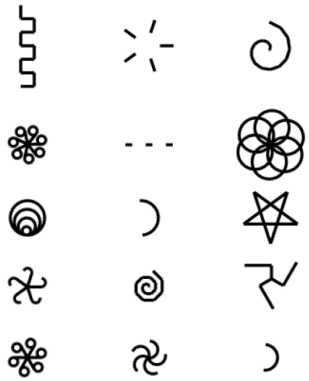
Currency

```
$100.25
$4.50
```

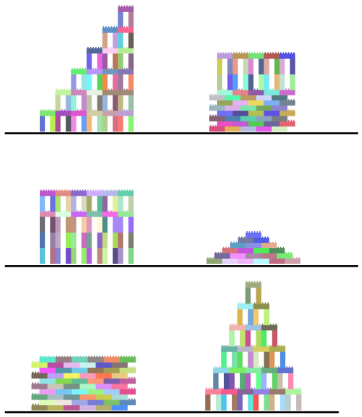
Dates

```
Y1775/0704
Y2000/0101
```

LOGO Graphics

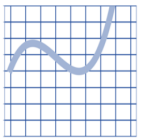


Block Towers

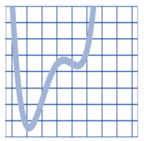


Symbolic Regression

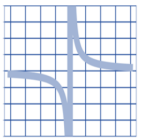
$\frac{(-2.4x - 0.9)}{(x - 4.4)(x - 0.9)}$



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



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



$\frac{4.9}{x}$

Recursive Programming

Filter

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

Index List

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

Length

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

Every Other

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

Physics

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

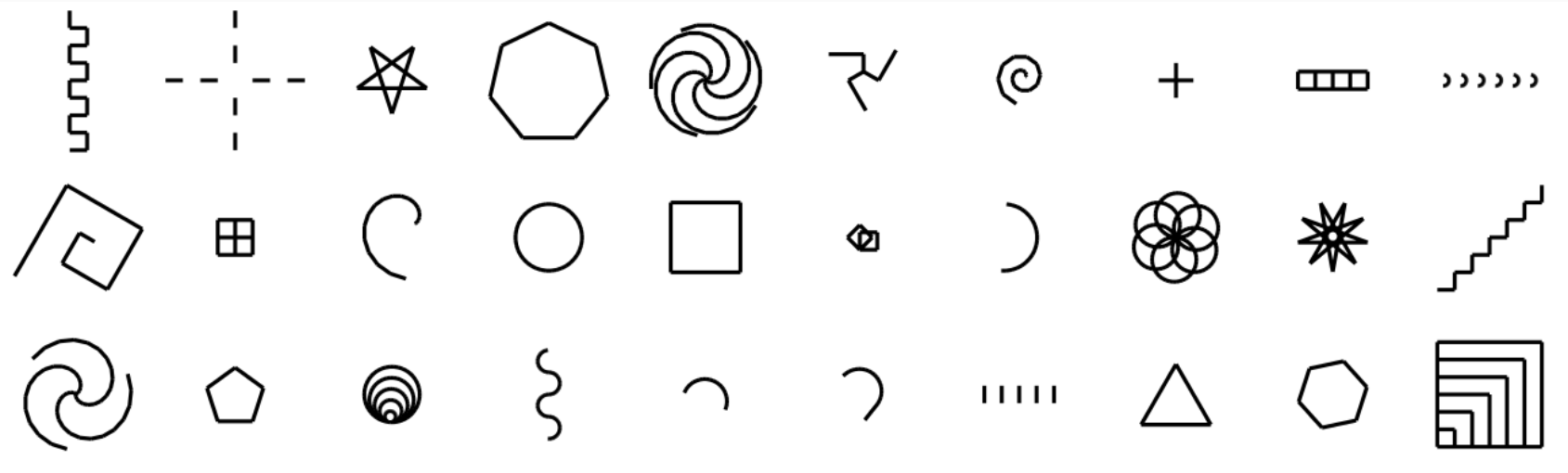
$$\vec{a} = \frac{1}{m}\sum_i \vec{F}_i$$

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1} - \vec{r_2}|^2}\widehat{r_1 - r_2}$$

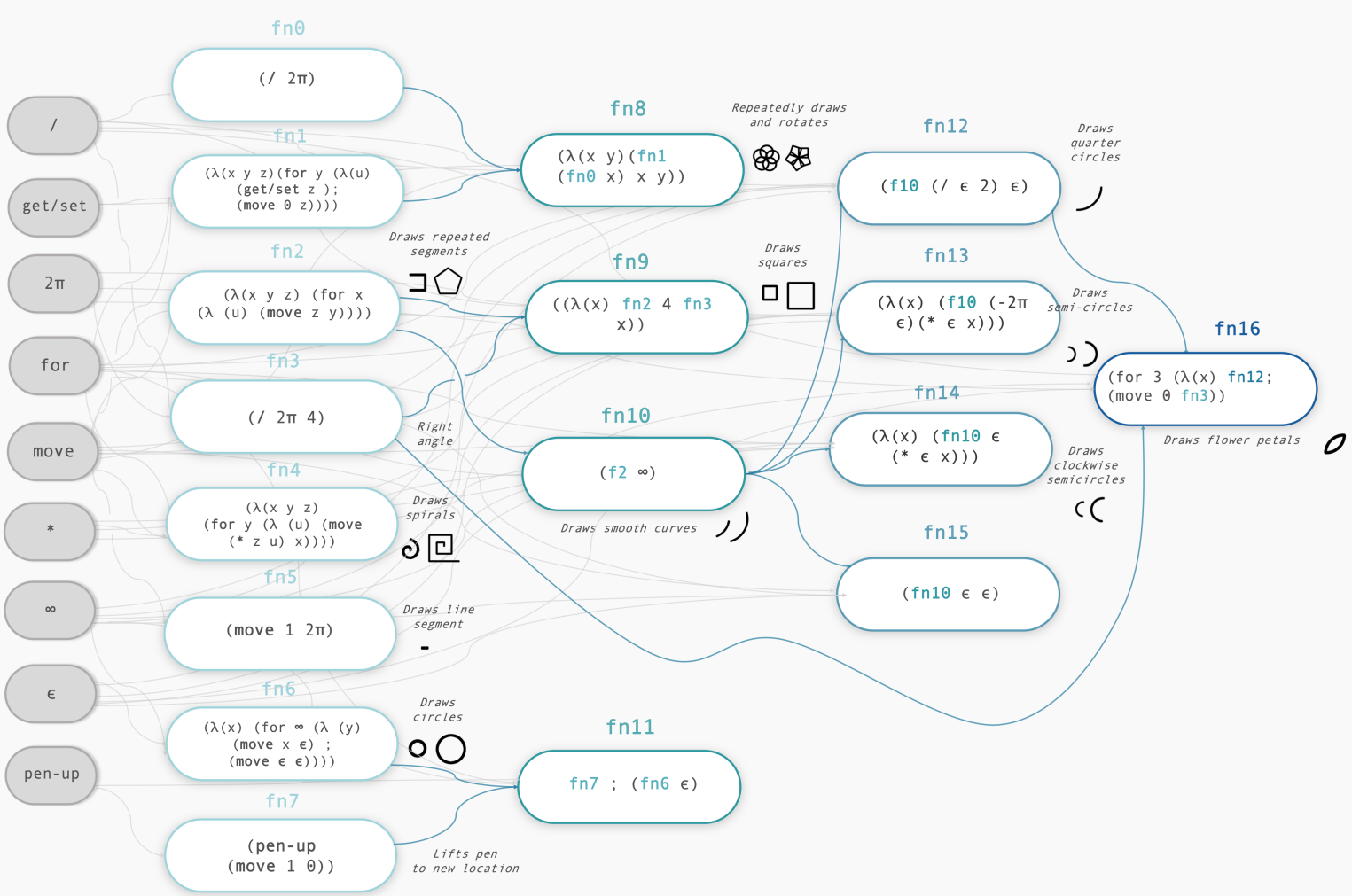
$$R_{total} = \left(\Sigma_i \frac{1}{R_i}\right)^{-1}$$

LOGO Graphics

30 out of 160 tasks

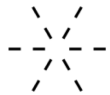


LOGO Graphics – learning an interpretable library of concepts



(fn8 5 (fn4 (* ϵ 2) ∞ ϵ))

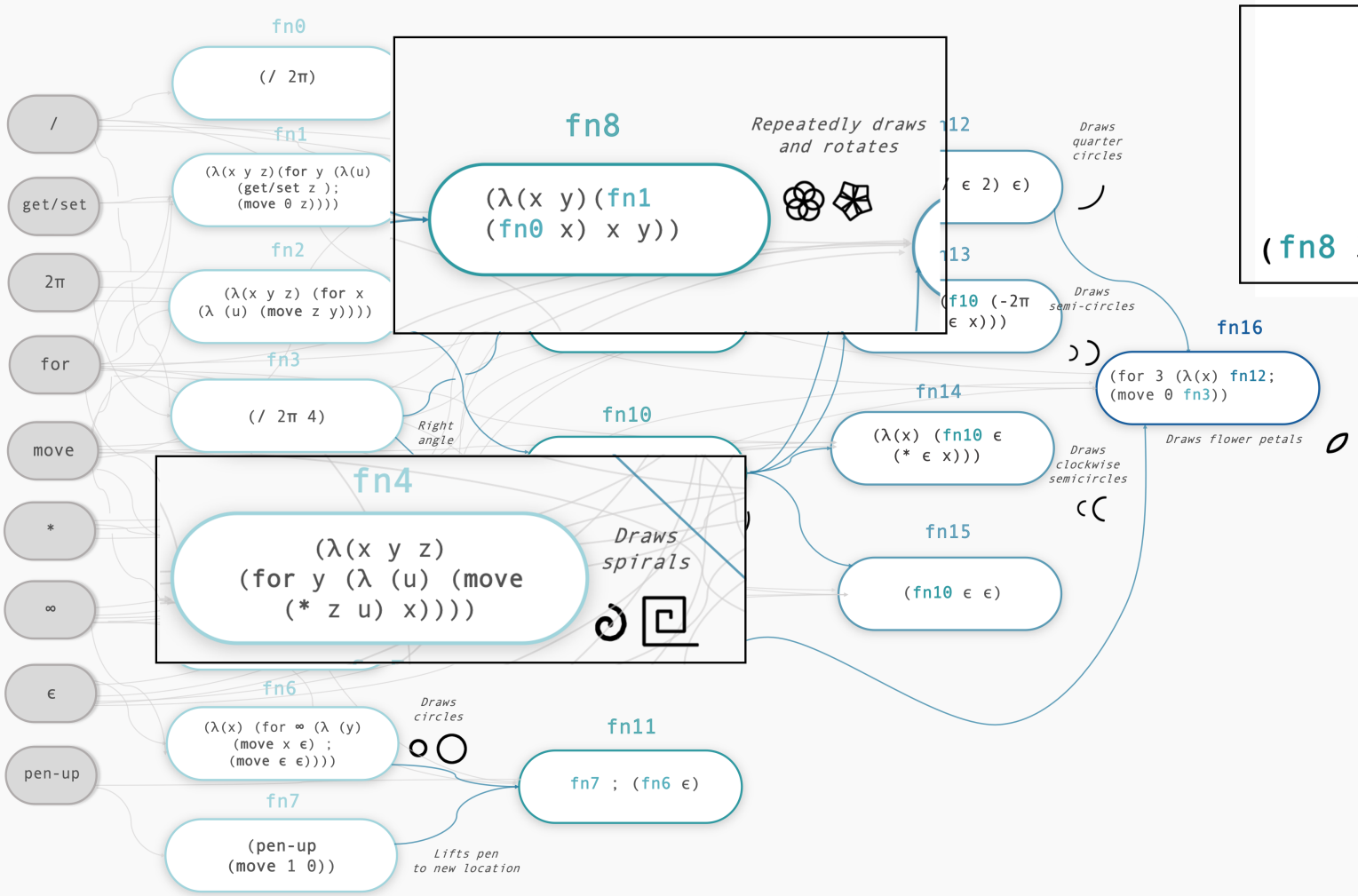
(for 7 (λ (x) (fn9 x)))

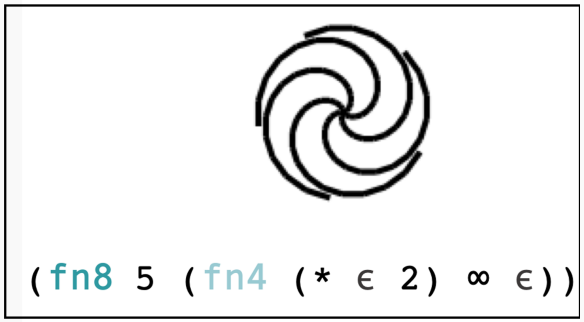


(fn8 6 (fn7 ; fn5 ; fn7 ; fn5))

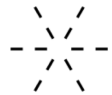
(move 0 (fn0 7)); fn5 ; (fn13 4)

LOGO Graphics – learning an interpretable library of concepts





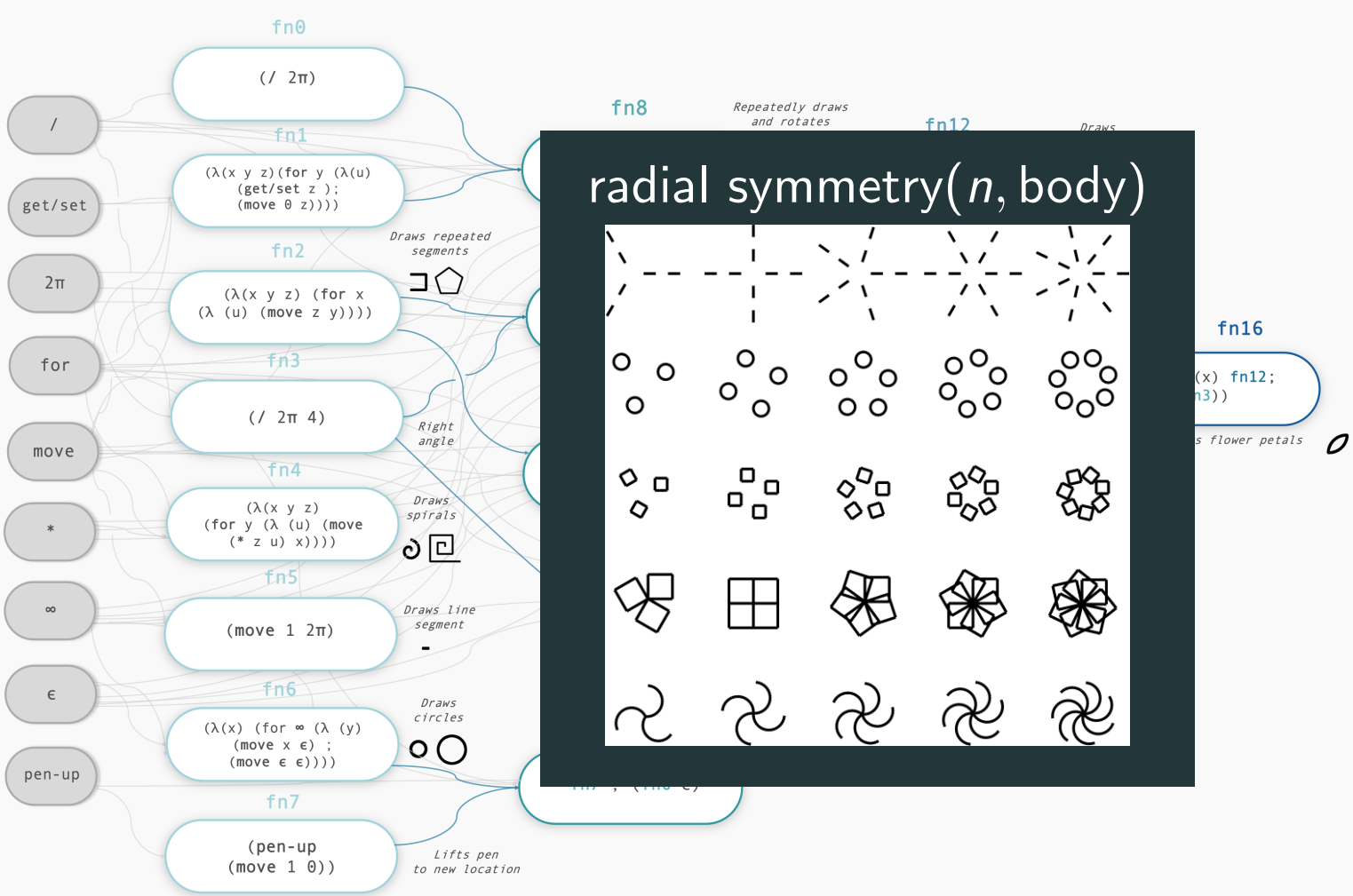
(for 7 (λ (x) (fn9 x)))



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

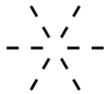
(move 0 (fn0 7)); fn5 ;
(fn13 4)

LOGO Graphics – learning an interpretable library of concepts



```
(fn8 5 (fn4 (* ϵ 2) ∞ ϵ))
```

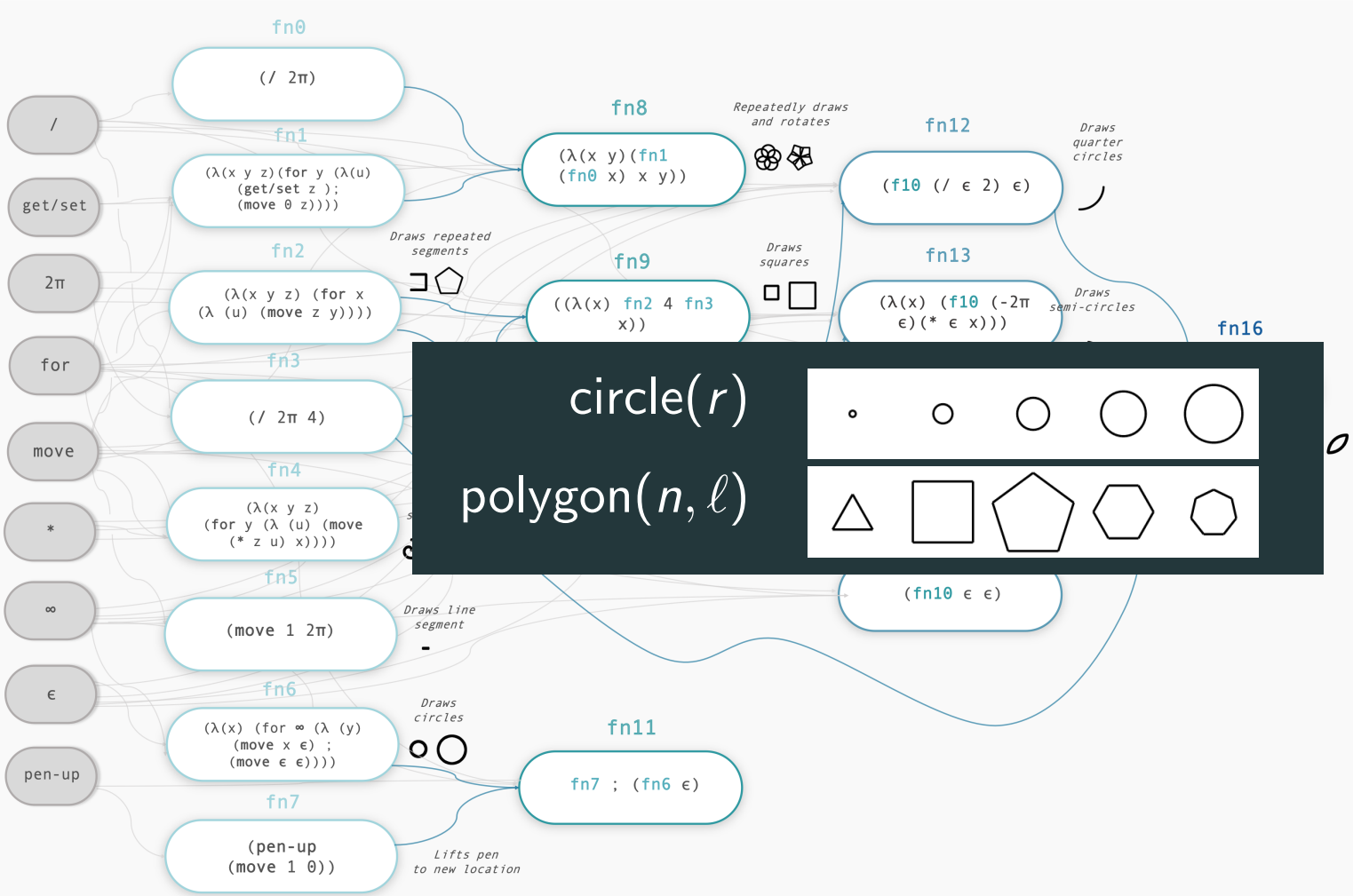
```
(for 7 (λ (x) (fn9 x)))
```



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

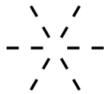
```
(move 0 (fn0 7)); fn5 ;
(fn13 4)
```

LOGO Graphics – learning an interpretable library of concepts



```
(fn8 5 (fn4 (* ϵ 2) ∞ ϵ))
```

```
(for 7 (λ (x) (fn9 x)))
```

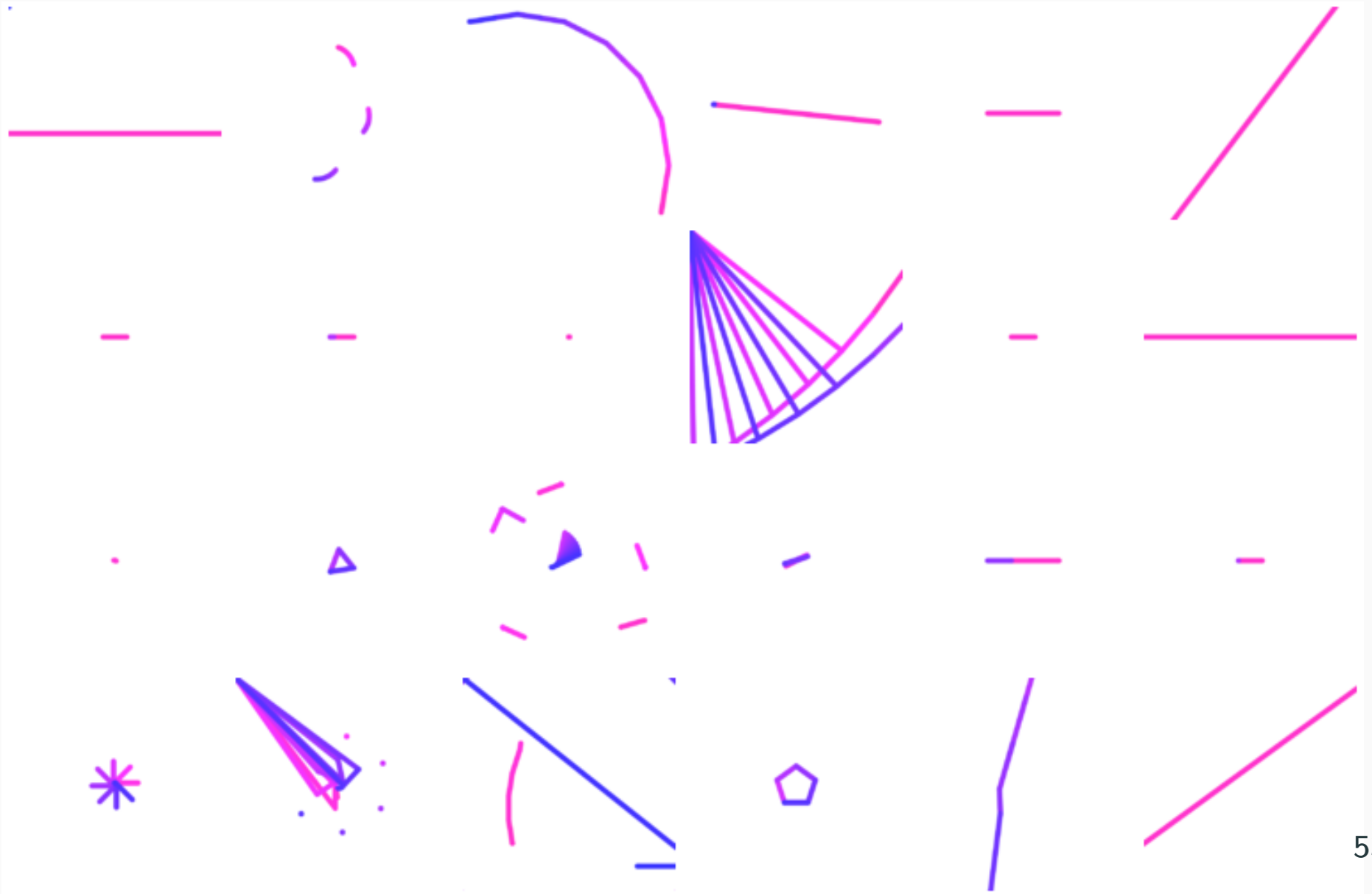


```
(fn8  6 (fn7 ; fn5 ;
fn7 ; fn5))
```

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

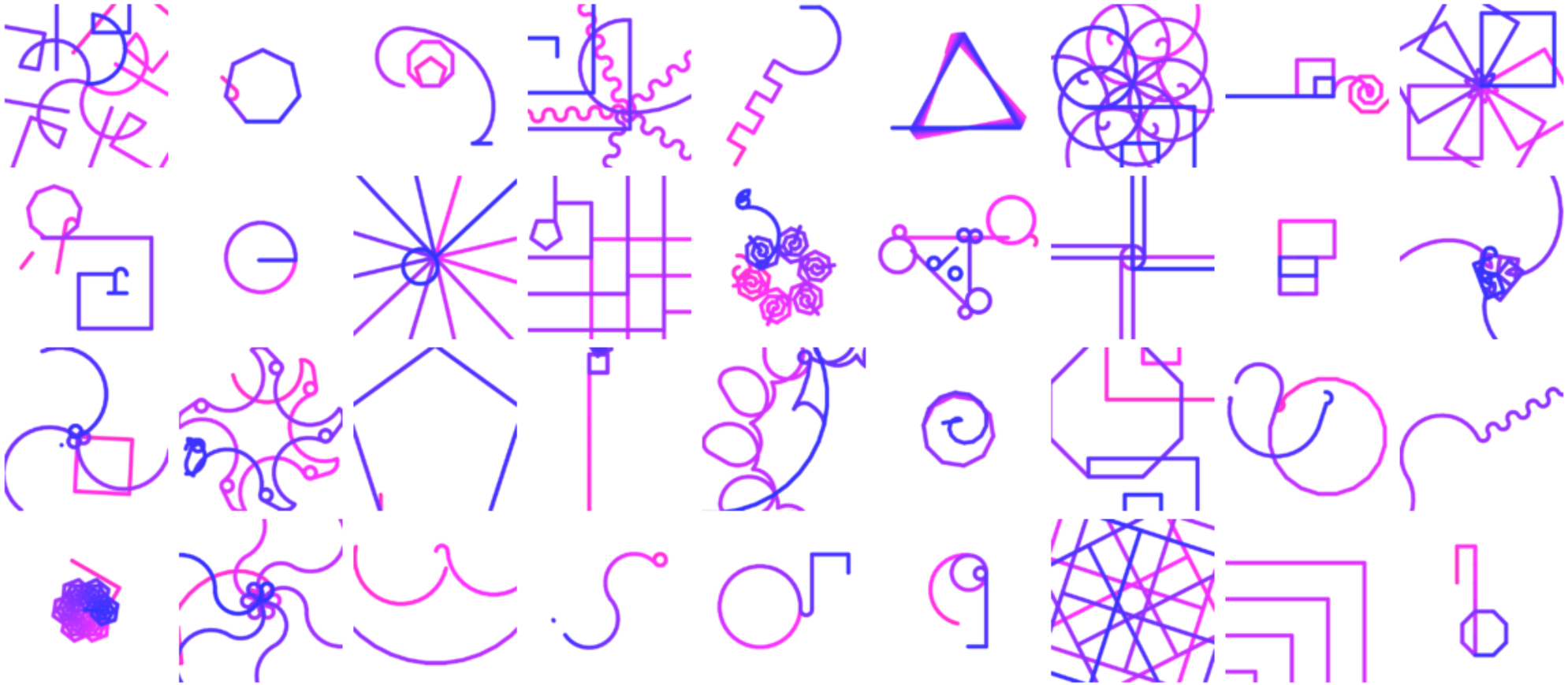
What does DreamCoder dream of?

before learning:



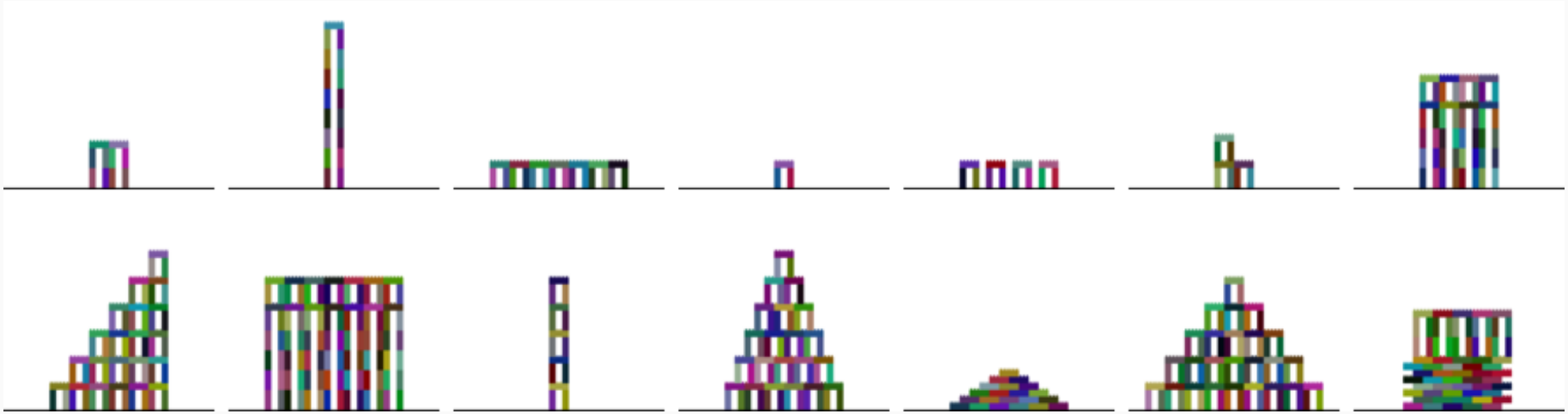
What does DreamCoder dream of?

after learning:



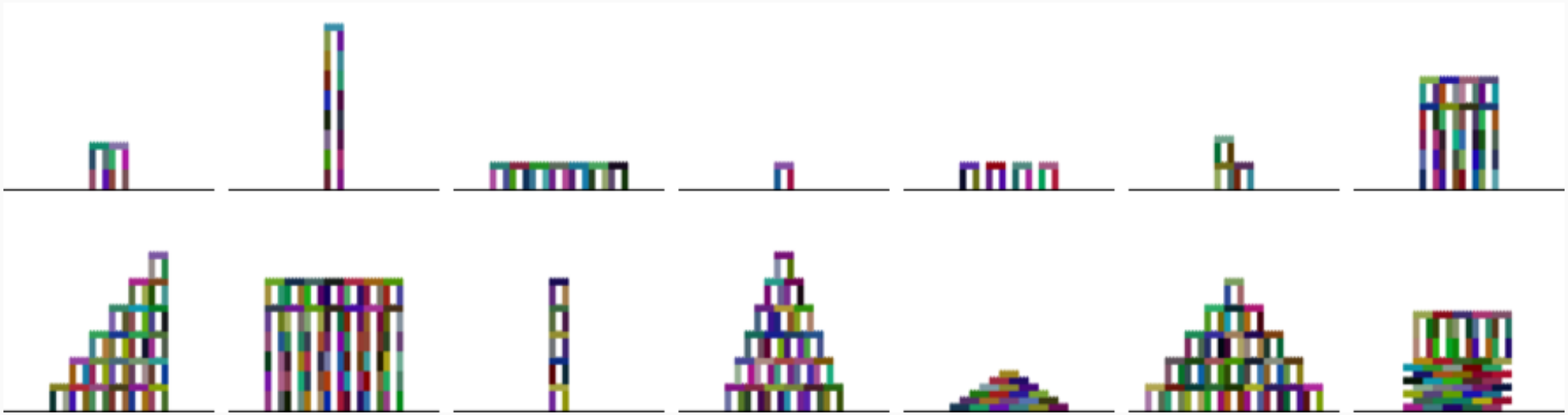
Planning to build towers

example tasks (112 total)



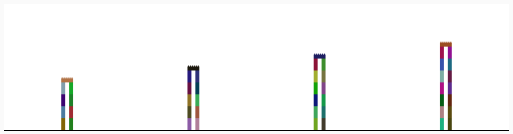
Planning to build towers

example tasks (112 total)

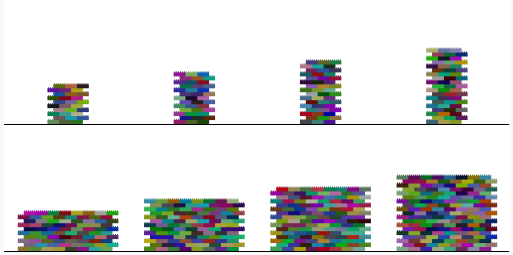


learned library routines ($\approx$ 20 total)

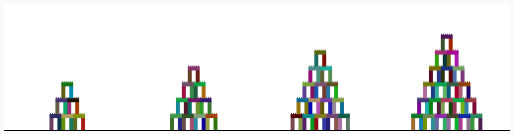
arch(h)



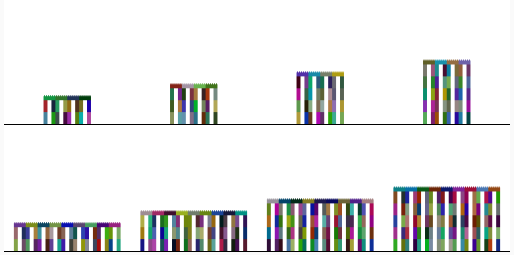
wall(w, h)



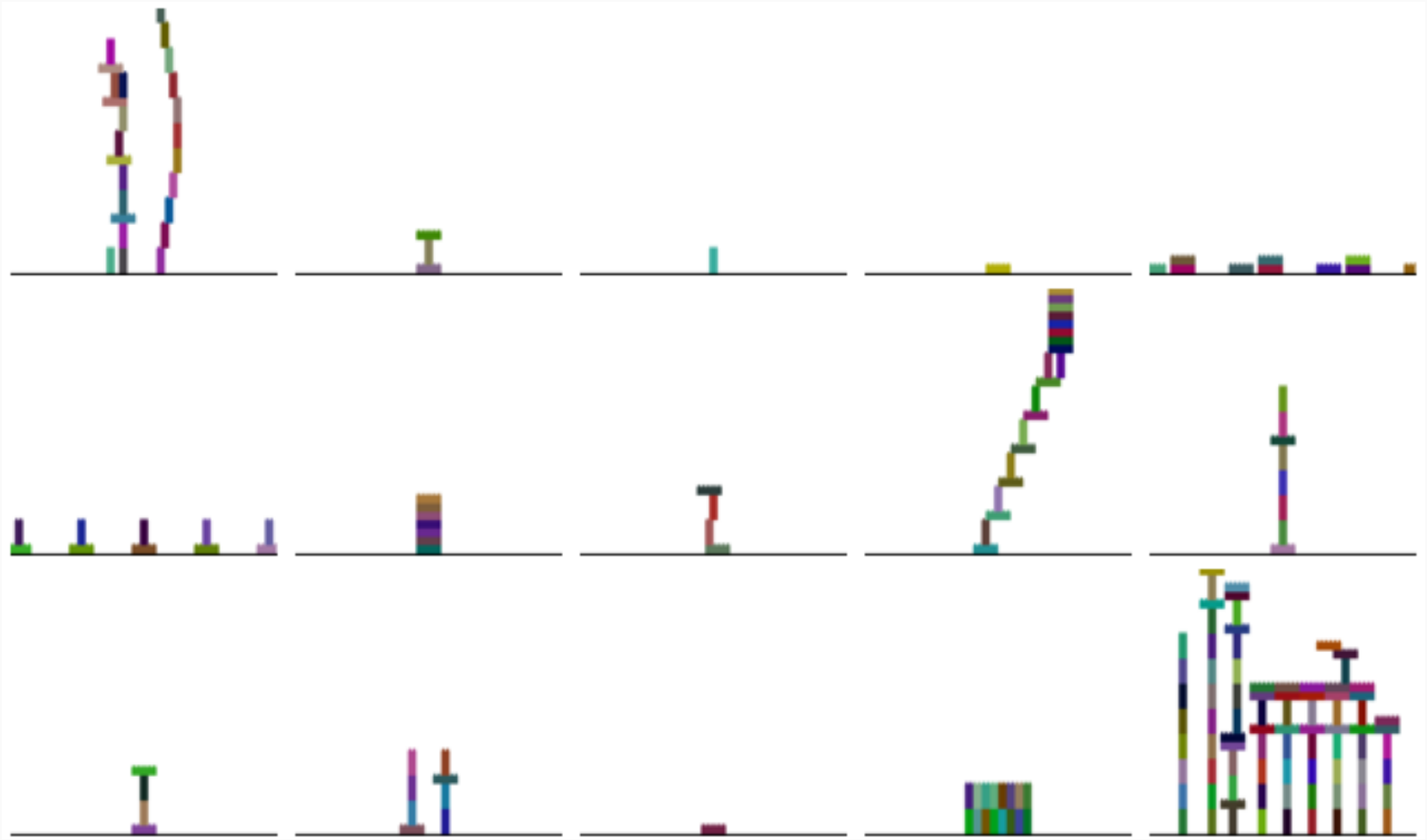
pyramid(h)



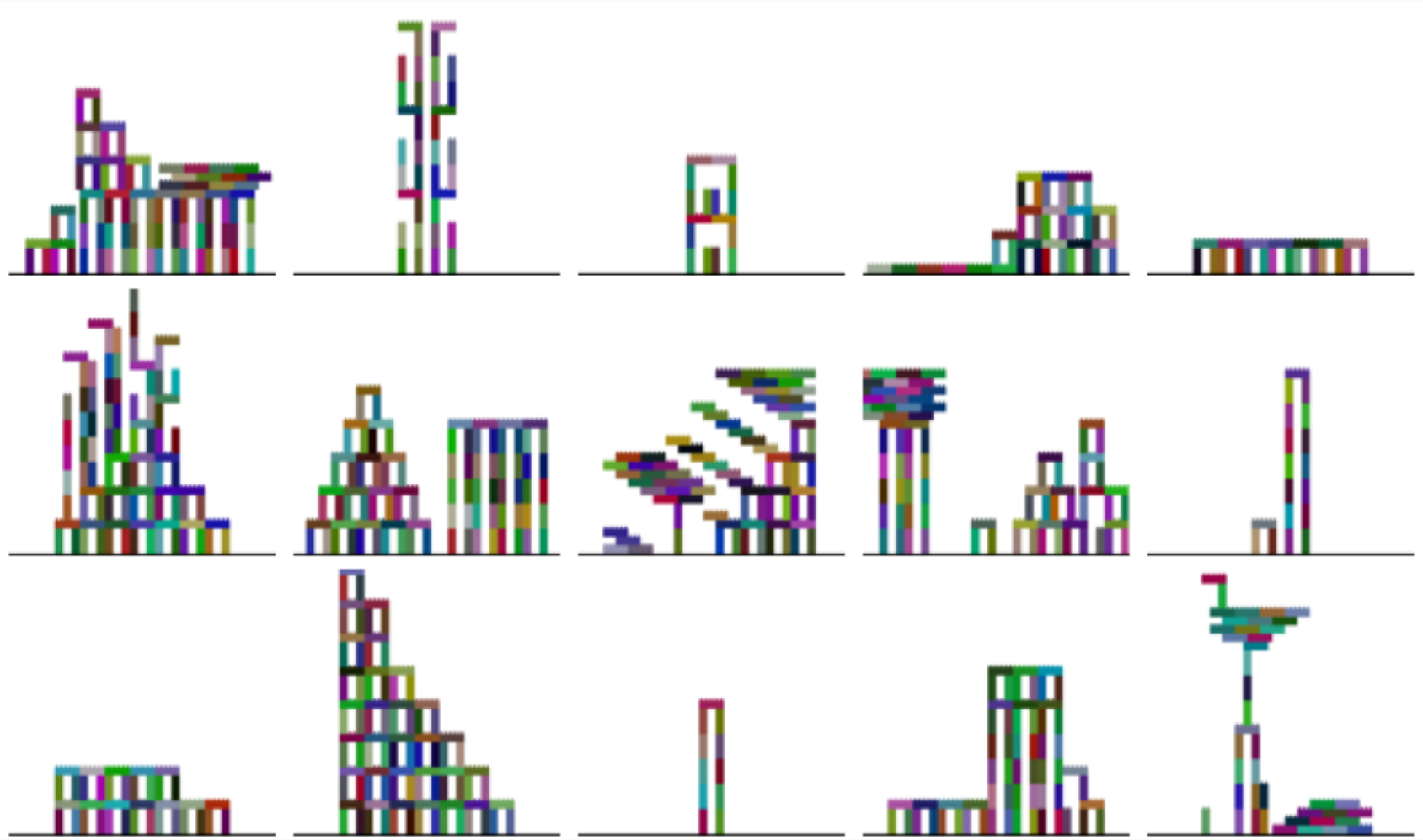
bridge(w, h)



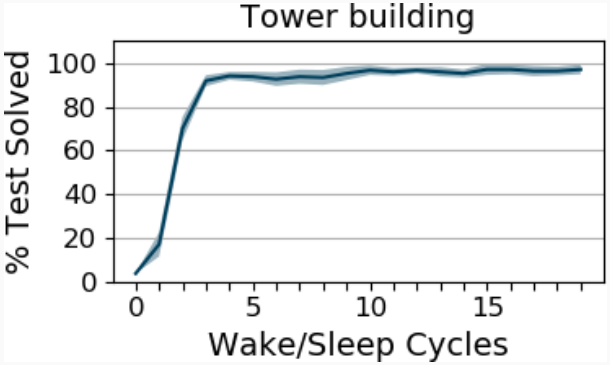
Dreams before learning



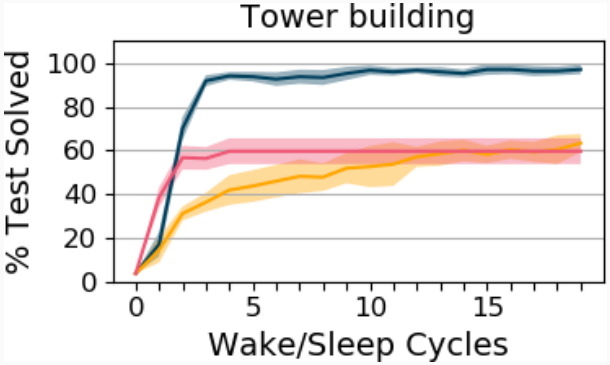
Dreams after learning

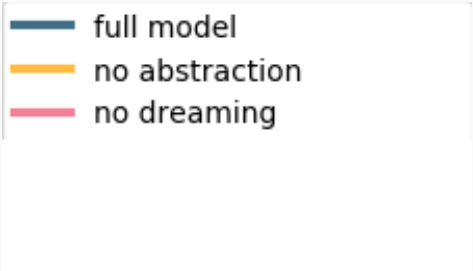


Learning dynamics

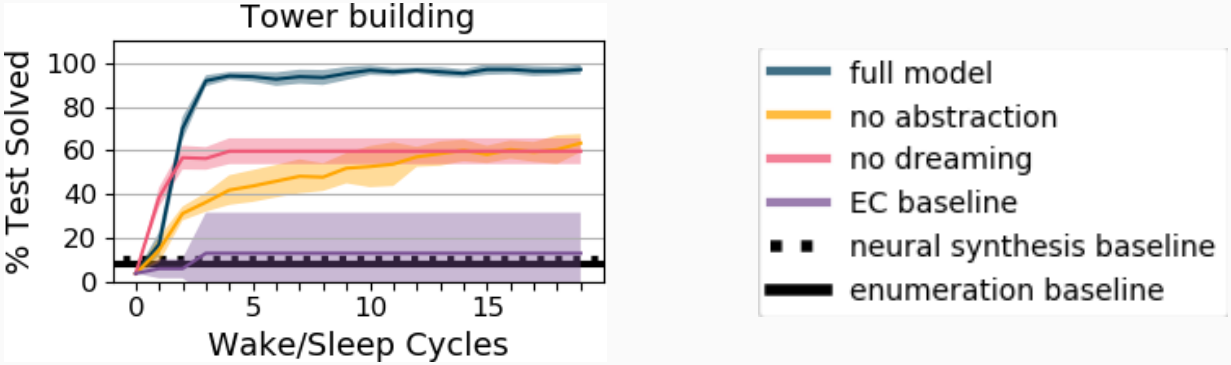


Learning dynamics



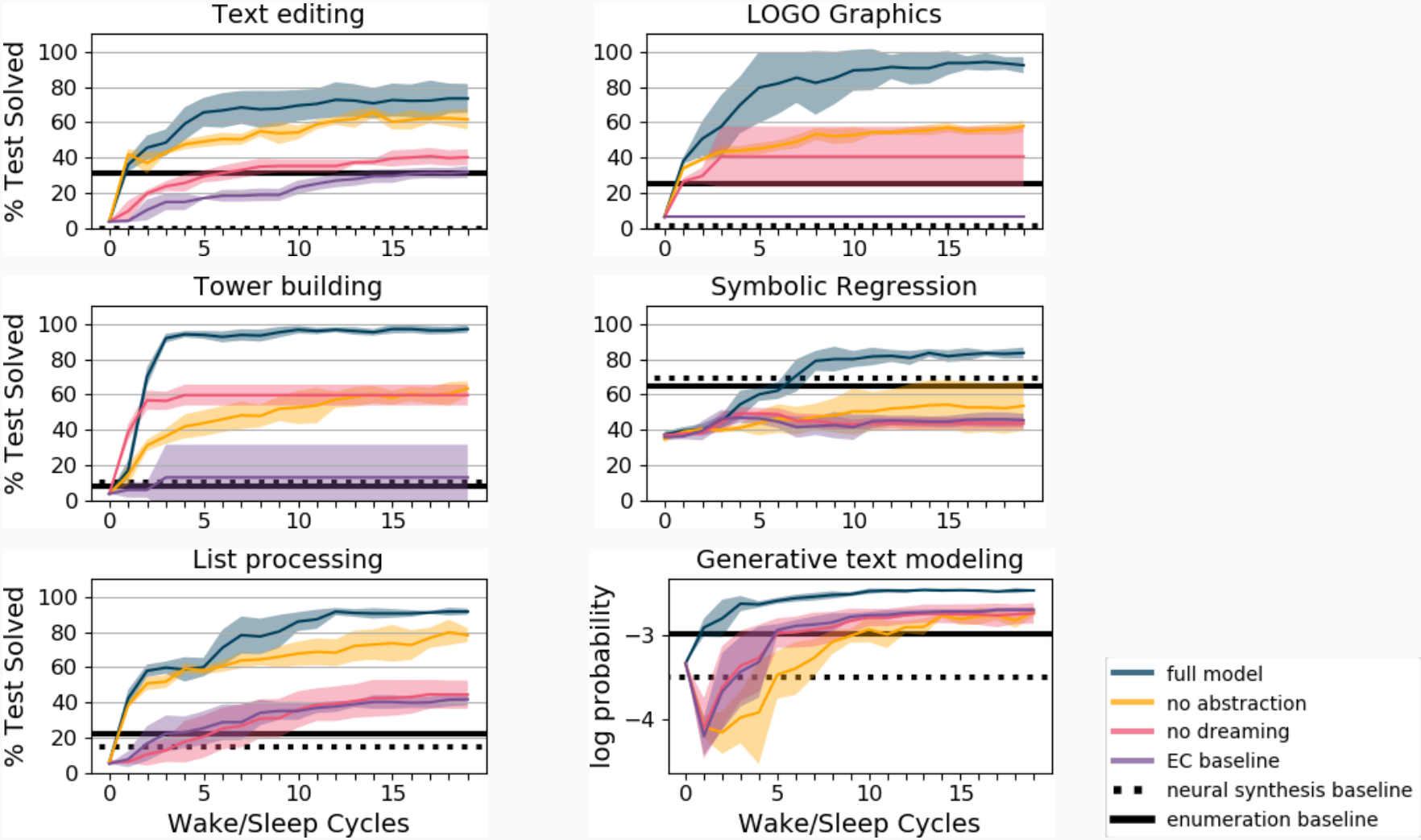


Learning dynamics

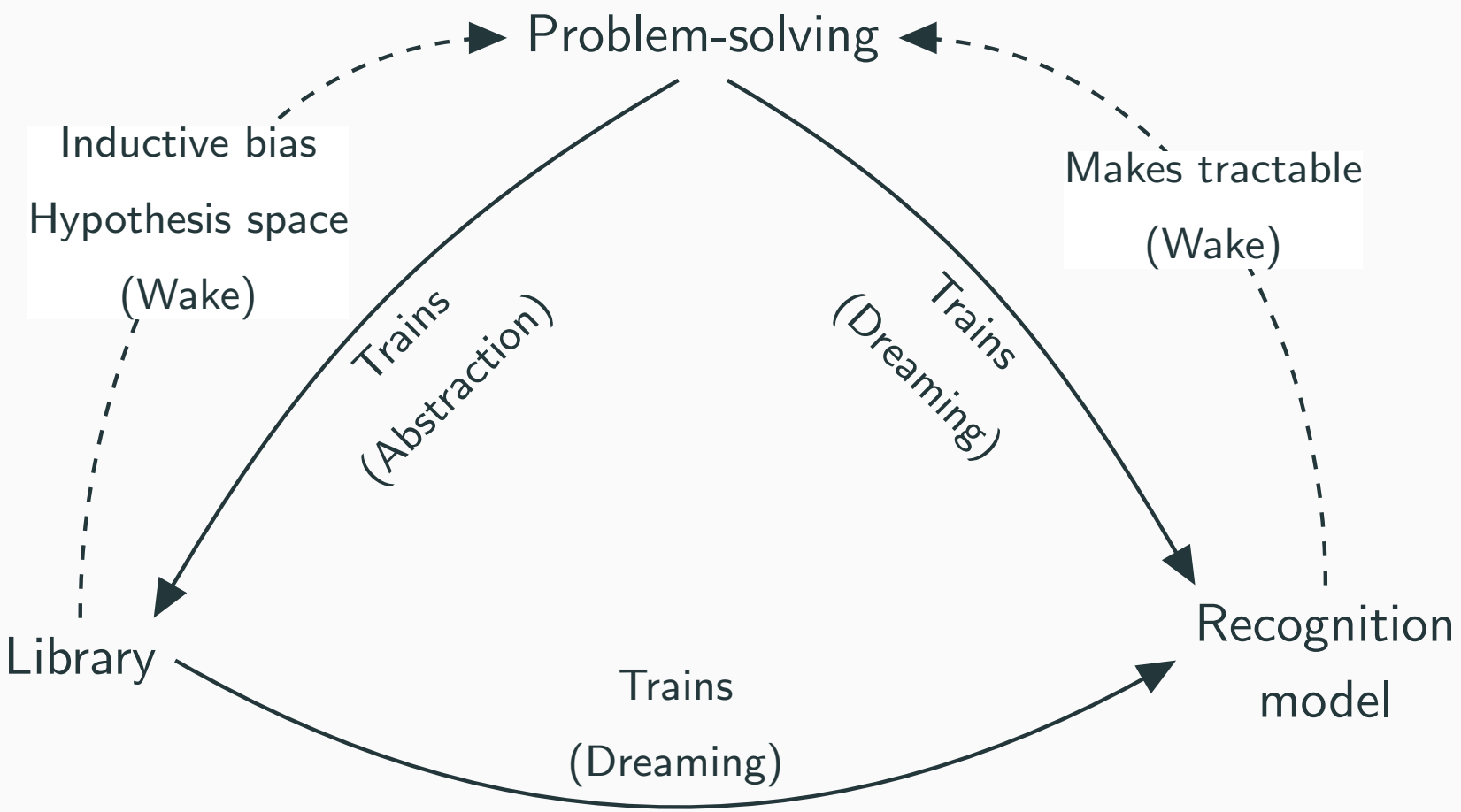


baselines: Exploration-Compression, EC (Dechter et al. 2013)
neural program synthesis, RobustFill (Devlin et al. 2017)
24 hours of brute-force enumeration

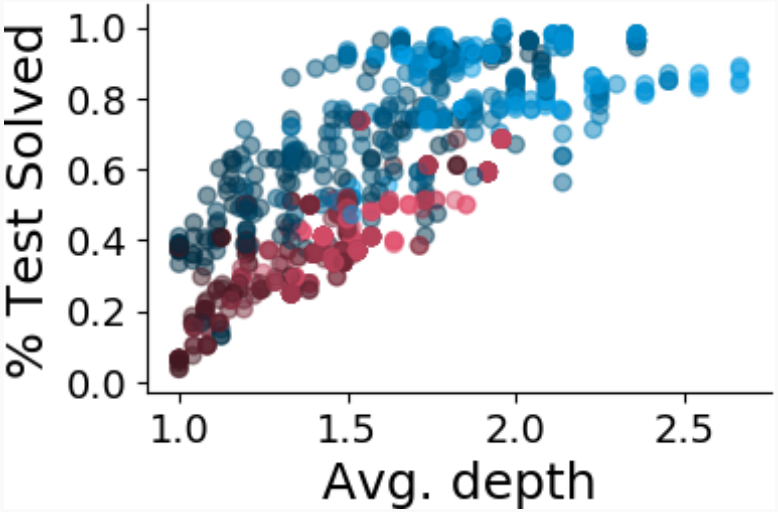
Learning dynamics

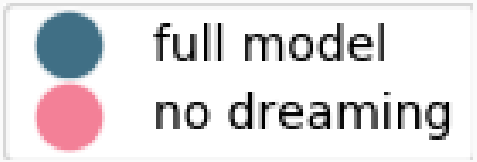


Synergy between dreaming and library learning



Evidence for dreaming bootstrapping better libraries

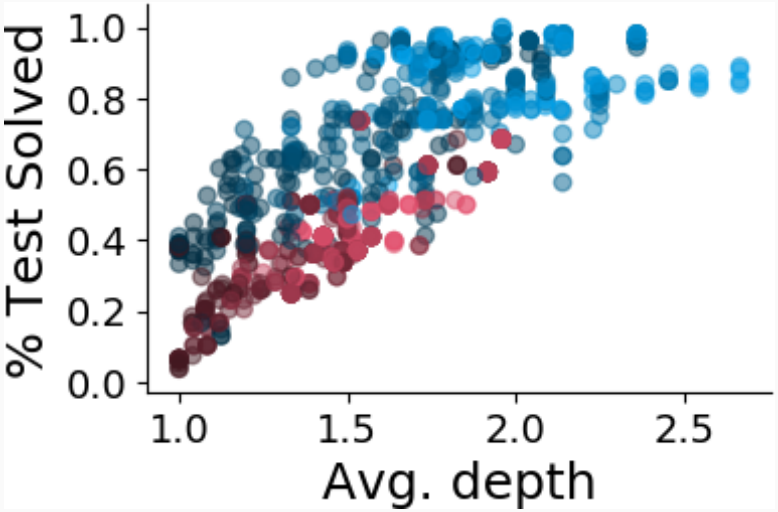


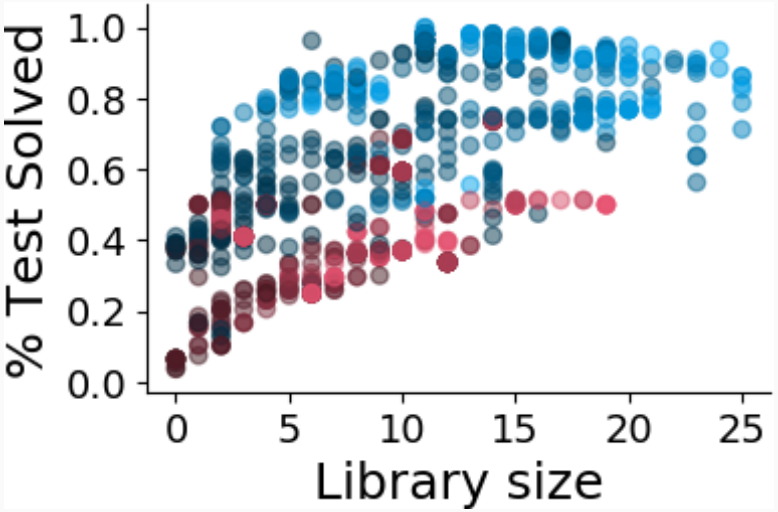


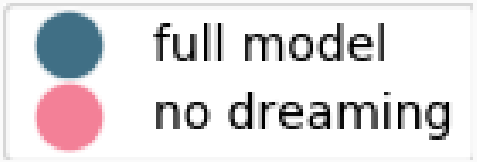
Darker: Early in learning

Lighter: Later in learning

Evidence for dreaming bootstrapping better libraries





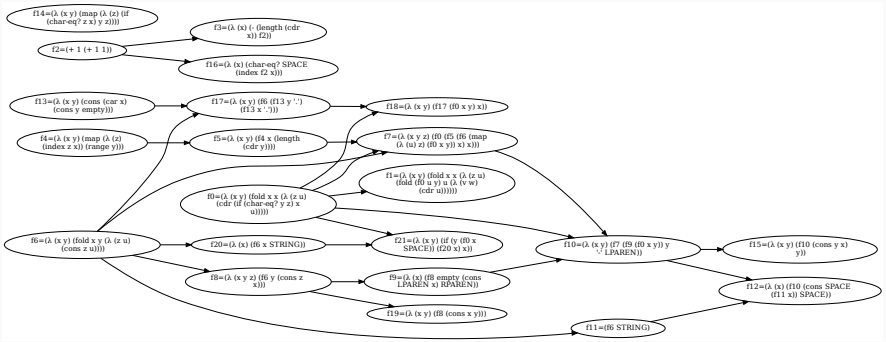


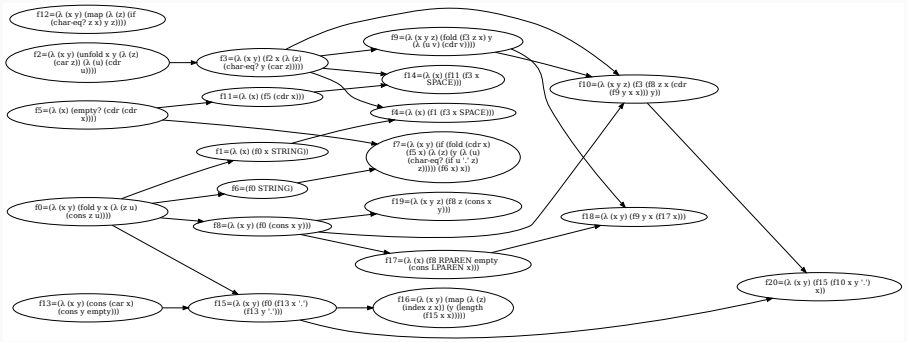
Darker: Early in learning

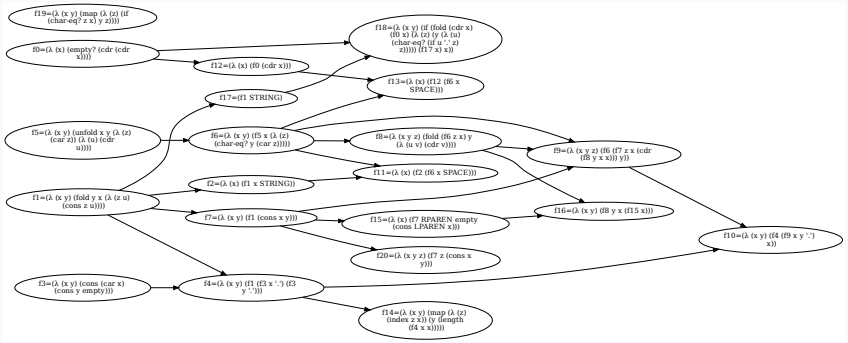
Lighter: Later in learning

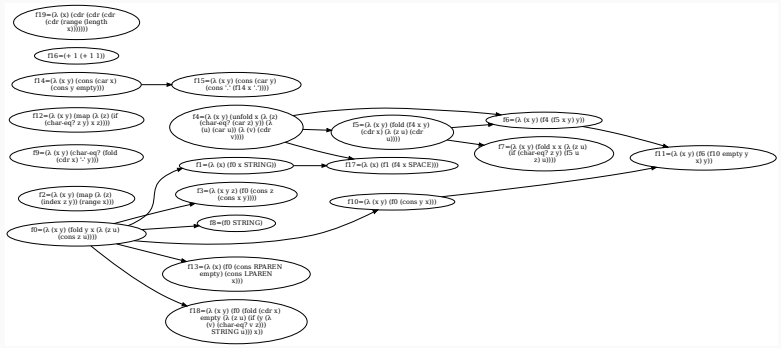
Library structure: Text Editing

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



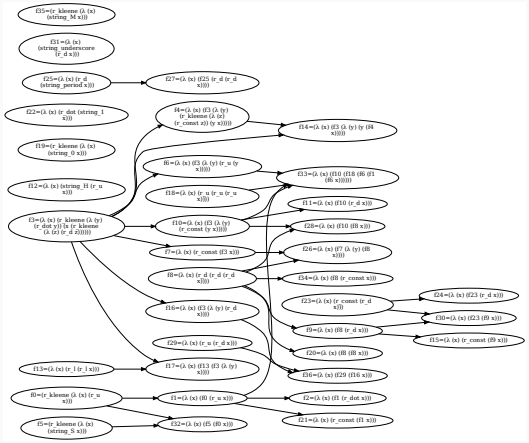


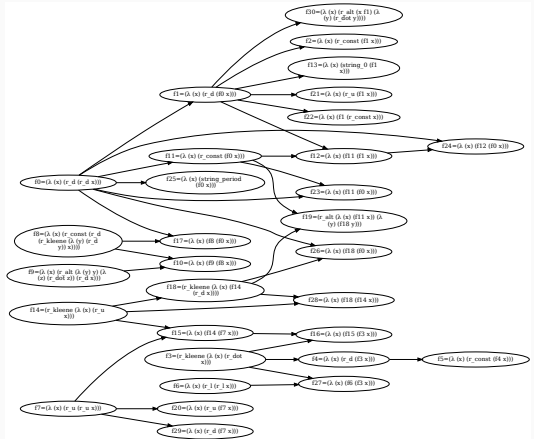




Library structure: Generating Text

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





From learning libraries, to learning languages

From learning libraries, to learning languages

functional programming $\rightarrow$ physics

From learning libraries, to learning languages

1950's Lisp $\rightarrow$ modern functional programming $\rightarrow$ physics

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

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

Work

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

Force in a Magnetic Field

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

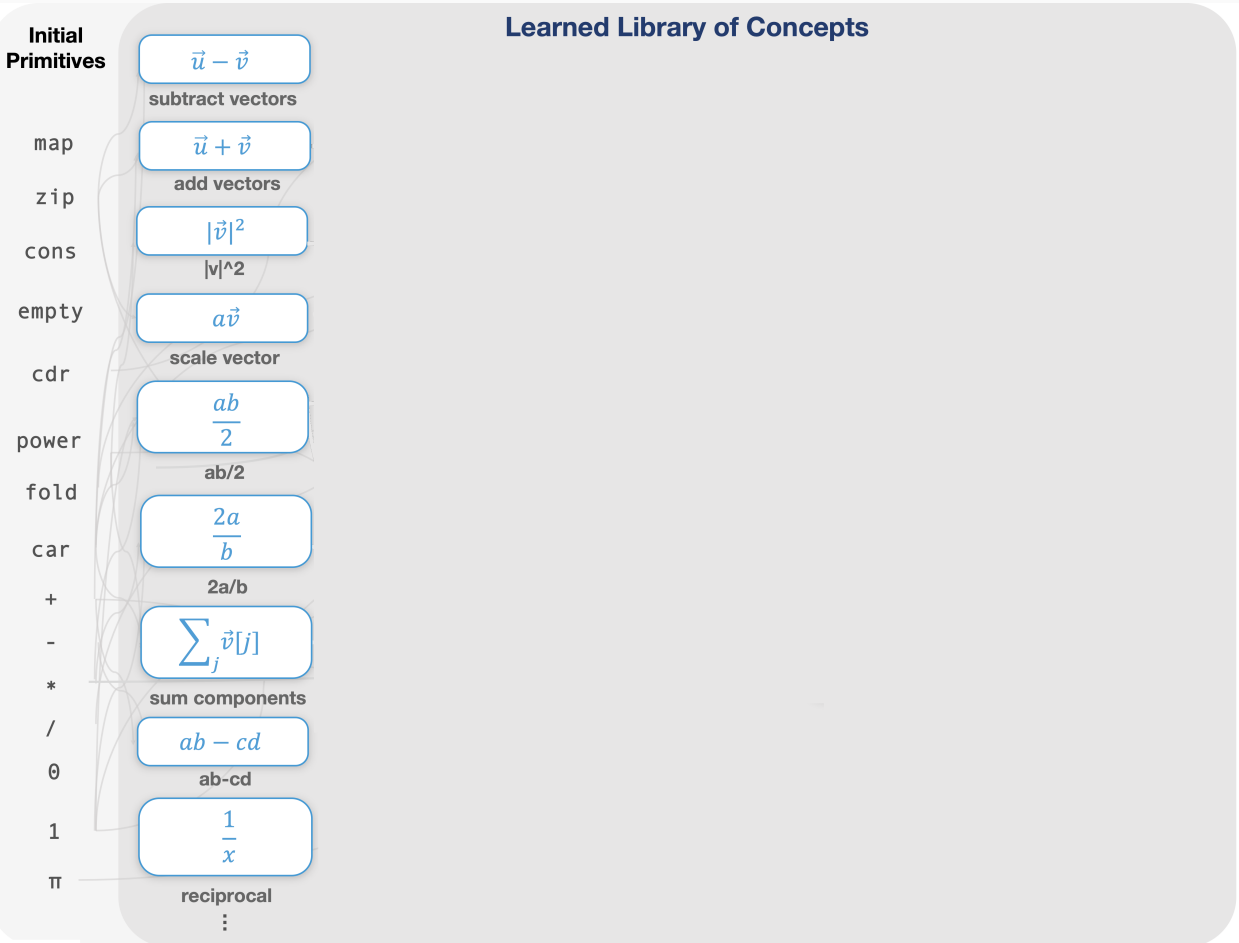
Kinetic Energy

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

Coulomb's Law

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1} - \vec{r_2}|^2}\widehat{r_1 - r_2}$$

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

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

Work

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

Force in a Magnetic Field

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

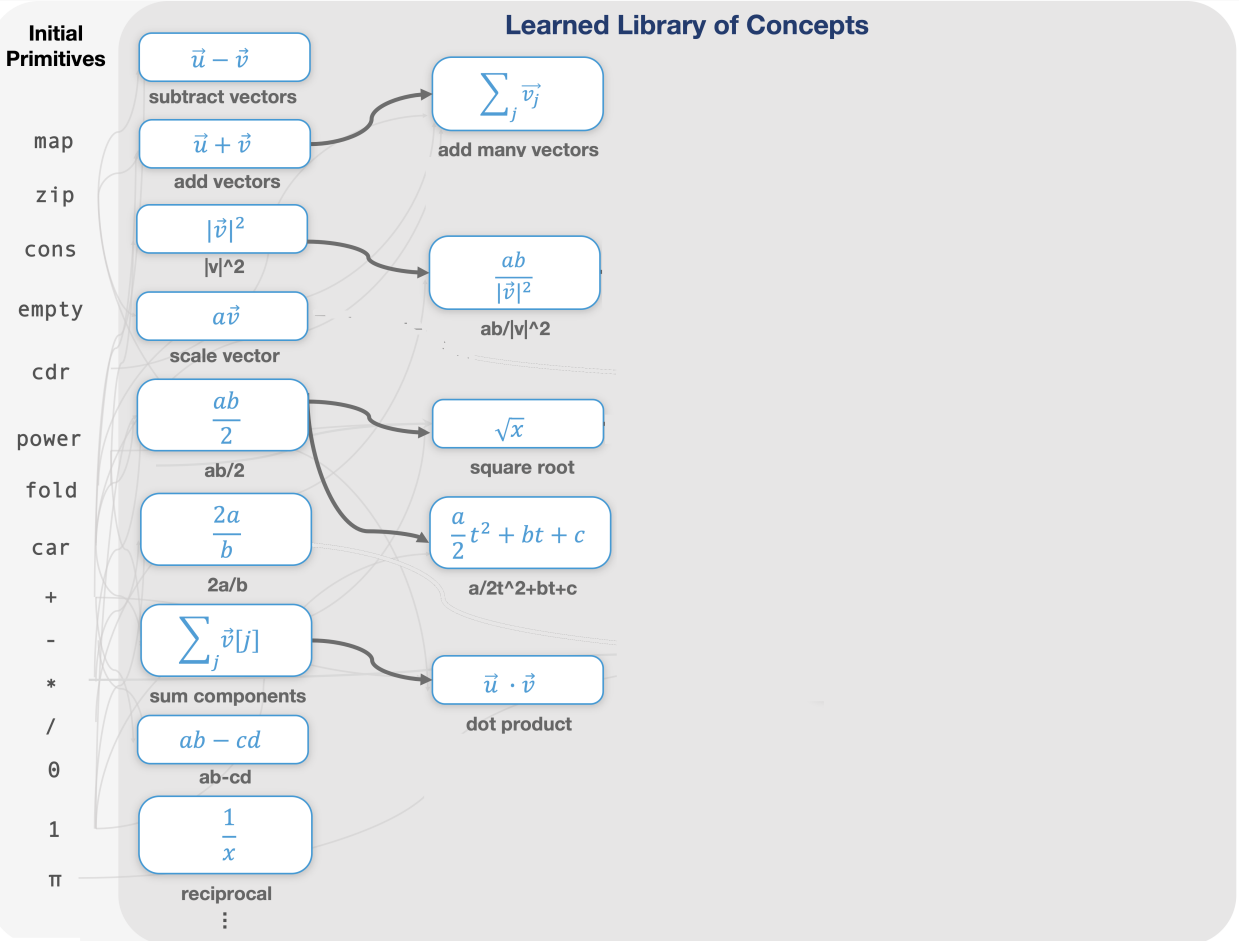
Kinetic Energy

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

Coulomb's Law

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1} - \vec{r_2}|^2}\widehat{r_1 - r_2}$$

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

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

Work

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

Force in a Magnetic Field

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

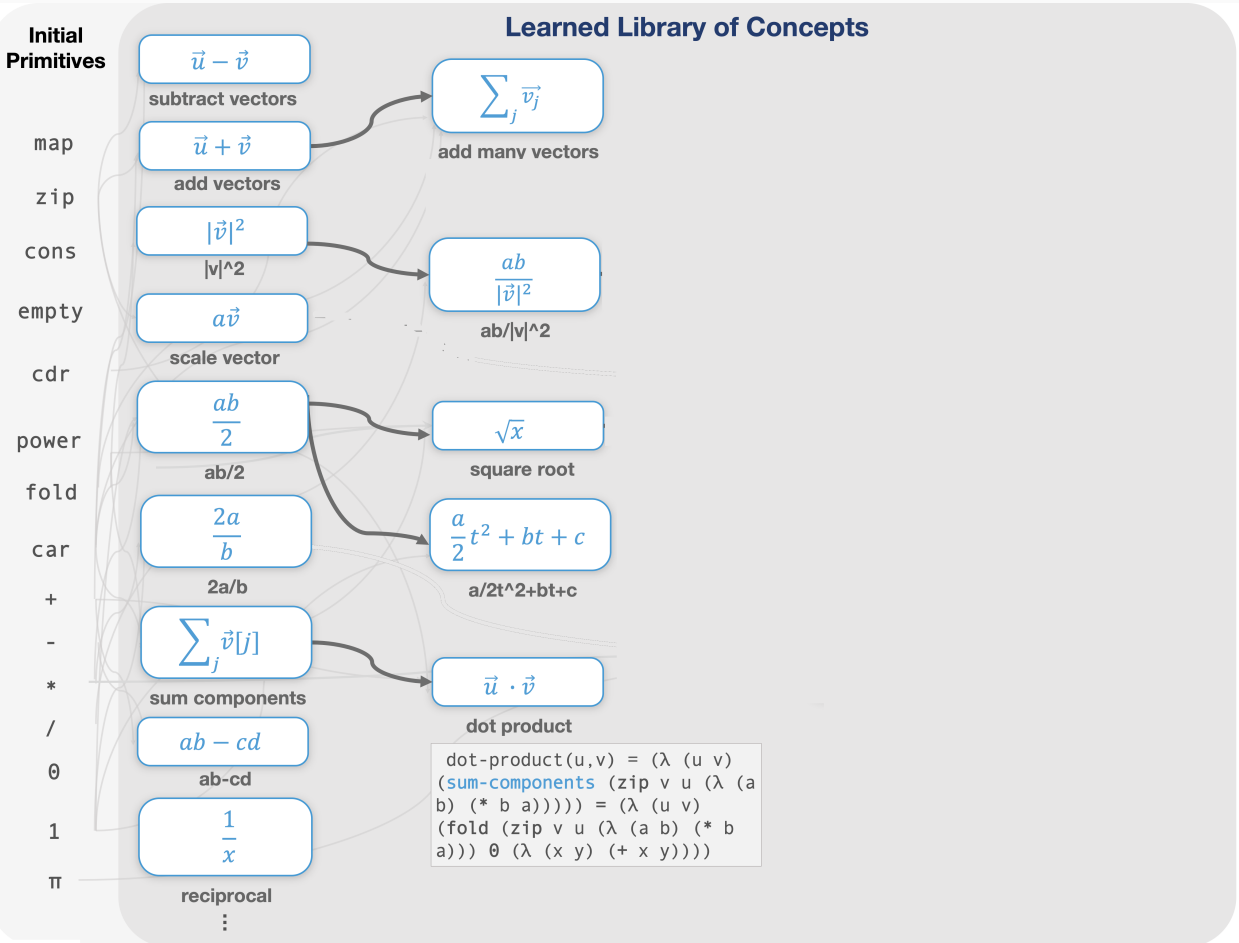
Kinetic Energy

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

Coulomb's Law

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1}-\vec{r_2}|^2}\widehat{r_1 - r_2}$$

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

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

Work

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

Force in a Magnetic Field

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

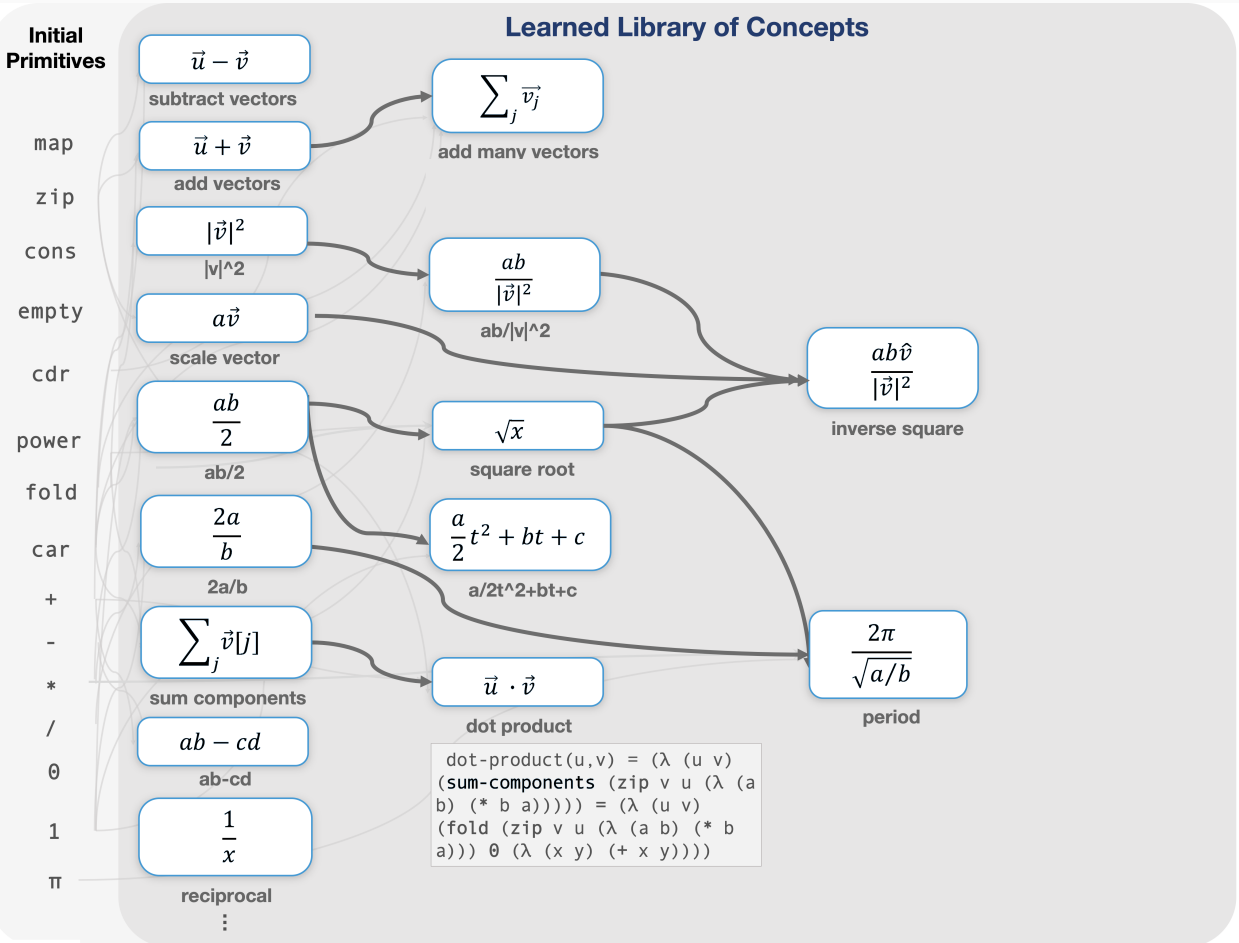
Kinetic Energy

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

Coulomb's Law

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1} - \vec{r_2}|^2}\widehat{r_1 - r_2}$$

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

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

Work

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

Force in a Magnetic Field

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

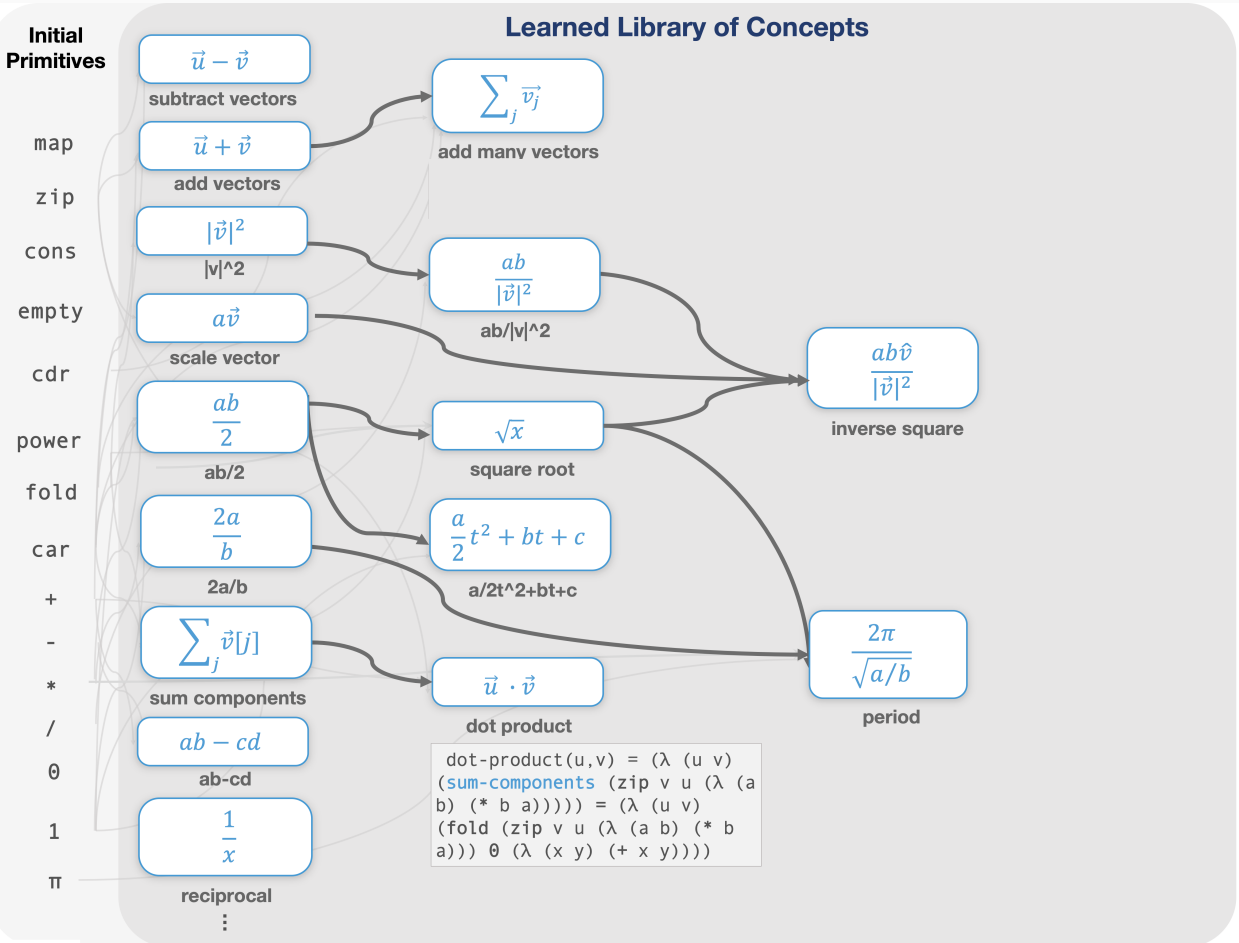
Kinetic Energy

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

Coulomb's Law

$$\vec{F}\propto\frac{q_1 q_2}{|\vec{r_1}-\vec{r_2}|^2}\widehat{r_1-r_2}$$

Growing languages for vector algebra and physics



Physics Equations

Newton's Second Law

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

```
(scale-vector(reciprocal m)
(add-many-vectors Fs))
```

Parallel Resistors

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

```
(reciprocal (sum-components
(map (λ(r) (reciprocal r))
Rs)))
```

Work

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

```
(dot-product F d)
```

Force in a Magnetic Field

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

```
(* q (ab-cd v_x b_y v_y b_x))
```

Kinetic Energy

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

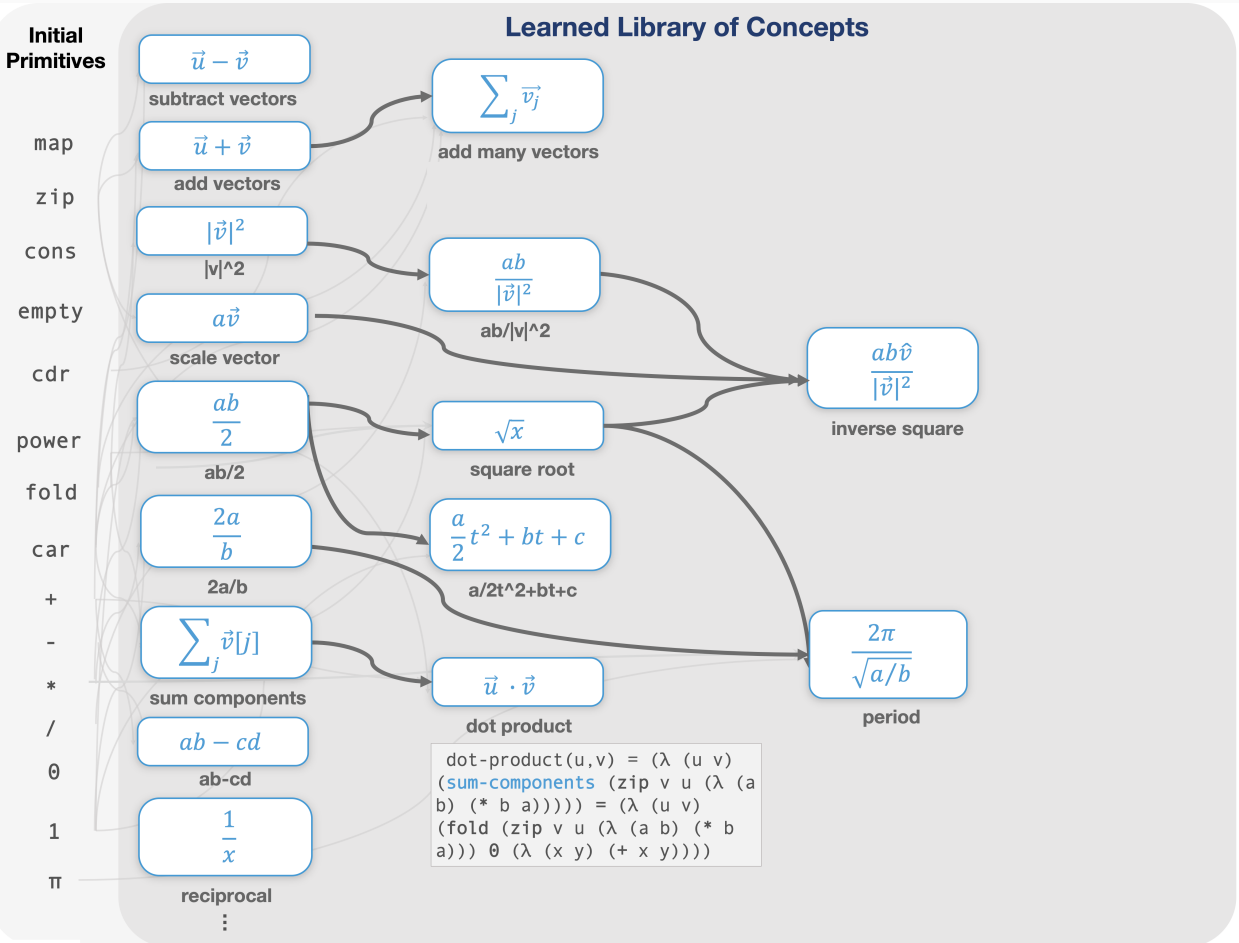
```
(ab/2 m (|v|^2 v))
```

Coulomb's Law

$$\vec{F}\propto\frac{q_1 q_2}{|\vec{r_1}-\vec{r_2}|^2}\widehat{r_1-r_2}$$

```
(inverse-square q_1 q_2
(subtract-vectors r_1 r_2))
```

Growing languages for vector algebra and physics



Physics Equations

Newton's Second Law

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

```
(scale-vector(reciprocal m)
(add-many-vectors Fs))
```

Parallel Resistors

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

```
(reciprocal (sum-components
(map (λ(r) (reciprocal r))
Rs)))
```

Work

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

```
(dot-product F d)
```

Force in a Magnetic Field

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

```
(* q (ab-cd v_x b_y v_y b_x))
```

Kinetic Energy

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

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

Coulomb's Law

$$\vec{F}\propto\frac{q_1 q_2}{|\vec{r_1}-\vec{r_2}|^2}\widehat{r_1-r_2}$$

```
(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))) 0 (λ (b
c) (+ (* b b)    c)))) (/ (* 1
1) (+ 1 1))) y) (fold (zip z u
(λ (d e)  (- d e))) 0 (λ (f g)
(+ (* f f) g)))) v)) (zip z u
(λ (h i) (- h i)))))
```

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

Recursive Programming Algorithms

Stutter

[■■] → [■■■■]
[■■■] → [■■■■■■]

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

Take every other

[■■■■] → [■■]
[■■■■■■] → [■■■]

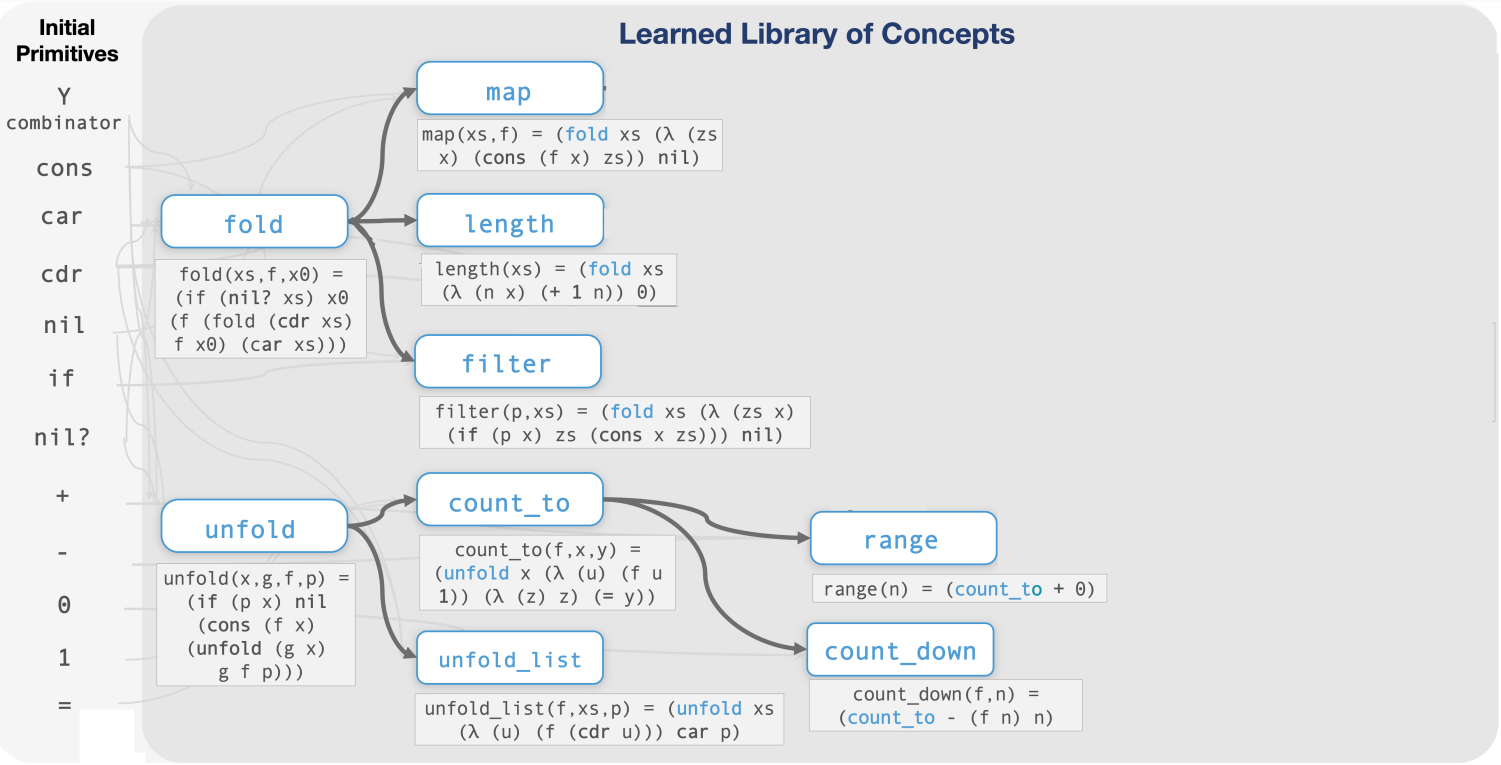
List lengths

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

List differences

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

Growing a language for recursive programming



Recursive Programming Algorithms

Stutter

[■■] → [■■■■]
[■■■] → [■■■■■■]

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

Take every other

[■■■■] → [■■]
[■■■■■■] → [■■■]

```
(unfold_list cdr A nil?)
```

List lengths

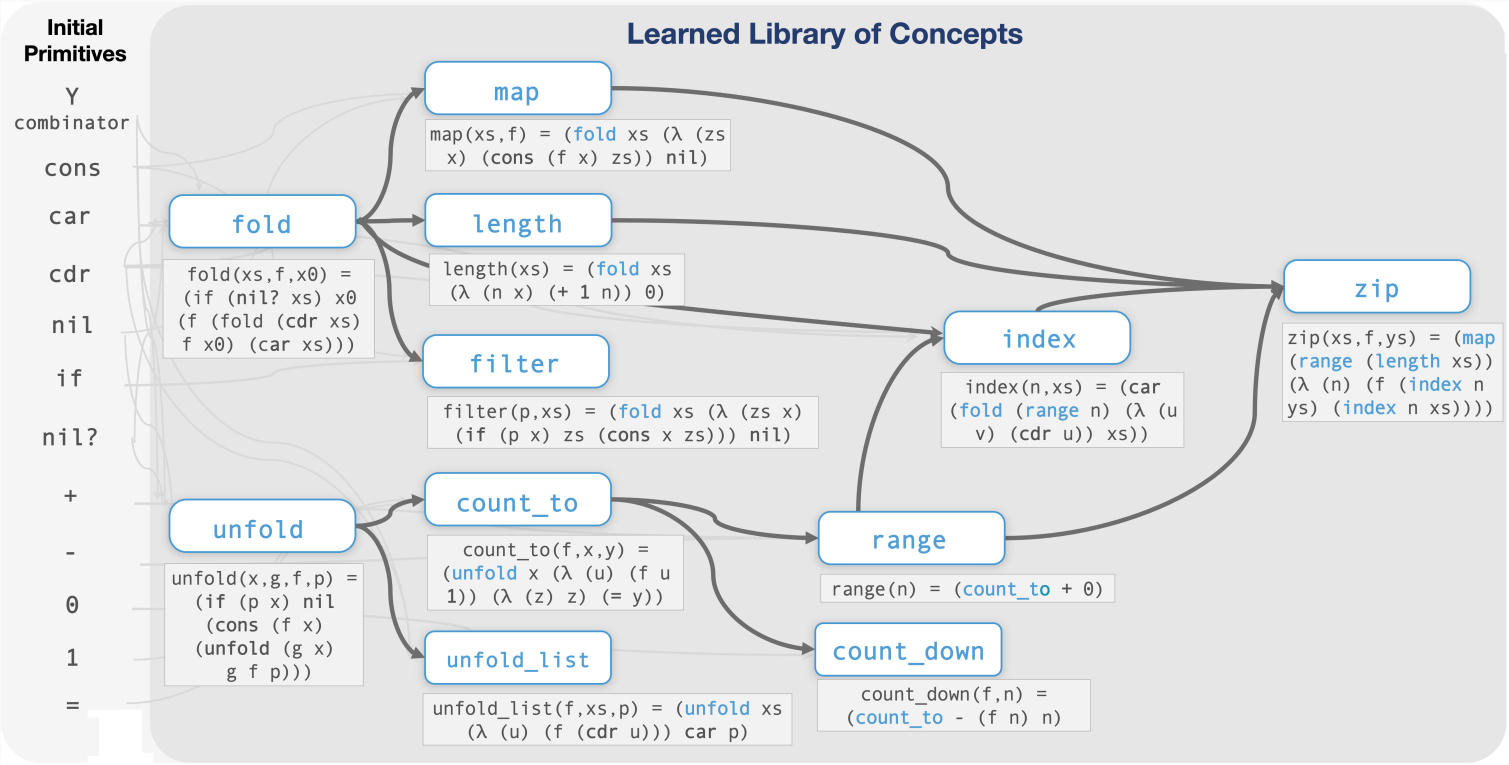
[[■■■], [■]] → [3 1]
[[■■], [], [■]] → [2 0 1]

```
(map A length)
```

List differences

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

Growing a language for recursive programming



Recursive Programming Algorithms

Stutter

[■■] → [■■■■]
[■■■] → [■■■■■■]

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

Take every other

[■■■■] → [■■]
[■■■■■■] → [■■■]

```
(unfold_list cdr A nil?)
```

List lengths

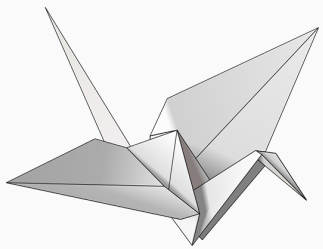
[[■■■], [■]] → [3 1]
[[■■], [], [■]] → [2 0 1]

```
(map A length)
```

List differences

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

```
(zip A - B )
```



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. 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

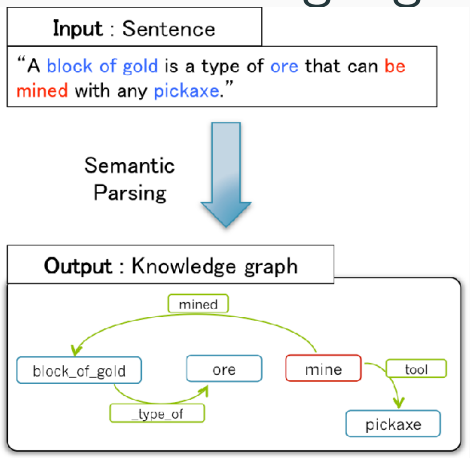
learning to learn

model discovery

the future

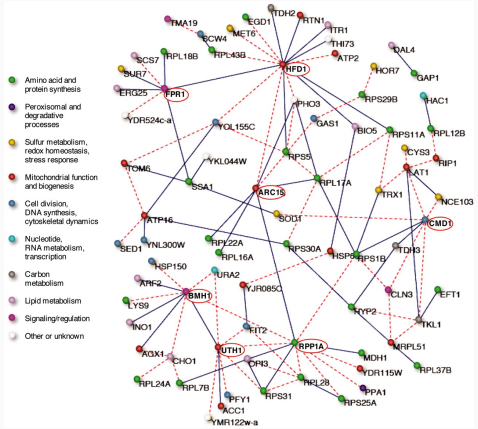
What's in reach

Library learning+ Natural language

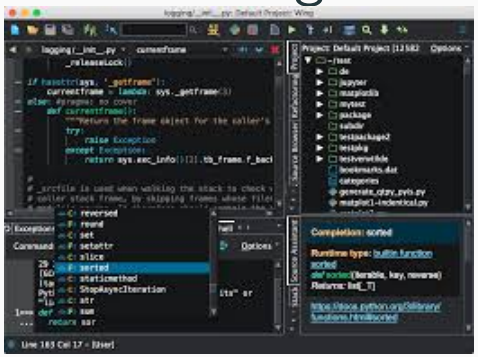


Modeling the physical world

Computer-Aided Science

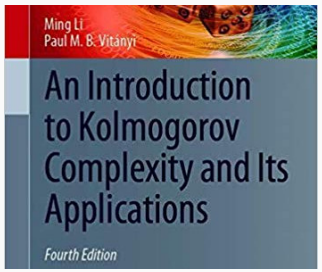


Synthesis for software engineers



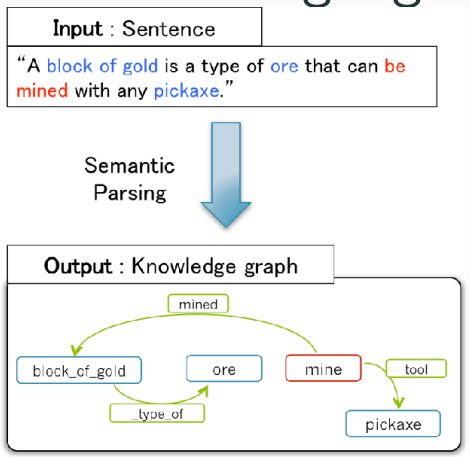
Theory for program induction

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

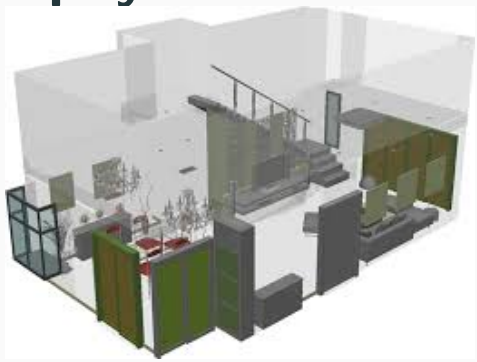


What's in reach

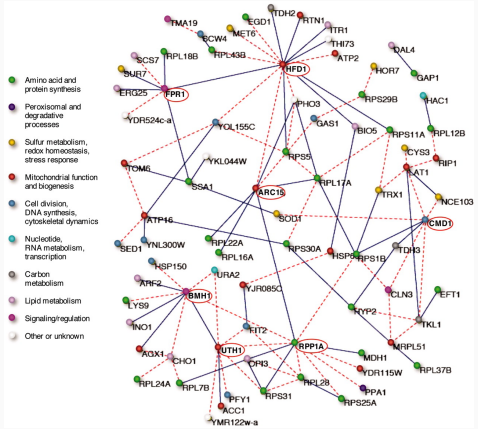
Library learning+ Natural language



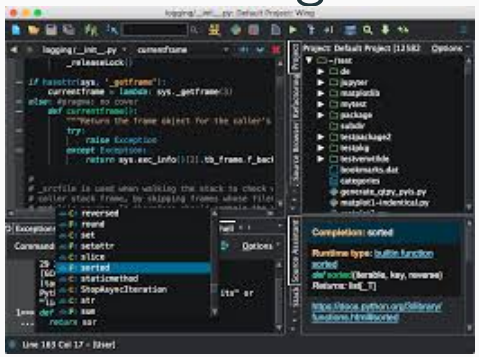
Modeling the physical world



Computer-Aided Science

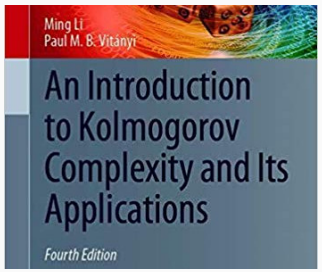


Synthesis for software engineers



Theory for program induction

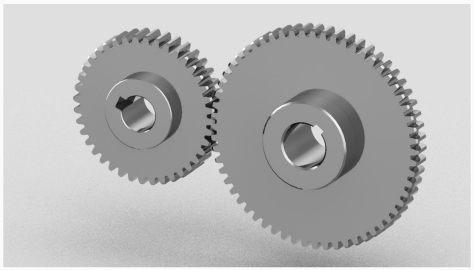
$$P_M(x) = \sum_{i=1}^{\infty} 2^{-|s_i(x)|}$$



Models of the physical world

hinge

gear



doorknob

Modeling the physical world

Built in | Learned Library | The world

2D geometry

symmetry

...

3D geometry

extrude

...

Rigid body

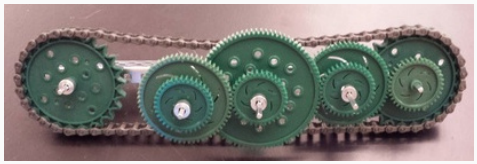
dynamics

...

Programming

for loop

...



Modeling the physical world

Built in | Learned Library | The world

2D geometry

symmetry

...

3D geometry

extrude

...

Rigid body

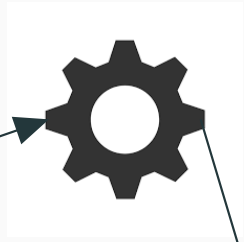
dynamics

...

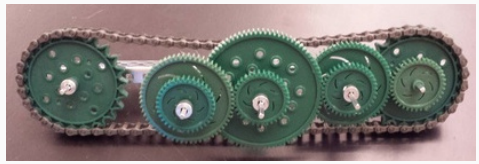
Programming

for loop

...



gear(n, ℓ, r)



Modeling the physical world

Built in | Learned Library | The world

2D geometry

symmetry

...

3D geometry

extrude

...

Rigid body

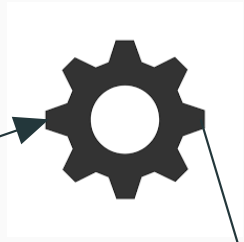
dynamics

...

Programming

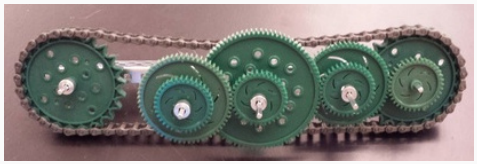
for loop

...



gear(n,ℓ,r)

geartrain(K,$\vec{n}$,$\vec{\ell}$,$\vec{r}$)



Modeling the physical world

Built in | Learned Library | The world

2D geometry

symmetry

...

3D geometry

extrude

...

Rigid body

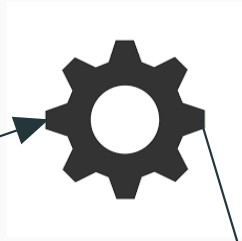
dynamics

...

Programming

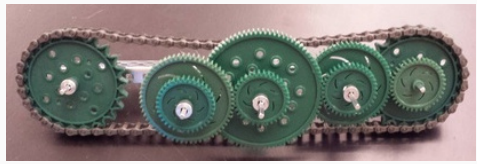
for loop

...



gear(n,ℓ,r)

geartrain(K,$\vec{n}$,$\vec{\ell}$,$\vec{r}$)



What's far off, but worthwhile?

A far-off, long-term goal for AI: Making machines that can learn to solve the full span of problems humans can master, while also learning from modest amounts of experience, and acquiring knowledge that can be interpreted and extended

using new devices

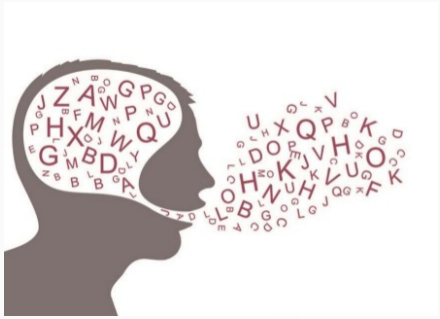
play

coding

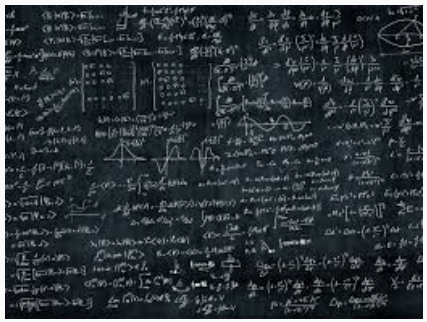
```
(MEMBER
 (LAMBDA (X L)
  (COND  ((NULL L) NIL)
         ((EQ X (FIRST L)) T)
         (T (MEMBER X (REST L))))))
```

Allen, Anatomy of Lisp, 1975

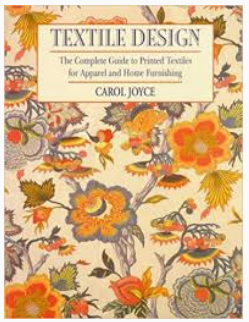
language



science



design



What's far off, but worthwhile?

A far-off, long-term goal for AI: Making machines that can learn to solve the full span of problems humans can master, while also learning from modest amounts of experience, and acquiring knowledge that can be interpreted and extended

List Processing

Sum List

```
[1 2 3] → 6
[4 6 8 1] → 17
```

Double

```
[1 2 3 4] → [2 4 6 8]
[6 5 1] → [12 10 2]
```

Check Evens

```
[0 2 3] → [T T F]
[2 4 9 6] → [T T F T]
```

Text Editing

Abbreviate

```
Allen Newell → A.N.
Herb Simon → H.S.
```

Drop Last Characters

```
jabberwocky → jabberw
copycat → cop
```

Extract

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

Regexes

Phone Numbers

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

Currency

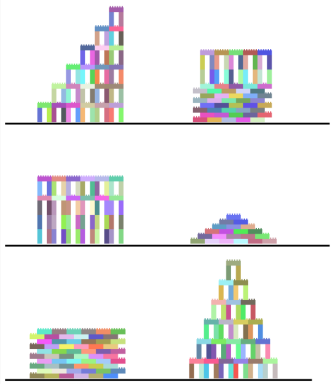
```
$100.25
$4.50
```

Dates

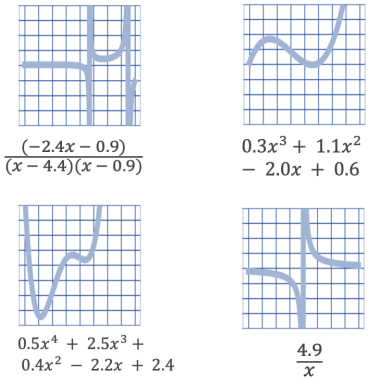
```
Y1775/0704
Y2000/0101
```

LOGO Graphics

Block Towers

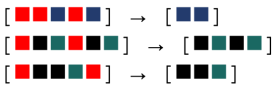


Symbolic Regression



Recursive Programming

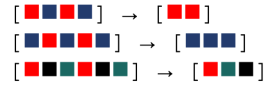
Filter



Length

Index List

Every Other



Physics

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

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

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1} - \vec{r_2}|^2}\widehat{r_1 - r_2}$$

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

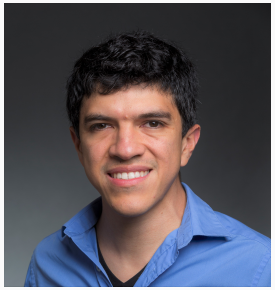
Collaborators

Tim O'Donnell

Josh Tenenbaum

Adam Albright

Armando Solar-Lezama



Max Nye

Cathy Wong

Yewen Pu

Dan Ritchie

Mathias Sable-Meyer

Lucas Morales

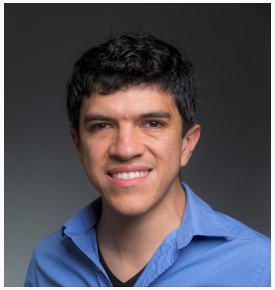
Collaborators

Tim O'Donnell

Josh Tenenbaum

Adam Albright

Armando Solar-Lezama



Max Nye

Cathy Wong

Yewen Pu

Dan Ritchie

Mathias Sable-Meyer

Lucas Morales

thank you