Program Induction:
Bridging AI and program synthesis

Kevin Ellis

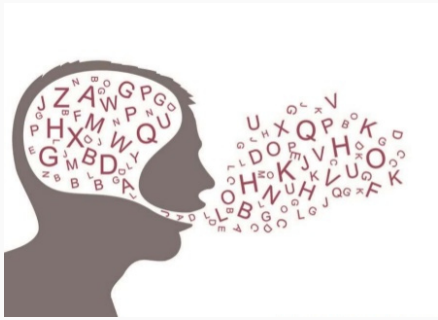
2020

MIT

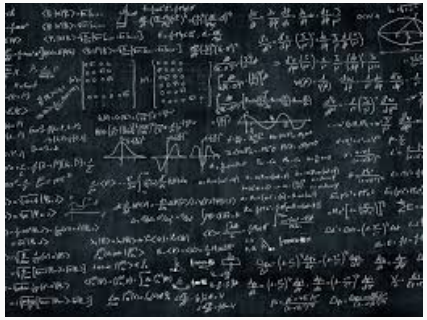
What computational problems are solved by intelligence?

an endless range of problems

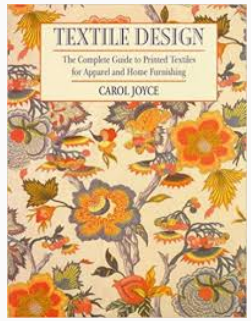
language



science

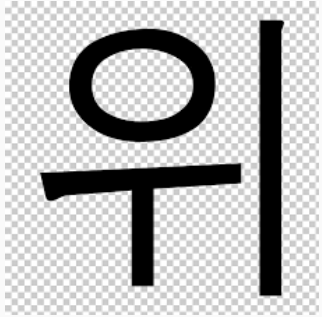


design



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 problems are solved by intelligence?

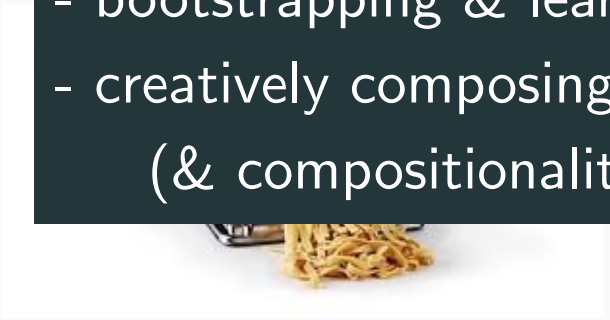
an endless range of problems

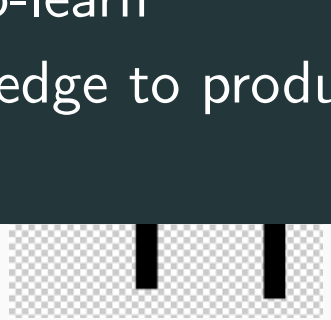
language

science

design

- learning from modest data/experience
- communicating & representing knowledge in understandable formats
- bootstrapping & learning-to-learn
- creatively composing knowledge to produce new concepts and artifacts (& compositionality)





engineering

play

What computational substrates can contribute to this picture?

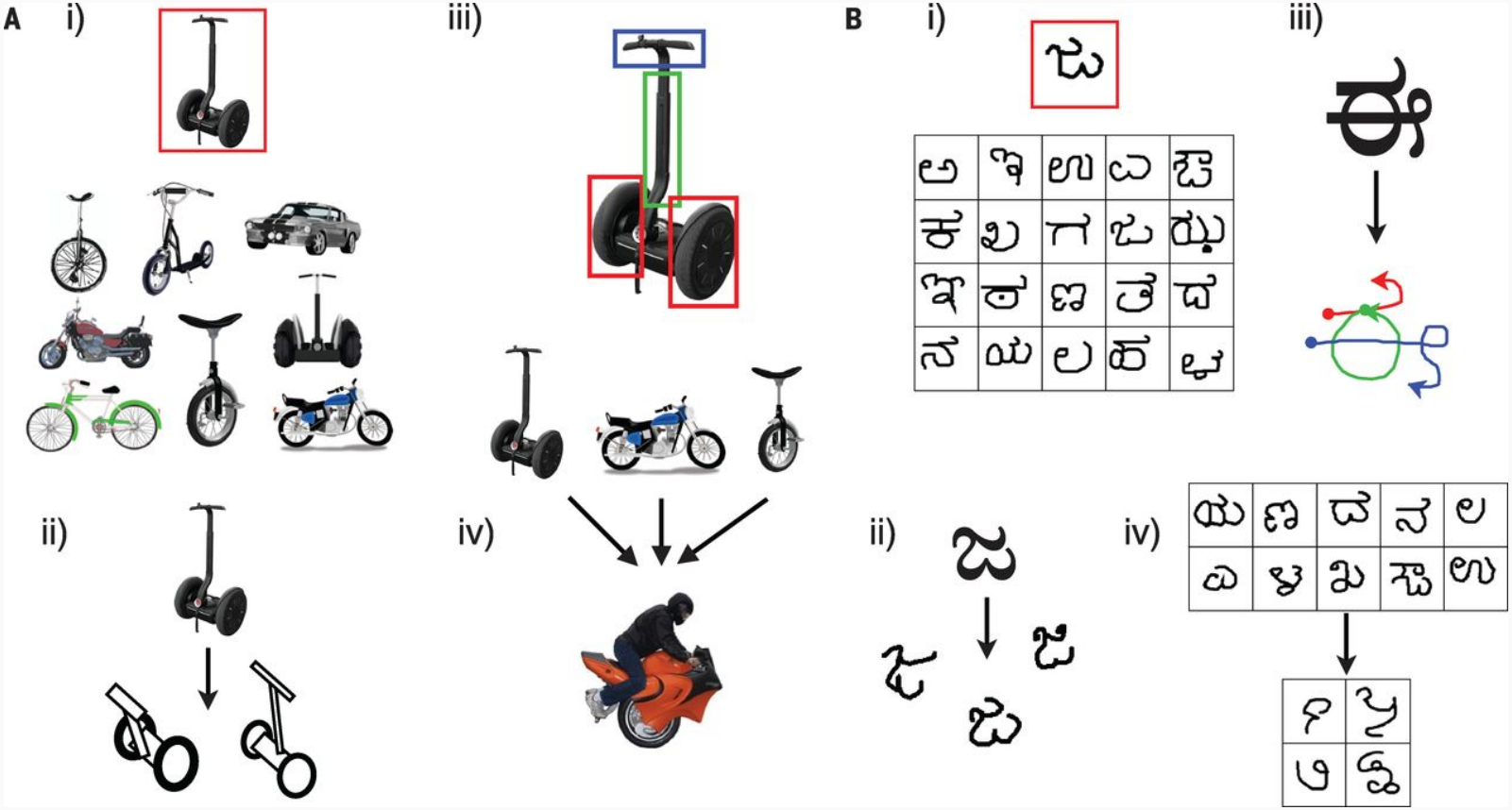
Three AI traditions: Symbolic, Probabilistic, Neural

What computational substrates can contribute to this picture?

Three AI traditions: Symbolic, Probabilistic, Neural

Program induction: roots in the works of Solomonoff, Fodor, McCarthy.

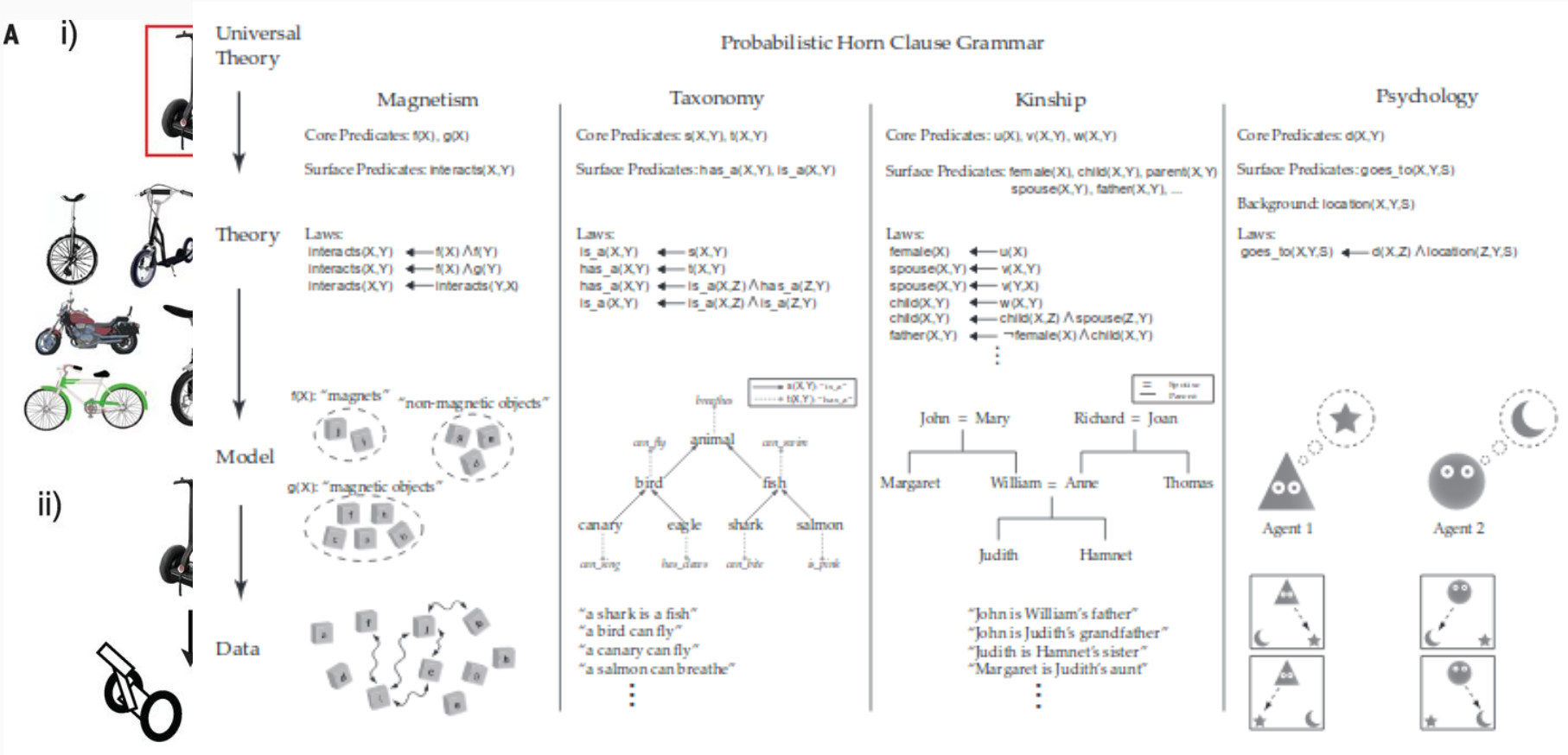
Combines and complements these three traditions.



What computational substrates can contribute to this picture?

Three AI traditions: Symbolic, Probabilistic, Neural

Program induction: roots in the works of Solomonoff, Fodor, McCarthy. Combines and complements these three traditions.

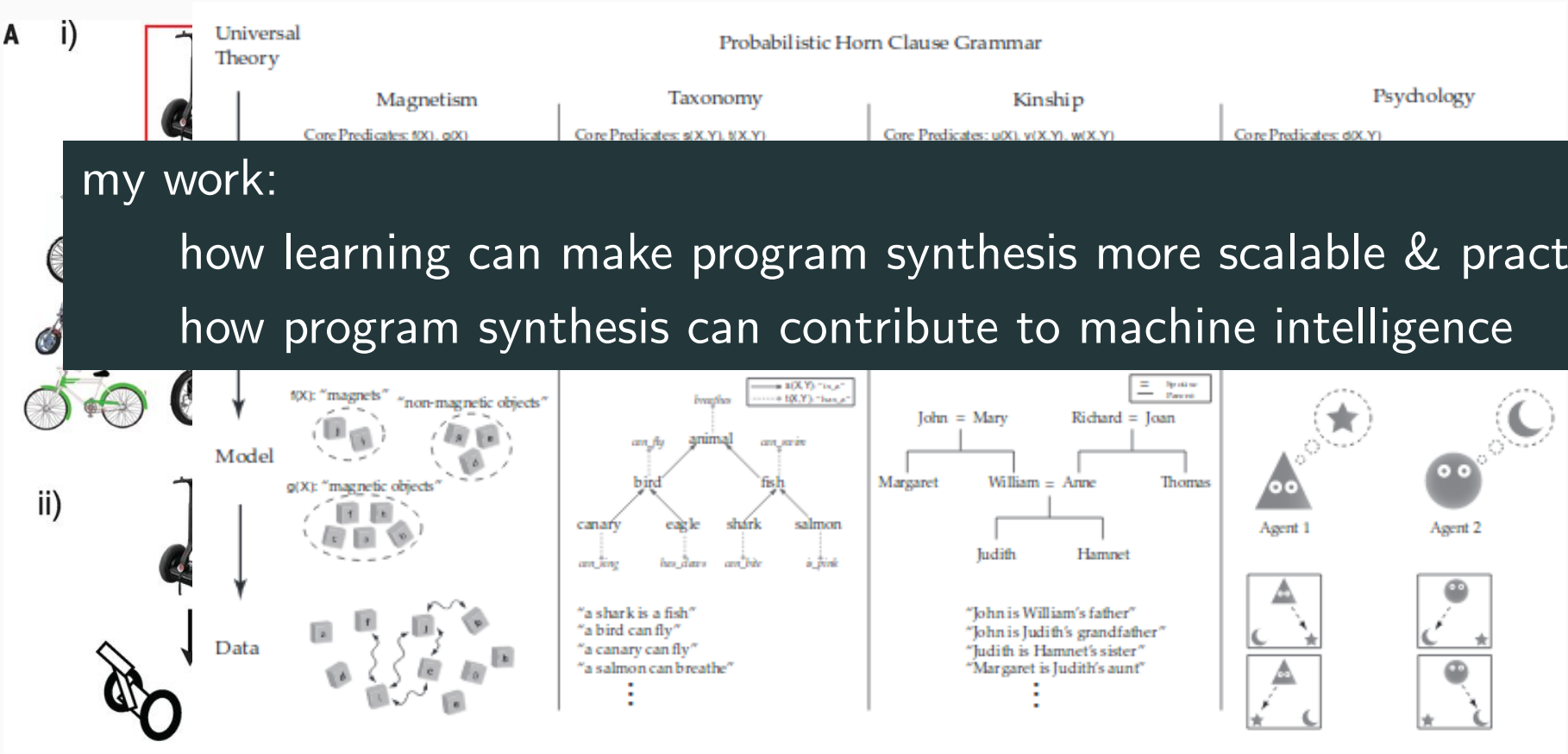


What computational substrates can contribute to this picture?

Three AI traditions: Symbolic, Probabilistic, Neural

Program induction: roots in the works of Solomonoff, Fodor, McCarthy.

Combines and complements these three traditions.



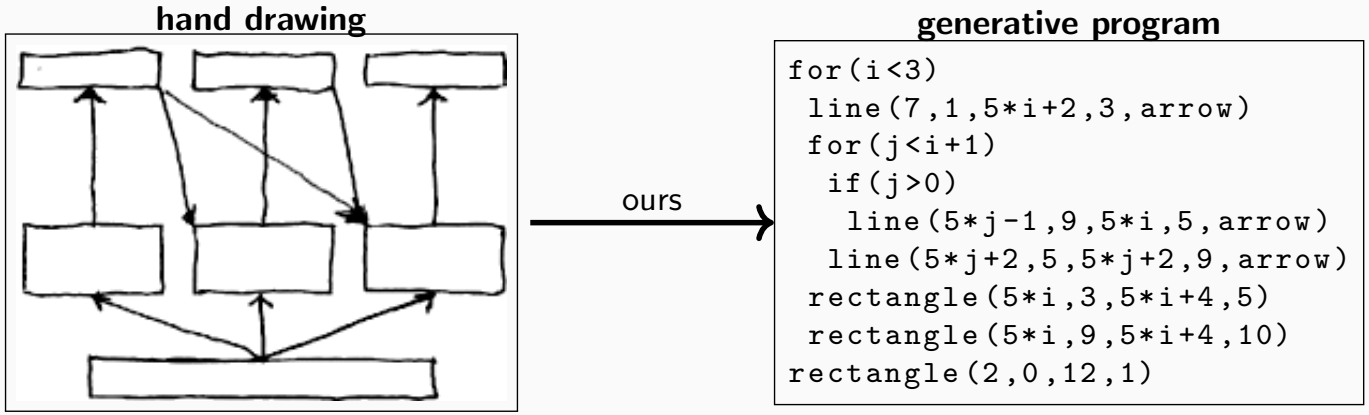
my work:

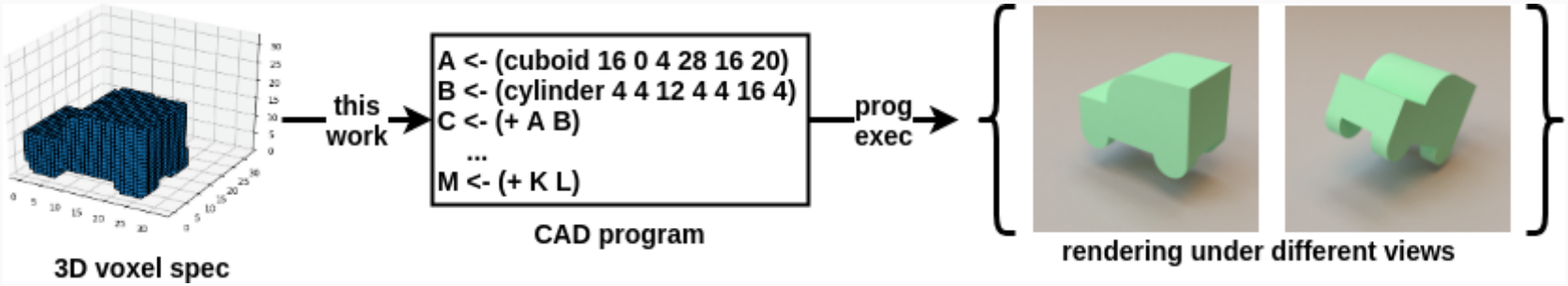
how learning can make program synthesis more scalable & practical

how program synthesis can contribute to machine intelligence

Perception, Learning-to-Learn, Synthesizing models

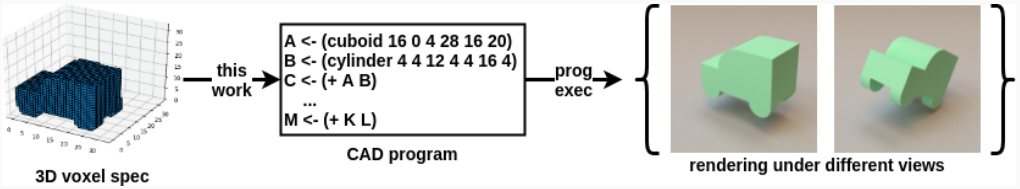
Theme #1: high-level scene understanding, pixels→programs





Perception, Learning-to-Learn, Synthesizing models

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



Theme #2: 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

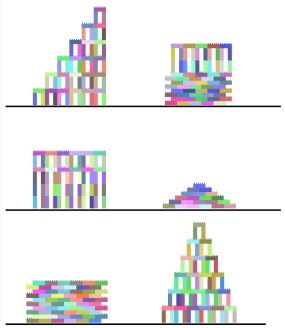
```
$100.25
$4.50
```

Dates

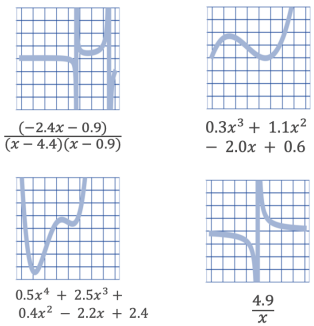
```
Y1775/0704
Y2000/0101
```

LOGO Graphics

Block Towers



Symbolic Regression



Recursive Programming

Filter

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

Index List

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

Length

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

Every Other

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

Physics

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

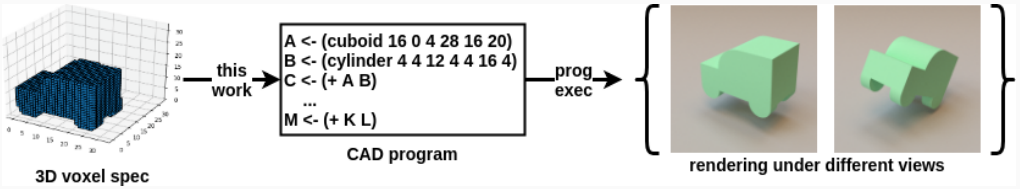
$$\vec{a} = \frac{1}{m}\sum_i \vec{F}_i$$

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

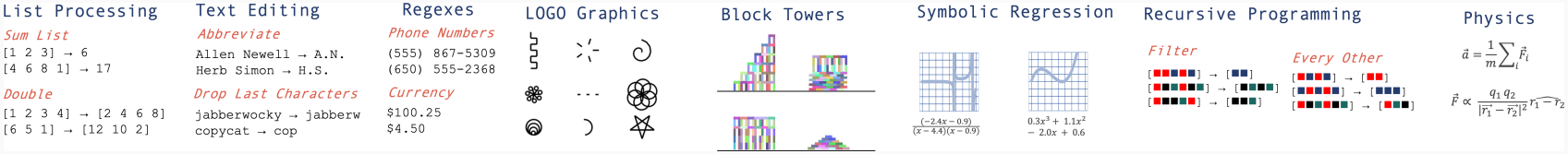
$$R_{total} = \left(\Sigma_i \frac{1}{R_i}\right)^{-1}$$

Perception, Learning-to-Learn, Synthesizing models

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



Theme #2: Learning to synthesize programs



Theme #3: Synthesizing human-understandable models

Language: Kerewe

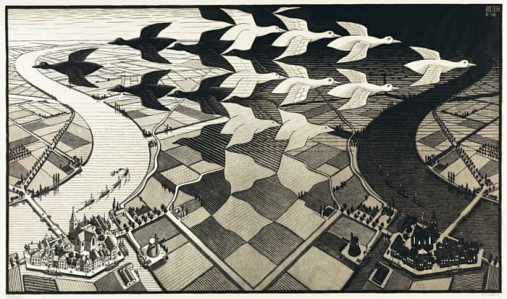
theory

INF→/ku/+stem+/a/, V e.o.→/ku/+stem+/ana/,
V for→/ku/+stem+/ila/, V for e.o.→/ku/+stem+/ilana/
V us→/kutú/+stem+/a/, V it→/kuki/+stem+/a/,
V for us→/kutú/+stem+/ila/, V it for us→/kukítú/+stem+/ila/

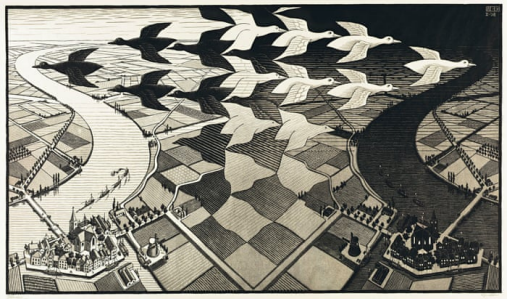
r_1: [ ] → [-hiTone] / [+hiTone] [-vowel]$_0$_
r_2: [+vowel] → [+hiTone] / [+hiTone] [-vowel]$_0$_[-vowel]

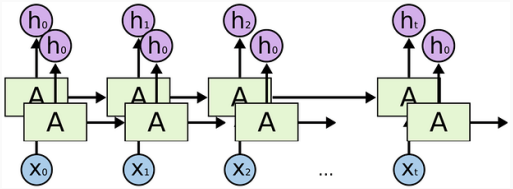
stems (unobserved)	observed data & unobserved derivation
⟨/gul/, BUY⟩	BUY US→/kutú/+/gul/+/a/→/kutúgula/$\xrightarrow{r_2}$[kutúgúla]
⟨/láb/, PASS⟩	PASS→/ku/+/láb/+/a/→[kulába]
⟨/sún/, PINCH⟩	PINCH IT FOR US→/kukítú/+/sún/+/ila/→ /kukítúsúnila/$\xrightarrow{r_1}$/kukítusunila/$\xrightarrow{r_2}$[kukítúsunila]
...	··· *total 56 words/8 inflections* ···

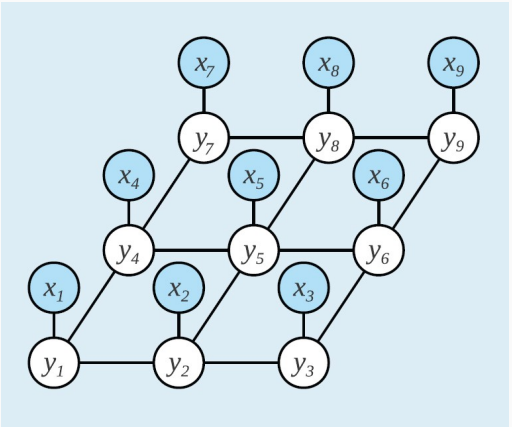
High-level, abstract visual abilities



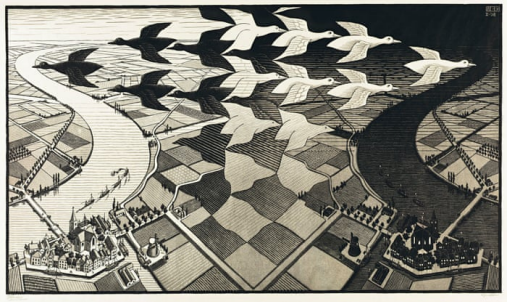
High-level, abstract visual abilities

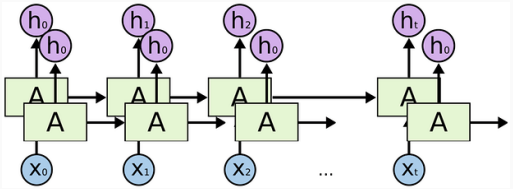


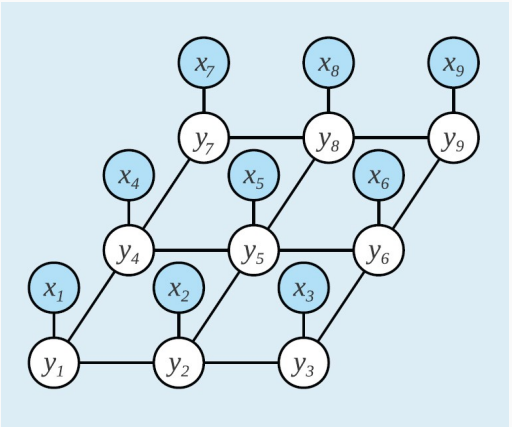




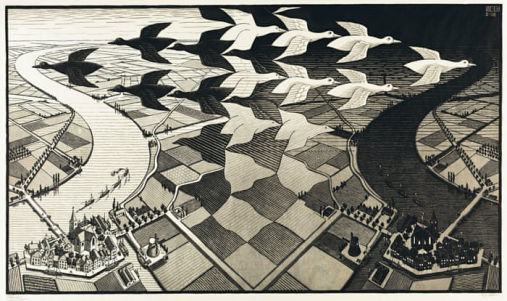
High-level, abstract visual abilities

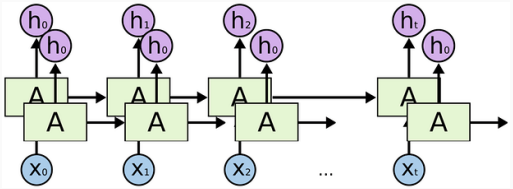


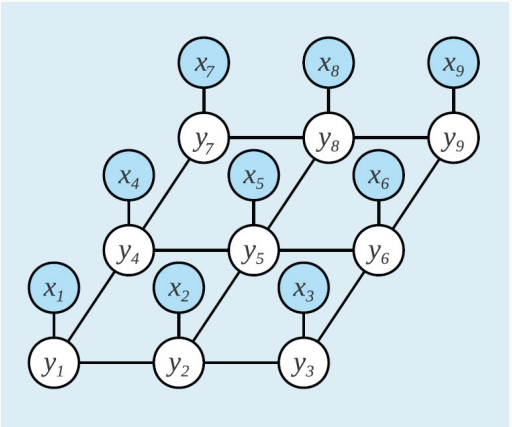


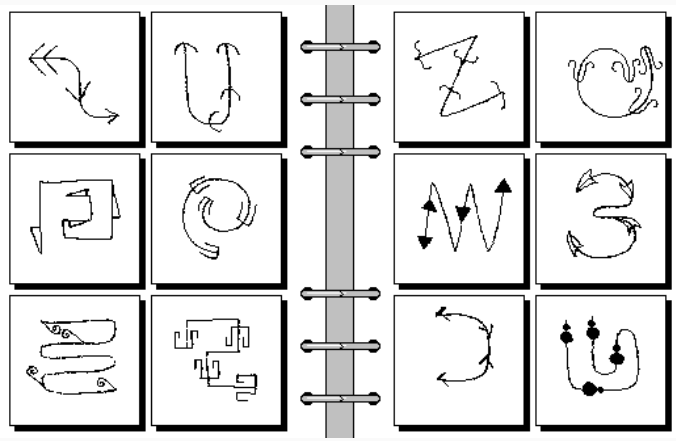


High-level, abstract visual abilities

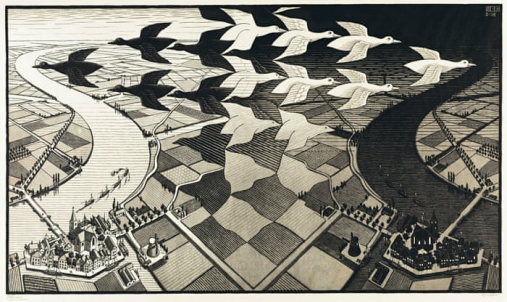


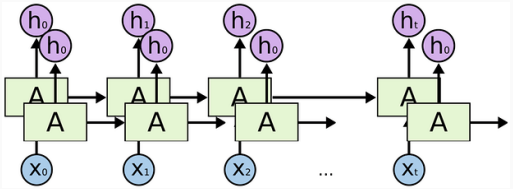


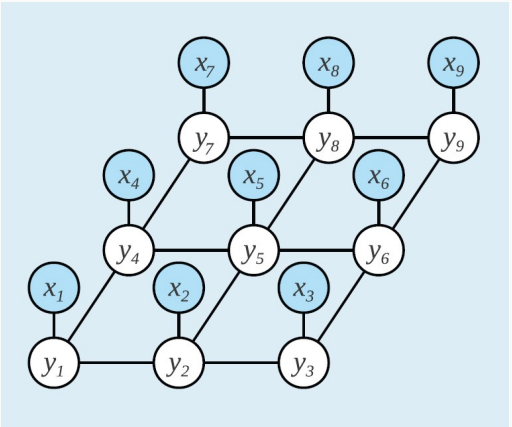


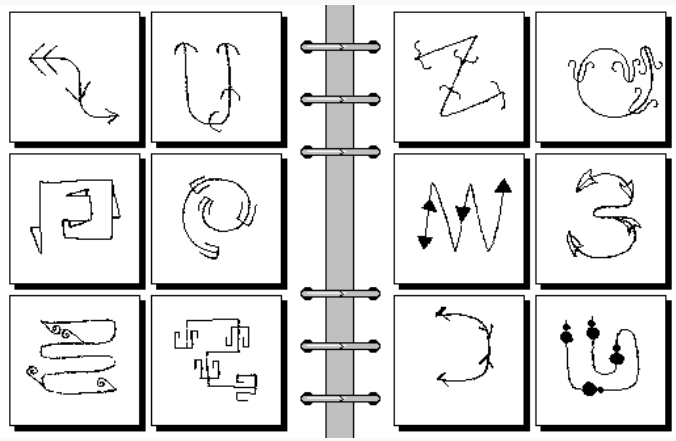


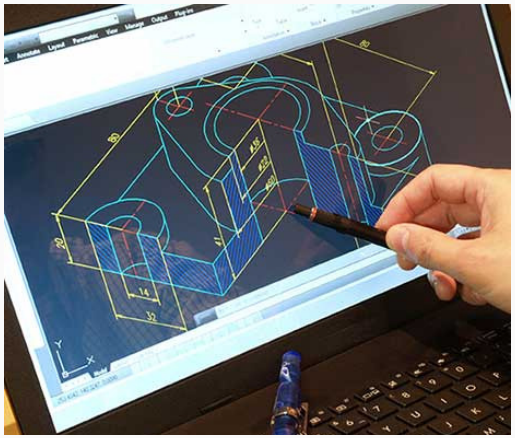
High-level, abstract visual abilities



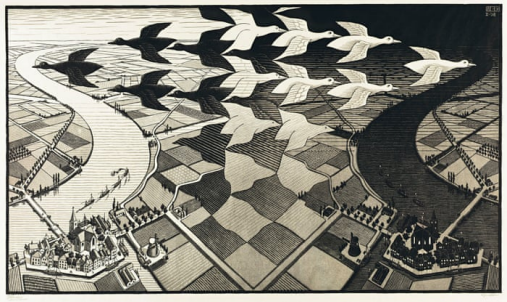


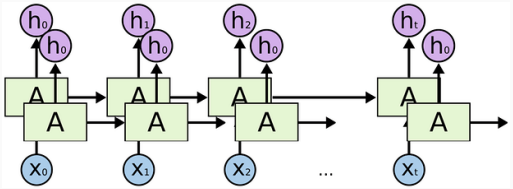


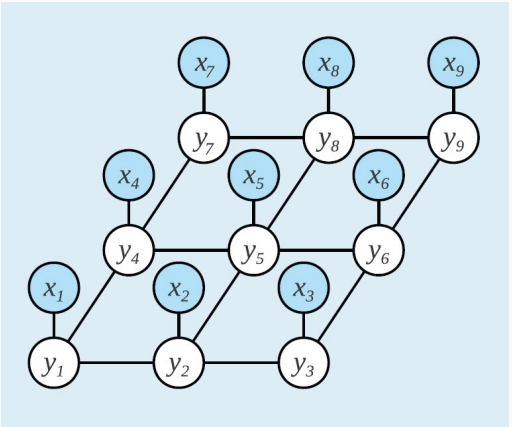


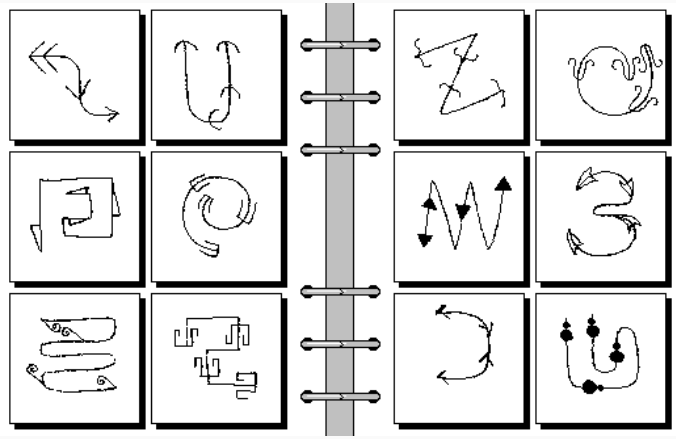


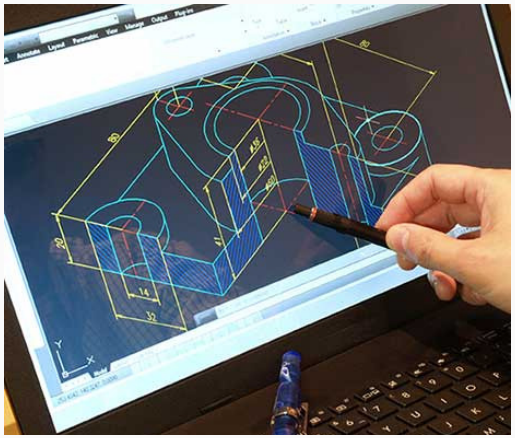
High-level, abstract visual abilities



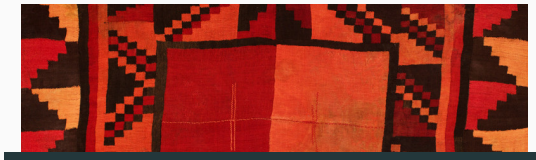


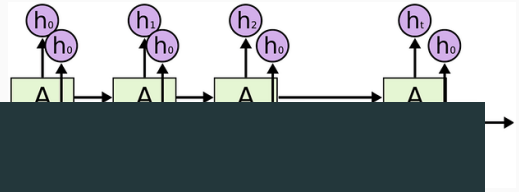






High-level, abstract visual abilities





why?

impute missing objects, extrapolate percepts,

learn visual concepts (‘arch’, ‘spiral’, HMM),

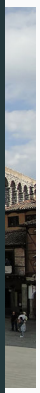
assist graphic design, assist 3D modeling

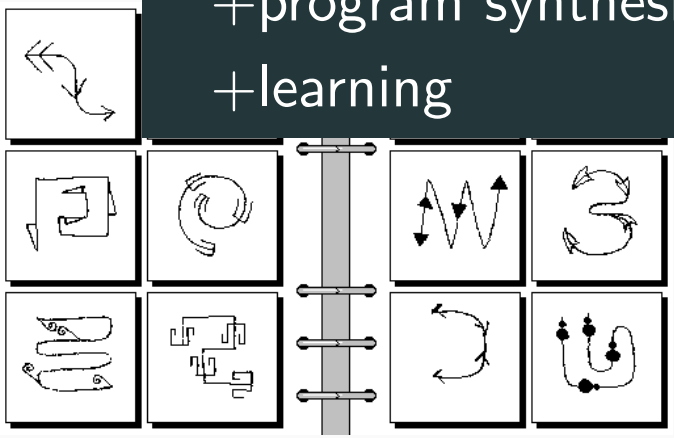
how?

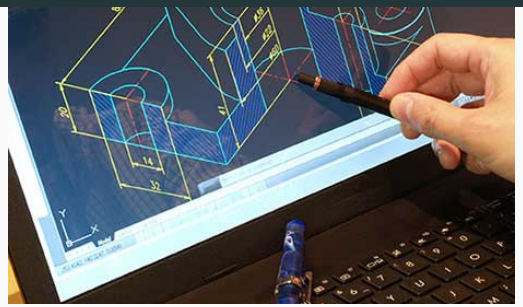
Bayesian inference of graphics program conditioned on image,

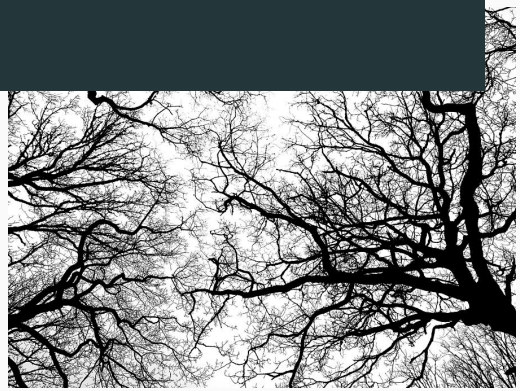
+program synthesis

+learning



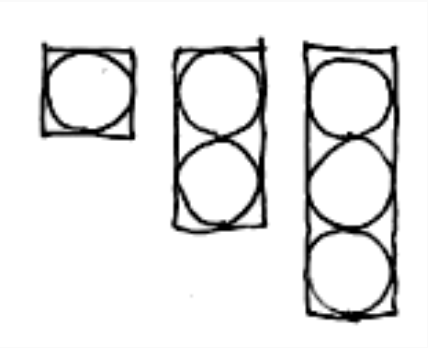






Learning to infer graphics programs from hand-drawn images

model infers program from drawing



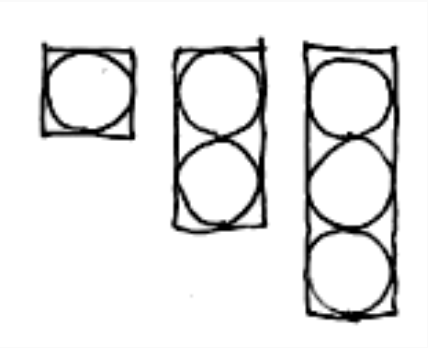
```
for (0 <= i < 3)
 rectangle(3*i,-2*i+4,
           3*i+2,6)
 for (0 <= j < i + 1)
  circle(3*i+1,-2*j+5)
```

Ellis, Ritchie, Solar-Lezama, Tenenbaum. NeurIPS 2018.

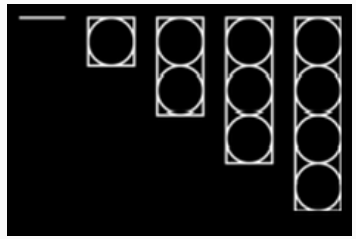
Learning to infer graphics programs from hand-drawn images

model infers program from drawing

zero-shot generalization / extrapolation



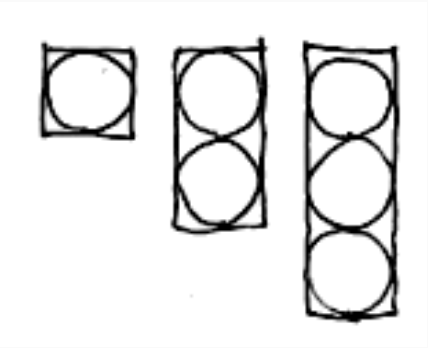
```
for (0 <= i < 3 + 1)
 rectangle(3*i,-2*i+4,
           3*i+2,6)
 for (0 <= j < i + 1)
  circle(3*i+1,-2*j+5)
```



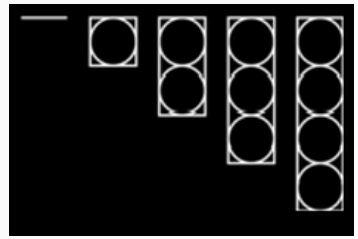
Learning to infer graphics programs from hand-drawn images

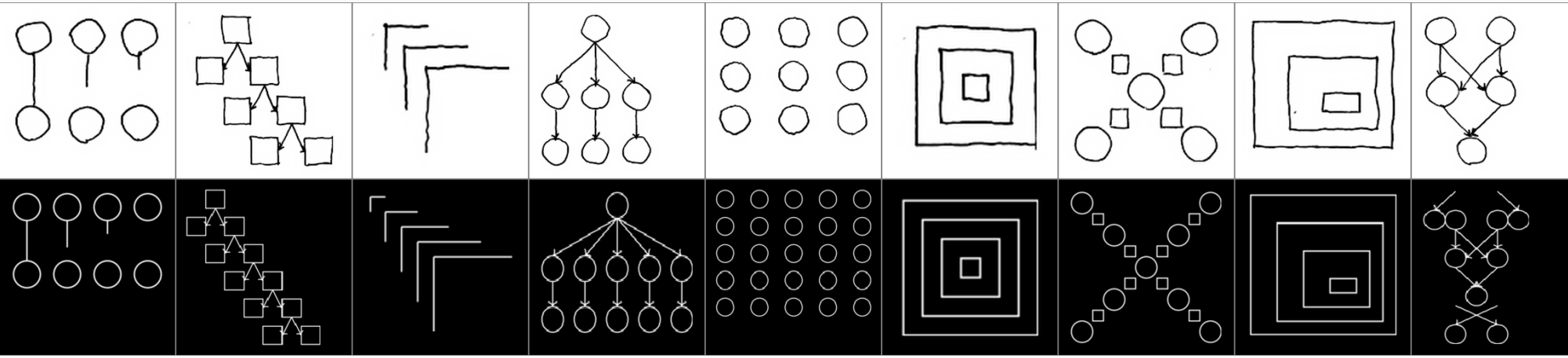
model infers program from drawing

zero-shot generalization / extrapolation

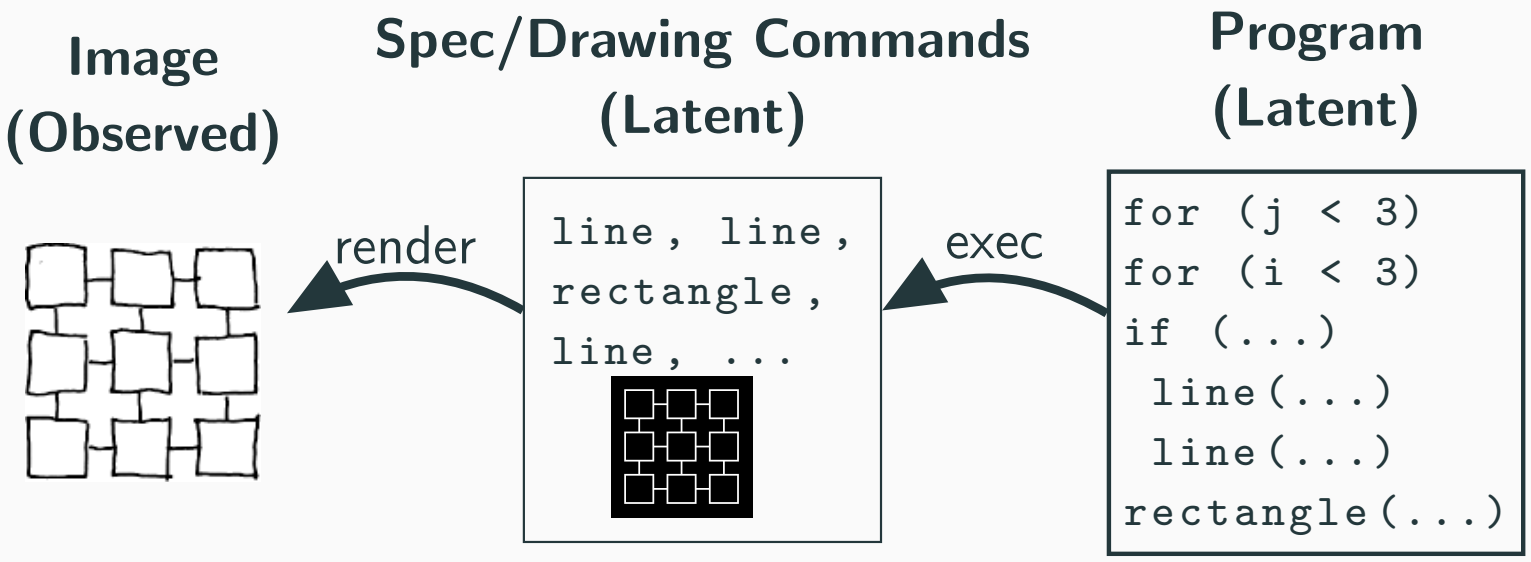


```
for (0 <= i < 3 + 1)
 rectangle(3*i,-2*i+4,
           3*i+2,6)
 for (0 <= j < i + 1)
  circle(3*i+1,-2*j+5)
```

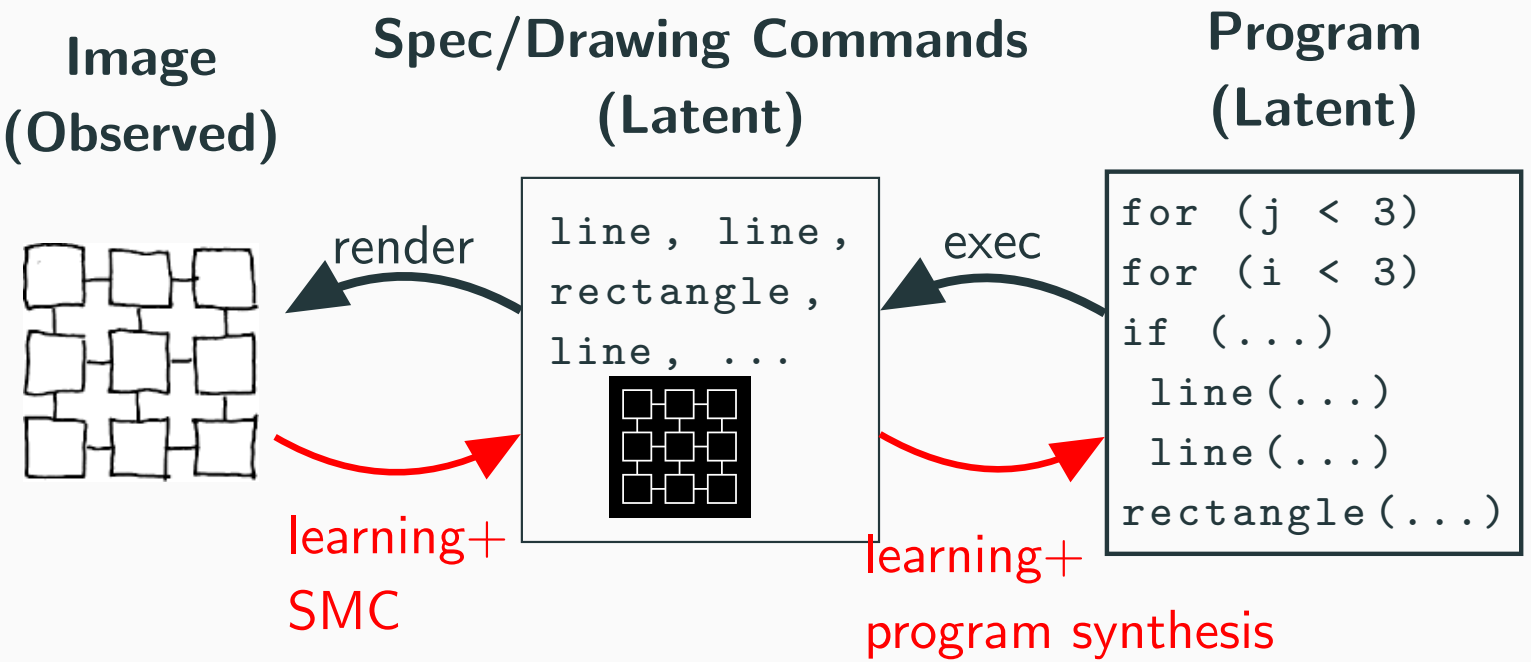




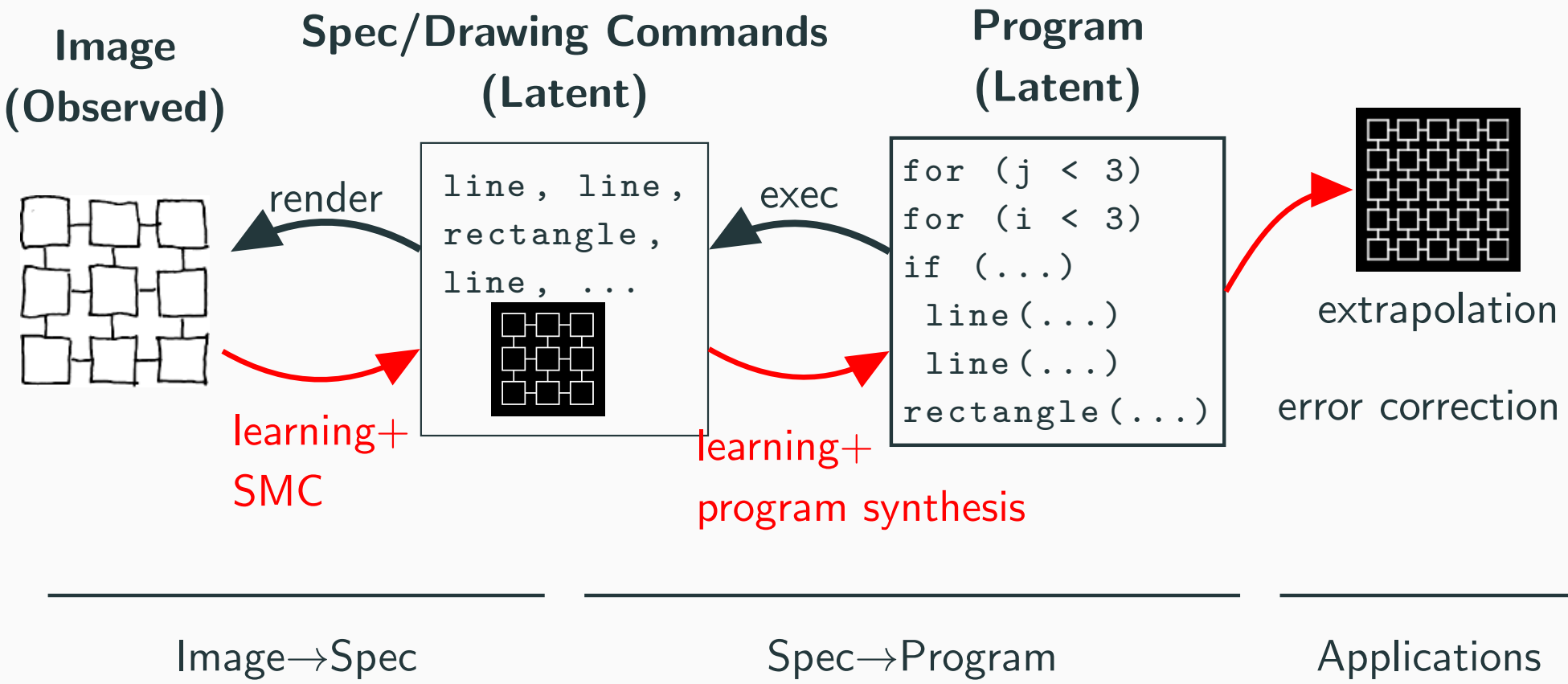
How to infer graphics programs from hand-drawn images



How to infer graphics programs from hand-drawn images



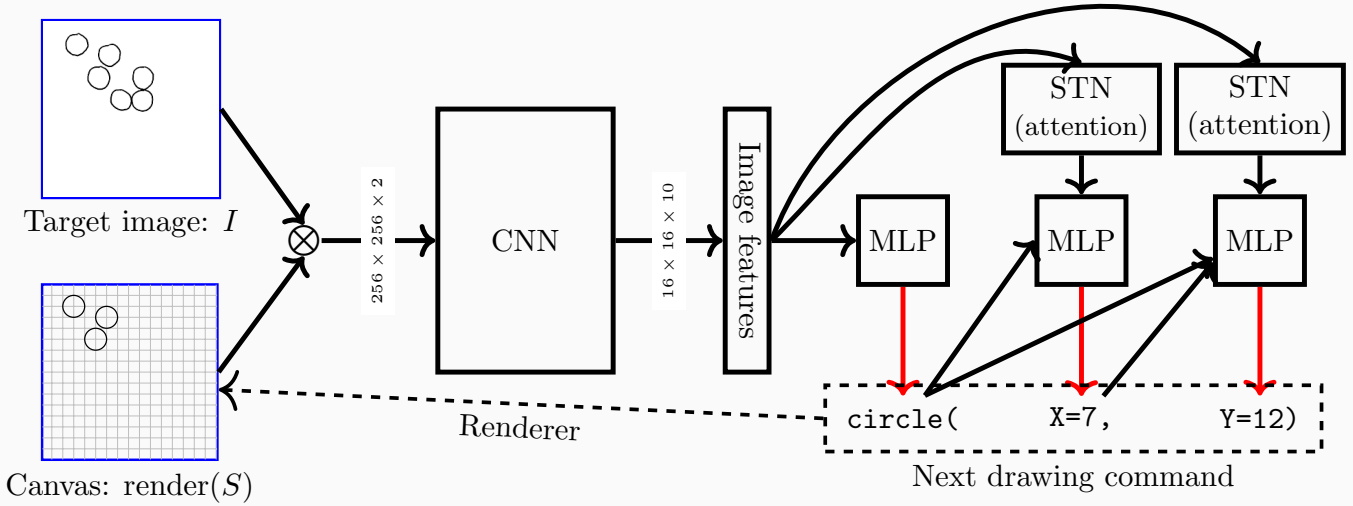
How to infer graphics programs from hand-drawn images



Parsing images into specs (LaTeX TikZ commands)

Neurally Guided Procedural Modeling (Ritchie et al 2016) +

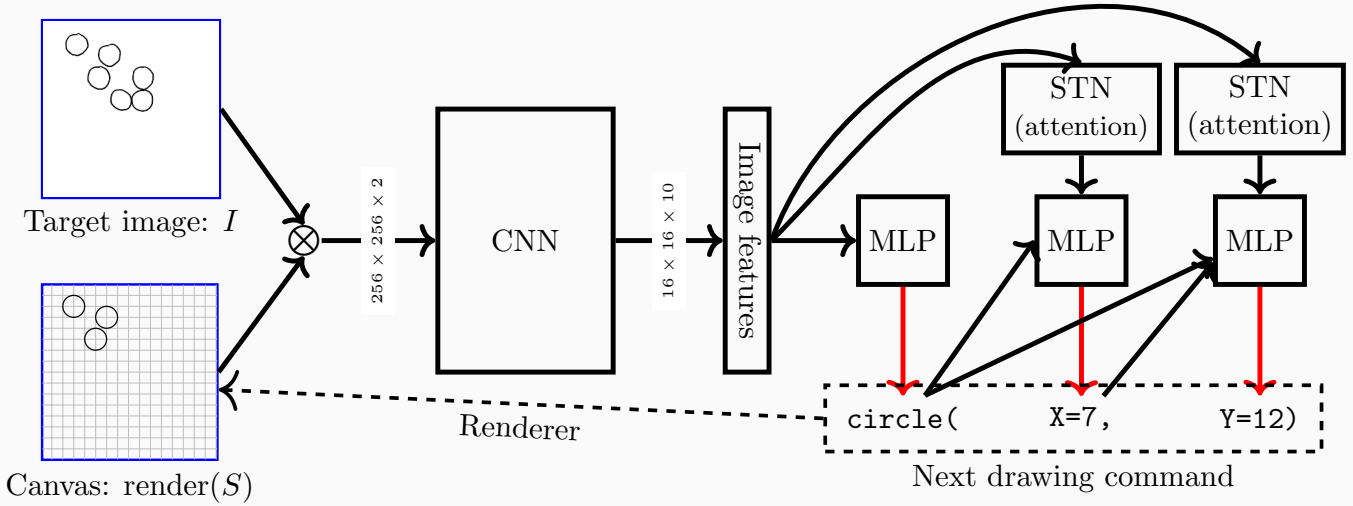
Attend, Infer, Repeat (Eslami et al 2016)

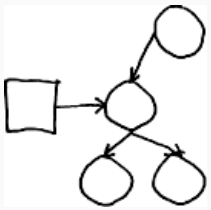


Parsing images into specs (LaTeX TikZ commands)

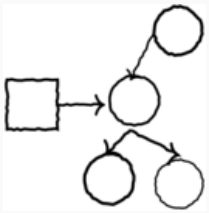
Neurally Guided Procedural Modeling (Ritchie et al 2016) +

Attend, Infer, Repeat (Eslami et al 2016)





hand drawing

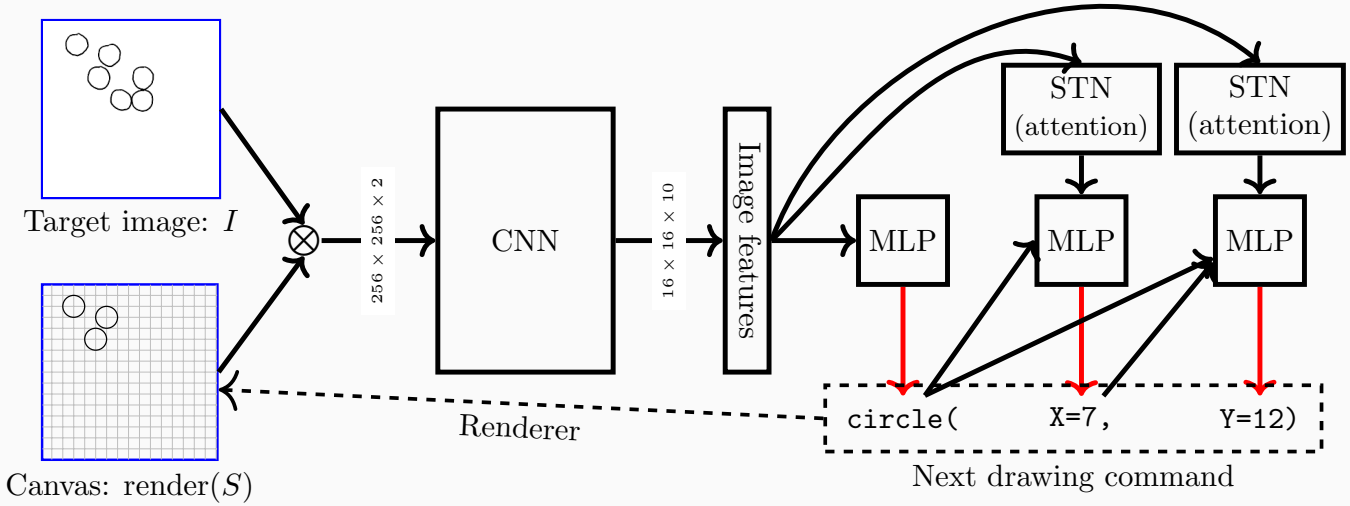


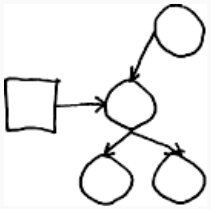
noisy rerender

Parsing images into specs (LaTeX TikZ commands)

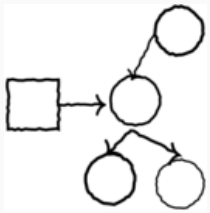
Neurally Guided Procedural Modeling (Ritchie et al 2016) +

Attend, Infer, Repeat (Eslami et al 2016)

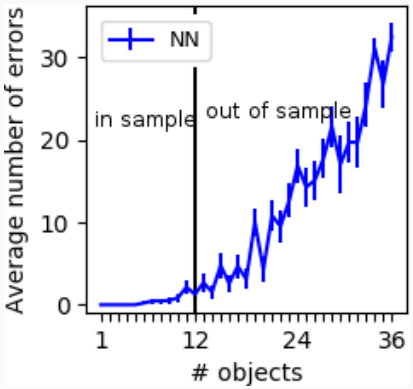




hand drawing



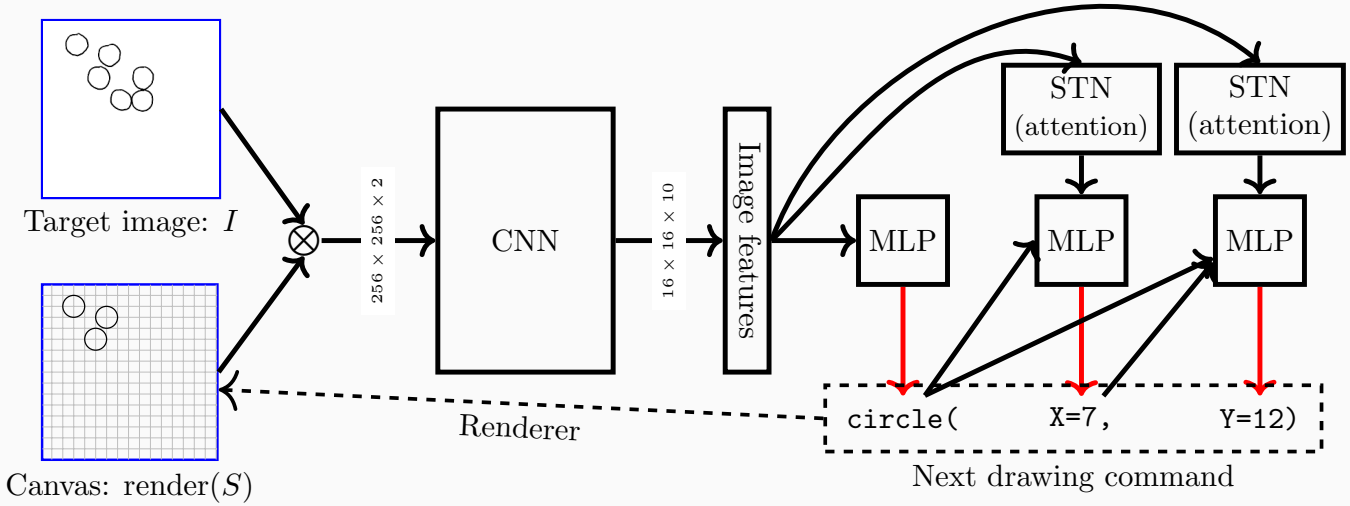
noisy rerender

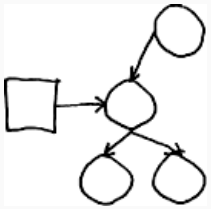


Parsing images into specs (LaTeX TikZ commands)

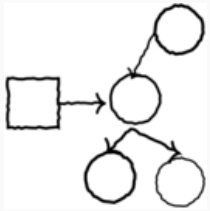
Neurally Guided Procedural Modeling (Ritchie et al 2016) +

Attend, Infer, Repeat (Eslami et al 2016)

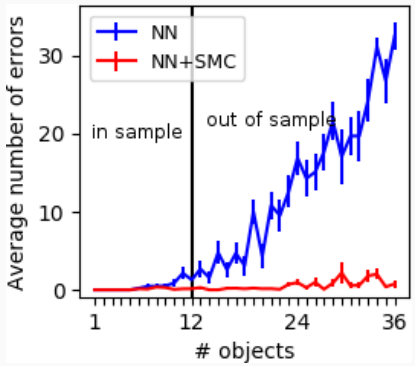




hand drawing



noisy rerender

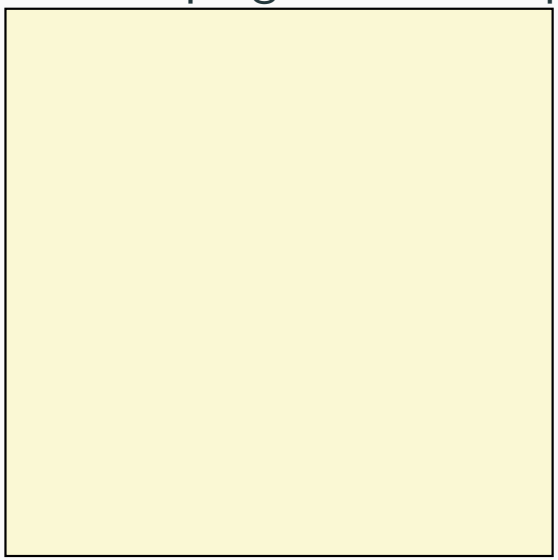


Learning to quickly synthesize programs

Learn search policy π(program subspace|spec)
Think of the subspace as an "ansatz"

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space

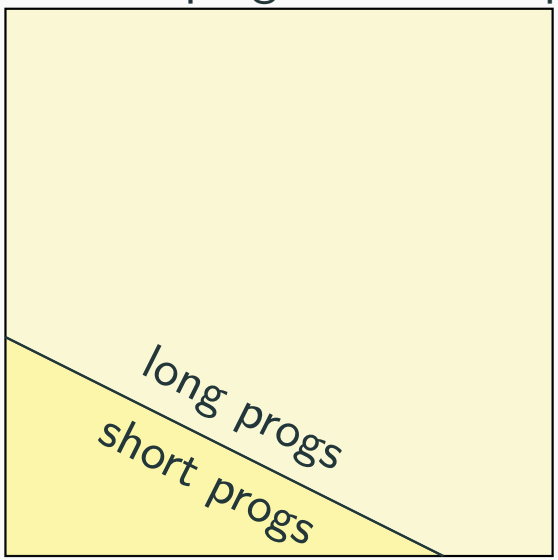


Learning to quickly synthesize programs

Learn search policy π(program subspace|spec)
Think of the subspace as an "ansatz"

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space

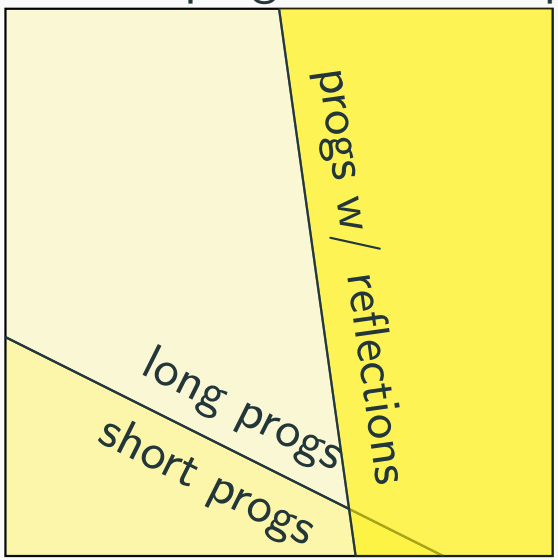


Learning to quickly synthesize programs

Learn search policy π(program subspace|spec)
Think of the subspace as an "ansatz"

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space

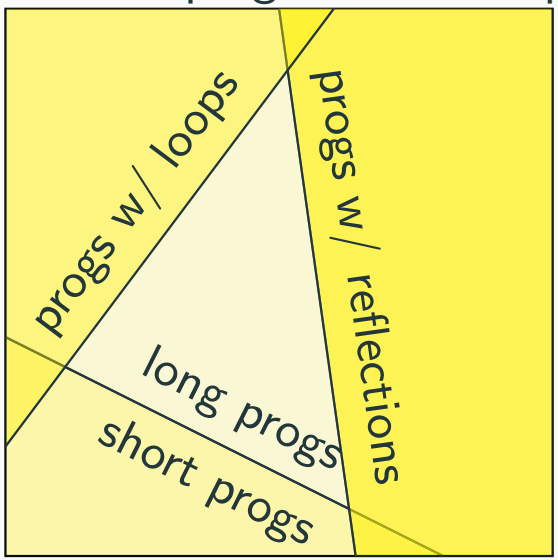


Learning to quickly synthesize programs

Learn search policy π(program subspace|spec)
Think of the subspace as an "ansatz"

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space

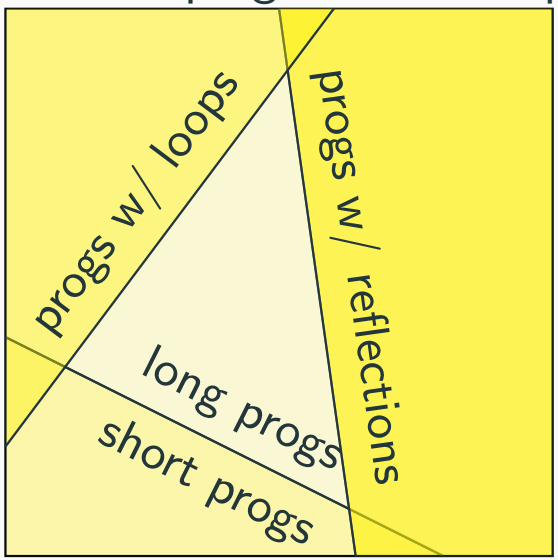


Learning to quickly synthesize programs

Learn search policy π(program subspace|spec)
Think of the subspace as an "ansatz"

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space



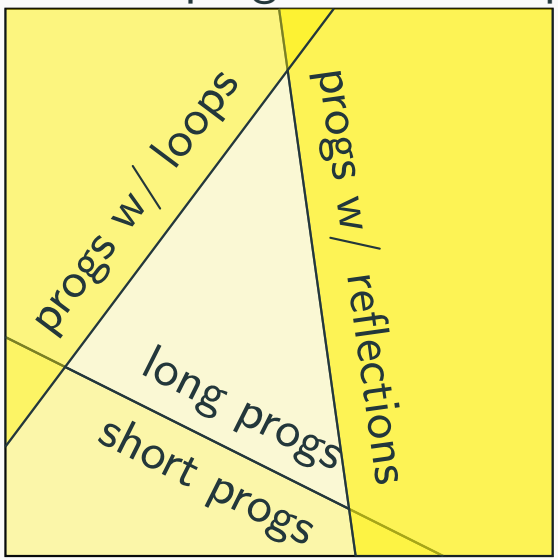
π(short, no loop/reflect|S) = ☐
π(long, loops|S) = ☐
π(long, no loop/reflect|S) = ☐
π(long, reflects|S) = ☐
etc.

Learning to quickly synthesize programs

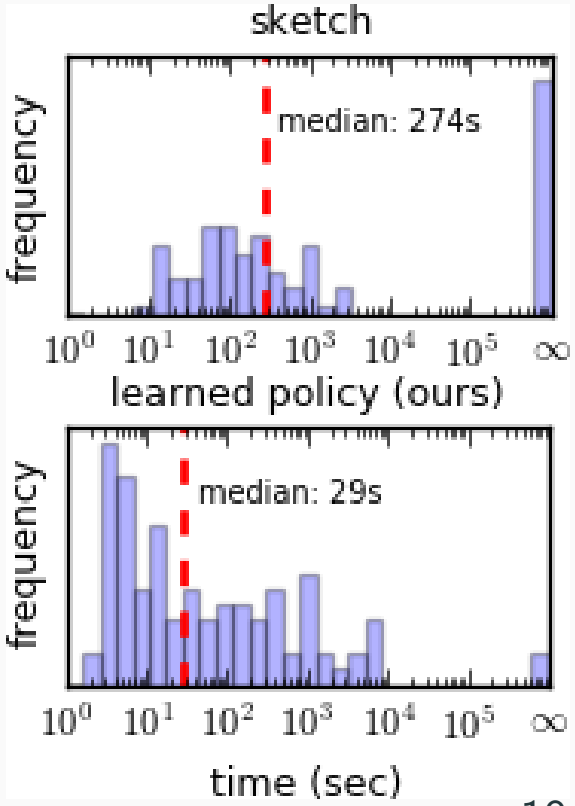
Learn search policy π(program subspace|spec)
Think of the subspace as an "ansatz"

OBJECTIVE: Small subspace for tractability while also being likely to contain good programs

Entire program search space

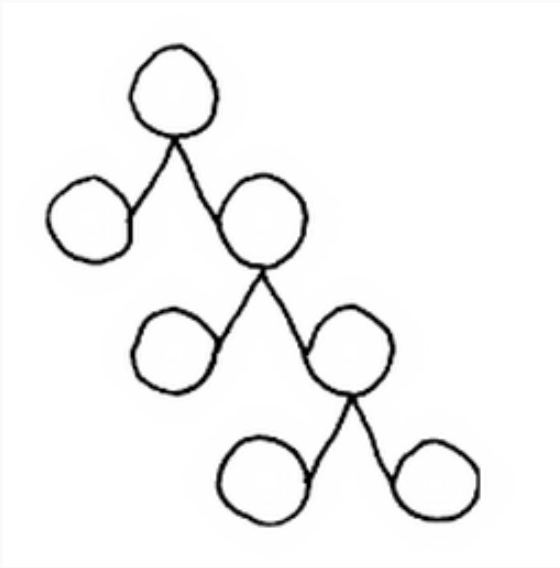


π(short, no loop/reflect|S) = ☐
π(long, loops|S) = ☐
π(long, no loop/reflect|S) = ☐
π(long, reflects|S) = ☐
etc.

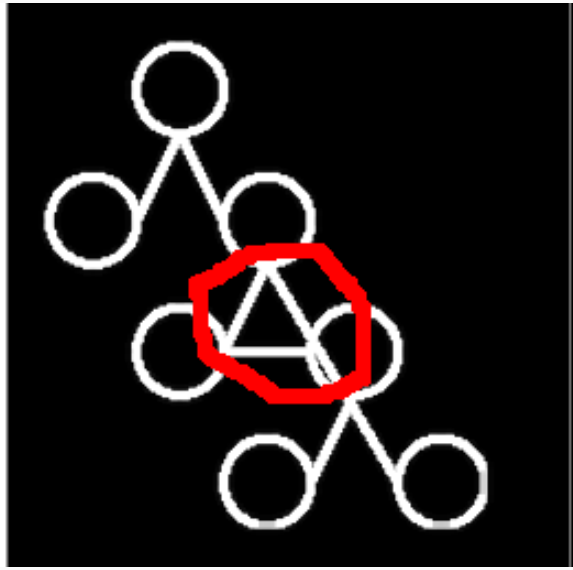


Top-down influences on perception

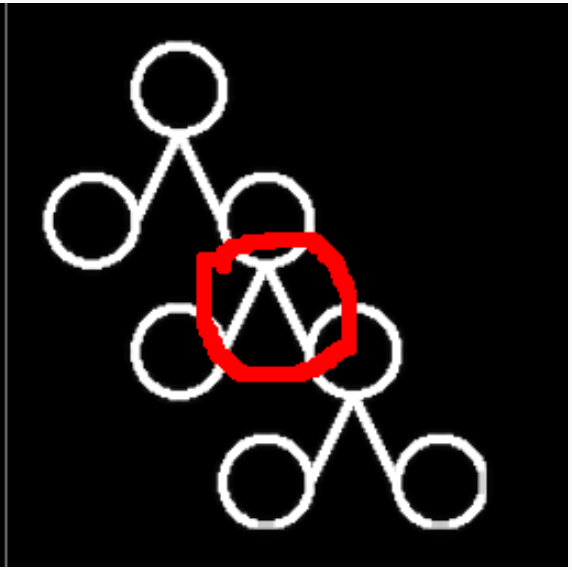
drawing



bottom-up neural net

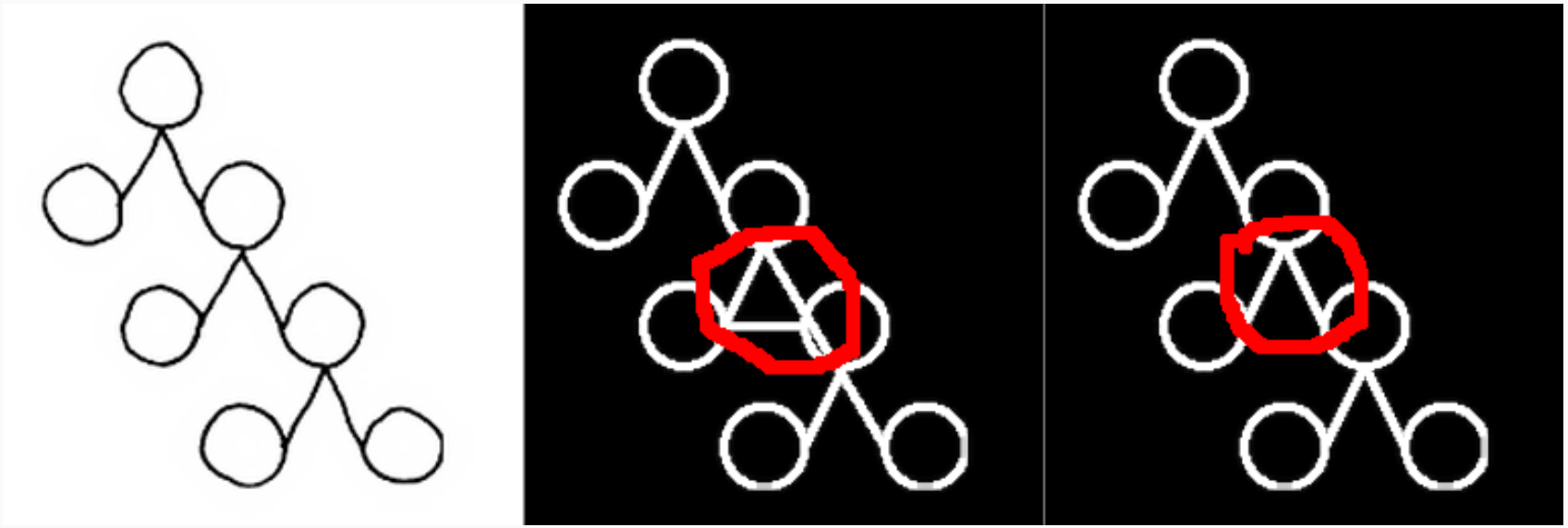


w/ top-down program bias



Top-down influences on perception

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

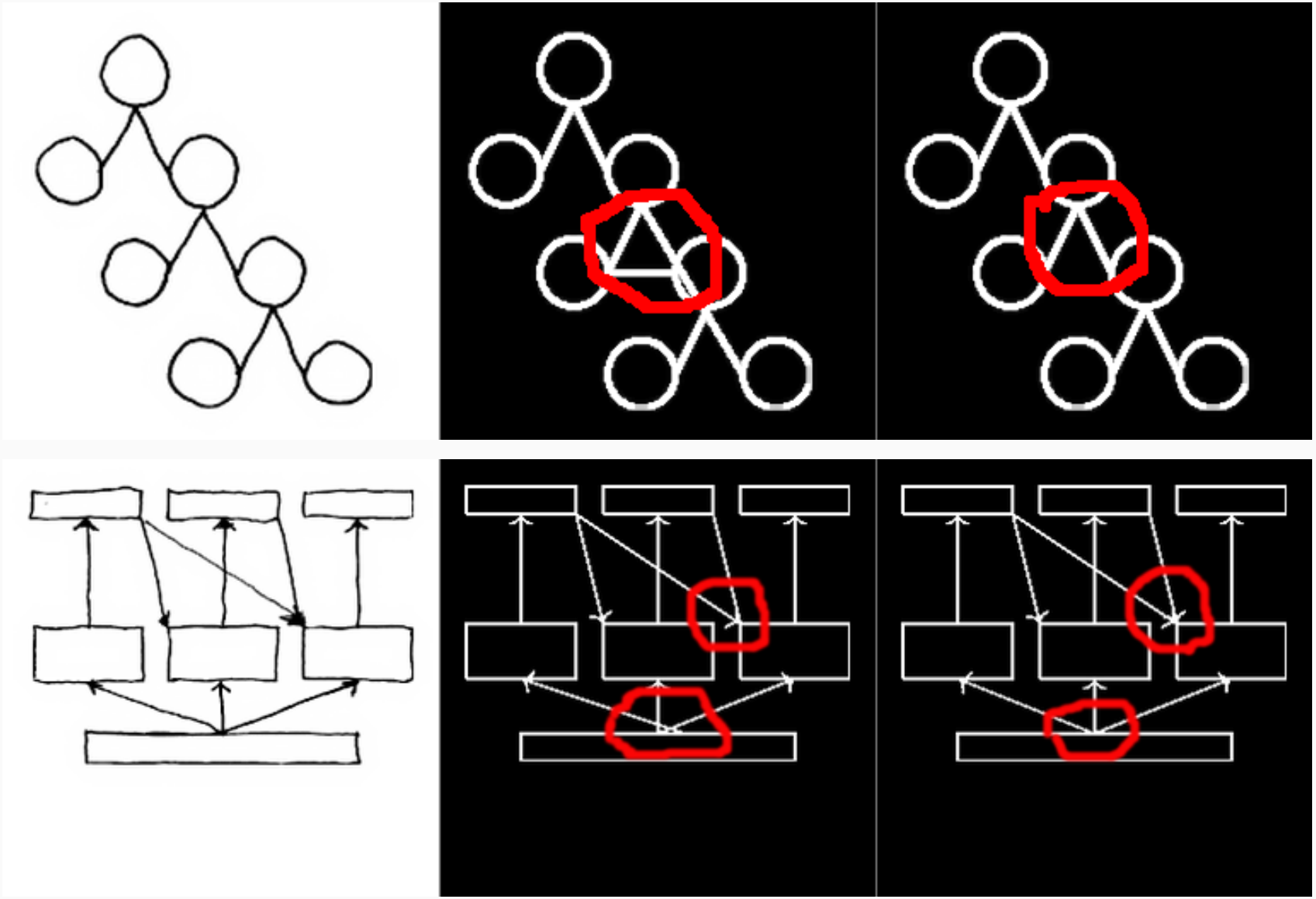


$$\hat{S}(I) = \arg\max_{S \in \mathcal{F}(I)} \mathbb{P}(I|\text{render}(S)) \times \mathbb{P}_\beta[\text{program}(S)]$$

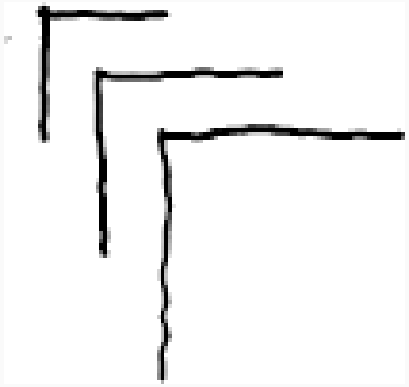
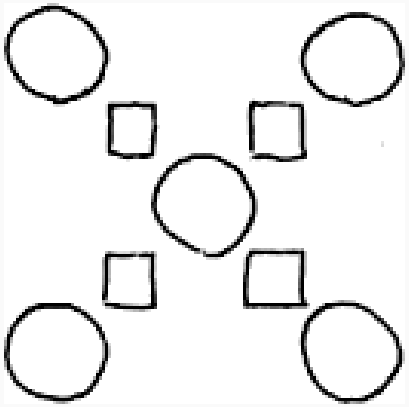
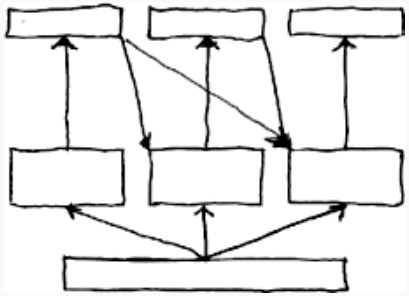
$$\beta^* = \arg\max_\beta \mathbb{E}\left[\log \frac{\mathbb{P}_\beta[\text{program}(S)] \times \mathbb{P}(I|\text{render}(S))}{\sum_{S' \in \mathcal{F}(I)} \mathbb{P}_\beta[\text{program}(S')] \times \mathbb{P}(I|\text{render}(S'))}\right]$$

Top-down influences on perception

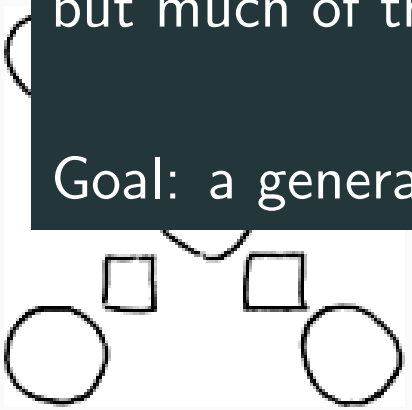
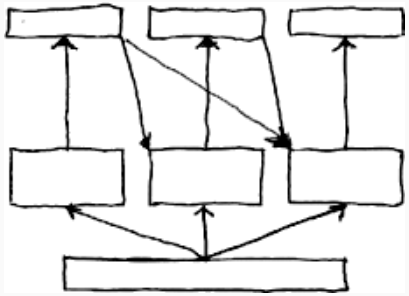
drawing | bottom-up neural net | w/ top-down program bias



Example programs

Drawing	Spec	Program
	`Line(2,15, 4,15)` `Line(4,9, 4,13)` `Line(3,11, 3,14)` `Line(2,13, 2,15)` `Line(3,14, 6,14)` `Line(4,13, 8,13)`	`for(i<3)` `  line(i,-1*i+6,` `  2*i+2,-1*i+6)` `  line(i,-2*i+4,i,-1*i+6)`
	`Circle(2,8)` `Rectangle(6,9, 7,10)` `Circle(8,8)` `Rectangle(6,12, 7,13)` `Rectangle(3,9, 4,10)` *... etc. ...; 9 lines*	`reflect(y=8)` `  for(i<3)` `  if(i>0)` `  rectangle(3*i-1,2,3*i,3)` `  circle(3*i+1,3*i+1)`
	`Line(3,10,3,14,arrow)` `Rectangle(11,8,15,10)` `Rectangle(11,14,15,15)` `Line(13,10,13,14,arrow)` *... etc. ...; 16 lines*	`for(i<3)` `  line(7,1,5*i+2,3,arrow)` `  for(j<i+1)` `  if(j>0)` `  line(5*j-1,9,5*i,5,arrow)` `  line(5*j+2,5,5*j+2,9,arrow)` `  rectangle(5*i,3,5*i+4,5)` `  rectangle(5*i,9,5*i+4,10)` `rectangle(2,0,12,1)`

Example programs

Drawing	Spec	Program
	`Line(2,15, 4,15)` `Line(4,9, 4,13)` `Line(3,11, 3,14)` `Line(2,13, 2,15)` `Line(3,14, 6,14)` `Line(4,13, 8,13)`	`for(i<3)` `line(i,-1*i+6,` `2*i+2,-1*i+6)` `line(i,-2*i+4,i,-1*i+6)`
	`Rectangle(3,9, 4,10)` *... etc. ...; 9 lines*	`rectangle(3*i-1,2,3*i,3)` `circle(3*i+1,3*i+1)`
	`Line(3,10,3,14,arrow)` `Rectangle(11,8,15,10)` `Rectangle(11,14,15,15)` `Line(13,10,13,14,arrow)` *... etc. ...; 16 lines*	`for(i<3)` `line(7,1,5*i+2,3,arrow)` `for(j<i+1)` `if(j>0)` `line(5*j-1,9,5*i,5,arrow)` `line(5*j+2,5,5*j+2,9,arrow)` `rectangle(5*i,3,5*i+4,5)` `rectangle(5*i,9,5*i+4,10)` `rectangle(2,0,12,1)`

Learning played a role...

but much of this system is specific to 2-D graphics

Goal: a general algorithm for learning to synthesize programs

Growing domain-specific knowledge

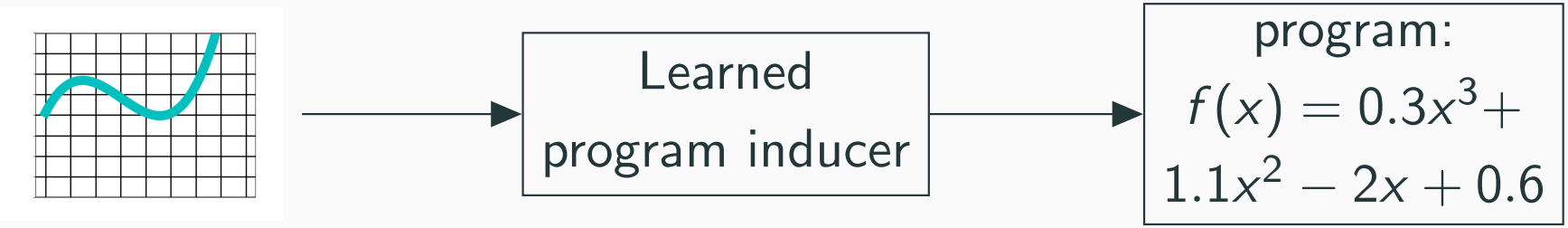
Goal: acquire domain-specific knowledge needed to induce a class of programs

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

Growing domain-specific knowledge

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

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



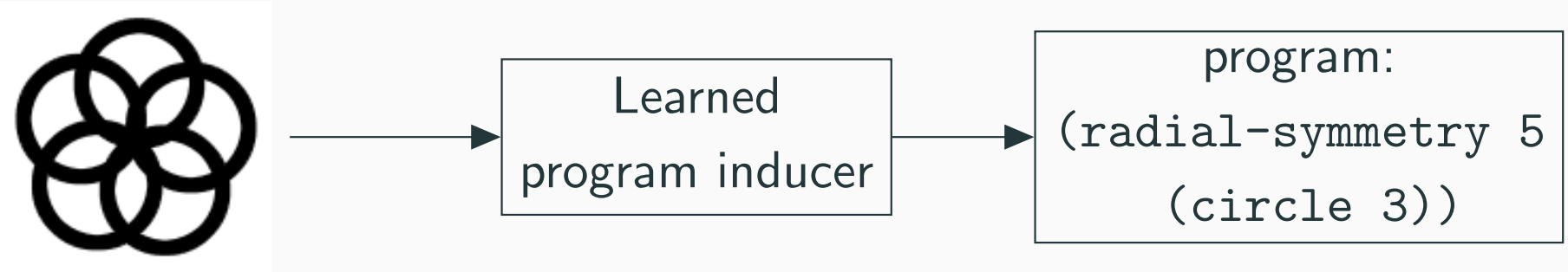
Concepts: x^3, $\alpha x + \beta$, etc

Inference strategy: neurosymbolic search for programs

Growing domain-specific knowledge

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

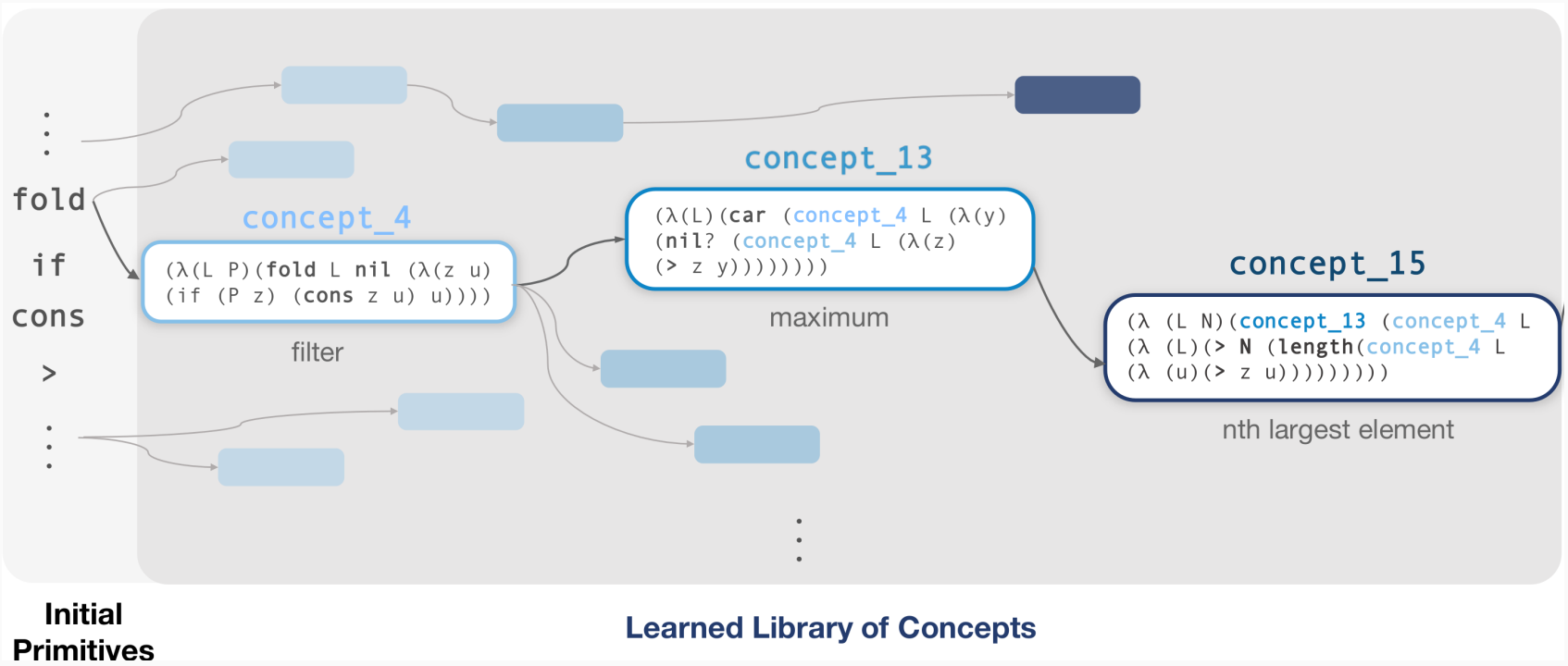
- Library of concepts (declarative knowledge; domain specific language; generative model over programs)
- Inference strategy (procedural knowledge; synthesis algorithm)



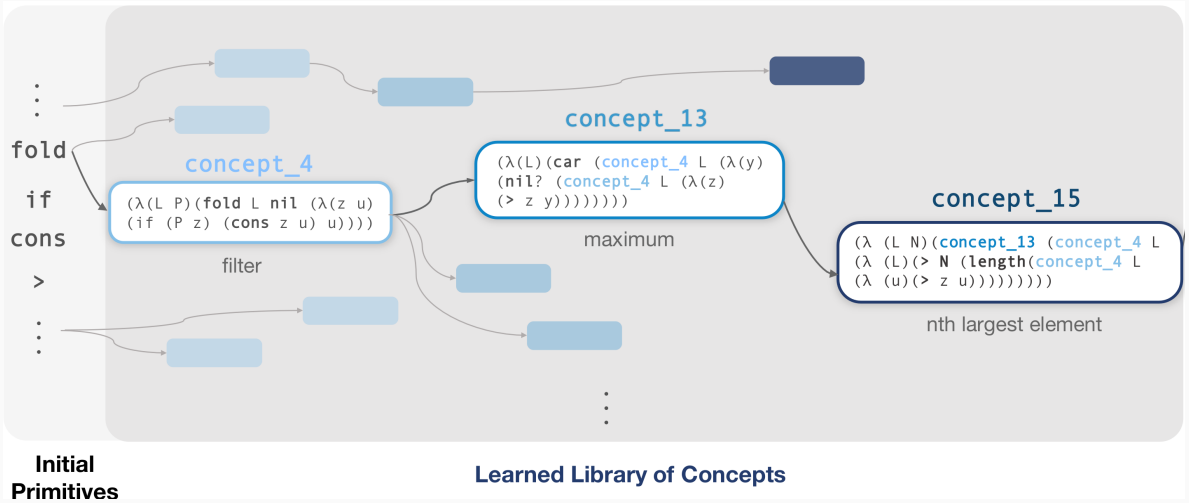
Concepts: `circle`, `radial-symmetry`, etc

Inference strategy: neurosymbolic search for programs

Library learning



Library learning



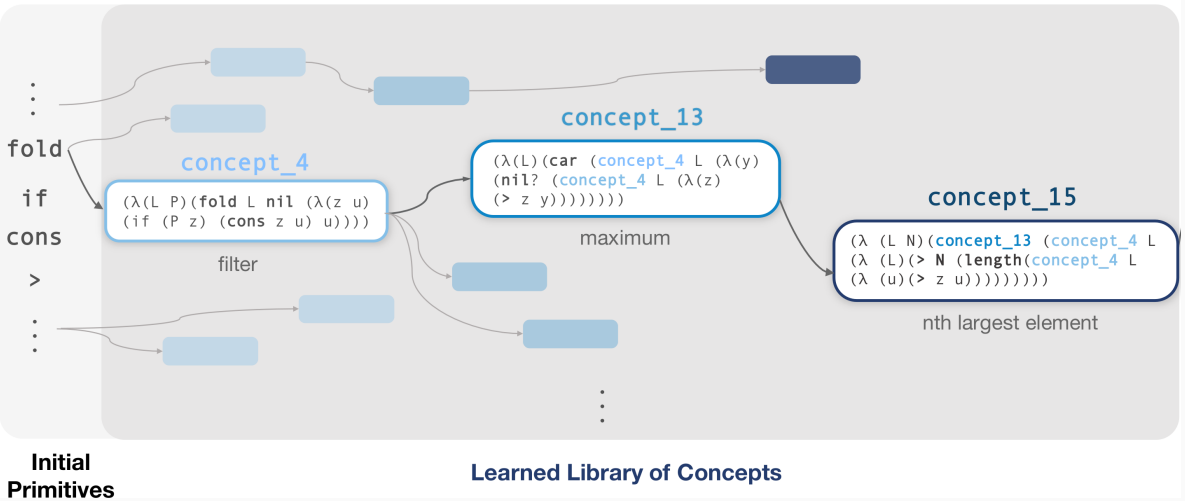
Problem: sort list

Solution:

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

get nth largest element where n = 1, 2, 3 ….length of list

Library learning



Problem: sort list

Solution:

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

get nth largest element where n = 1, 2, 3 ….length of list

Solution in initial primitives:

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

Discovered Problem Solutions

DreamCoder

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

DreamCoder

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

List Processing

Sum List

```
[1 2 3] → 6
[4 6 8 1] → 17
```

Double

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

Check Evens

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

Text Editing

Abbreviate

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

Drop Last Characters

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

Extract

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

Regexes

Phone Numbers

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

Currency

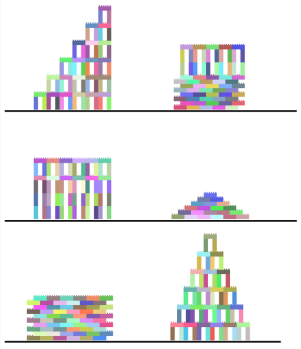
```
$100.25
$4.50
```

Dates

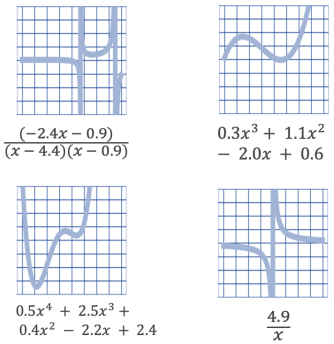
```
Y1775/0704
Y2000/0101
```

LOGO Graphics

Block Towers



Symbolic Regression



Recursive Programming

Filter

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

Length

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

Index List

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

Every Other

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

Physics

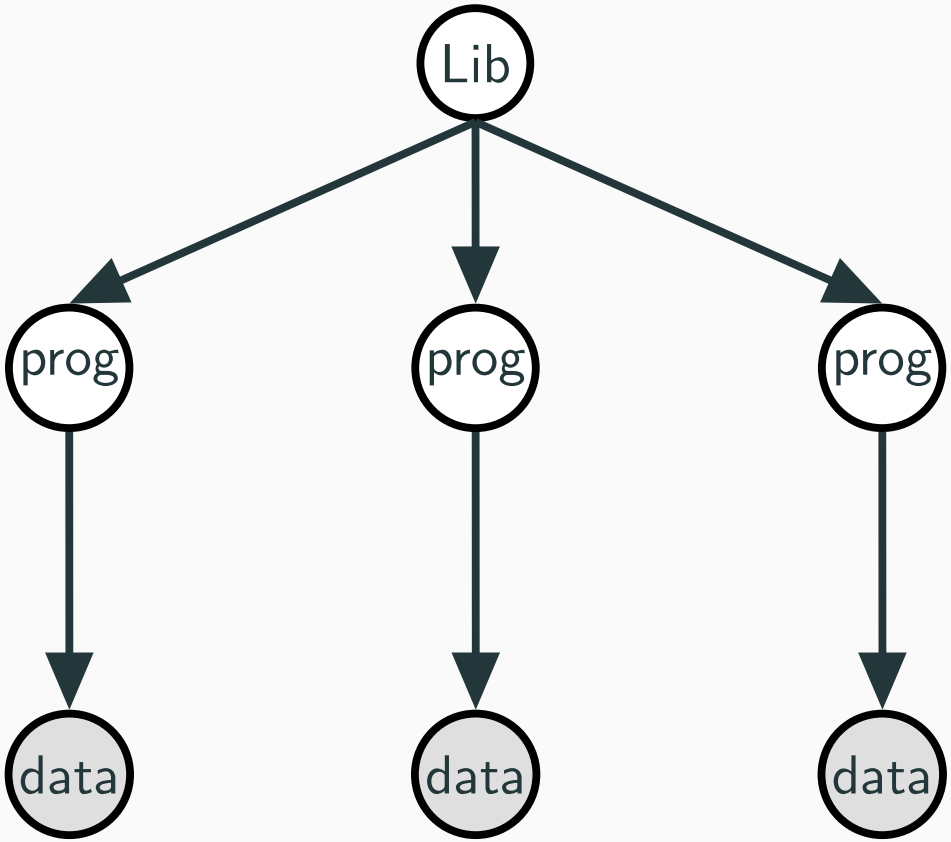
$$KE = \frac{1}{2}m|\vec{v}|^2$$

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

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

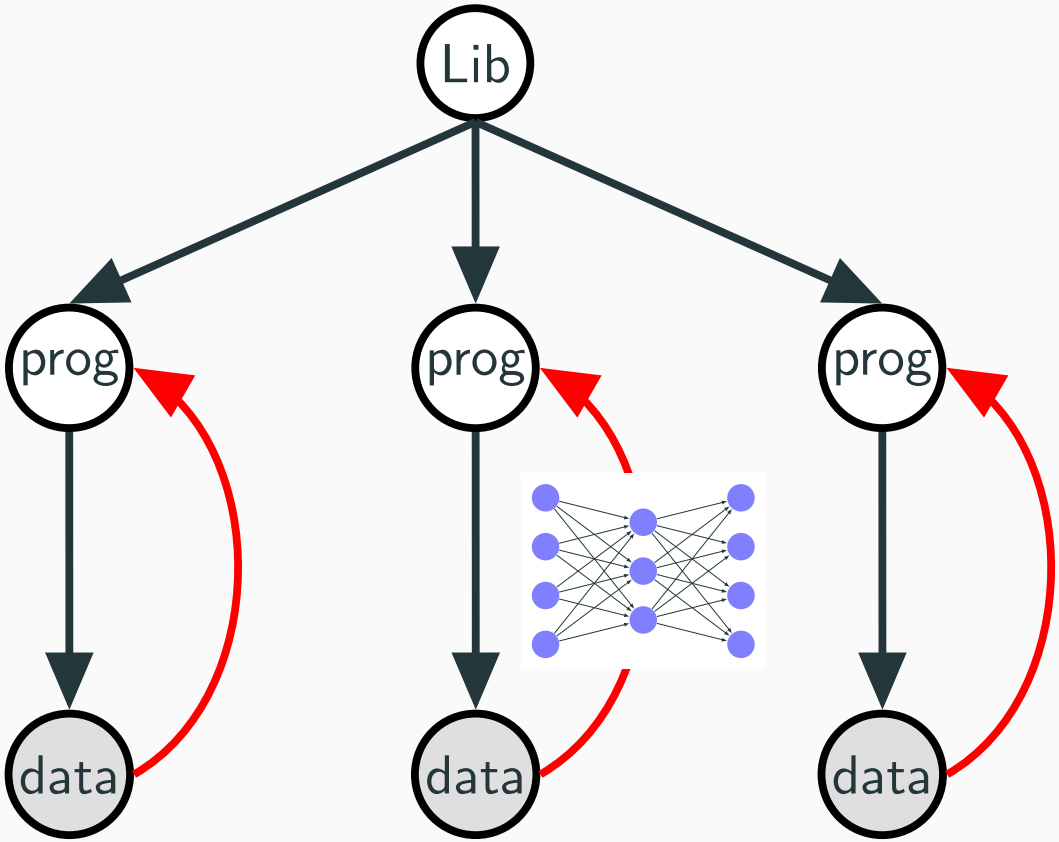
$$R_{total} = \left(\Sigma_i \frac{1}{R_i}\right)^{-1}$$

Library learning as Bayesian inference



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

Library learning as amortized Bayesian inference

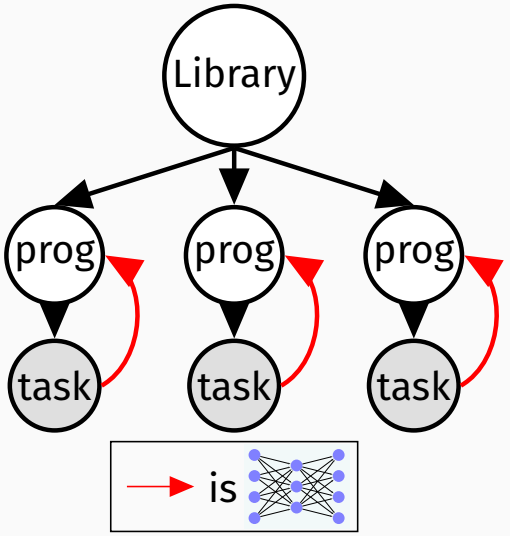


amortized inference +

better program representation (Lisp) +

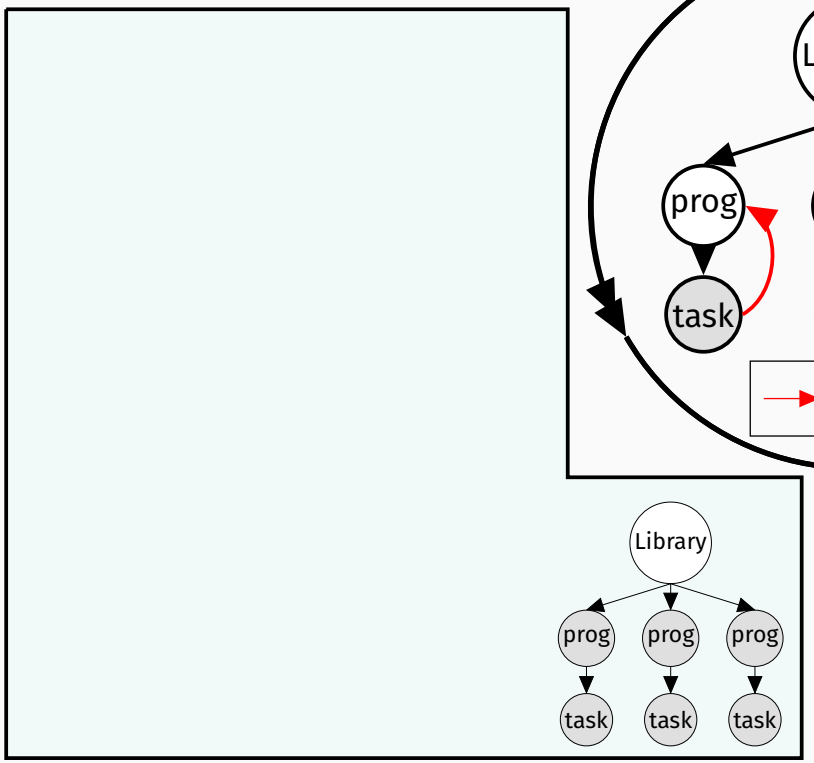
library learning via program analysis +

new neural inference network for program synthesis

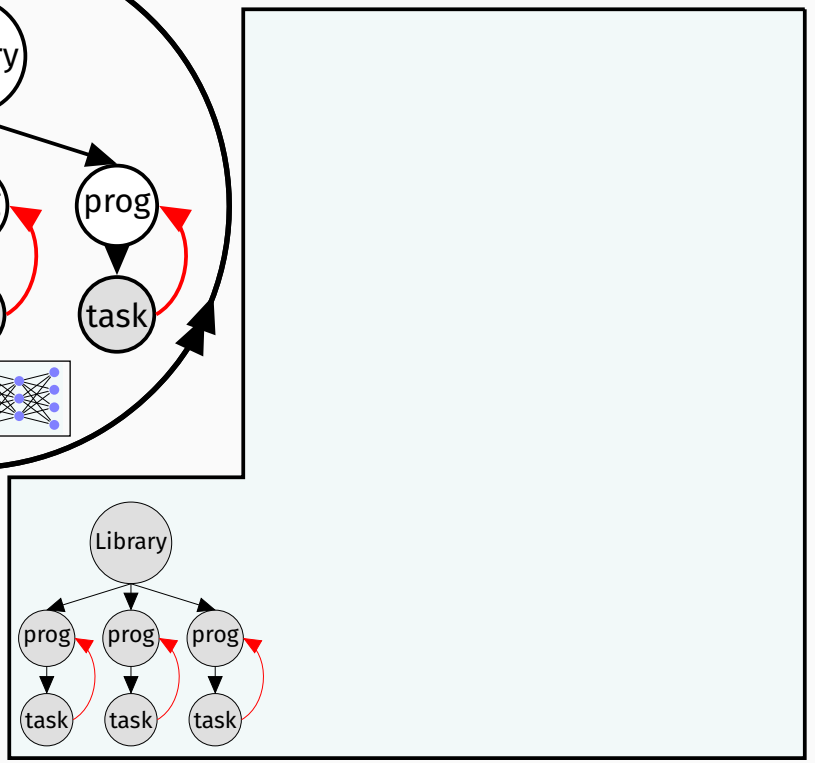

Library
prog
prog
prog
task
task
task
is

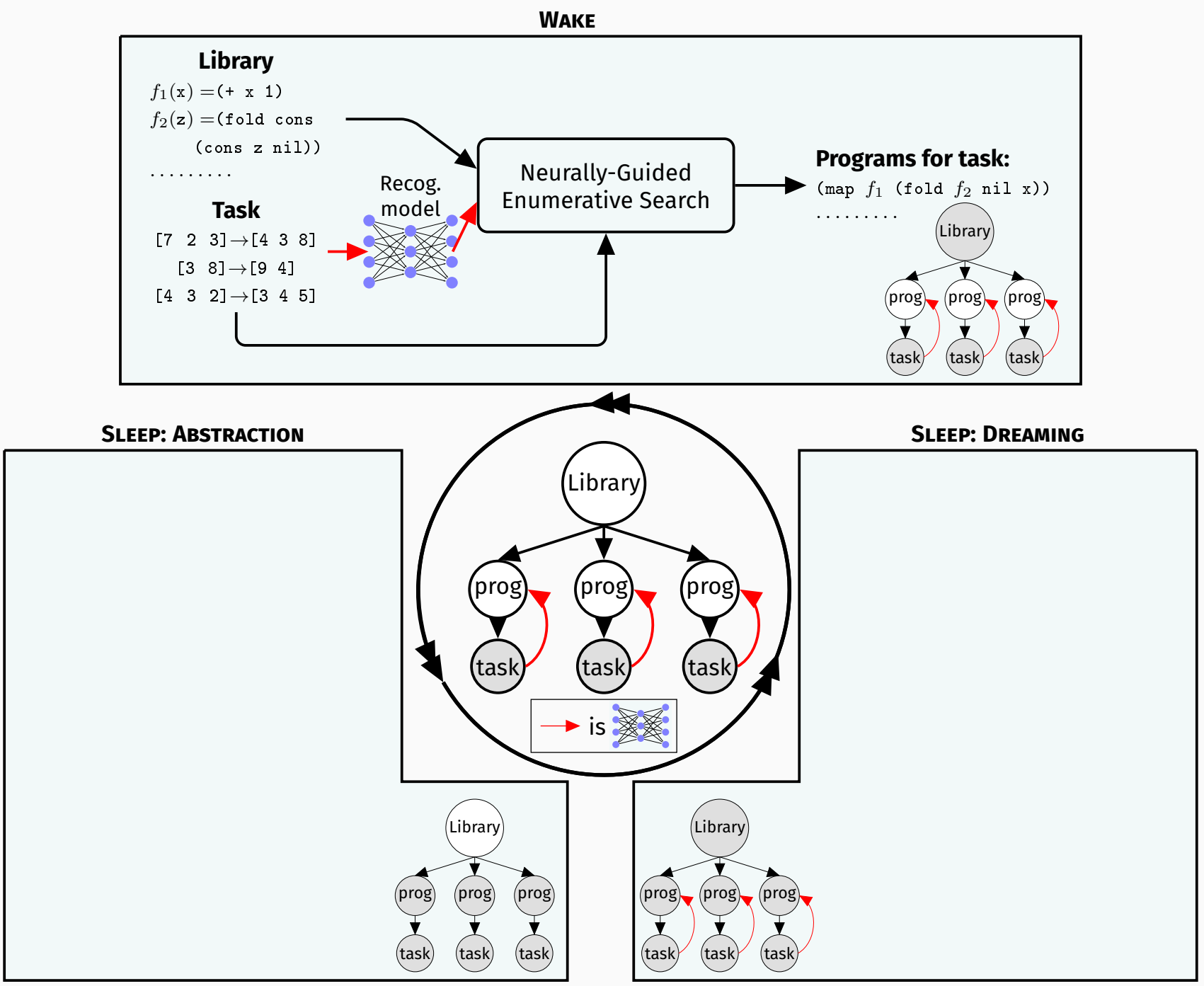
Wake

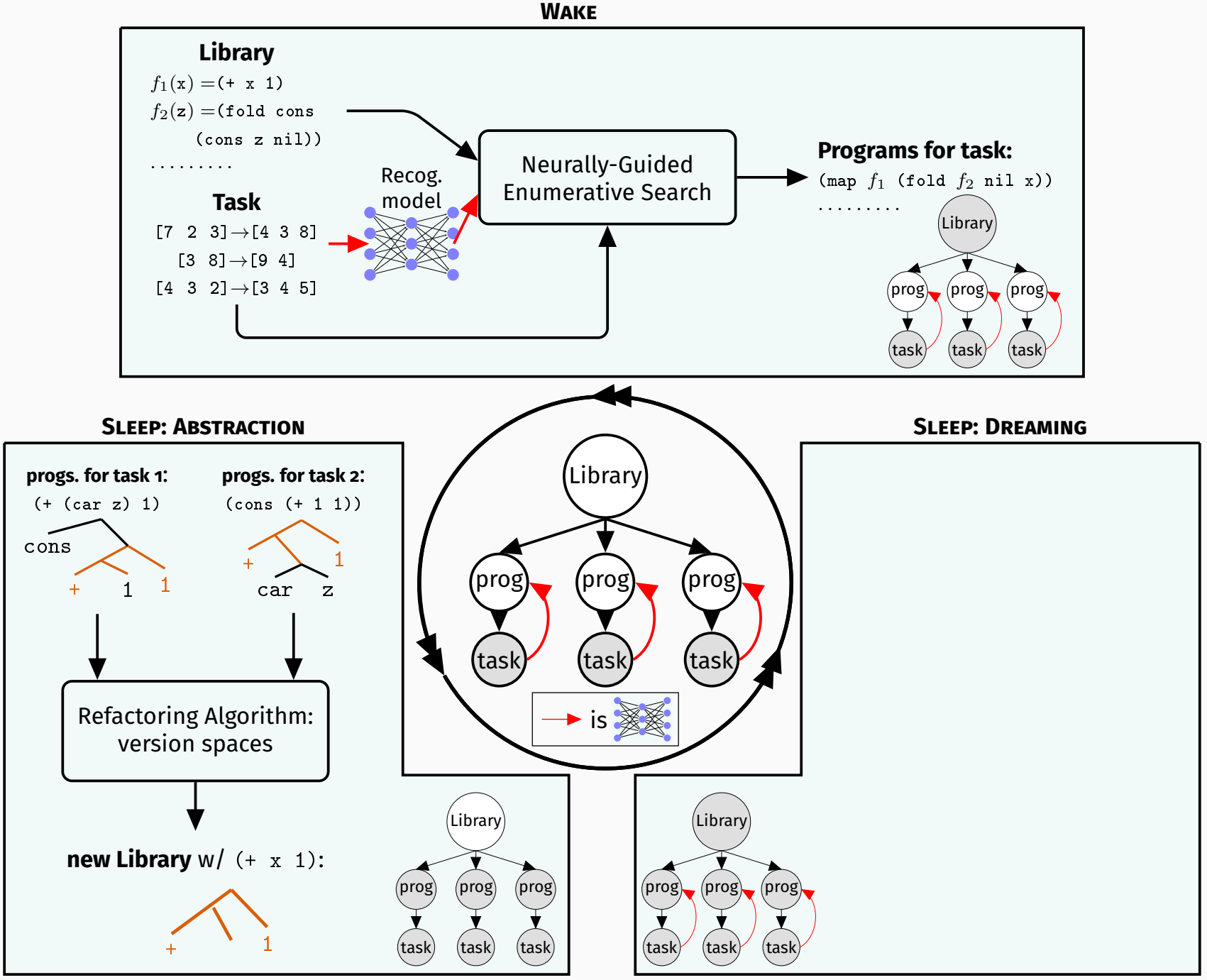
Sleep: Abstraction

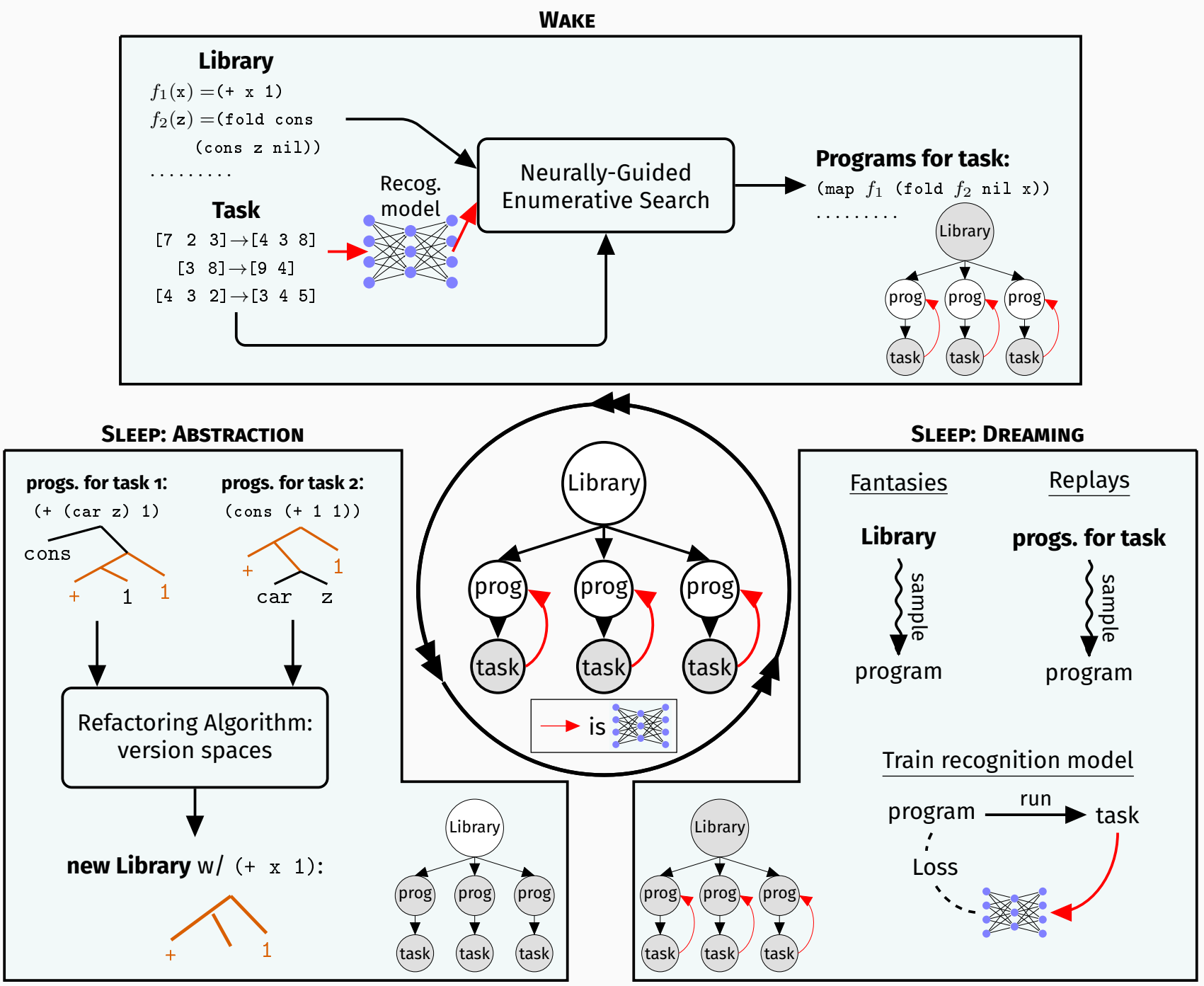


Sleep: Dreaming

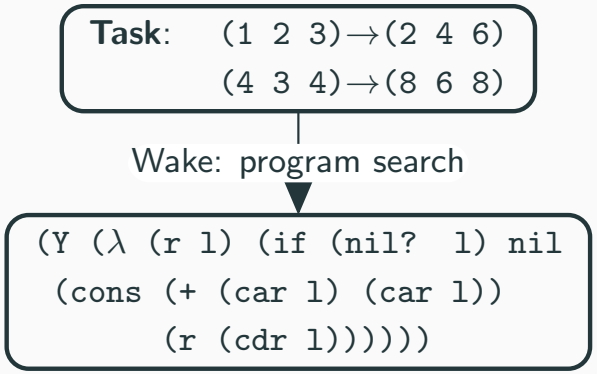


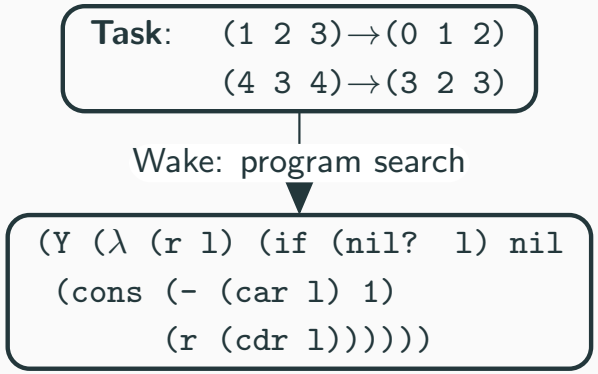

WAKE
Library
f_1(x) =(+ x 1)
f_2(z) =(fold cons
(cons z nil))
.....
Task
[7 2 3]→[4 3 8]
[3 8]→[9 4]
[4 3 2]→[3 4 5]
Recog. model
Neurally-Guided Enumerative Search
Programs for task:
(map f_1 (fold f_2 nil x))
.....
Library
prog
task
SLEEP: ABSTRACTION
SLEEP: DREAMING
Library
prog
task
is
Library
prog
task
Library
prog
task


WAKE
Library
f_1(x) =(+ x 1)
f_2(z) =(fold cons
(cons z nil))
.....
Task
[7 2 3]→[4 3 8]
[3 8]→[9 4]
[4 3 2]→[3 4 5]
Recog. model
Neurally-Guided Enumerative Search
Programs for task:
(map f_1 (fold f_2 nil x))
.....
Library
prog
task
SLEEP: ABSTRACTION
progs. for task 1:
(+ (car z) 1)
progs. for task 2:
(cons (+ 1 1))
cons
car
z
Refactoring Algorithm: version spaces
new Library w/ (+ x 1):
SLEEP: DREAMING
is

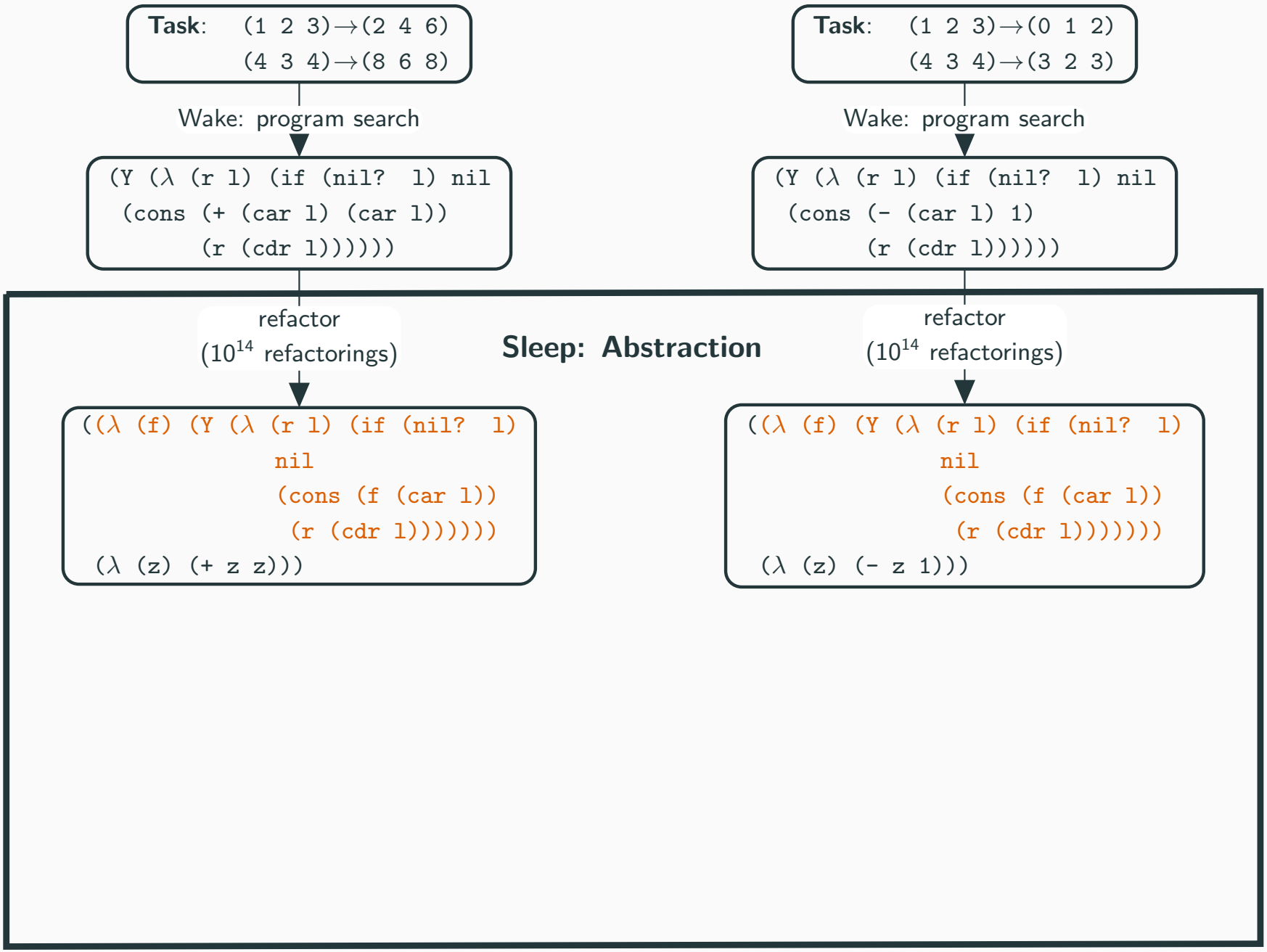

WAKE
Library
f_1(x) =(+ x 1)
f_2(z) =(fold cons
(cons z nil))
.....
Task
[7 2 3]→[4 3 8]
[3 8]→[9 4]
[4 3 2]→[3 4 5]
Recog. model
Neurally-Guided Enumerative Search
Programs for task:
(map f_1 (fold f_2 nil x))
.....
Library
prog
task
SLEEP: ABSTRACTION
progs. for task 1:
(+ (car z) 1)
cons
progs. for task 2:
(cons (+ 1 1))
car z
Refactoring Algorithm: version spaces
new Library w/ (+ x 1):
Library
is
SLEEP: DREAMING
Fantasies
Replays
Library
progs. for task
sample
program
Train recognition model
program
run
task
Loss

Abstraction Sleep: Growing the library via refactoring

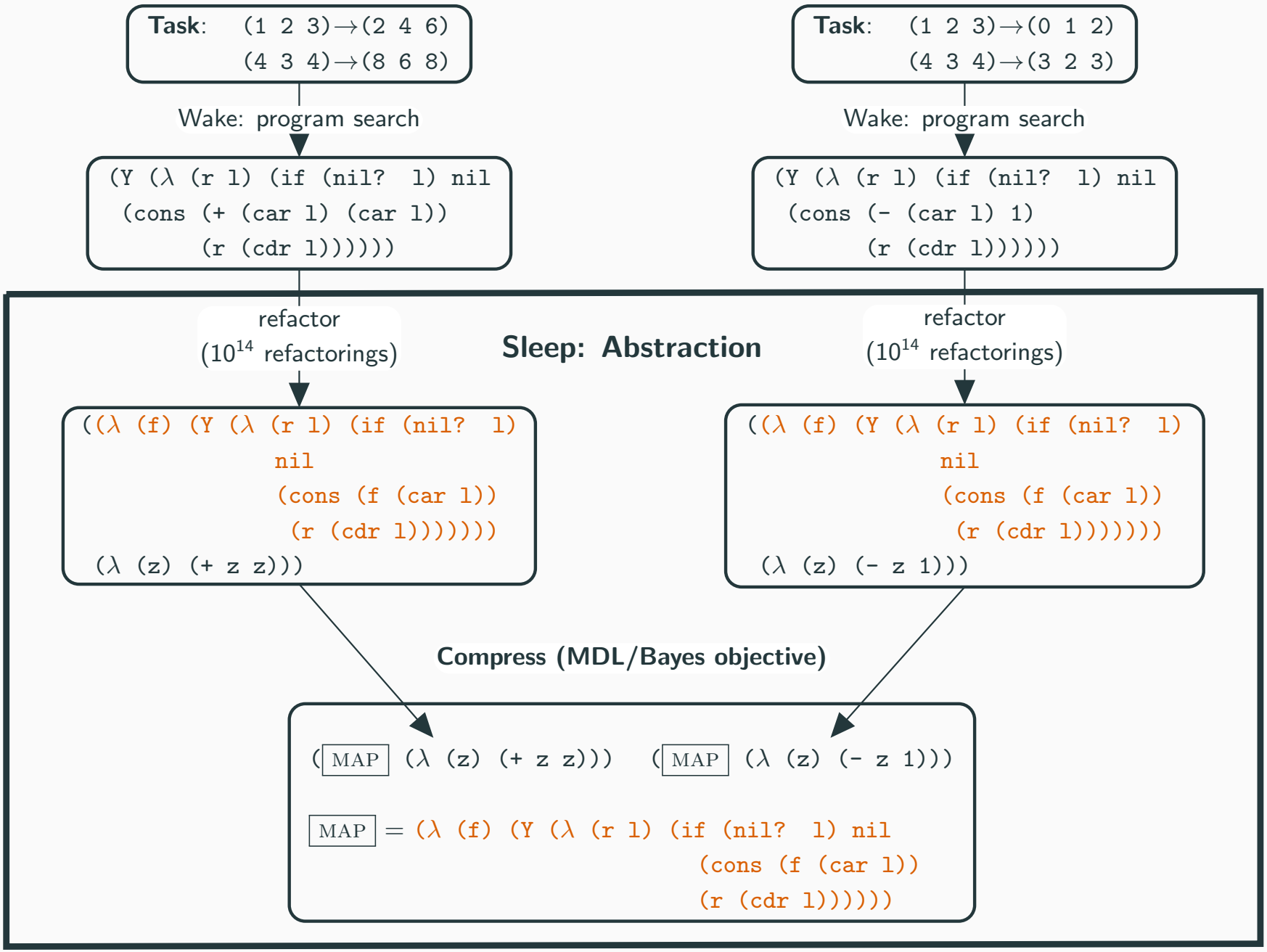




Abstraction Sleep: Growing the library via refactoring



Abstraction Sleep: Growing the library via refactoring



Version space algebra for refactoring

$$
\begin{aligned}
\text{Expr} \rightarrow\ & \text{var} \\
\mid\ & \lambda\text{var.Expr} \\
\mid\ & (\text{Expr Expr}) \\
\mid\ & \text{primitive}
\end{aligned}
$$

Version space algebra for refactoring

$$
\begin{array}{l}
\text{Expr} \rightarrow \text{var} \\
\quad \mid \lambda\text{var.Expr} \\
\quad \mid (\text{Expr Expr}) \\
\quad \mid \text{primitive}
\end{array}
\qquad
\begin{array}{ll}
\text{VS} \rightarrow \text{var} & \\
\quad \mid \lambda\text{var.VS} & \\
\quad \mid (\text{VS VS}) & \\
\quad \mid \text{primitive} & \\
\quad \mid \text{VS} \uplus \text{VS} & \textit{nondeterministic choice} \\
\quad \mid \Lambda & \textit{choose any expression} \\
\quad \mid \varnothing & \textit{choose no expression}
\end{array}
$$

Version space algebra for refactoring

$$
\begin{array}{lll}
\text{Expr} \to \text{var} & \text{VS} \to \text{var} & \\
\quad \mid \lambda\text{var}.\text{Expr} & \quad \mid \lambda\text{var}.\text{VS} & \\
\quad \mid (\text{Expr Expr}) & \quad \mid (\text{VS VS}) & \\
\quad \mid \text{primitive} & \quad \mid \text{primitive} & \\
 & \quad \mid \text{VS} \uplus \text{VS} & \textit{nondeterministic choice} \\
 & \quad \mid \Lambda & \textit{choose any expression} \\
 & \quad \mid \varnothing & \textit{choose no expression}
\end{array}
$$

what version spaces mean:

$$
\begin{array}{ll}
[\![\text{var}]\!] = \{\text{var}\} & [\![v_1 \uplus v_2]\!] = \{e : v \in \{v_1, v_2\}, e \in [\![v]\!]\} \\
[\![\lambda x.v]\!] = \{\lambda x.e : e \in [\![v]\!]\} & [\![(v_1\, v_2)]\!] = \{(e_1 e_2) : e_1 \in [\![v_1]\!], e_2 \in [\![v_2]\!]\} \\
[\![\varnothing]\!] = \varnothing & [\![\Lambda]\!] = \Lambda
\end{array}
$$

Using version spaces

$$
\begin{aligned}
\text{VS} \rightarrow\ & \text{var} \mid \lambda\text{var.VS} \mid (\text{VS VS}) \mid \text{primitive} \\
\mid\ & \text{VS} \uplus \text{VS} && \textit{nondeterministic choice} \\
\mid\ & \Lambda && \textit{choose any expression} \\
\mid\ & \varnothing && \textit{choose no expression}
\end{aligned}
$$

Using version spaces

$$\begin{aligned}
\text{VS} \rightarrow\ & \text{var} \mid \lambda\text{var.VS} \mid (\text{VS VS}) \mid \text{primitive} \\
\mid\ & \text{VS} \uplus \text{VS} && \textit{nondeterministic choice} \\
\mid\ & \Lambda && \textit{choose any expression} \\
\mid\ & \varnothing && \textit{choose no expression}
\end{aligned}$$

exploit the fact that $e \in [\![v]\!]$ can be efficiently computed:

$$\textsc{refactor}(v|\text{Lib}) = \begin{cases} e, \text{ if } e \in \text{Lib and } e \in [\![v]\!] \\ \textsc{refactor}'(v|\text{Lib}), \text{ otherwise.} \end{cases}$$

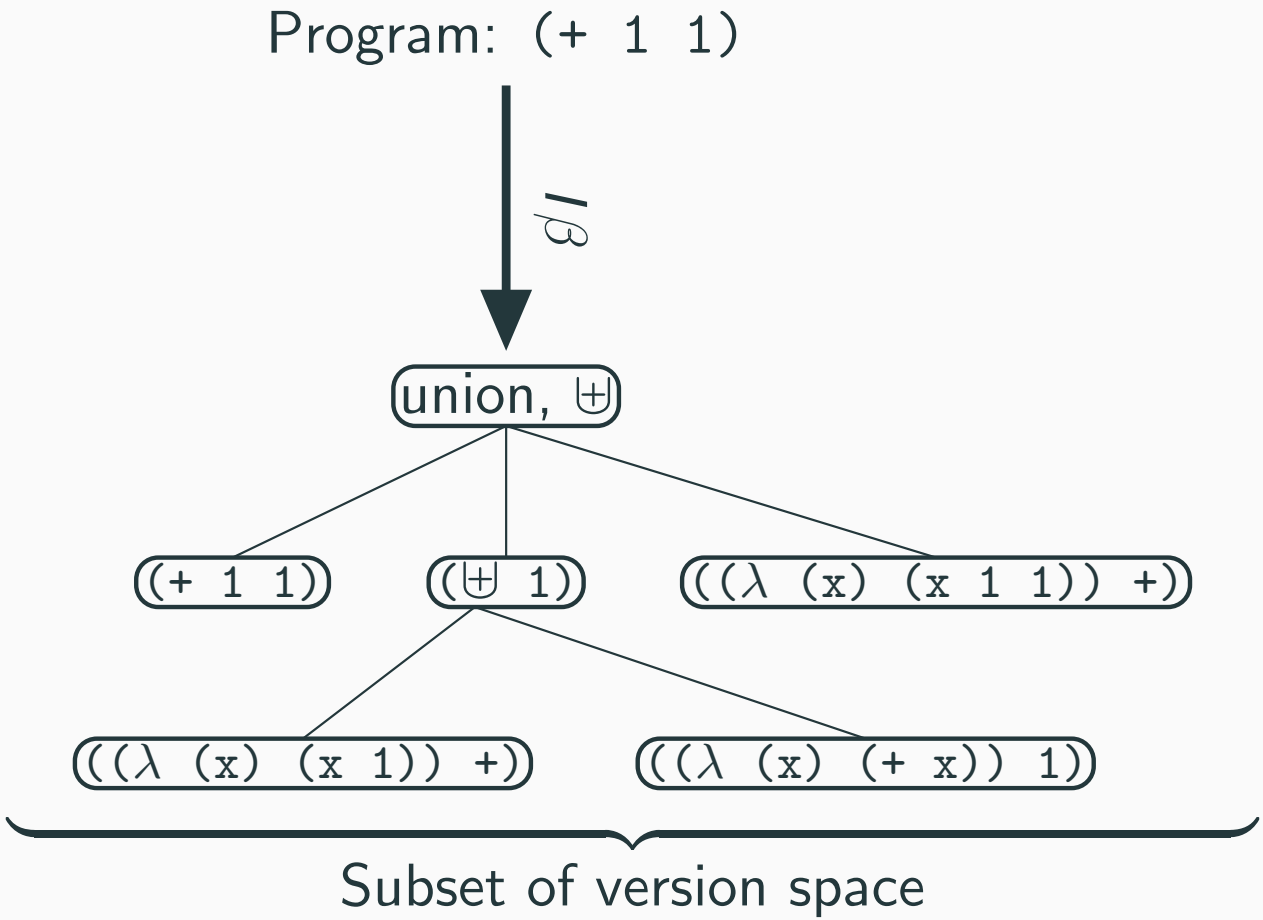
$$\textsc{refactor}'(v|\text{Lib}) = v, \text{ if } v \text{ is a primitive or variable}$$

$$\textsc{refactor}'(\lambda x.b|\text{Lib}) = \lambda x.\textsc{refactor}(b|\text{Lib})$$

$$\textsc{refactor}'(v_1\, v_2|\text{Lib}) = (\textsc{refactor}(v_1|\text{Lib})\ \ \textsc{refactor}(v_2|\text{Lib}))$$

Using version spaces

$$\begin{aligned} \text{VS} \rightarrow\ & \text{var} \mid \lambda\text{var.VS} \mid (\text{VS VS}) \mid \text{primitive} \\ \mid\ & \text{VS} \uplus \text{VS} && \textit{nondeterministic choice} \\ \mid\ & \Lambda && \textit{choose any expression} \\ \mid\ & \varnothing && \textit{choose no expression} \end{aligned}$$



Using version spaces

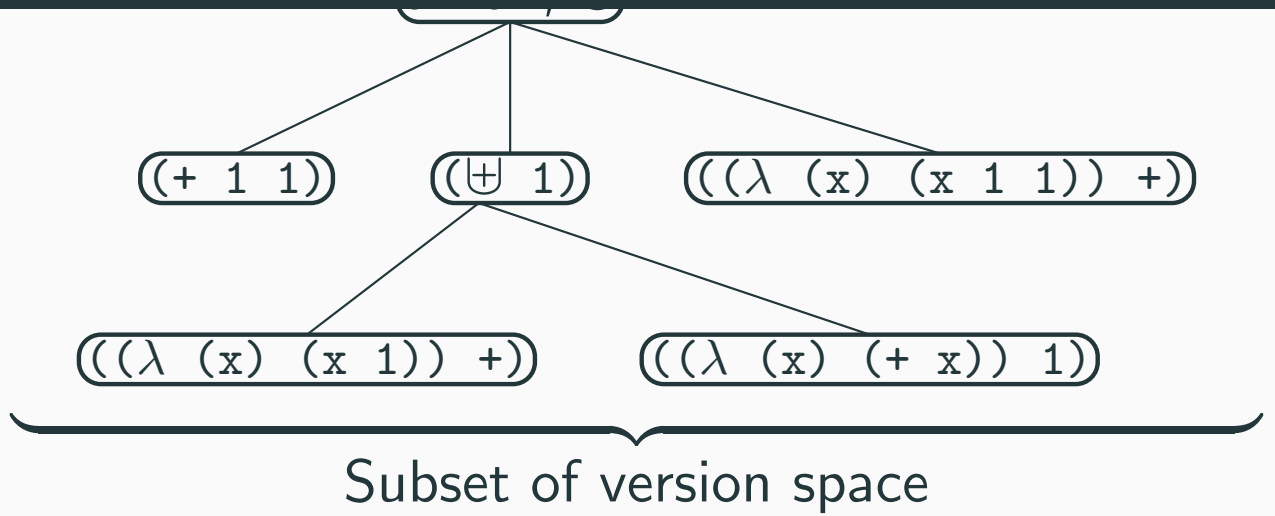
$$\begin{aligned} VS \rightarrow\ & \text{var} \mid \lambda\text{var}.VS \mid (VS\ VS) \mid \text{primitive} \\ \mid\ & VS \uplus VS && \textit{nondeterministic choice} \\ \mid\ & \Lambda && \textit{choose any expression} \\ \mid\ & \varnothing && \textit{choose no expression} \end{aligned}$$

completeness: $I\beta$ gets all the refactorings

let $v_2 = I\beta(v_1)$ and $e_1 \in [\![v_1]\!]$. for any $e_2 \longrightarrow e_1$ then $e_2 \in [\![v_2]\!]$

consistency: $I\beta$ only gets valid refactorings

let $v_2 = I\beta(v_1)$ and $e_2 \in [\![v_2]\!]$. then there is a $e_1 \in [\![v_1]\!]$ where $e_2 \longrightarrow e_1$



DreamCoder Domains

List Processing

Sum List

```
[1 2 3] → 6
[4 6 8 1] → 17
```

Double

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

Check Evens

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

Text Editing

Abbreviate

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

Drop Last Characters

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

Extract

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

Regexes

Phone Numbers

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

Currency

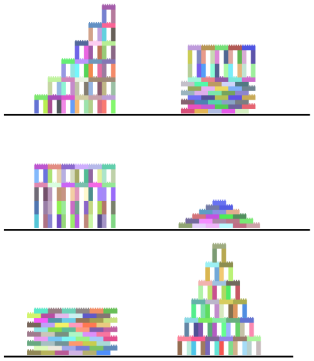
```
$100.25
$4.50
```

Dates

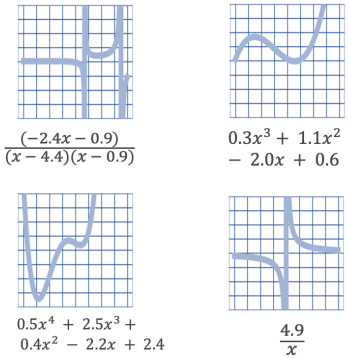
```
Y1775/0704
Y2000/0101
```

LOGO Graphics

Block Towers

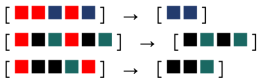


Symbolic Regression



Recursive Programming

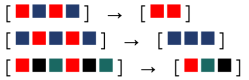
Filter



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

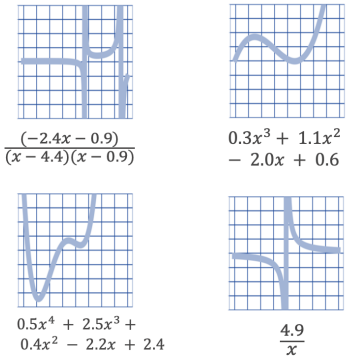
Dates

Y1775/0704
Y2000/0101

LOGO Graphics

Block Towers

Symbolic Regression



Recursive Programming

Filter

[ ] → [ ]
[ ] → [ ]
[ ] → [ ]

Index List

0, [ ] →
1, [ ] →
1, [ ] →

Length

[ ] → 4
[ ] → 6
[ ] → 3

Every Other

[ ] → [ ]
[ ] → [ ]
[ ] → [ ]

Physics

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

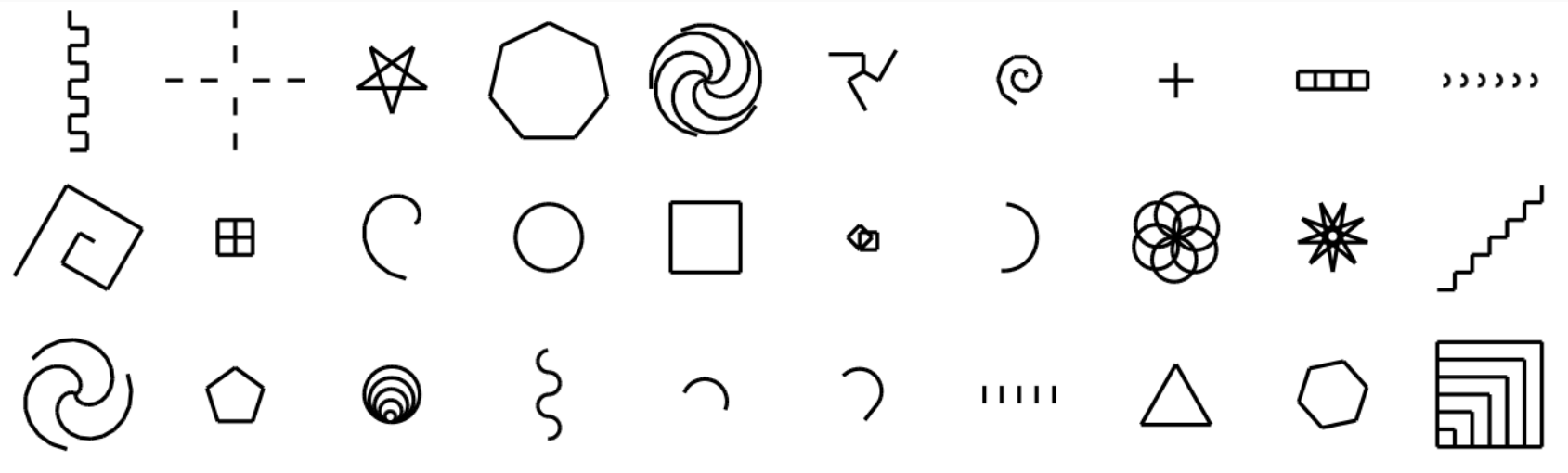
$$\vec{a} = \frac{1}{m}\sum_i \vec{F}_i$$

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

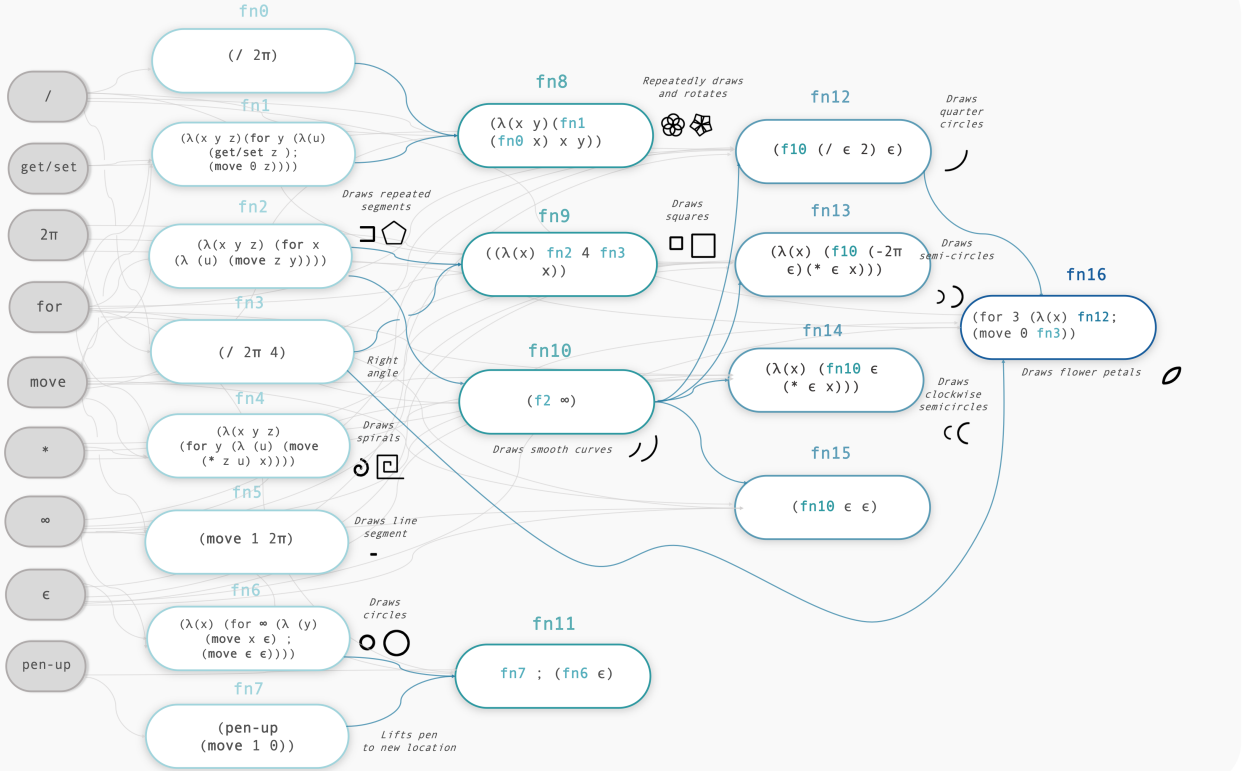
$$R_{total} = \left(\Sigma_i \frac{1}{R_i}\right)^{-1}$$

LOGO Graphics

30 out of 160 tasks



LOGO Graphics – learning interpretable library of concepts



```
(fn8 5 (fn4 (* ϵ 2) ∞ ϵ))
```

```
(for 2 (λ (x)
(fn14 2); (fn13 2)))
```

```
(for 7 (λ (x) (fn9 x)))
```

```
(fn10 (* (fn0 7) 3) 3)
```

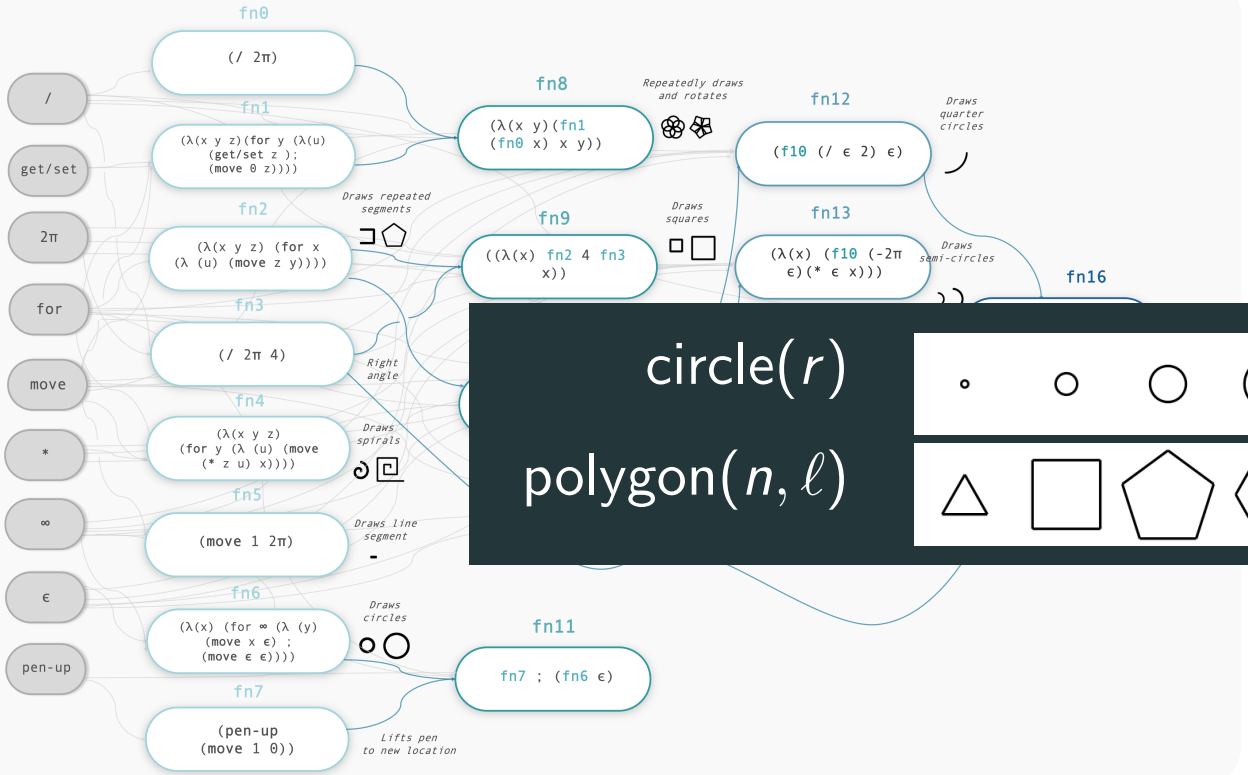
```
(fn8  6 (fn7 ; fn5 ;
fn7 ; fn5))
```

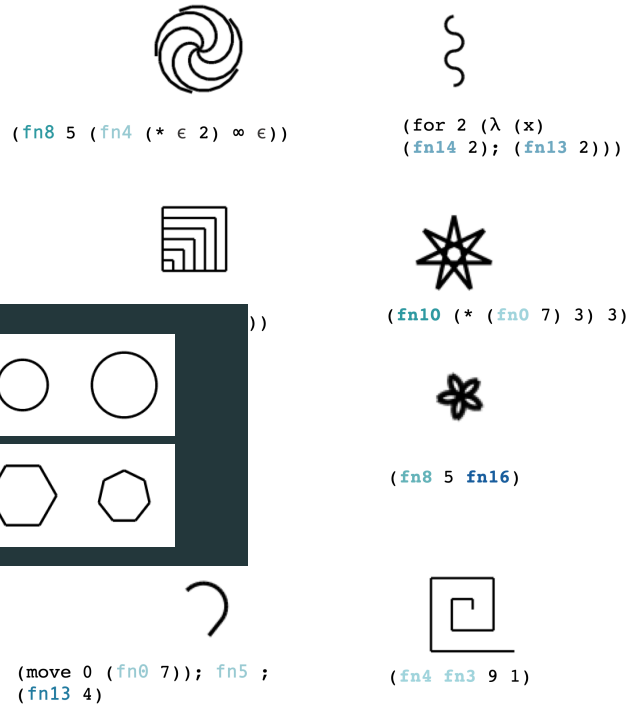
```
(fn8 5 fn16)
```

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

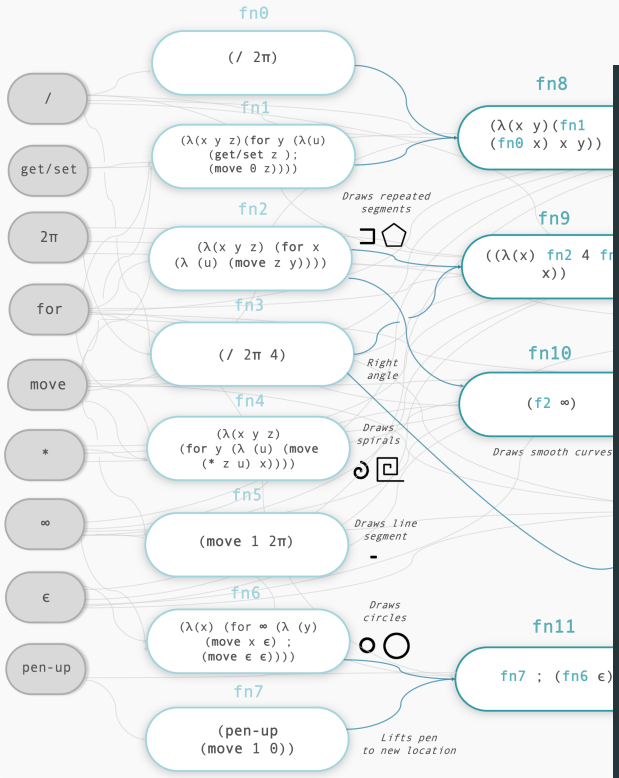
```
(fn4 fn3 9 1)
```

LOGO Graphics – learning interpretable library of concepts

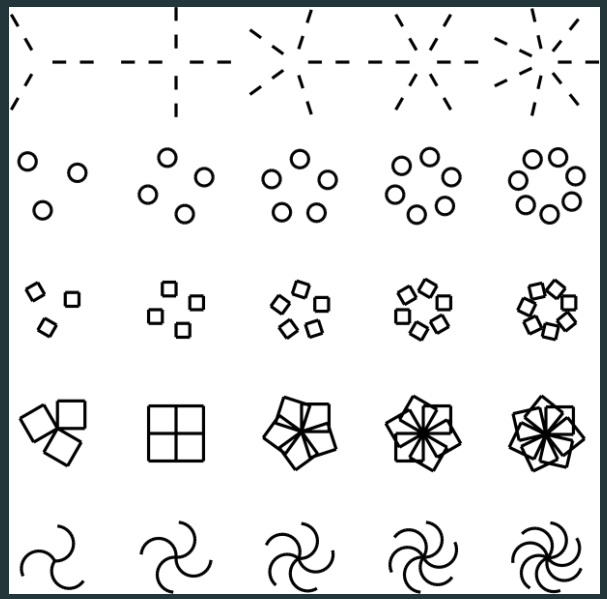


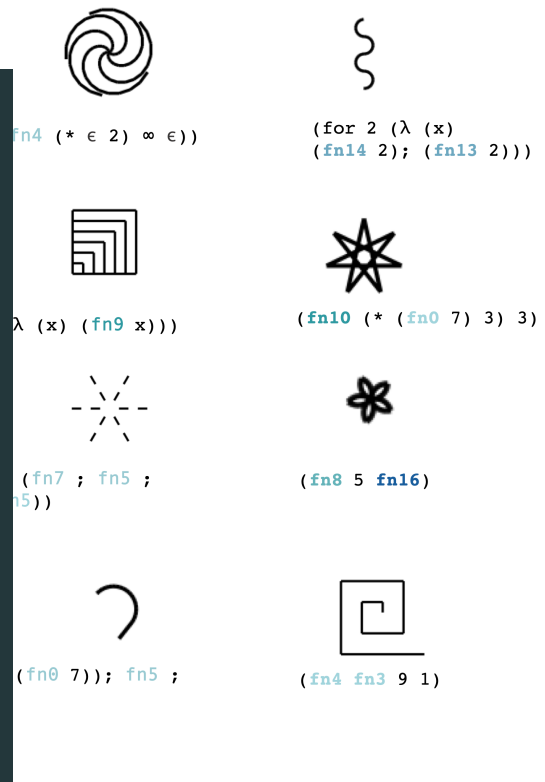


LOGO Graphics – learning interpretable library of concepts



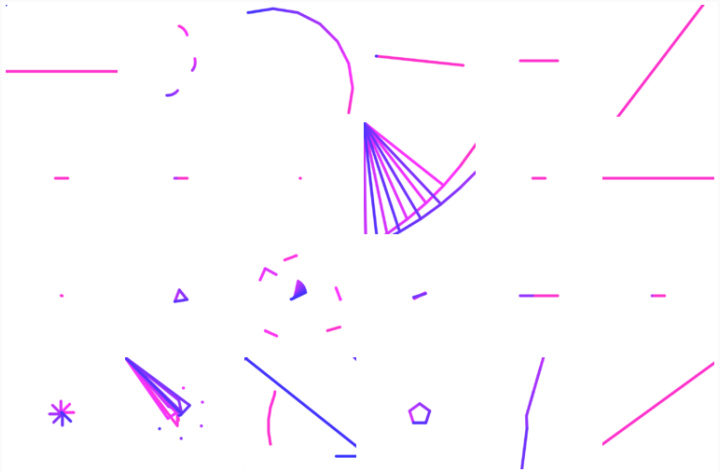
radial symmetry(n, body)



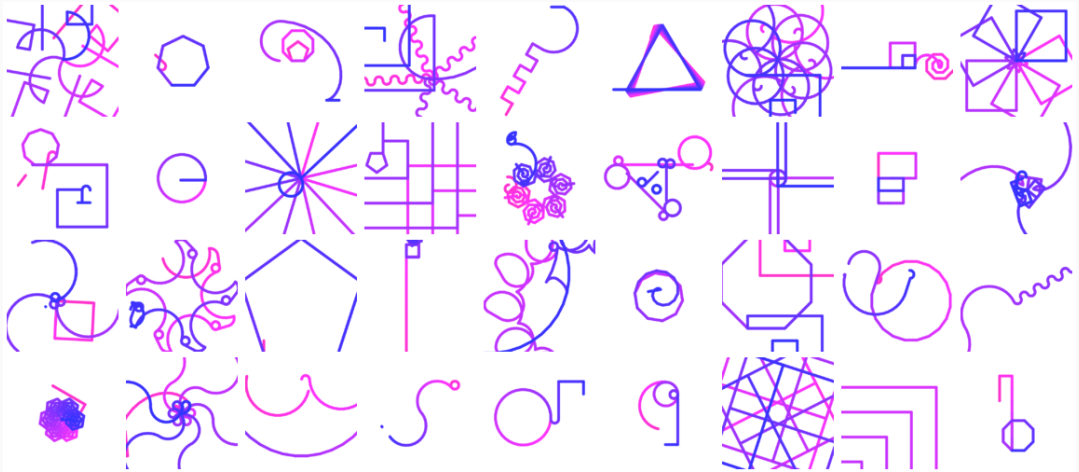


what does DreamCoder dream of?

before learning

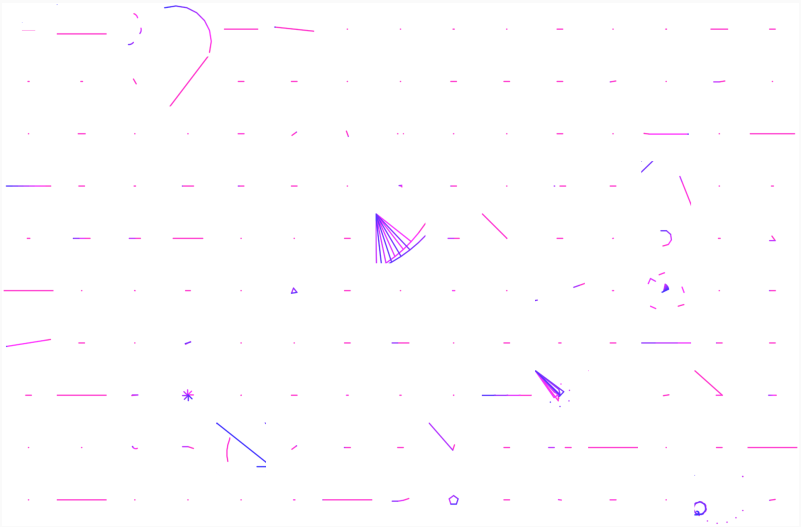


after learning

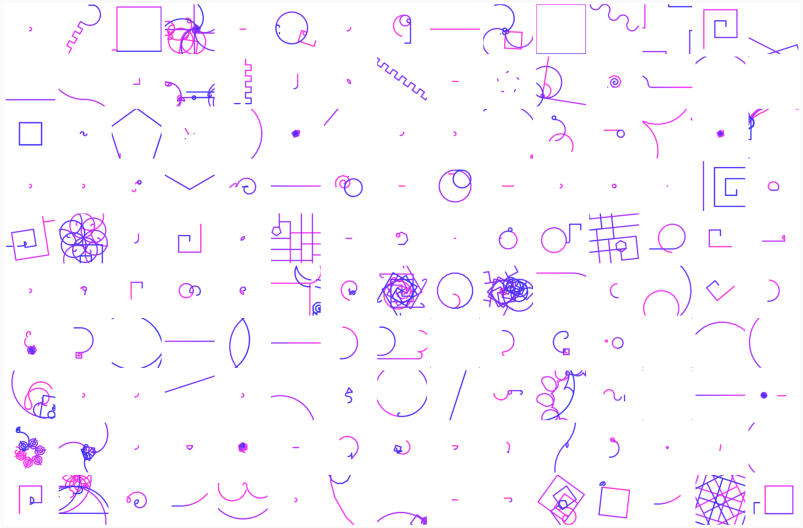


what does DreamCoder dream of?

before learning

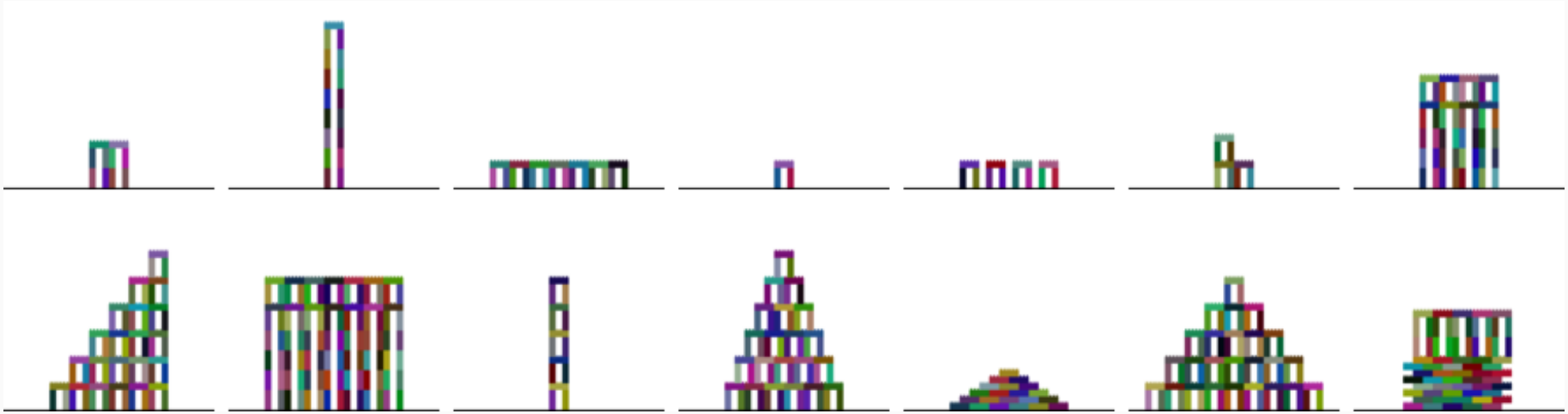


after learning



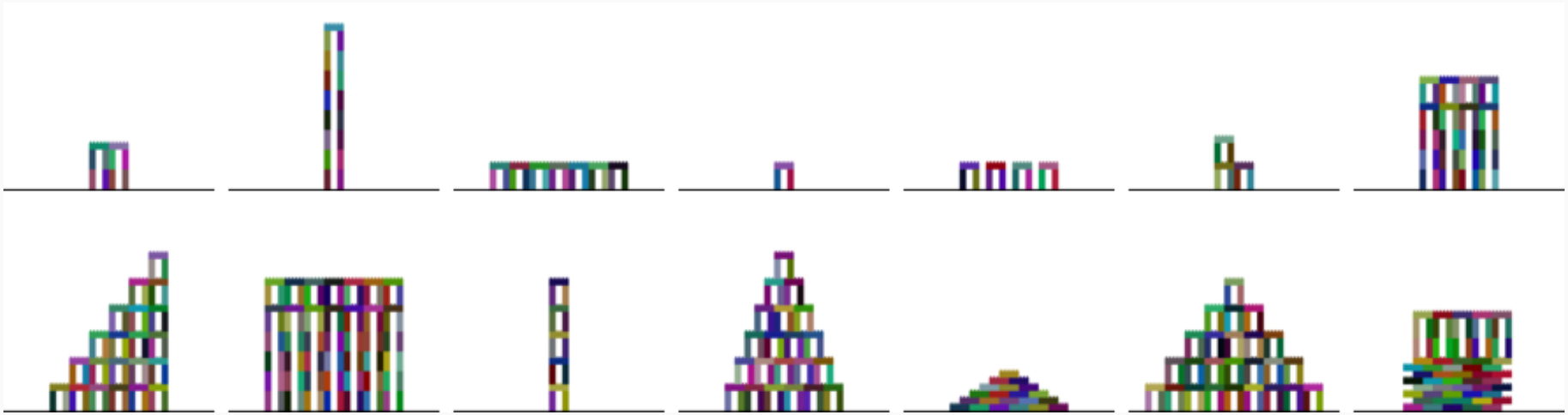
Planning to build towers

example tasks (112 total)



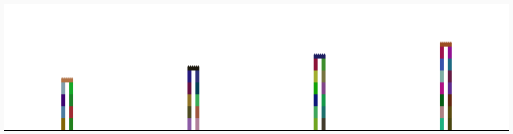
Planning to build towers

example tasks (112 total)

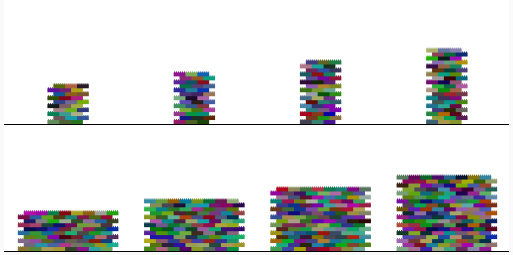


learned library routines ($\approx$ 20 total)

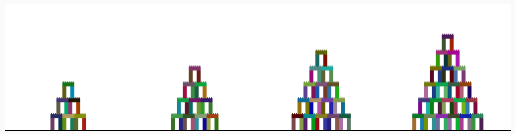
arch(h)



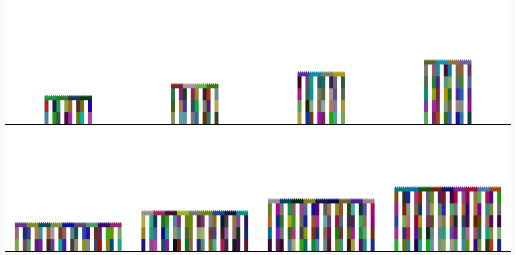
wall(w, h)



pyramid(h)

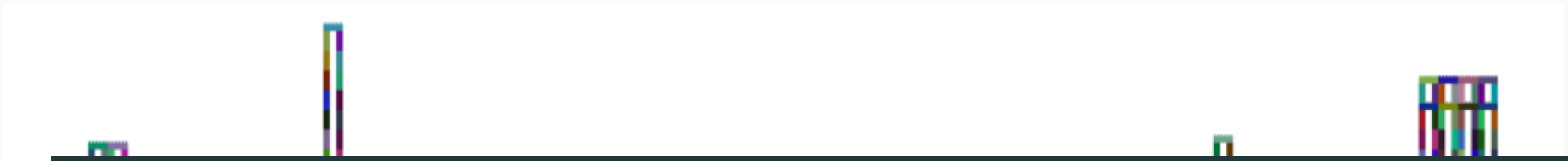


bridge(w, h)



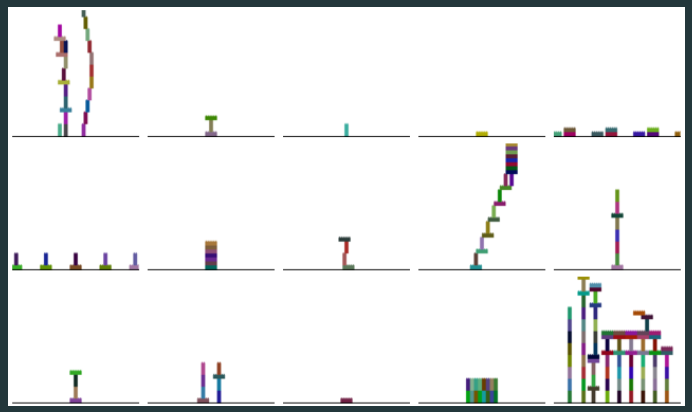
Planning to build towers

example tasks (112 total)

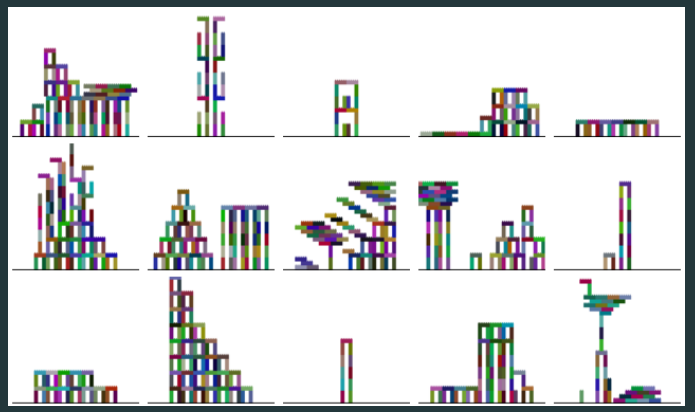


lea

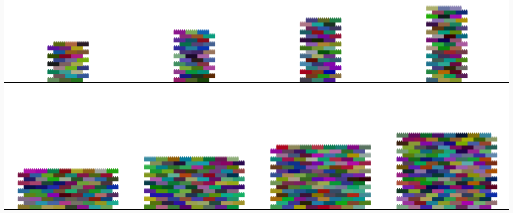
dreams before learning



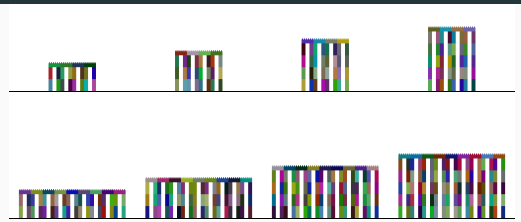
dreams after learning



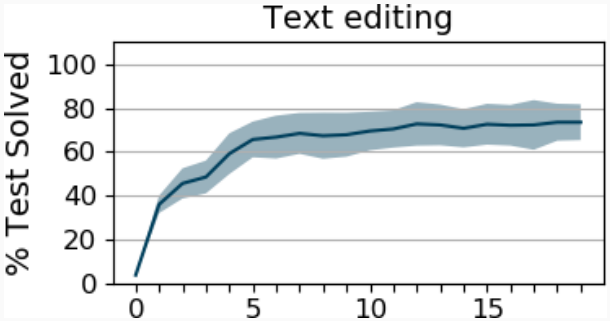
wall(w, h)

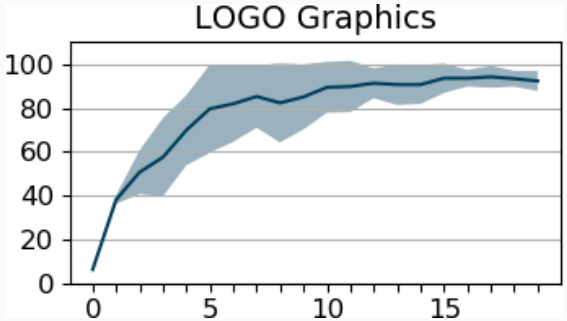


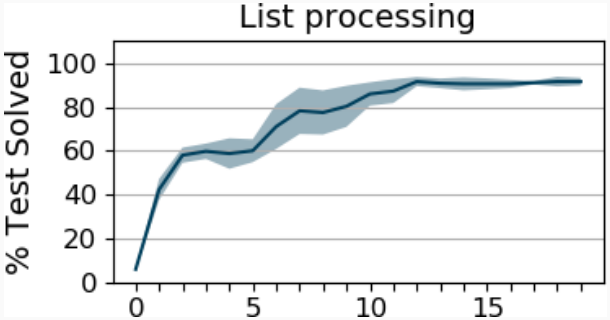
bridge(w, h)

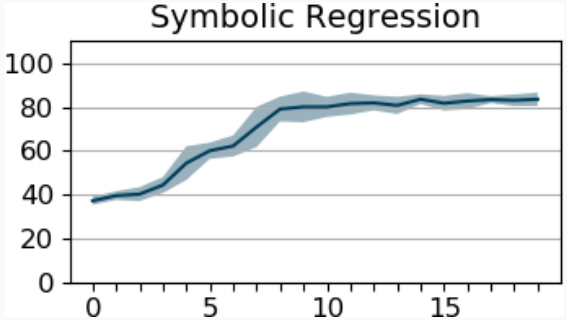


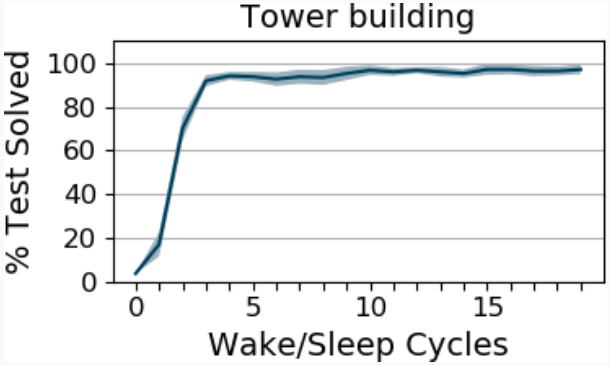
synergy between dreaming and library learning

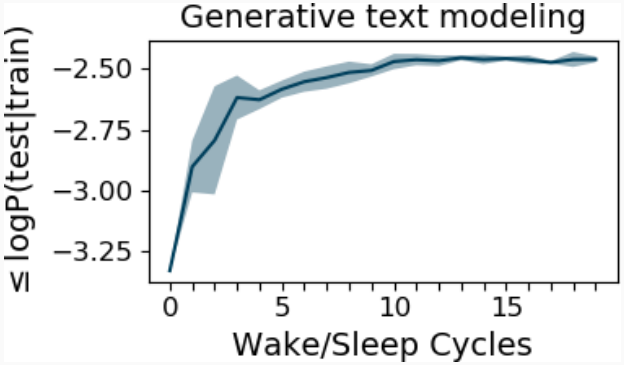


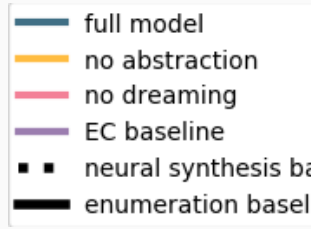




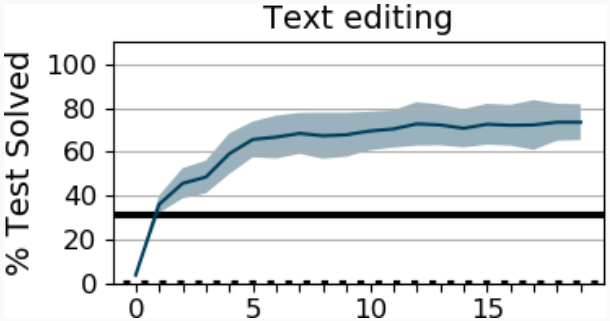


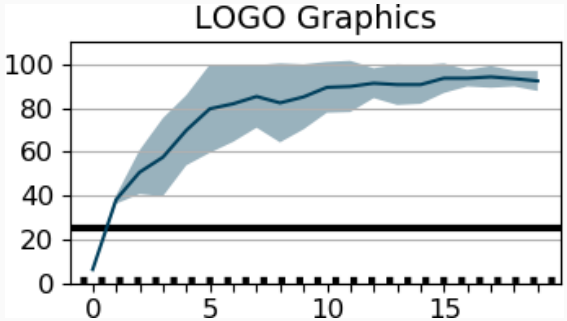


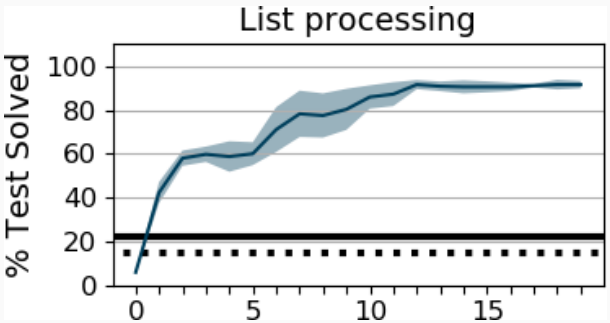


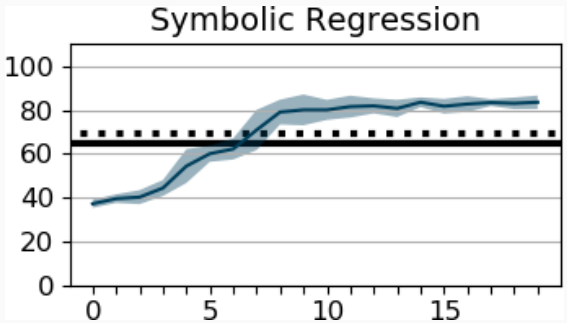


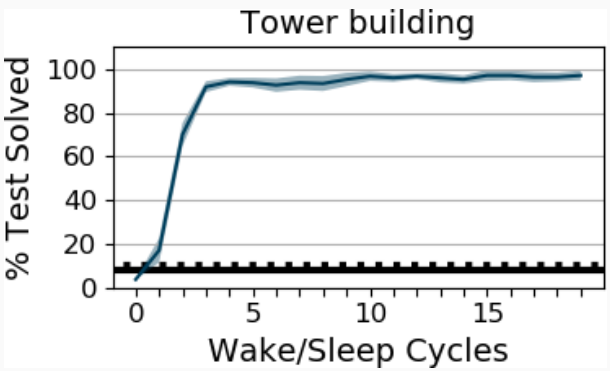
synergy between dreaming and library learning

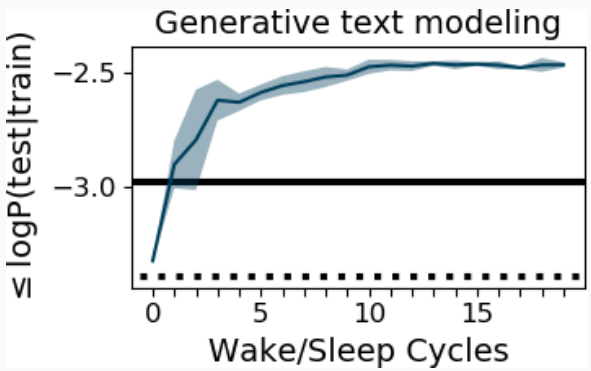


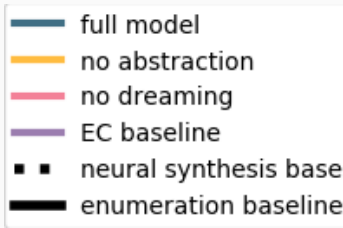




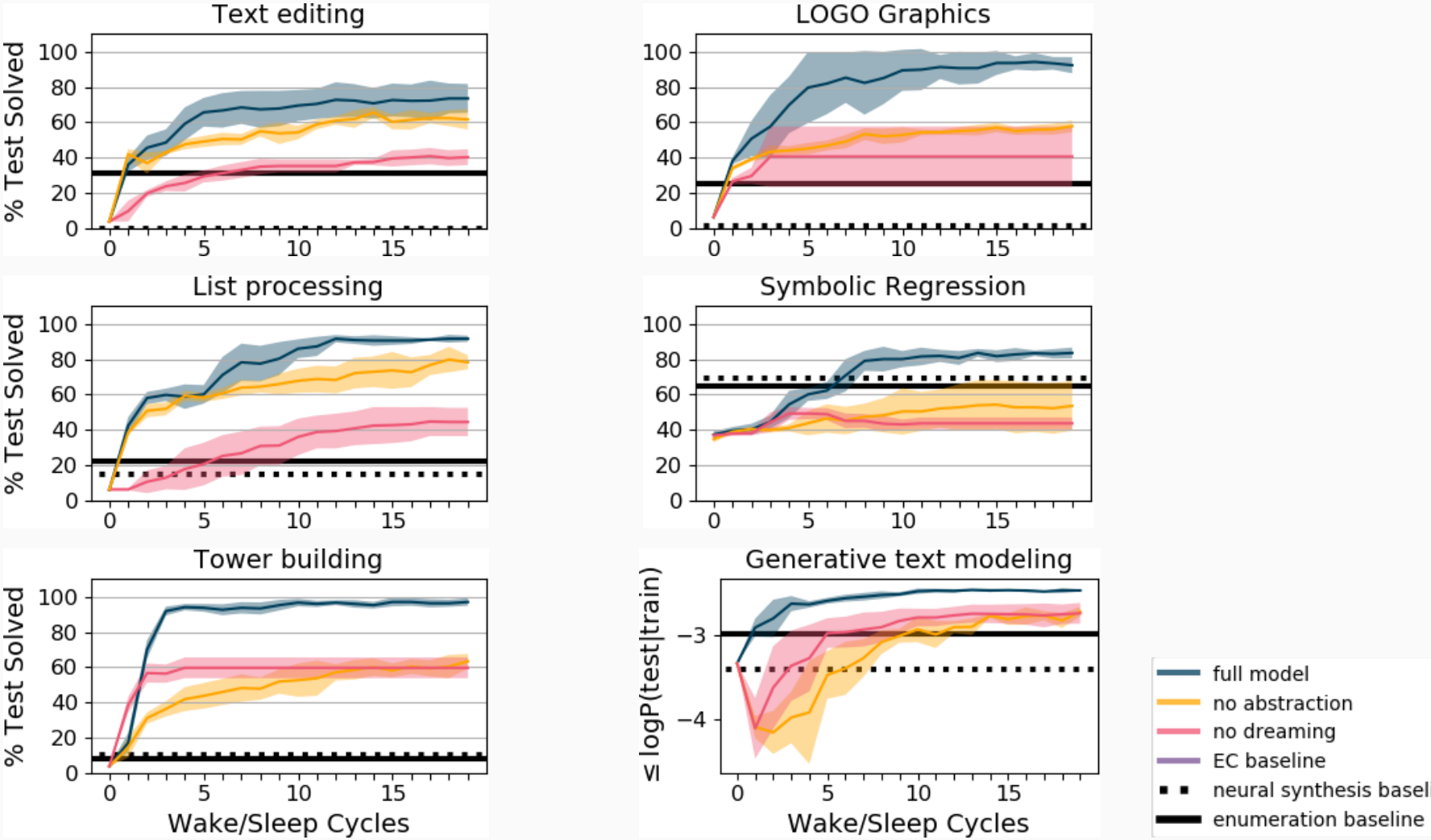




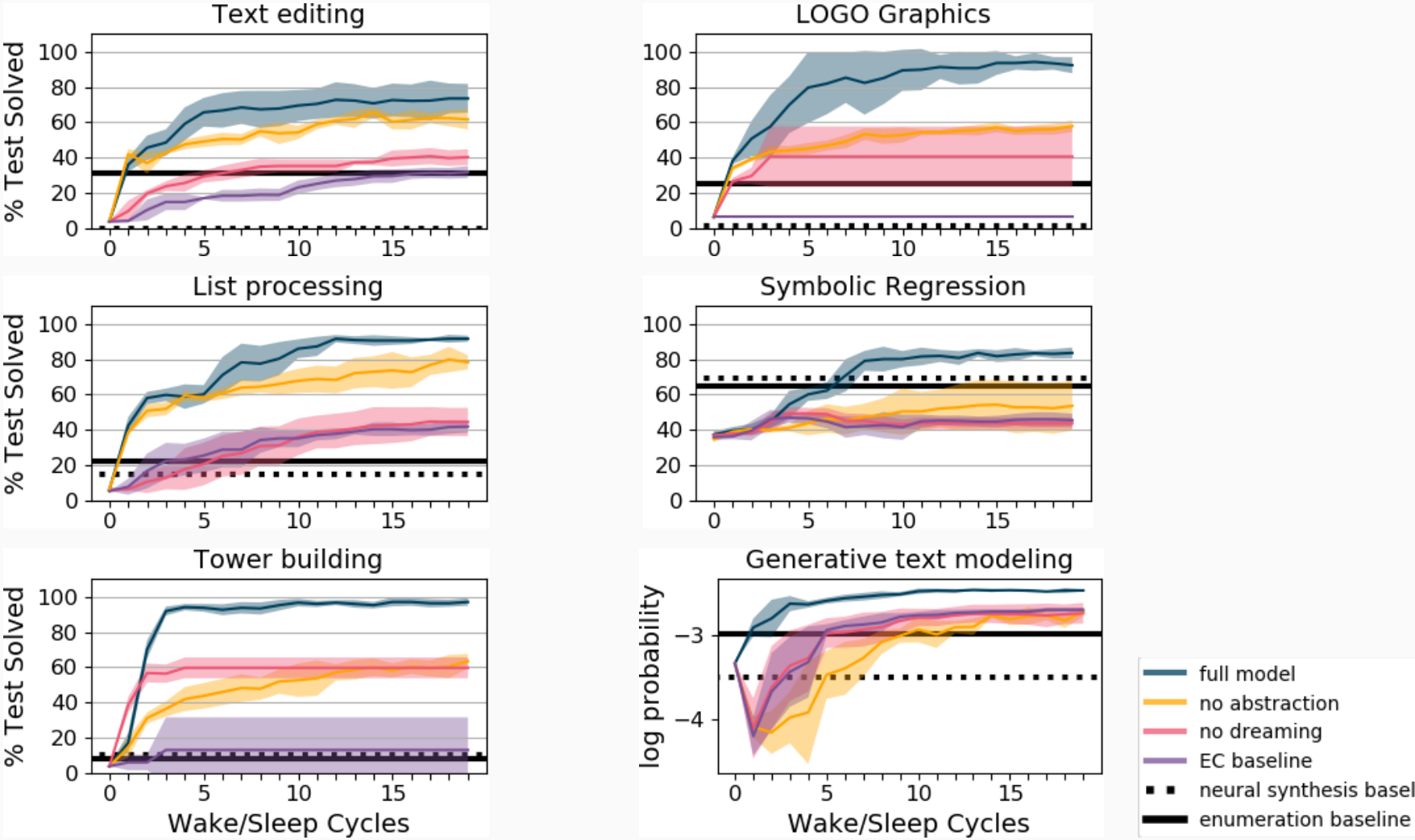




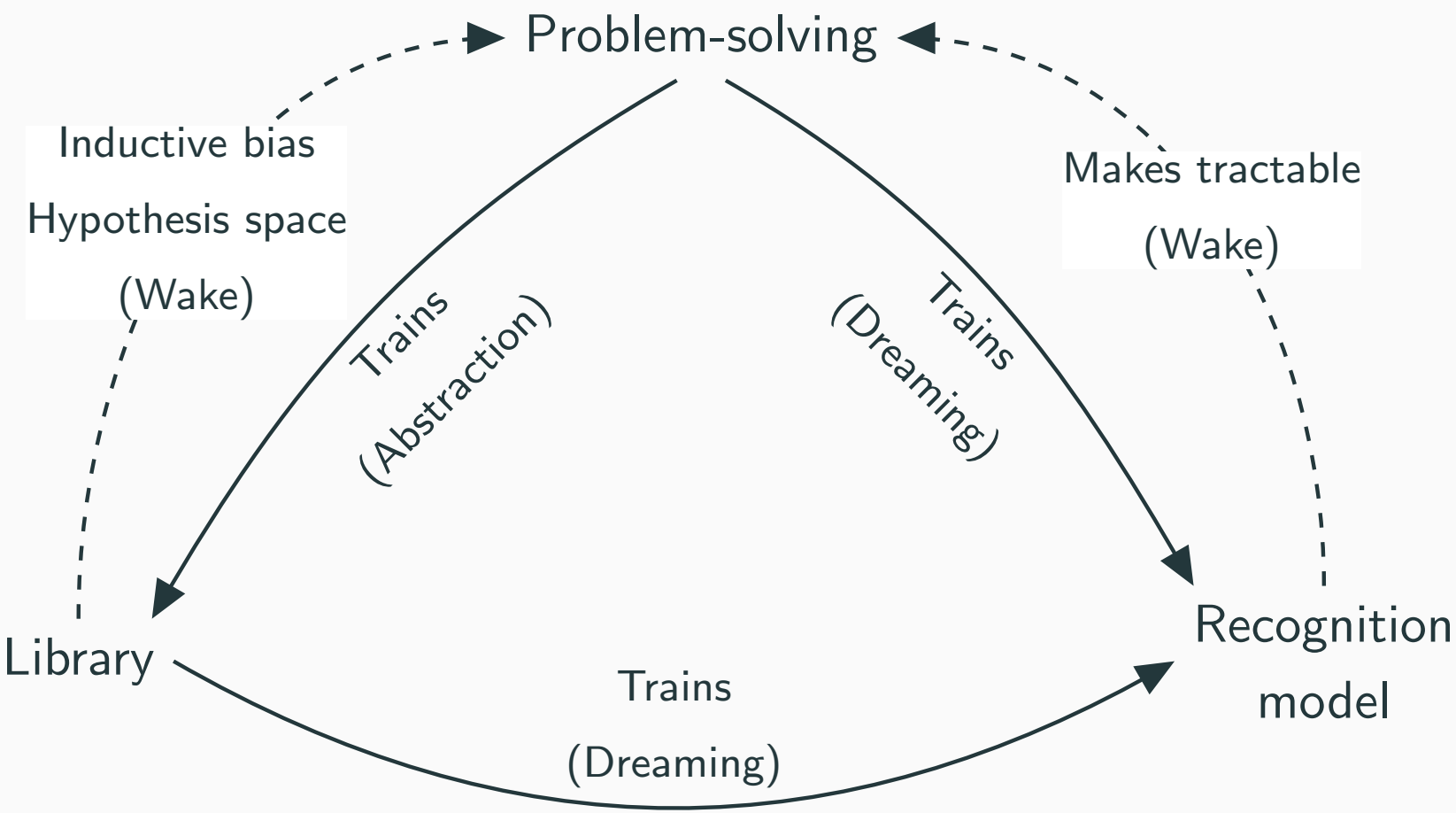
synergy between dreaming and library learning



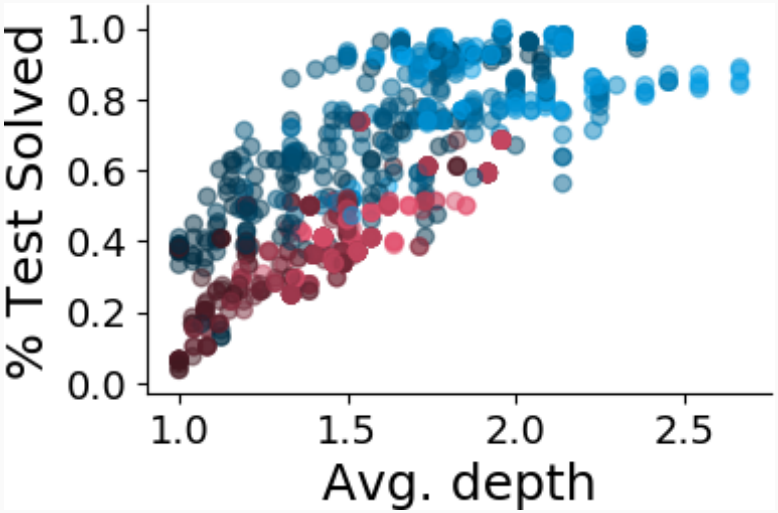
synergy between dreaming and library learning

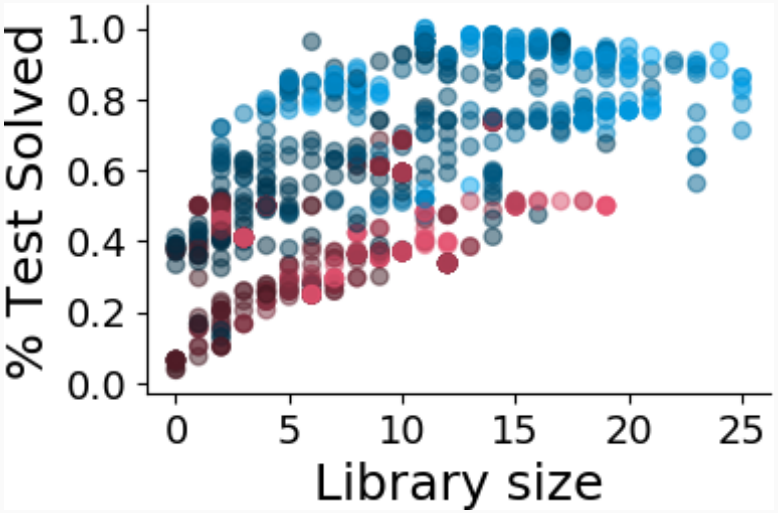


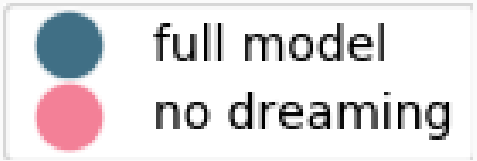
synergy between dreaming and library learning



Evidence for dreaming bootstrapping better libraries







Dark→Light: Early in learning→Later in learning

From learning libraries to learning languages

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

- "beginner:" basic domain-specific procedures, only easiest problems have short solutions
- "expert:" learned library allowing hardest problems to have short meaningful solutions

From learning libraries to learning languages

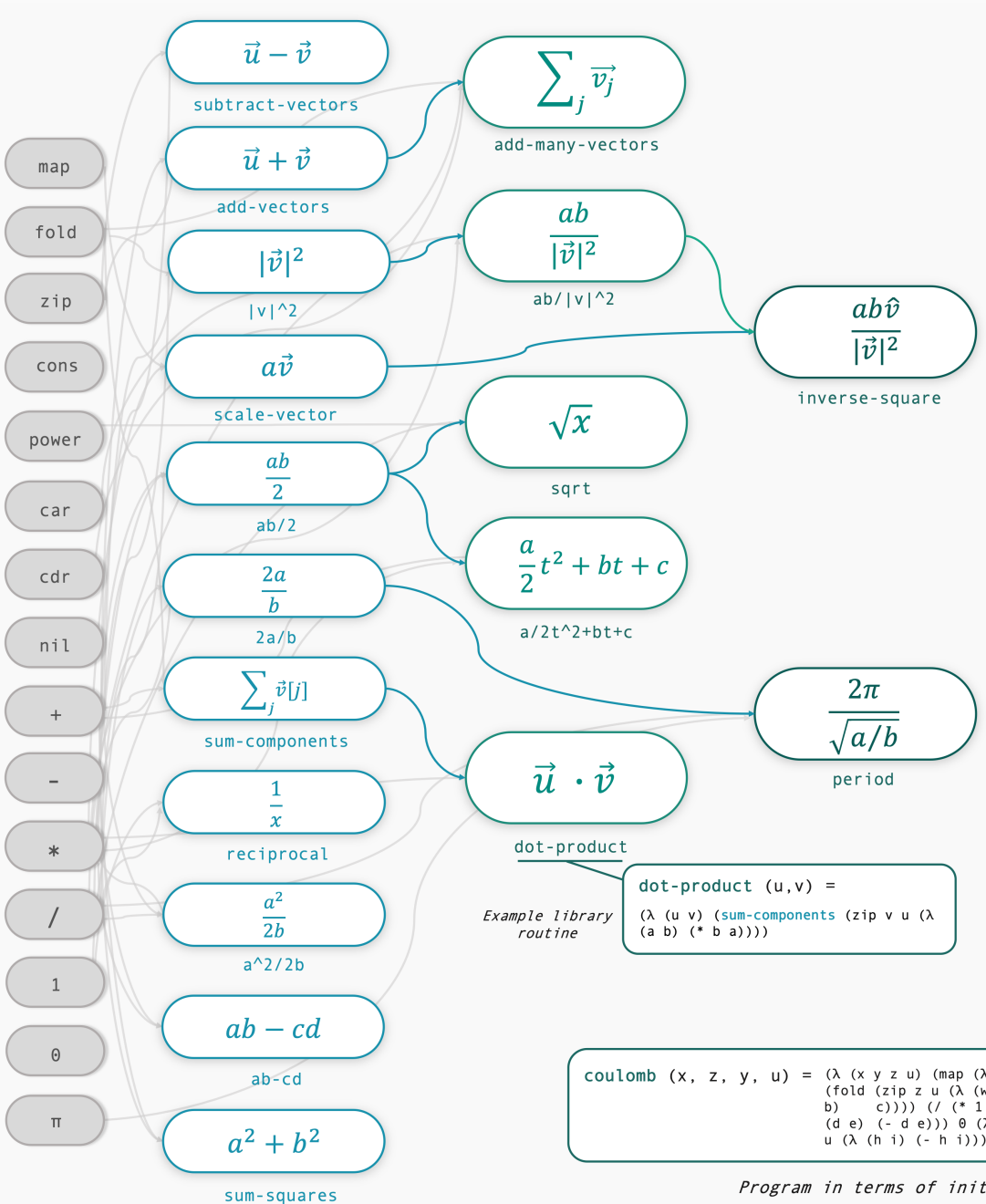
these experiments study how DreamCoder grows from a "beginner" state to "expert":

- "beginner:" basic domain-specific procedures, only easiest problems have short solutions
- "expert:" learned library allowing hardest problems to have short meaningful solutions

go beyond: start with generic arithmetic & control flow, learn fundamentals of domain

- Physics from recursive higher-order functions
- Recursive higher-order language from 1959 proto-Lisp & Y-combinator

Rediscovering vector algebra



Electric Field to Charge Flux

$$\vec{E}=\rho\vec{J}$$

```
(scale-vector p J)
```

Newton's Second Law

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

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

Parallel Resistors

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

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

Kinetic Energy

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

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

Energy in a Capacitor

$$U=\frac{1}{2}CV^2$$

```
(a^2/2b  v (reciprocal c))
```

Ballistic Motion

$$x(t)=\frac{a}{2}t^2+v_0t+x_0$$

```
(a/2t^2+bt+c c b a t)
```

Force in a Magnetic Field

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

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

Work

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

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

Coulomb's Law

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

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

Center of Mass

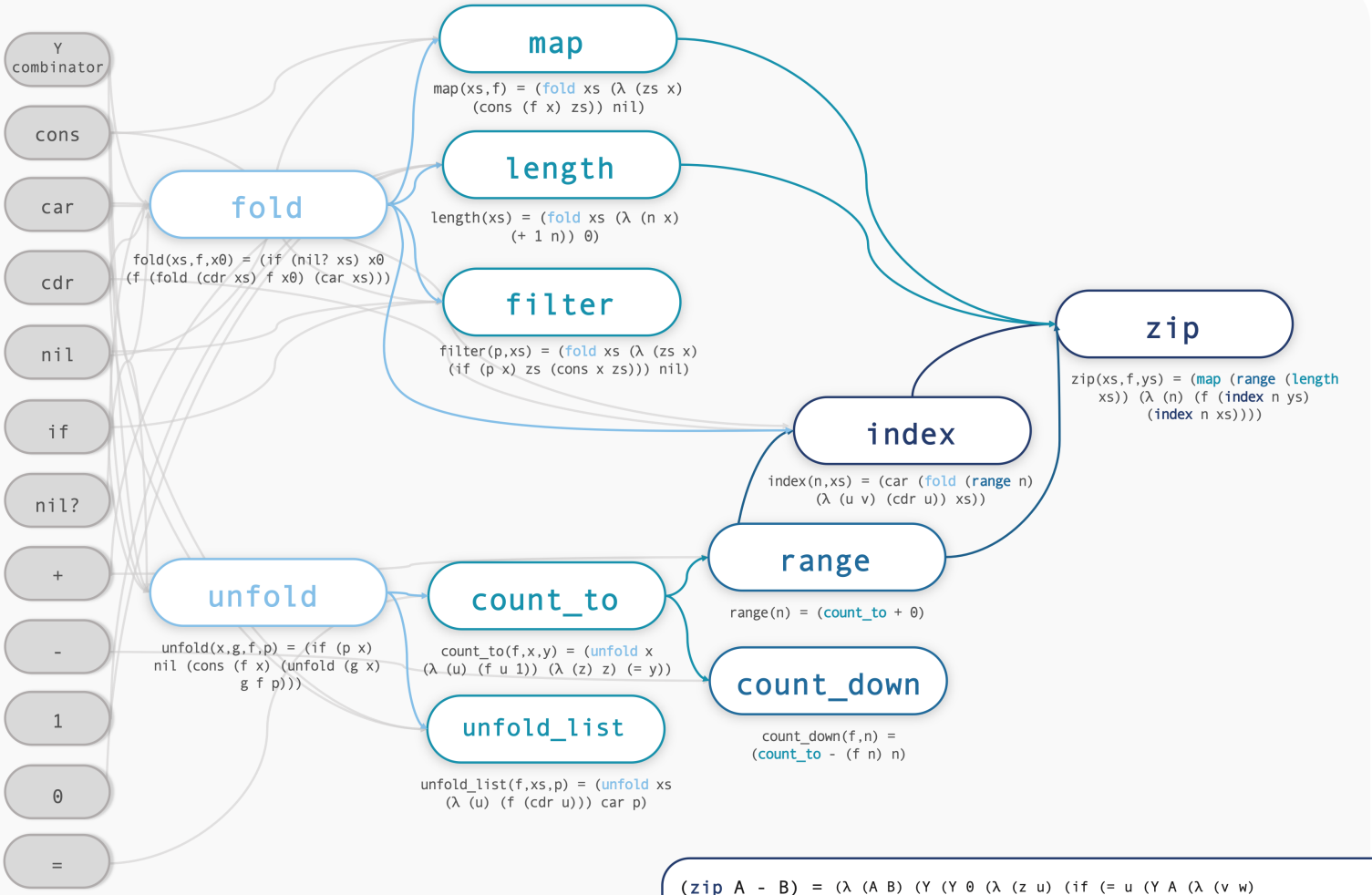
$$X_{CM}=\frac{\sum_i m_i x_i}{\sum_i m_i}$$

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

```
coulomb (x, z, y, u) = (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)))))
```

Program in terms of initial library

Rediscovering origami programming



Stutter

[■■] → [■■■■]
[■■■] → [■■■■■■]

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

Index List

0, [■■■■■] → ■
1, [■■■■■] → ■

```
(index n A)
```

Take Every Other

[■■■■] → [■■]
[■■■■■■] → [■■■]

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

List Lengths

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

```
(map A length)
```

Differences

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

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

```
(zip A - B) = (λ (A B) (Y (Y 0 (λ (z u) (if (= u (Y A (λ (v w)
(if (nil? w) 0 (+ 1 (v (cdr w)))))))) nil (cons u (z (+ u 1))))))
(λ (a b) (if (nil? b) nil (cons (- (car (Y (Y 0 (λ (c d) (if (= d
(car b)) nil (cons d (c (+ d 1)))))))) (λ (e f) (if (nil? f) A (cdr
(e (cdr f)))))))) (car (Y (Y 0 (λ (g h) (if (= h (car b)) nil (cons
h (g (+ h 1)))))))) (λ (i j) (if (nil? j) B (cdr (i (cdr j))))))))
(a (cdr b)))))))
```

Program in terms of initial library

From learning libraries to using interpreters

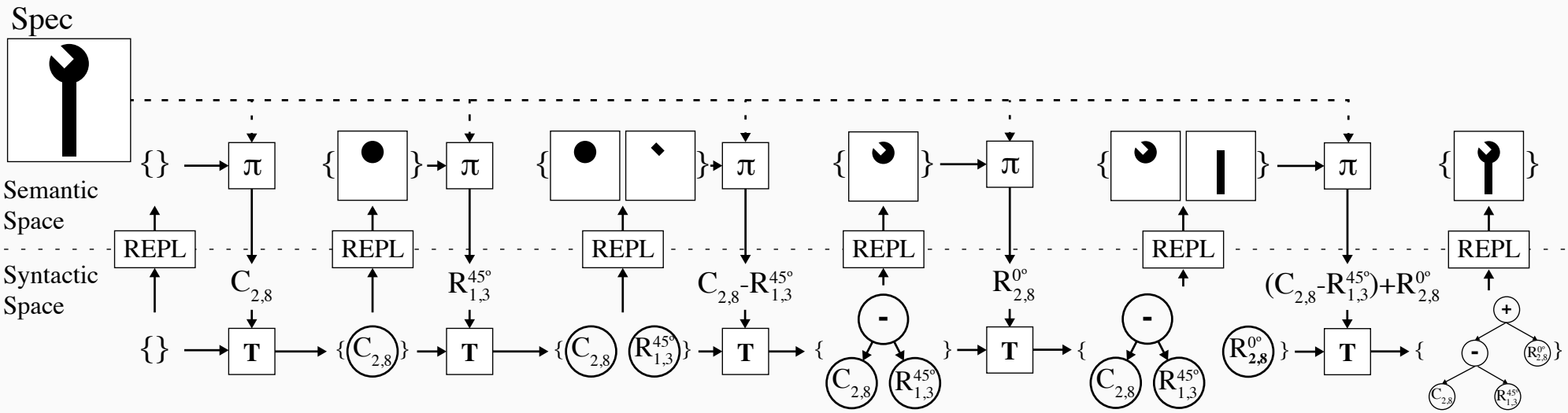
DreamCoder: library building as inspiration

Software engineers: build libraries, use interpreters, version control, debuggers, ...

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

*equal contribution

Synthesis with a REPL



REPL: Bridges syntax and semantics

π: policy, writes code conditioned on REPL state

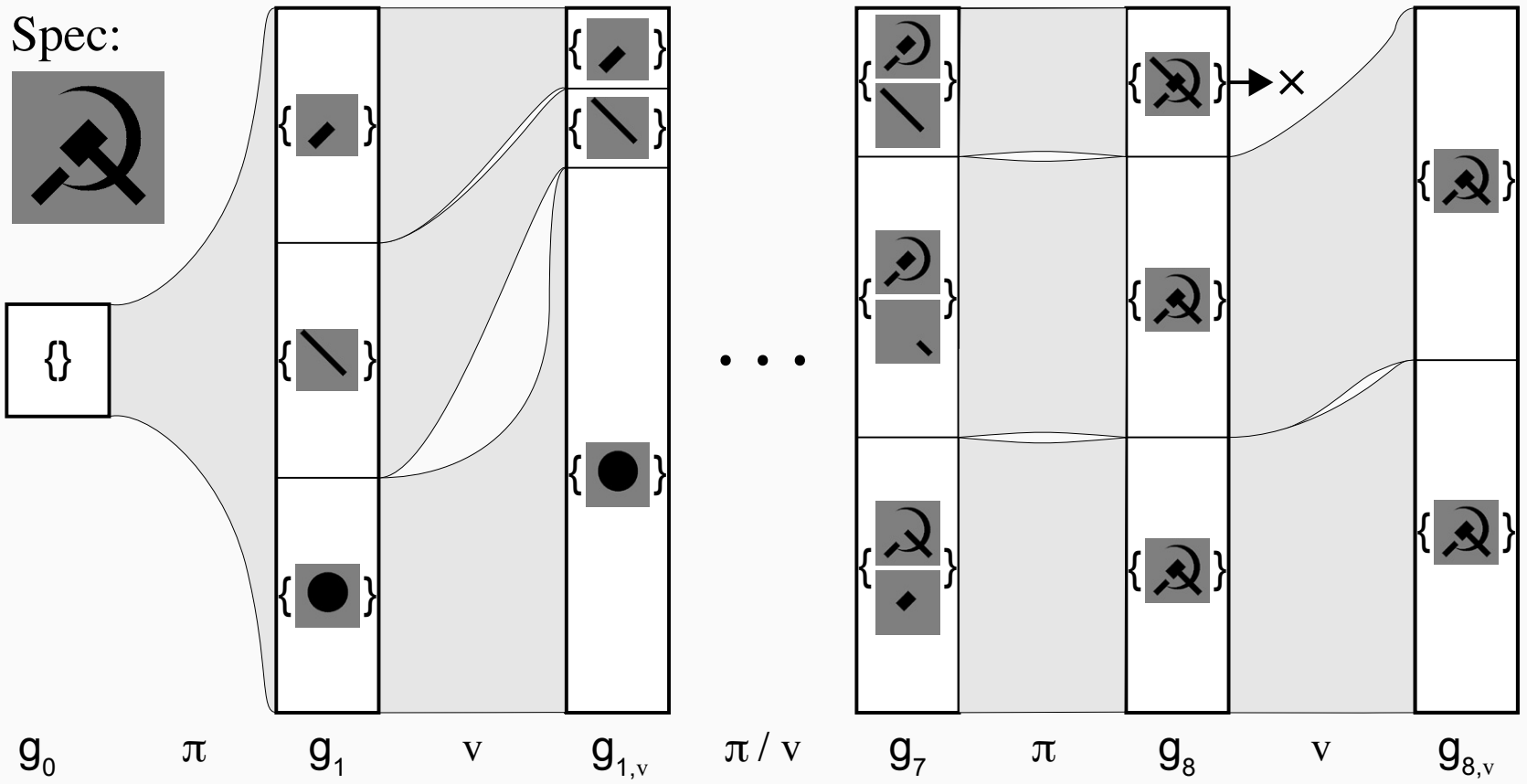
T: Markov Decision Process transition function

Spec: Image to draw

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

Interleaving policy+value+REPL in stochastic tree search

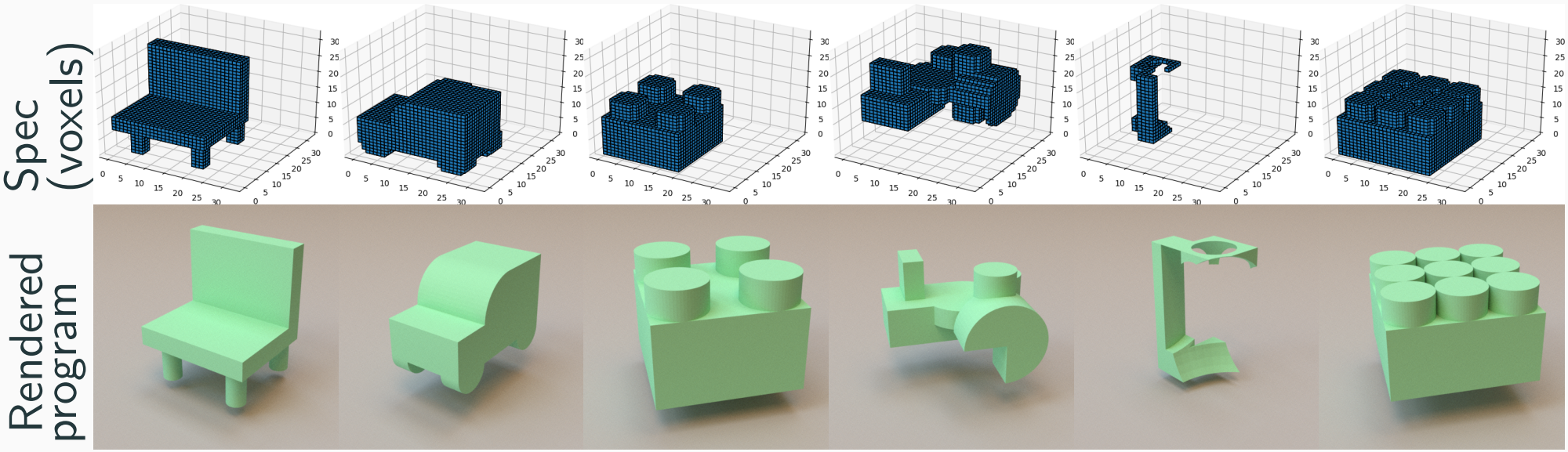
c.f. AlphaGo, TD-Gammon



Scaling to long programs

Branching factor: > 1.3 million per line of code

Successfully synthesizes >20-line programs in seconds (>100 tokens)



Scaling to long programs

Branching factor: > 400 actions

Successfully synthesizes 40-action programs

Spec:	
6/12/2003	$\rightarrow$ date: 12 mo: 6 year: 2003
3/16/1997	$\rightarrow$ date: 16 mo: 3 year: 1997
Held out test instance:	
12/8/2019	$\rightarrow$ date: 8 mo: 12 year: 2019
Results:	
Ours	$\rightarrow$ **date: 8 mo: 12 year: 2019**
RobustFill	$\rightarrow$ date:12/8/2019