

GPS / Program Induction

December 16, 2019

Human program induction everywhere

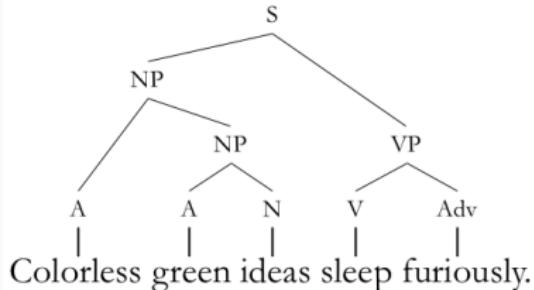
```
(MEMBER  
  (LAMBDA (X L)  
    (COND ((NULL L) NIL)  
          ((EQ X (FIRST L)) T)  
          (T (MEMBER X (REST L))))))
```

Allen, Anatomy of Lisp, 1975

Human program induction everywhere

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

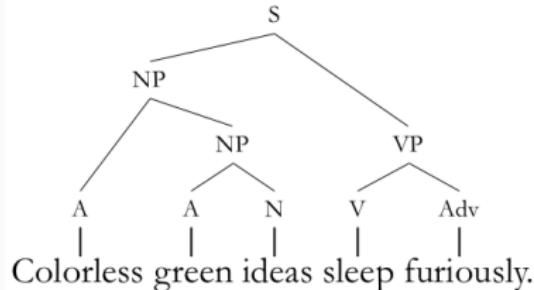
Allen, Anatomy of Lisp, 1975



Human program induction everywhere

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

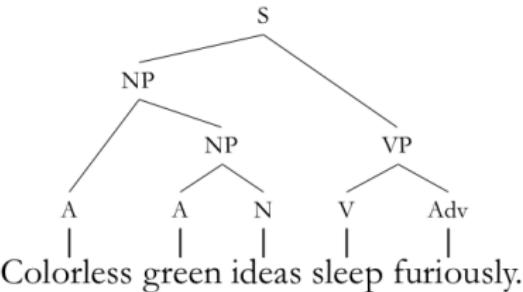
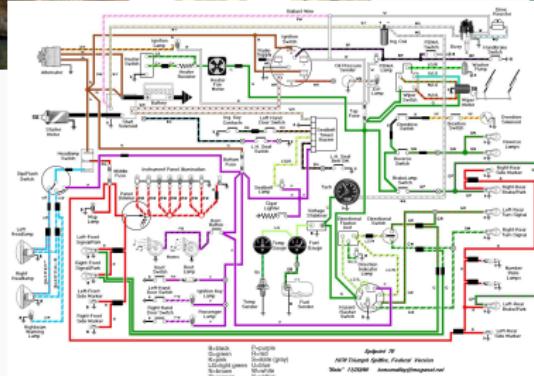
Allen, Anatomy of Lisp, 1975



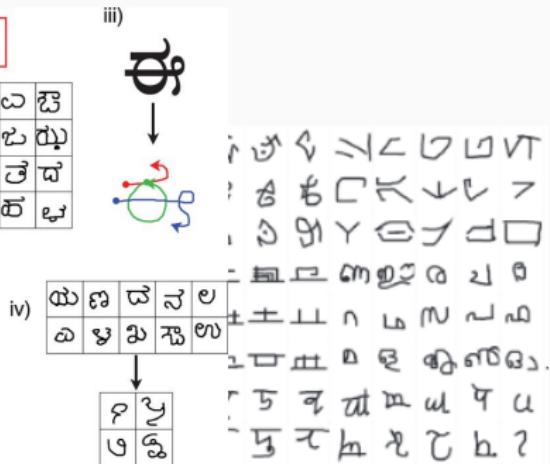
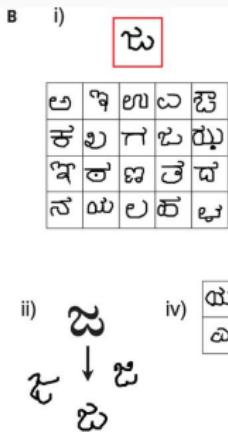
Human program induction everywhere

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

Allen, Anatomy of Lisp, 1975



Human program induction everywhere



ବିଚାରଣା ପାଇଁ ଏହା କିମ୍ବା କିମ୍ବା କିମ୍ବା କିମ୍ବା
 ଏହା କିମ୍ବା କିମ୍ବା କିମ୍ବା କିମ୍ବା କିମ୍ବା କିମ୍ବା

Exploration-Compression

Dechter et al. 2013

What if we don't have the right programming language for solving our problems?

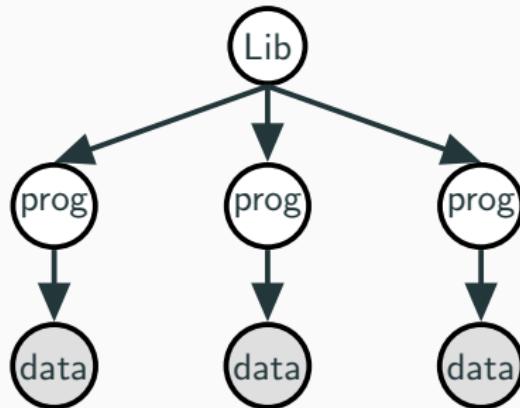
Exploration-Compression

Dechter et al. 2013

What if we don't have the right programming language for solving our problems?

Idea: Iteratively grow the language, alternating problem-solving and language-growing. Grow the language by “compressing out” reused program components, and cache those components in a learned library

Exploration-Compression as Bayesian inference

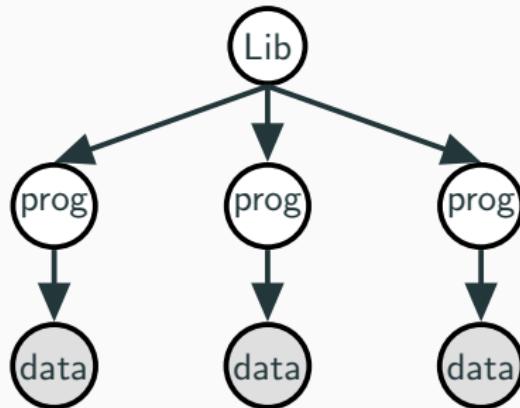


Explore: Enumerate programs built from library (Lib). Keep program if it solves a task (explains an observed datum)

Compress: Update library by incorporating reused subtrees of programs found during Explore. maximize:

$$\mathbb{P} [\text{Lib}] \prod_{\text{task}} \sum_{\substack{\text{prog solving task} \\ \text{prog found during Explore}}} \mathbb{P} [\text{prog} | \text{Lib}]$$

Exploration-Compression as Bayesian inference



Explore: Enumerate programs built from library (Lib). Keep program if it solves a task (explains an observed datum)

Compress: Update library by incorporating reused subtrees of programs found during Explore. *minimize:*

$$\underbrace{-\log \mathbb{P}[\text{Lib}]}_{\text{size of library}} + \sum_{\text{task}} -\log \underbrace{\max_{\substack{\text{prog solving task} \\ \text{prog found during Explore}}} \mathbb{P}[\text{prog}|\text{Lib}]}_{\text{size of program solving task wrt library}}$$

Compression in action

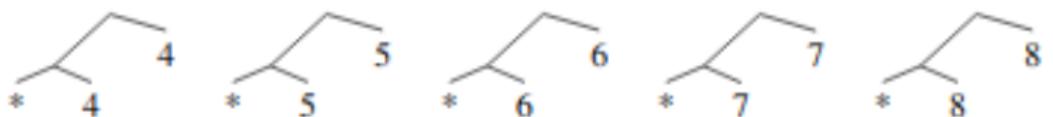
Suppose you need to build programs that calculate 16, 25, 36, 49, 64

Intuitively, here we should invent a new function called “square”

Compression in action

Suppose you need to build programs that calculate 16, 25, 36, 49, 64

Intuitively, here we should invent a new function called “square”



(16)

(25)

(36)

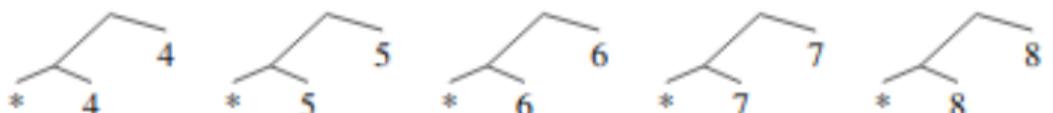
(49)

(64)

Compression in action

Suppose you need to build programs that calculate 16, 25, 36, 49, 64

Intuitively, here we should invent a new function called “square”



symbols w/o square (top): $3 \times 5 = 15$

symbols w/ square (bottom): $3 + 2 \times 5 = 12$

Bootstrap learning dynamics

At first, almost nothing is solvable. System bootstraps itself by solving easier problems first, reusing motifs discovered in their solutions to progress toward harder problems

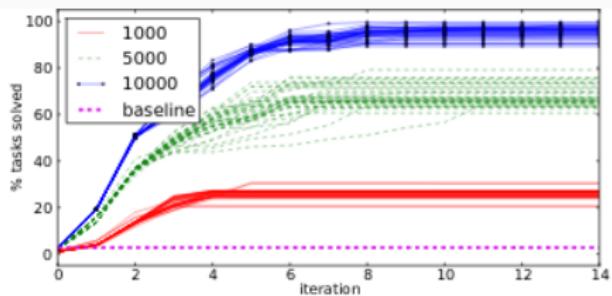


Figure 2: Learning curves as a function of frontier size. As frontier size is increased, curves plateau closer to 100% performance. A baseline search over 150000 expressions only hits 3% of the tasks (dashed pink line).

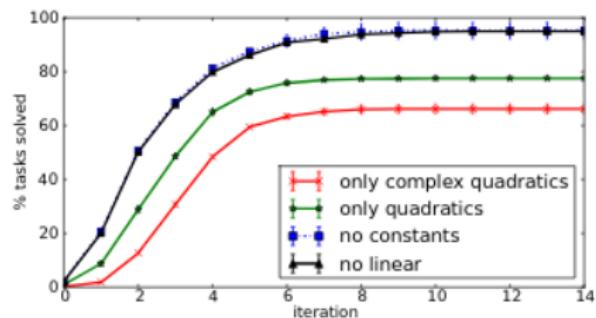


Figure 3: How do different task sets affect learning curves? Learning curves at a frontier size of 10000 for different task sets.

Hinges on:

- sufficient search effort (frontier size, left)
- a graded mixture of difficulty (right)

Learning interpretable concepts in the library

func.	CL expression	schematic
NOT	(S NAND I)	
AND	(C B NAND) (B (S NAND I)) \rightarrow (C B NAND) (B NOT)	
OR	((B (C (B (S NAND NAND)))) (S NAND I) \rightarrow ((B (C (B (S NAND NAND)))) NOT)	
E_1	S B (S NAND)	
E_2	(B (C B NAND) S)	
	(S NAND) (S NAND I)	

Growing domain-specific knowledge

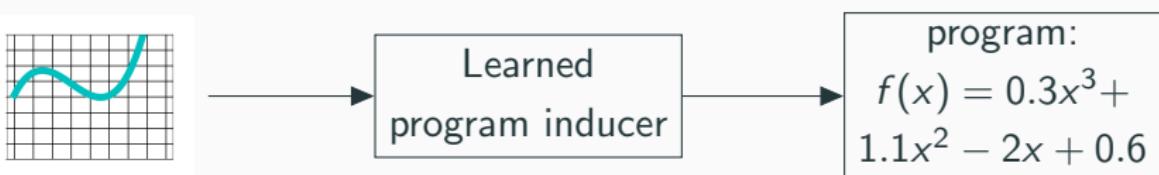
Goal: acquire domain-specific knowledge needed to induce a class of programs

- 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)



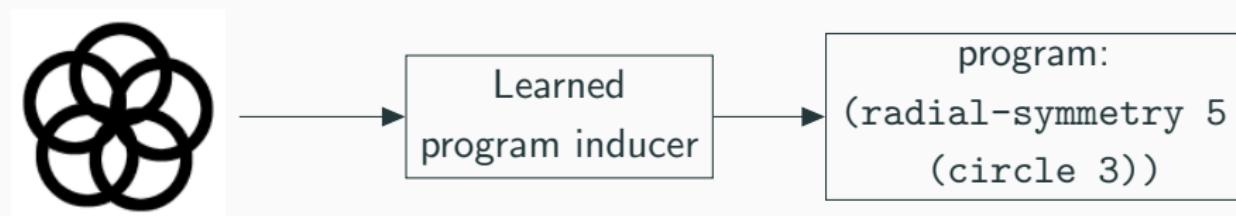
Concepts: $x^3, \alpha x + \beta$, etc

Inference strategy: neurosymbolic search for programs

Growing domain-specific knowledge

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

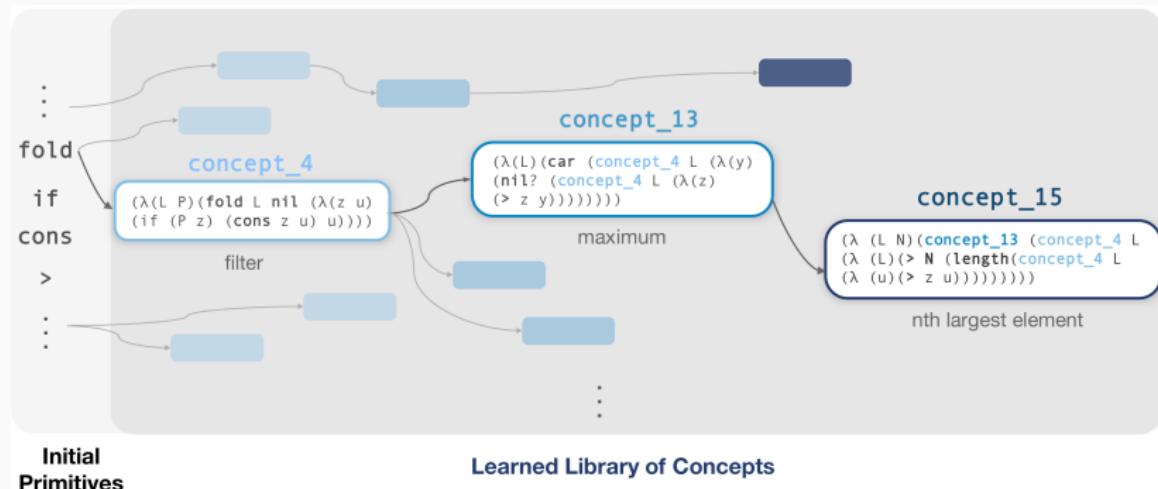
- Library of concepts (declarative knowledge; domain specific language; generative model over programs)
- Inference strategy (procedural knowledge; synthesis algorithm)



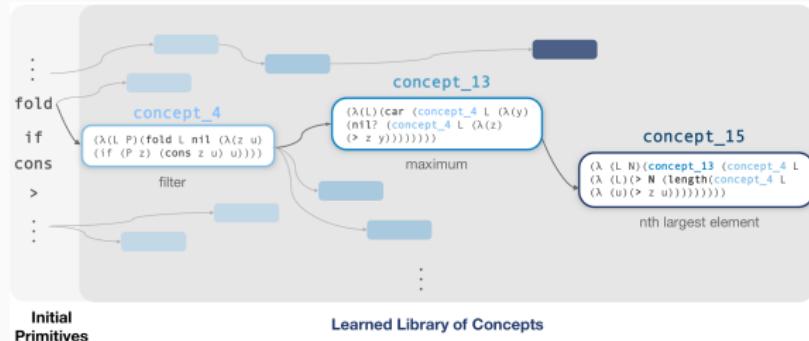
Concepts: circle, radial-symmetry, etc

Inference strategy: neurosymbolic search for programs

Library learning



Library learning



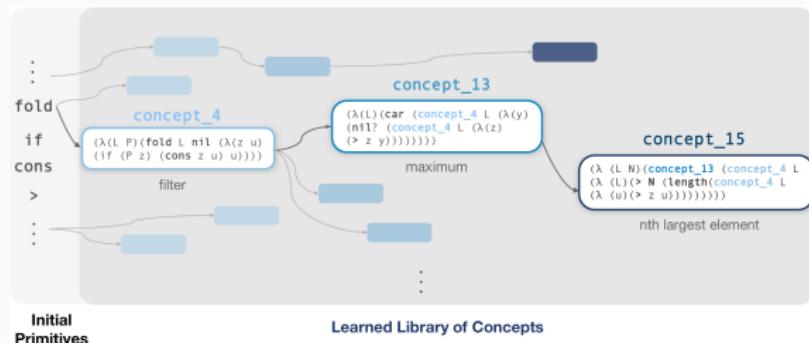
Problem: sort list

Solution:

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

get nth largest element where $n = 1, 2, 3 \dots \text{length of list}$

Library learning



Problem: sort list

Solution:

$(map\ (\lambda(n)\ (concept_15\ L\ (+\ 1\ n)))\ (range\ (length\ L))))$
get nth largest element where n = 1, 2, 3 ... length of list

Solution in initial primitives:

```
(λ(x) (map (λ(y) (car (fold x nil (λ(z u) (if (gt? (+ y 1) (length (fold x nil (λ(v w) (if (gt? z v) (cons v 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
 $[1 \ 2 \ 3] \rightarrow 6$
 $[4 \ 6 \ 8 \ 1] \rightarrow 17$

Double

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

Check Evens

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

Text Editing

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

Drop Last Characters

jabberwocky → jabberw
copycat → cop

Extract

see spot(run) → run
a (bee) see → bee

Regexes

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

Currency

\$100.25
\$4.50

Dates

Y1775/0704
Y2000/0101

LOGO Graphics



Physics

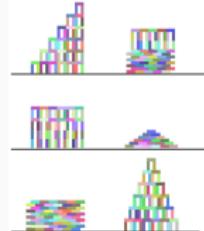
$$KE = \frac{1}{2}m|\vec{v}|^2$$

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

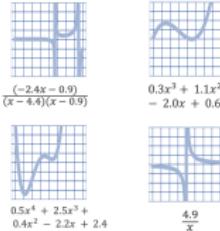
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \hat{r}_1 - \hat{r}_2$$

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

Block Towers



Symbolic Regression



Recursive Programming

Filter

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

Length

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

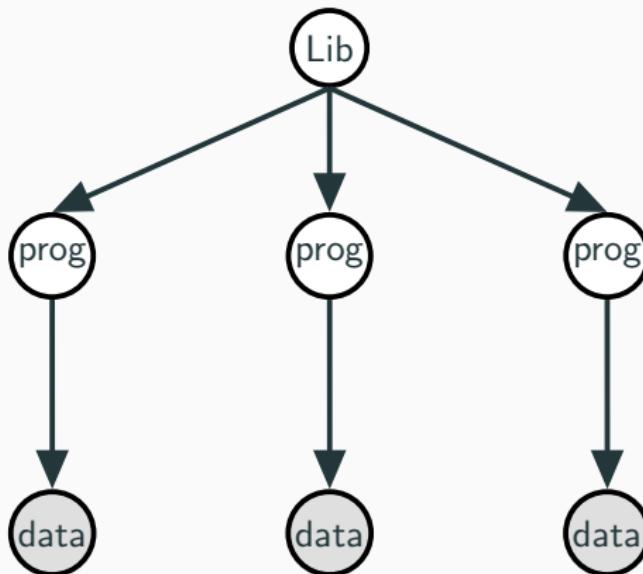
Index List

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

Every Other

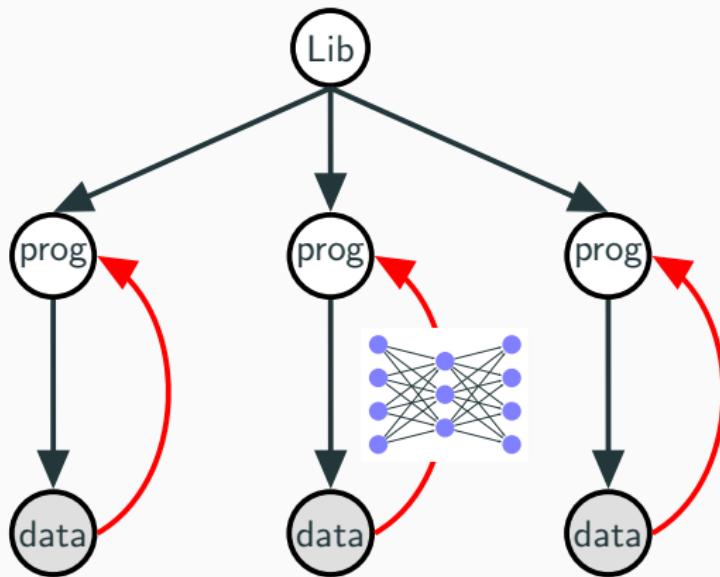
$[■■■■■■■■■] \rightarrow [■■■]$
 $[■■■■■■■■■] \rightarrow [■■■■]$
 $[■■■■■■■■■] \rightarrow [■■■■■]$

Library learning as Bayesian inference

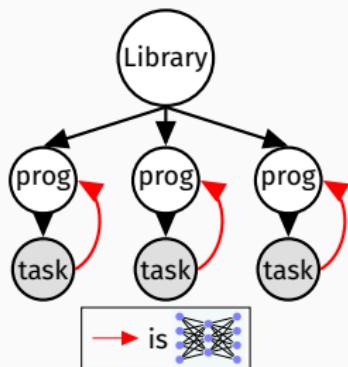


[Dechter et al., 2013] [Liang et al, 2010]; [Lake et al, 2015]

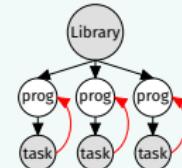
Library learning as amortized Bayesian inference



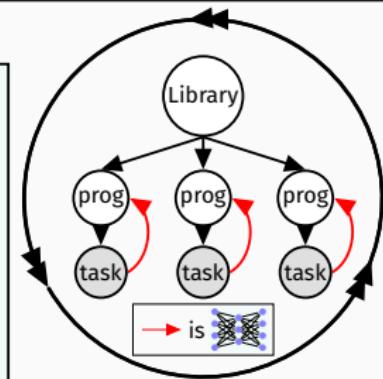
amortized inference +
better program representation (Lisp) +
library learning via program analysis +
new neural inference network for program synthesis



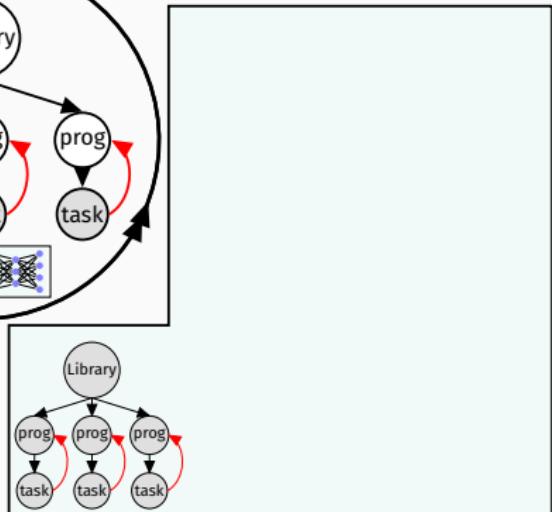
WAKE



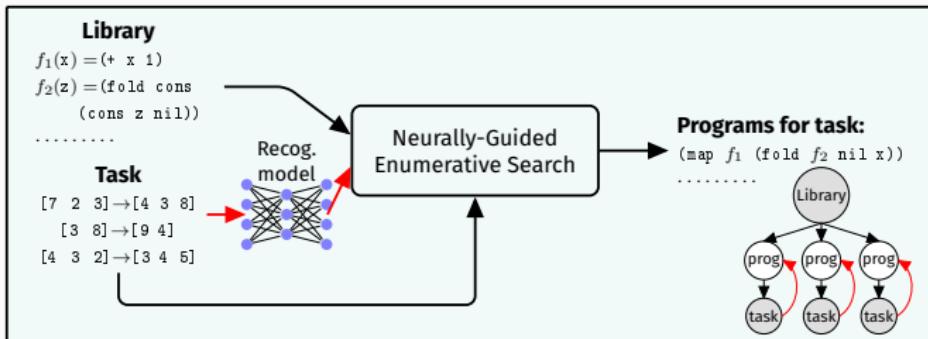
SLEEP: ABSTRACTION



SLEEP: DREAMING



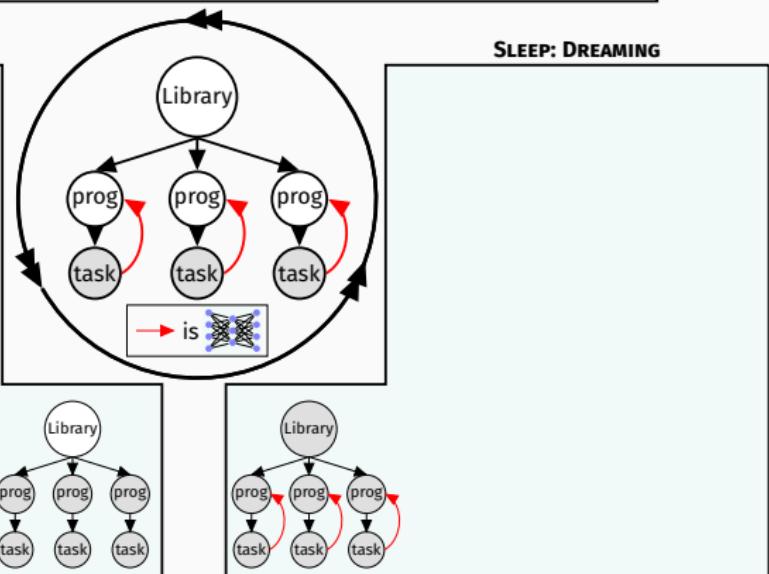
WAKE



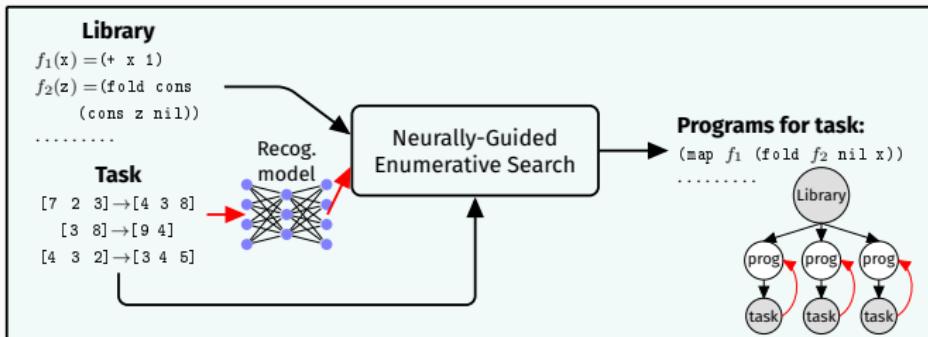
SLEEP: ABSTRACTION



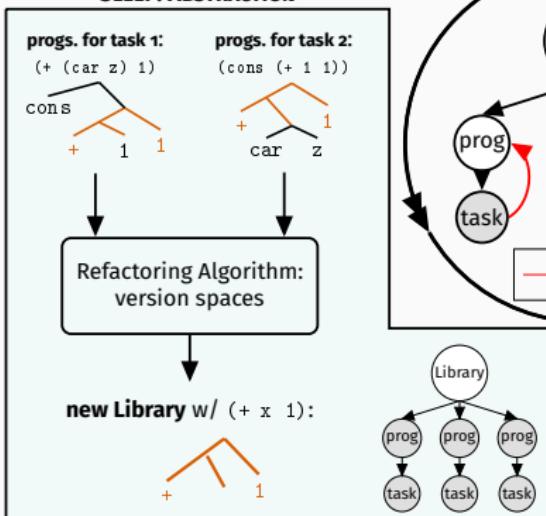
SLEEP: DREAMING



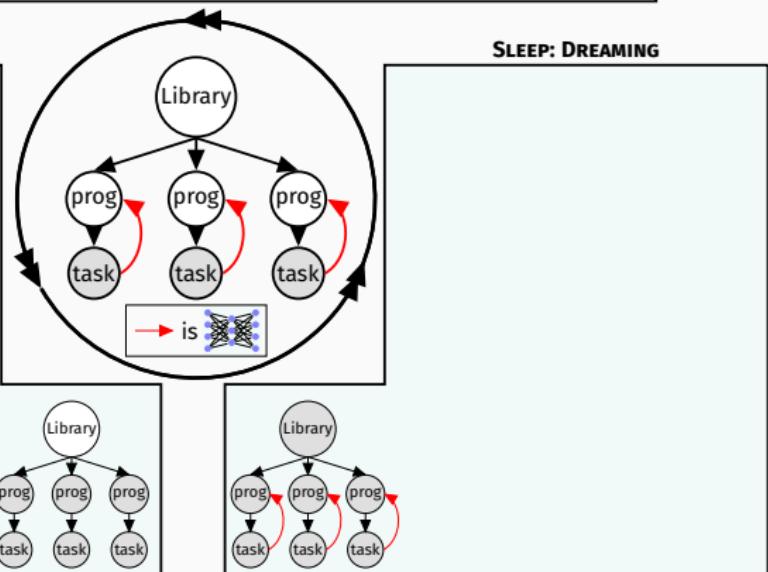
WAKE



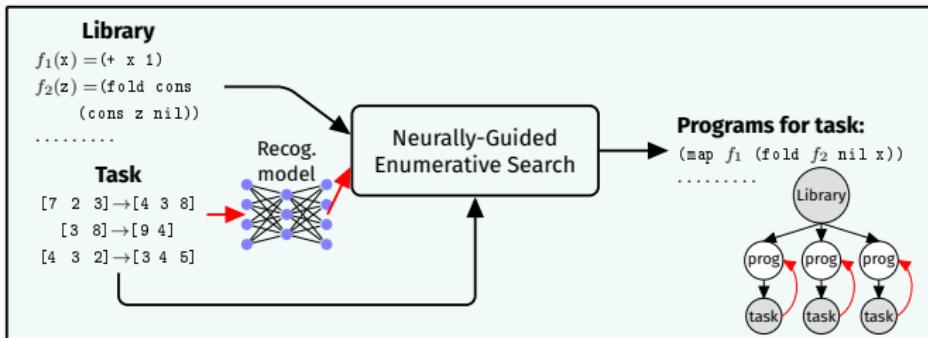
SLEEP: ABSTRACTION



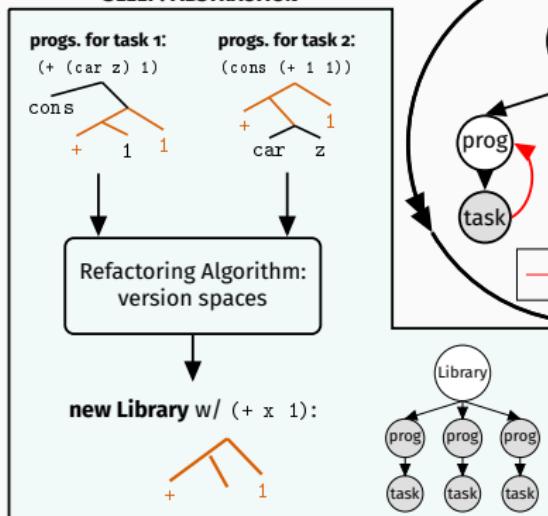
SLEEP: DREAMING



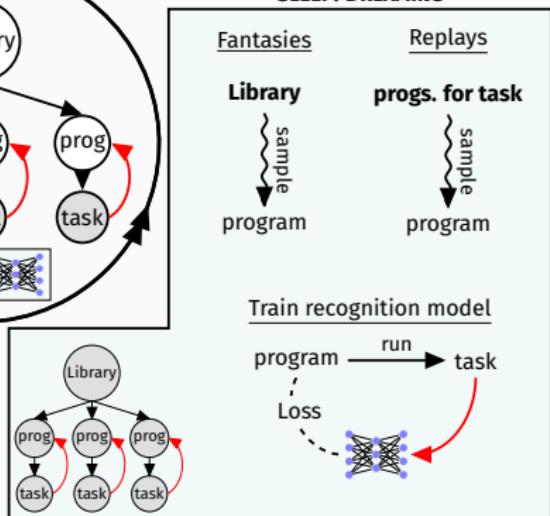
WAKE



SLEEP: ABSTRACTION



SLEEP: DREAMING



Abstraction Sleep: Growing the library via refactoring

Task: $(1\ 2\ 3) \rightarrow (2\ 4\ 6)$
 $(4\ 3\ 2) \rightarrow (8\ 6\ 8)$

Wake: program search

```
(Y (\lambda (r l) (if (nil? l) nil  
                      (cons (+ (car l) (car l))  
                            (r (cdr l)))))))
```

Task: $(1\ 2\ 3) \rightarrow (0\ 1\ 2)$
 $(4\ 3\ 2) \rightarrow (3\ 2\ 3)$

Wake: program search

```
(Y (\lambda (r l) (if (nil? l) nil  
                      (cons (- (car l) 1)  
                            (r (cdr l)))))))
```

Abstraction Sleep: Growing the library via refactoring

Task: $(1\ 2\ 3) \rightarrow (2\ 4\ 6)$
 $(4\ 3\ 2) \rightarrow (8\ 6\ 8)$

Wake: program search

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

Task: $(1\ 2\ 3) \rightarrow (0\ 1\ 2)$
 $(4\ 3\ 2) \rightarrow (3\ 2\ 3)$

Wake: program search

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

refactor

$(10^{14}$ refactorings)

Sleep: Abstraction

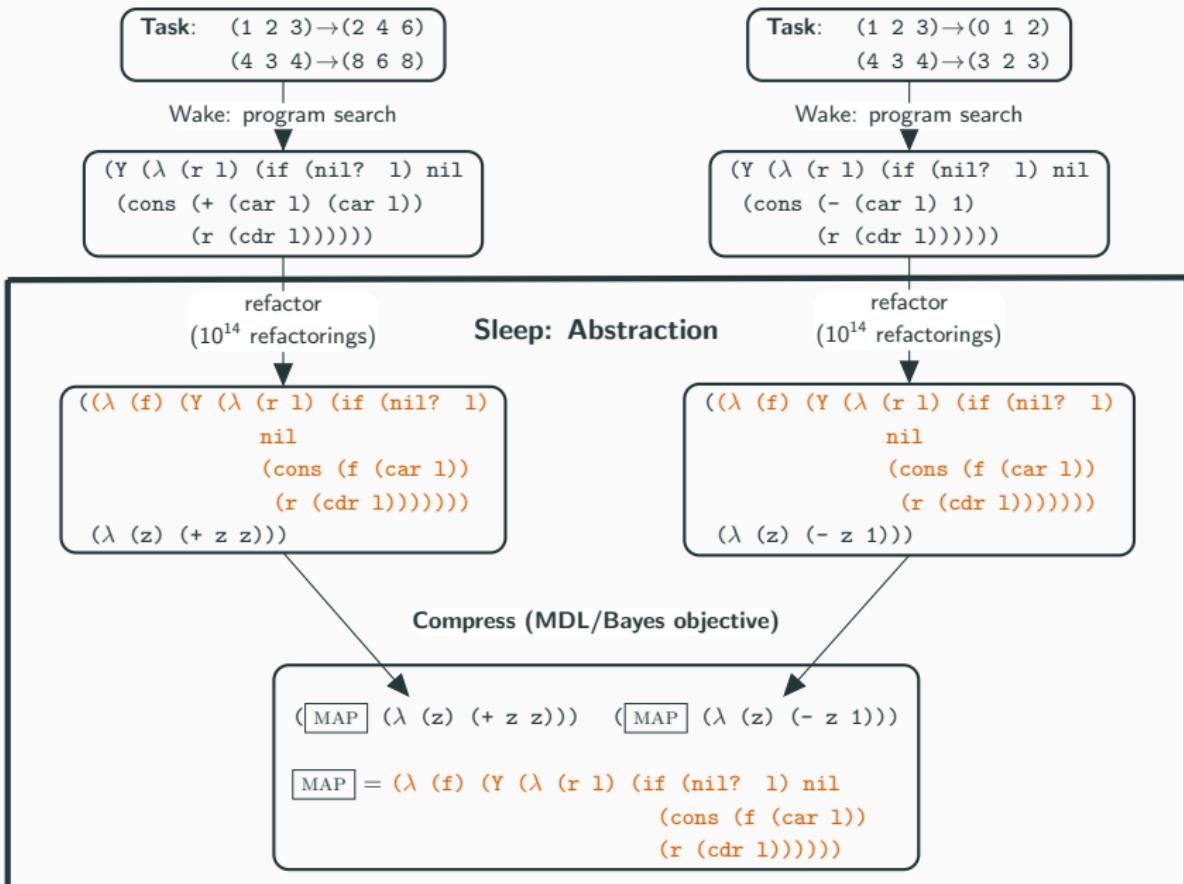
```
((λ (f) (Y (λ (r 1) (if (nil? 1)  
                           nil  
                           (cons (f (car 1))  
                                 (r (cdr 1)))))))  
  (λ (z) (+ z z)))
```

refactor

$(10^{14}$ refactorings)

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

Abstraction Sleep: Growing the library via refactoring



Version space algebra for refactoring

$\text{Expr} \rightarrow \text{var}$

- | $\lambda\text{var}.\text{Expr}$
- | $(\text{Expr } \text{Expr})$
- | primitive

Version space algebra for refactoring

$\text{Expr} \rightarrow \text{var}$

- | $\lambda\text{var}.\text{Expr}$
- | $(\text{Expr } \text{Expr})$
- | primitive

$\text{VS} \rightarrow \text{var}$

- | $\lambda\text{var}.\text{VS}$
- | $(\text{VS } \text{VS})$
- | primitive
- | $\text{VS} \uplus \text{VS}$ *nondeterministic choice*
- | Λ *choose any expression*
- | \emptyset *choose no expression*

Version space algebra for refactoring

$\text{Expr} \rightarrow \text{var}$

| $\lambda \text{var}. \text{Expr}$
| ($\text{Expr} \ \text{Expr}$)
| primitive

$\text{VS} \rightarrow \text{var}$

| $\lambda \text{var}. \text{VS}$
| ($\text{VS} \ \text{VS}$)
| primitive
| $\text{VS} \uplus \text{VS}$ *nondeterministic choice*
| Λ *choose any expression*
| \emptyset *choose no expression*

what version spaces mean:

$$[\![\text{var}]\!] = \{\text{var}\}$$

$$[\![v_1 \uplus v_2]\!] = \{e : v \in \{v_1, v_2\}, e \in [\![v]\!]\}$$

$$[\![\lambda x. v]\!] = \{\lambda x. e : e \in [\![v]\!]\}$$

$$[\![(v_1 v_2)]\!] = \{(e_1 e_2) : e_1 \in [\![v_1]\!], e_2 \in [\![v_2]\!]\}$$

$$[\![\emptyset]\!] = \emptyset$$

$$[\![\Lambda]\!] = \Lambda$$

Using version spaces

$VS \rightarrow var \mid \lambda var.VS \mid (VS\ VS) \mid primitive$

| $VS \cup VS$ *nondeterministic choice*

| Λ *choose any expression*

| \emptyset *choose no expression*

Using version spaces

$$\begin{aligned} \text{VS} \rightarrow & \text{ var } | \lambda \text{var}. \text{VS} | (\text{VS } \text{VS}) | \text{ primitive} \\ & | \text{VS} \sqcup \text{VS} \quad \textit{nondeterministic choice} \\ & | \Lambda \quad \textit{choose any expression} \\ & | \emptyset \quad \textit{choose no expression} \end{aligned}$$

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

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

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

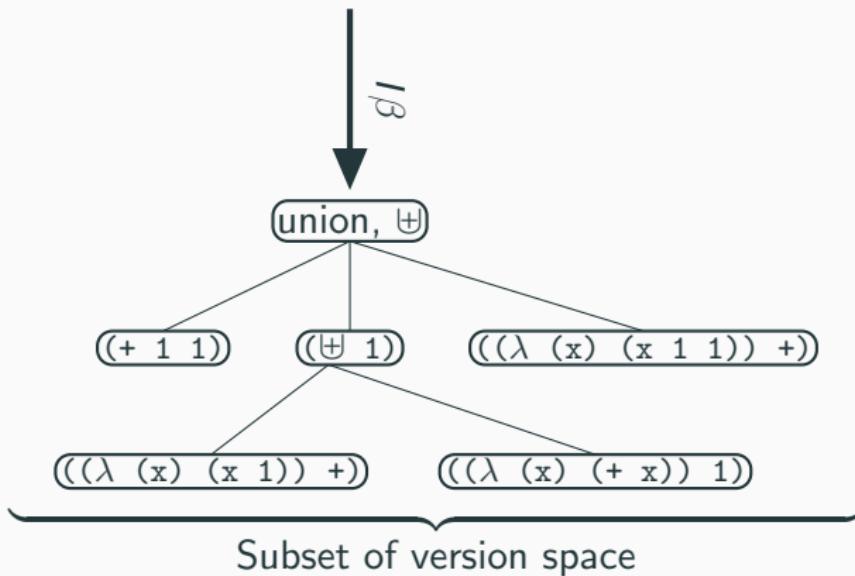
$$\text{REFACTOR}'(\lambda x. b|\text{Lib}) = \lambda x. \text{REFACTOR}(b|\text{Lib})$$

$$\text{REFACTOR}'(v_1 v_2|\text{Lib}) = (\text{REFACTOR}(v_1|\text{Lib}) \text{ REFACTOR}(v_2|\text{Lib}))$$

Using version spaces

$VS \rightarrow var \mid \lambda var.VS \mid (VS \vee VS) \mid primitive$
| $VS \oplus VS$ *nondeterministic choice*
| Λ *choose any expression*
| \emptyset *choose no expression*

Program: $(+ 1 1)$



Using version spaces

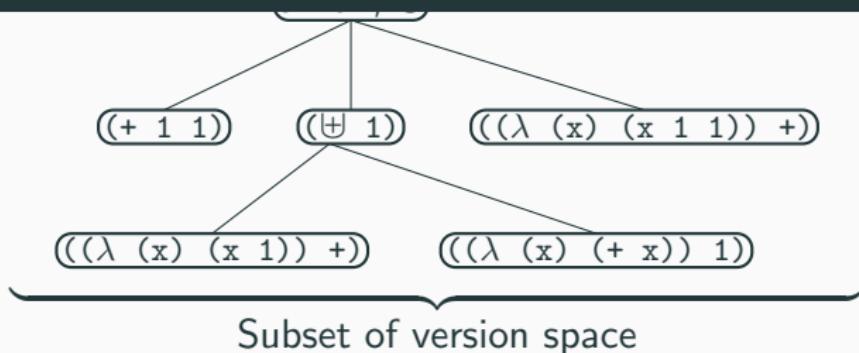
$VS \rightarrow var \mid \lambda var.VS \mid (VS \: VS) \mid primitive$
| $VS \sqcup VS$ *nondeterministic choice*
| Λ *choose any expression*
| \emptyset *choose no expression*

completeness: $I\beta$ gets all the refactorings

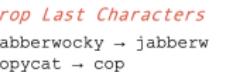
let $v_2 = I\beta(v_1)$ and $e_1 \in \llbracket v_1 \rrbracket$. for any $e_2 \rightarrow e_1$ then $e_2 \in \llbracket v_2 \rrbracket$

consistency: $I\beta$ only gets valid refactorings

let $v_2 = I\beta(v_1)$ and $e_2 \in \llbracket v_2 \rrbracket$. then there is a $e_1 \in \llbracket v_1 \rrbracket$ where $e_2 \rightarrow e_1$



DreamCoder Domains

List Processing	Text Editing	Regexes	LOGO Graphics
Sum List [1 2 3] → 6 [4 6 8 1] → 17	Abbreviate Allen Newell → A.N. Herb Simon → H.S.	Phone Numbers (555) 867-5309 (650) 555-2368	↶ ↷ ↸ ↹
Double [1 2 3 4] → [2 4 6 8] [6 5 1] → [12 10 2]	Drop Last Characters jabberwocky → jabberw copycat → cop	Currency \$100.25 \$4.50	✿ ⋯ ⋮ ⋆
Check Evens [0 2 3] → [T T F] [2 4 9 6] → [T T F T]	Extract see spot(run) → run a (bee) see → bee	Dates Y1775/0704 Y2000/0101	⊗ ⊖ ⊙ ⊚
Block Towers 	Symbolic Regression  $(-2.4x - 0.9)$ $(x - 4.4)(x - 0.9)$ $0.3x^3 + 1.1x^2 - 2.0x + 0.6$	Recursive Programming Filter [■■■■■] → [■■] [■■■■■■] → [■■■■] [■■■■■■■] → [■■■■■] Length [■■■■■] → 4 [■■■■■■] → 6 [■■■■] → 3	Physics $KE = \frac{1}{2} m \vec{v} ^2$ $\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$ $\vec{F} \propto \frac{q_1 q_2}{ \vec{r}'_1 - \vec{r}'_2 ^2} \hat{\vec{r}} = \hat{\vec{r}}$ $R_{total} = \left(\sum_i \frac{1}{R_i} \right)^{-1}$

DreamCoder Domains

List Processing

Sum List

[1 2 3] → 6

[4 6 8 1] → 17

Double

[1 2 3 4] → [2 4 6 8]

[6 5 1] → [12 10 2]

Check Evens

[0 2 3] → [T T F]

[2 4 9 6] → [T T F T]

Text Editing

Abbreviate

Allen Newell → A.N.

Herb Simon → H.S.

Drop Last Characters

jabberwocky → jabberw

copycat → cop

Extract

see spot(run) → run

a (bee) see → bee

Regexes

Phone Numbers

(555) 867-5309

(650) 555-2368

Currency

\$100.25

\$4.50

Dates

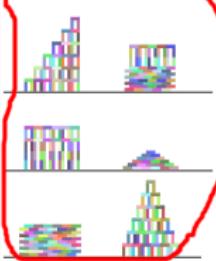
Y1775/0704

Y2000/0101

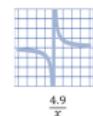
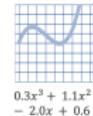
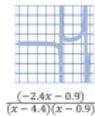
LOGO Graphics



Block Towers



Symbolic Regression



Recursive Programming

Filter

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

Length

[■■■■] → 4
[■■■■■] → 6
[■■■] → 3

Index List

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

Every Other

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

Physics

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

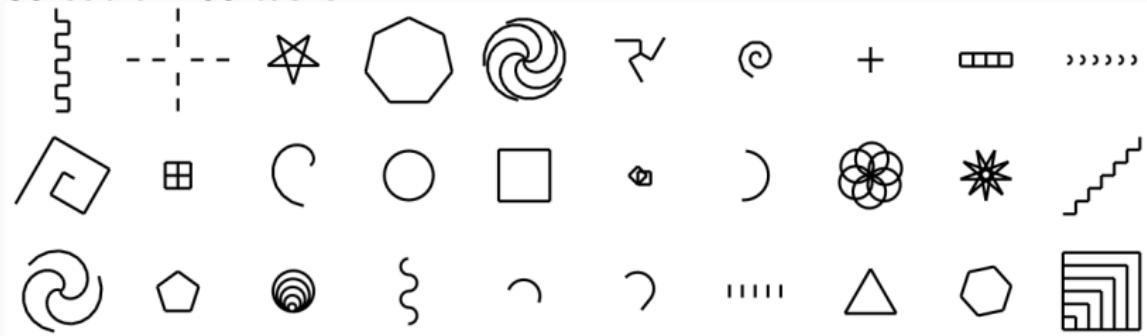
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

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

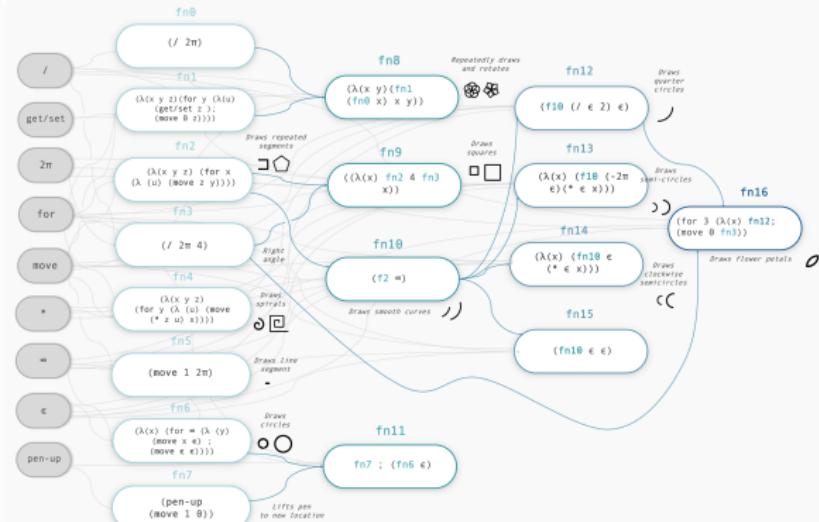
$$V_{total} = \left(\sum_i \frac{1}{R_i} \right)^{-1}$$

LOGO Graphics

30 out of 160 tasks



LOGO Graphics – learning interpretable library of concepts



$\{\text{fn8 } 5 \cdot (\text{fn4} \cdot (* \cdot 2) \cdot \epsilon)\}$

$\{\text{for } 2 \cdot (\lambda(x) (\text{fn14 } 2); (\text{fn13 } 2))\}$

$\{\text{for } 7 \cdot (\lambda(x) (\text{fn9 } x))\}$

$\{\text{fn10} \cdot (* \cdot (\text{fn0 } 7) \cdot 3) \cdot 3\}$

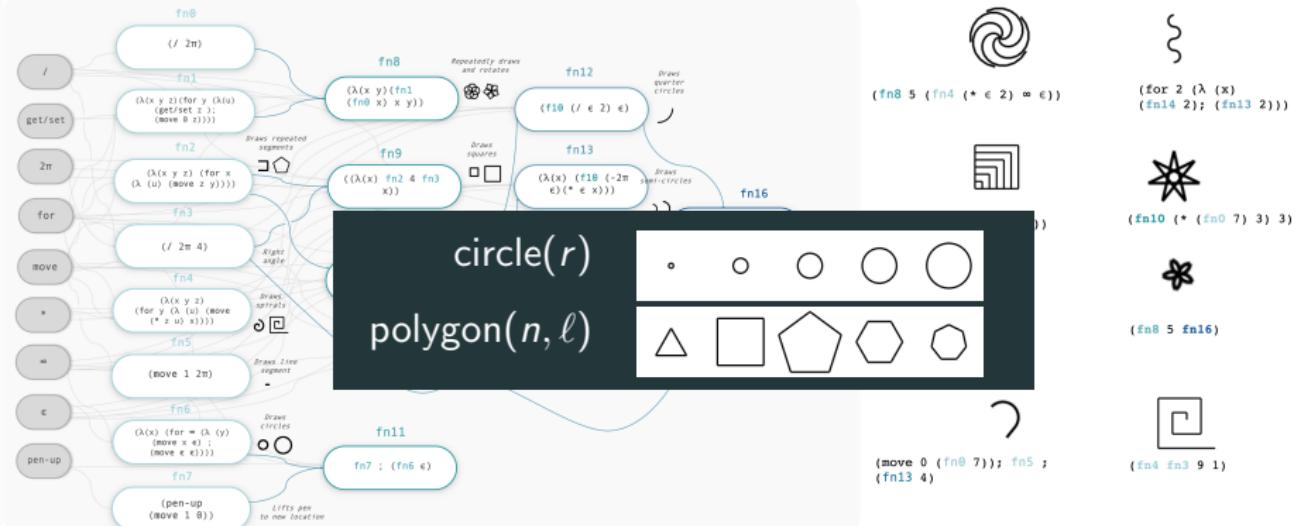
$\{\text{fn8 } 6 \cdot (\text{fn7} ; \text{fn5} ; \text{fn7} ; \text{fn5})\}$

$\{\text{fn8 } 5 \cdot \text{fn16}\}$

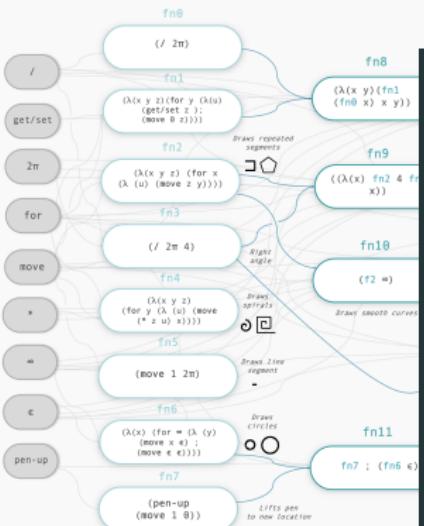
$\{\text{move 0 } (\text{fn0 } 7); \text{fn5} ; (\text{fn13 } 4)\}$

$\{\text{fn4 } \text{fn3 } 9 \cdot 9\}$

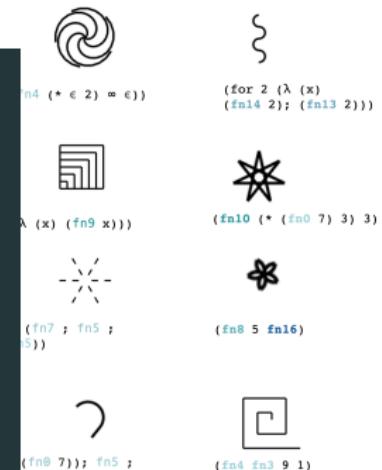
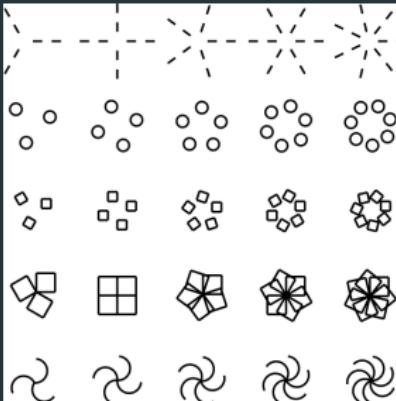
LOGO Graphics – learning interpretable library of concepts



LOGO Graphics – learning interpretable library of concepts



radial symmetry(*n*, body)

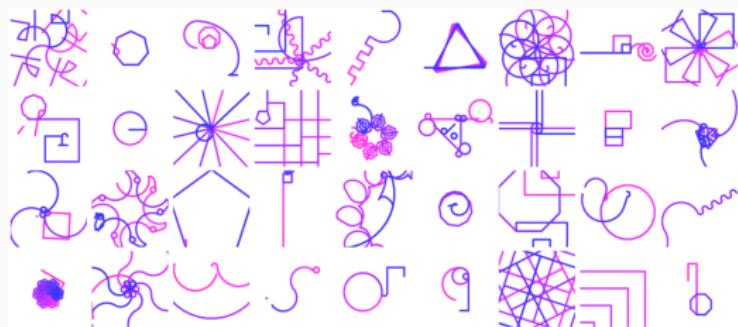


what does DreamCoder dream of?

before learning

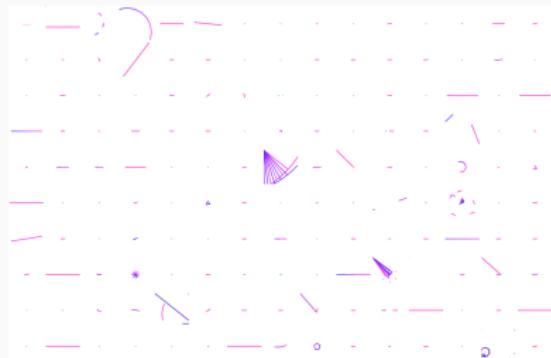


after learning

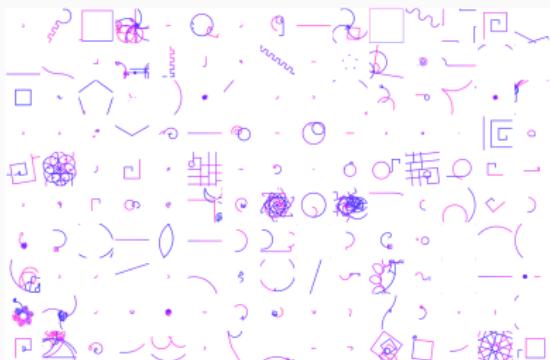


what does DreamCoder dream of?

before learning

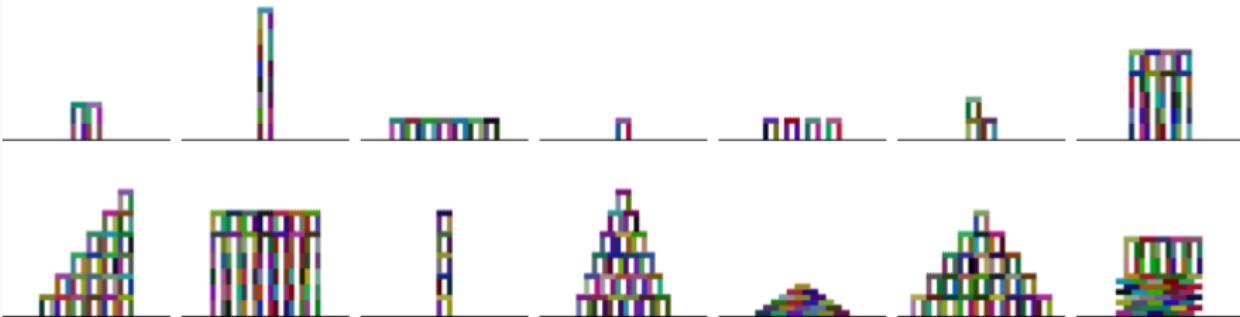


after learning



Planning to build towers

example tasks (112 total)

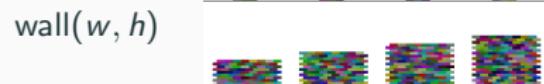
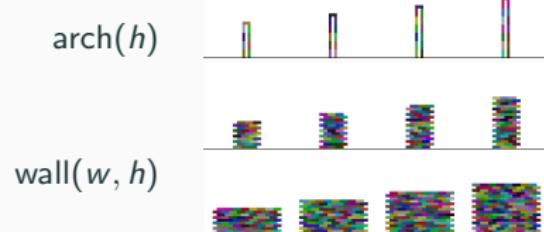


Planning to build towers

example tasks (112 total)

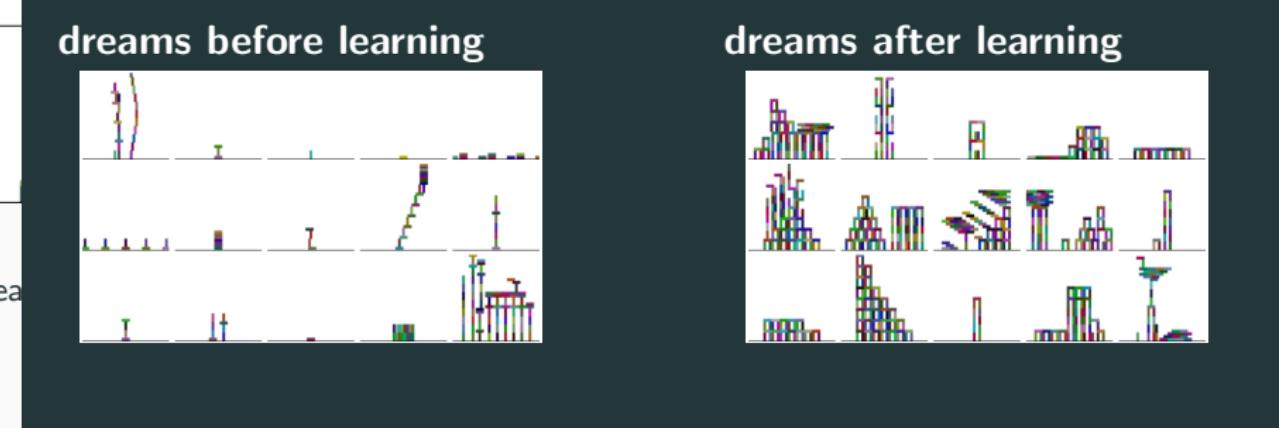


learned library routines (≈ 20 total)

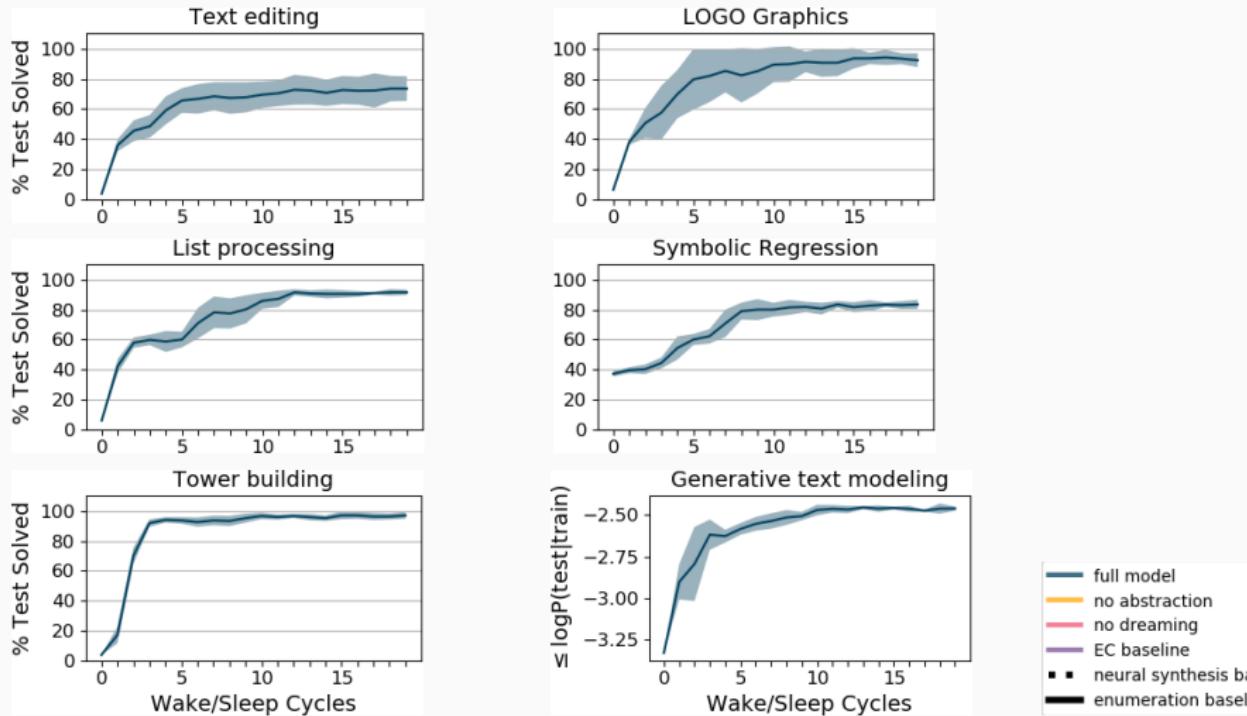


Planning to build towers

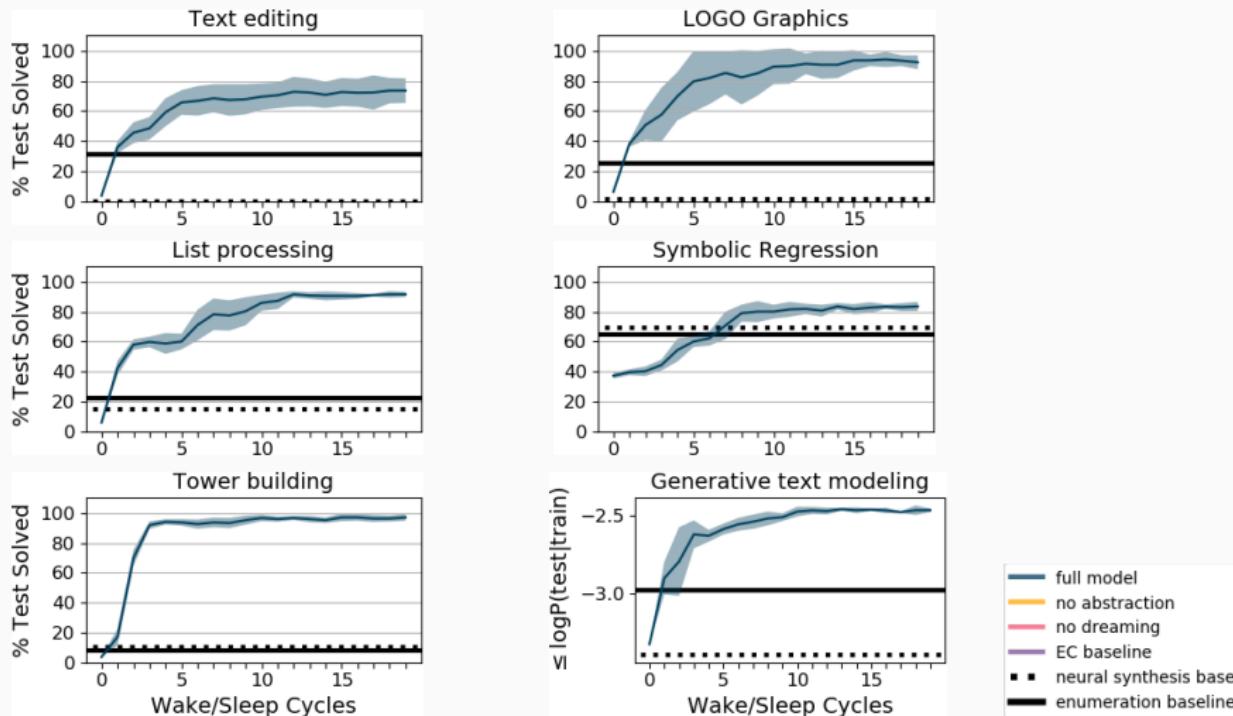
example tasks (112 total)



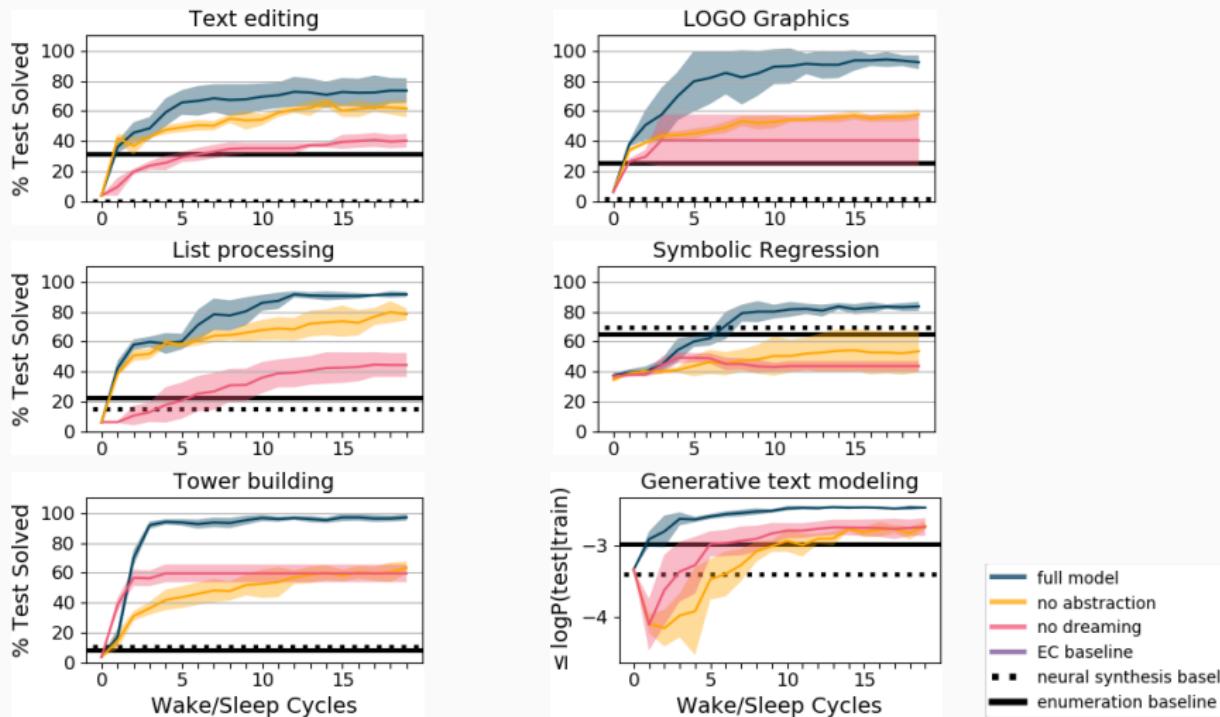
synergy between dreaming and library learning



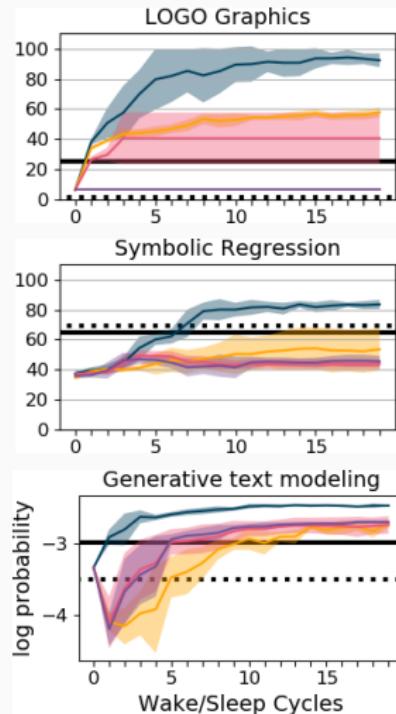
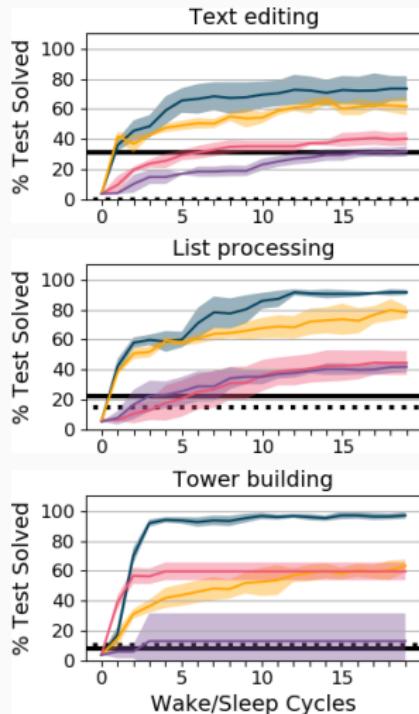
synergy between dreaming and library learning



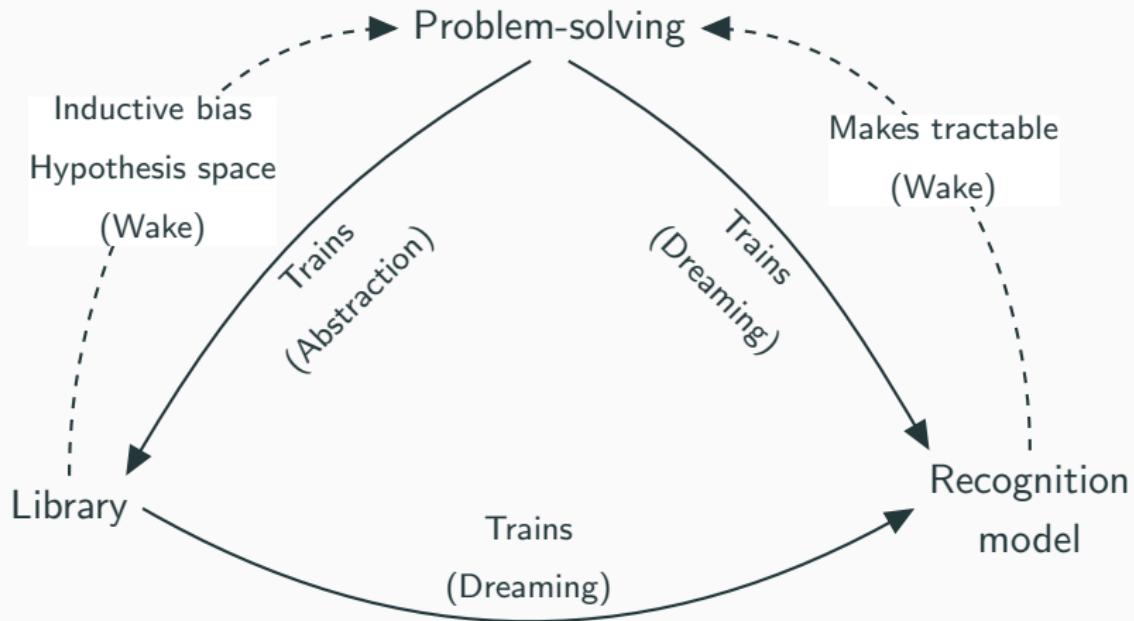
synergy between dreaming and library learning



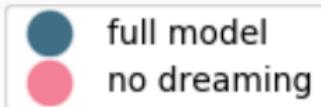
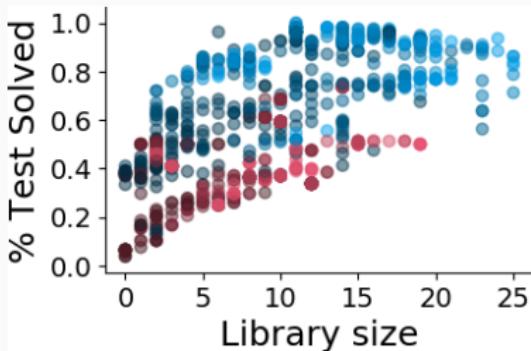
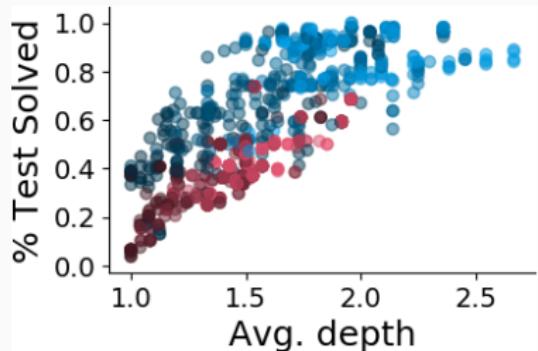
synergy between dreaming and library learning



synergy between dreaming and library learning



Evidence for dreaming bootstrapping better libraries



Dark→Light: Early in learning→Later in learning