Quantitative Reasoning and a Bayesian View of Synthesis

with inputs on slides from Armando Solar-Lezama

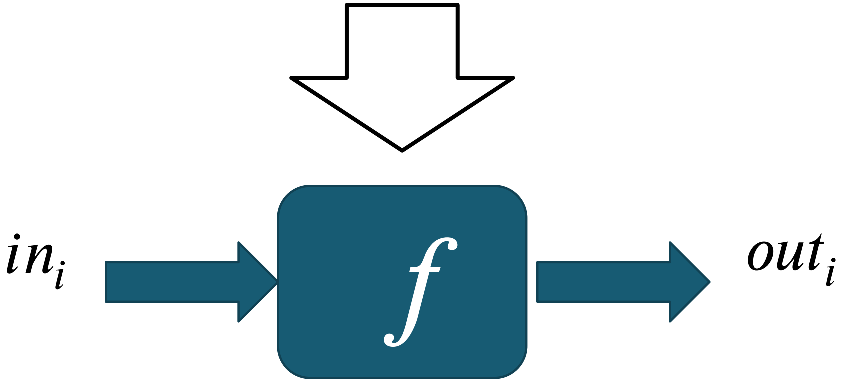
Bayes Theorem

$$P(A \mid B) = \frac{P(B \mid A)\,P(A)}{P(B)}$$

$$P(A \cap B) = P(B \mid A)\,P(A) = P(A \mid B)\,P(B)$$

Programming by Example

$$[(in_0, out_0), (in_1, out_1), \ \ldots (in_k, out_k)]$$



Problem is hopelessly underspecified

- Many semantically distinct programs can satisfy the examples

Bayesian View of PBE

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

$$P(f \mid evidence) = \frac{P(evidence \mid f)P(f)}{P(evidence)}$$

I/O Examples

Background Prob/Prior

For our purpose of finding an optimal f, we can ignore P(evidence) in the denominator

P(evidence) != 0

Bayesian View of PBE

Find the best f given the evidence

$$P(f \mid evidence) = \frac{P(evidence \mid f)\,P(f)}{P(evidence)}$$

Find the best f given the evidence

$$\underset{f}{\operatorname{argmax}} P(f \mid evidence) = \underset{f}{\operatorname{argmax}} \frac{P(evidence \mid f)\,P(f)}{P(evidence)}$$

$$= \underset{f}{\operatorname{argmax}}\; P(evidence \mid f)\,P(f)$$

WARNING: $P(evidence)$ better not be zero!

$P(evidence \mid f)$

z is a normalization constant, crucial for making sure these are probabilities, but unimportant from the point of view of optimiziation.

Perfectly captured I/O examples

- $P(evidence \mid f) = P\left(\left[(in_i, out_i)\right]_i \mid f\right) = \begin{cases} 1/z & \forall_i\ f(in_i) = out_i \\ 0 & otherwise \end{cases}$
- With a uniform $P(f)$ this reduces to finding any function that works

$P(f)$

So far we have been using a uniform P

- $$P(f) = \begin{cases} 1/Z & if\ f\ belongs\ to\ the\ space\ of\ programs \\ 0 & otherwise \end{cases}$$

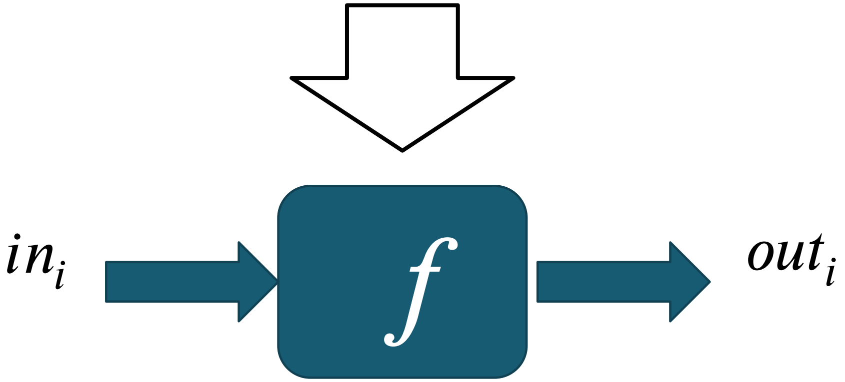
Shortest programs are better than longer programs

- $$P(f) = \begin{cases} \frac{1}{Z} * e^{-len(f)} & if\ f\ belongs\ to\ the\ space\ of\ programs \\ 0 & otherwise \end{cases}$$

Could we learn $P(f)$?

Programming by Example

$$[(in_0, out_0), (in_1, out_1), \ \ldots (in_k, out_k)]$$



Problem is hopelessly underspecified

- Many semantically distinct programs can satisfy the examples

$$P\left(f \,\middle|\, [(in_i, out_i)]_i\right) \approx P_f(f) * P_{io}\left([(in_i, out_i)]_i \,\middle|\, f\right)$$

$P(evidence \mid f)$ for Synthesis under errors

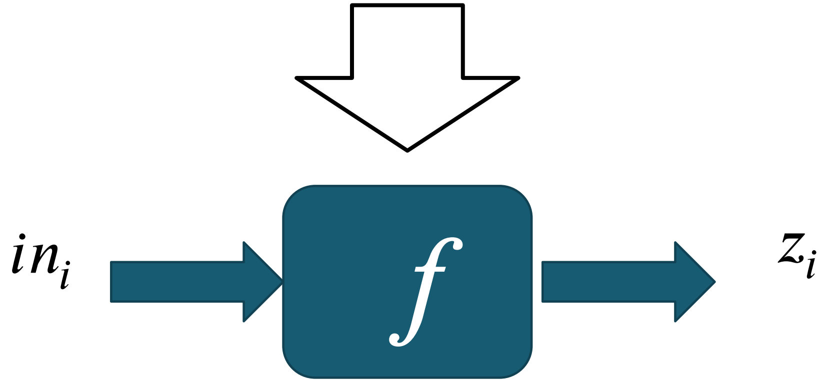
$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Imperfectly captured independent I/O examples

- $P(evidence \mid f) = P\left([(in_i, out_i)]_i \mid f \right) = \prod_i P_{o|z}\left(out_i \;\middle|\; f,\ in_i\right) P(in_i))$
- For the purposes of maximizing $P(f)$, $P(in_i)$ can be ignored if all inputs are equally likely

Learning from noisy data

$$[(in_0, out_0), (in_1, out_1),\ \ldots (in_k, out_k)]$$



Need to trade off quality of f against faithfulness to data

- This requires an error model

$$P\left(f \;\middle|\; \left[(in_i, out_i)\right]_i\right) \approx P_f(f) * \prod_i P_{o|z}\left(out_i \;\middle|\; f,\ in_i\right)$$

Example: Off-by-one Errors

Suppose we know off-by-one errors are possible in the data

- $$P_{o|z}(out_i \mid f,\ in_i) = \begin{cases} 0.5 & f(in_i) = out_i \\ 0.25 & f(in_i) = out_i \pm 1 \\ 0 & else \end{cases}$$

If $p(f)$ is uniform, this reduces to

- “Discard programs that are more than one-off on any input”
- “From the remaining programs, select the one that matches the most examples”

Off-by-one Errors

Suppose we know off-by-one errors are possible in the data but others are possible as well.

- $$P_{o|z}(out_i \mid f,\ in_i) = \begin{cases} \frac{1}{Z}0.5 & f(in_i) = out_i \\ \frac{1}{z}0.25 & f(in_i) = out_i \pm 1 \\ \epsilon & else \end{cases}$$

Non-uniform $P(f)$

Trade off $P_{o|z}\left(out_i \;\middle|\; f,\ in_i\right)$ against $P\left(f\right)$

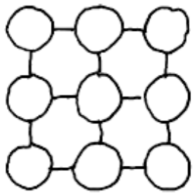
- A solution that misses more outputs may still be preferable if it has much higher probability

Learning to Infer Graphics Programs from Hand-Drawn Images

Kevin Ellis, Daniel Ritchie, Josh Tenenbaum

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, Josh Tenenbaum, *Learning to Infer Graphics Programs from Hand-Drawn Images*, 2018

From images to programs



Hand Drawn Figure

NN + Search

```
Circle(5,8)
Circle(2,8)
Circle(8,11)
Line(2,9, 2,10)
Circle(8,8)
Line(3,8, 4,8)
Line(3,11, 4,11)
Line(8,9, 8,10)
Circle(5,14)
```

... etc. ...; 21 lines

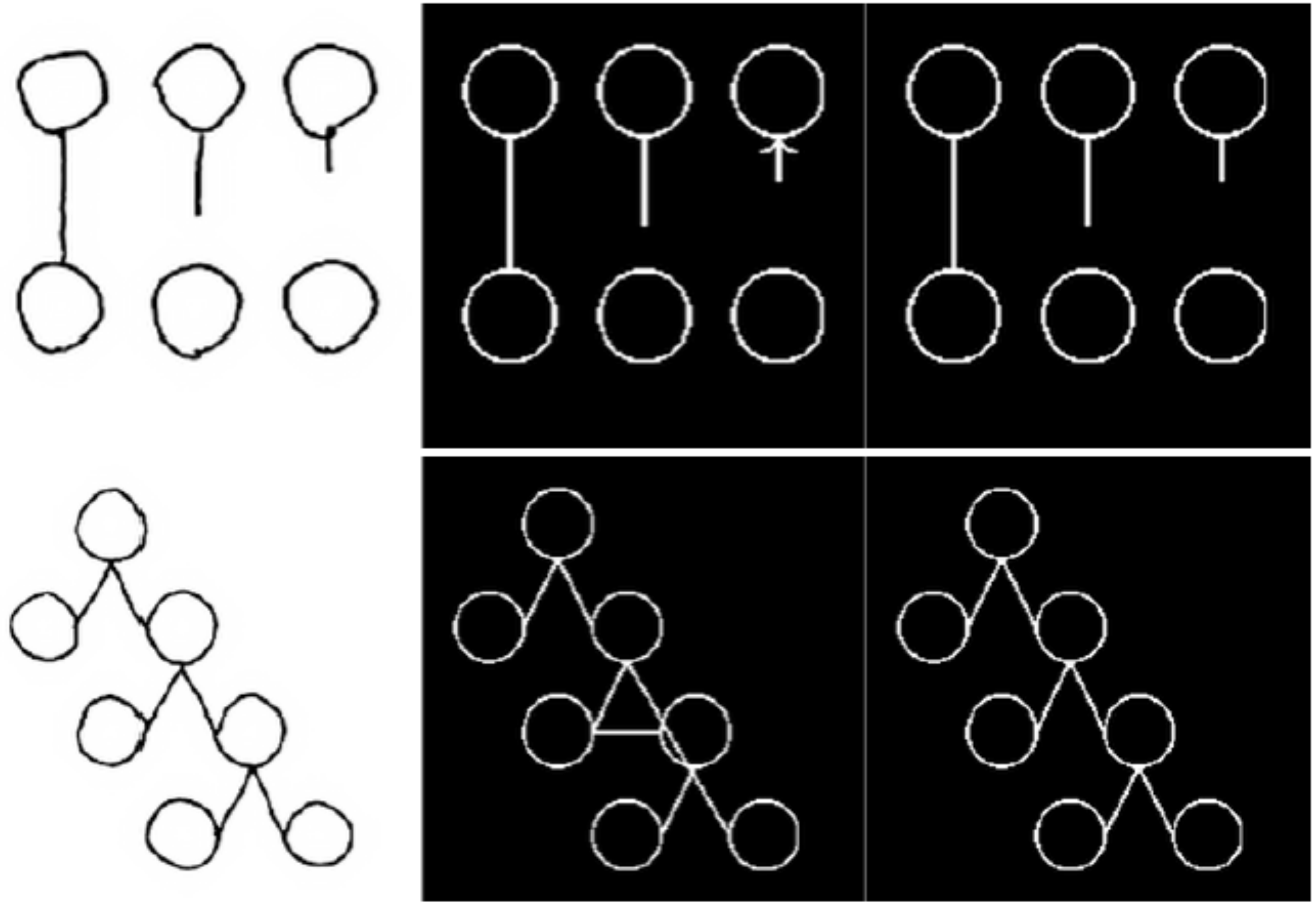
Description of
elements in the drawing

Synthesis

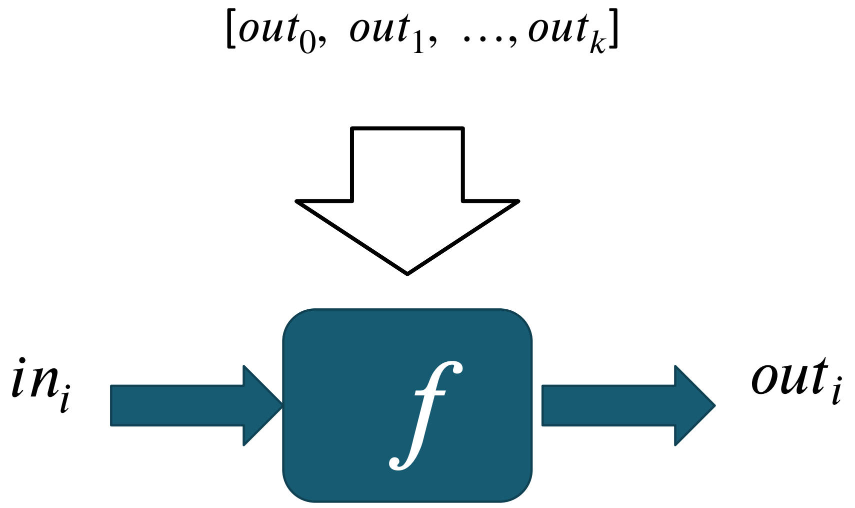
```
for(i<3)
 for(j<3)
  if(j>0)
   line(-3*j+8,-3*i+7,
        -3*j+9,-3*i+7)
  line(-3*i+7,-3*j+8,
       -3*i+7,-3*j+9)
  circle(-3*j+7,-3*i+7)
```

Program representation
of drawing

Why? Correcting errors in perception



Unsupervised learning



This is hopelessly underspecified

- Can we identify the process that generated the sequence?

Kevin Ellis, Armando Solar-Lezama, Joshua B. Tenenbaum, *Unsupervised Learning by Program Synthesis*, 2015

Unsupervized learning

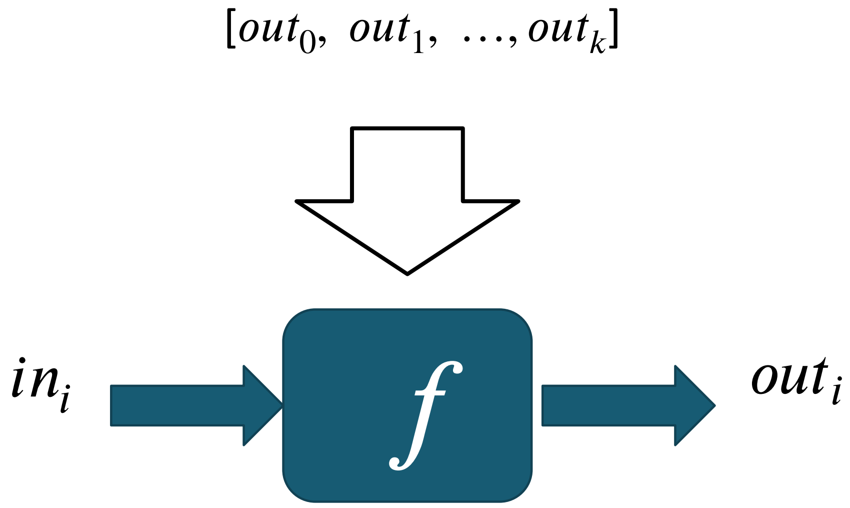
$$P\Big(f,\ [in_i]\ \mid\ [out_i]\Big) = \frac{P\Big(\ [out_i]\ \mid\ f,\ [in_i]\ \Big)P(f,\ [in_i])}{P([out_i])}$$

Assuming independence:

$$P(\ [out_i]\ \mid\ f,\ [in_i]) = \Pi_i\ P\big(out_i\ \mid\ f,\ in_i\big)$$

$$P\Big(f,\ [in_i]\Big) = P\big(f\big) * \Pi_i\ P\big(in_i\big)$$

Unsupervised learning



This is hopelessly underspecified

- Can we identify the process that generated the sequence?

$$P\left(f,\ [in_i]_i \,\middle|\, [out_i]_i\right) \approx P_f(f) * \prod_i P_{o|z}\left(out_i \mid f, in_i\right) * P_{in}(in_i)$$

To marginalize or not to Marginalize

$$P\left(f,\ [in_i]_i \middle| [out_i]_i\right)$$

Probability that a given function and inputs were the cause for an observed series of outputs

$$\sum_{[in_i]_i} P\left(f,\ [in_i]_i \middle| [out_i]_i\right) P([in_i]_i)$$

Probability that a given function is consistent with the observed outputs

Which of the two functions above should you be optimizing?

- Formulation on the left is easier to solve for
  - especially with symbolic methods

Maximum Likelihood vs Sampling

Often your goal is to find the most likely f

- $\max_f P_f(f \mid \ldots)$

For some situations, sampling from $P_f(f \mid \ldots)$ is better

- The most likely is not necessarily the one you want
- E.g. in PBE the function the user has in mind may not be the "best"

Inversion, Rejection, Relationship and Approximation

Isn't there a whole field looking into this?

Machine learning has been studying these problems for a while

What we bring to the table:

- Flexible spaces of functions
- Complex distributions
- Powerful symbolic search techniques

Machine Learning for Synthesis

- DeepCoder : Learning to Write Programs,
    - Balog et al. ICLR 2017

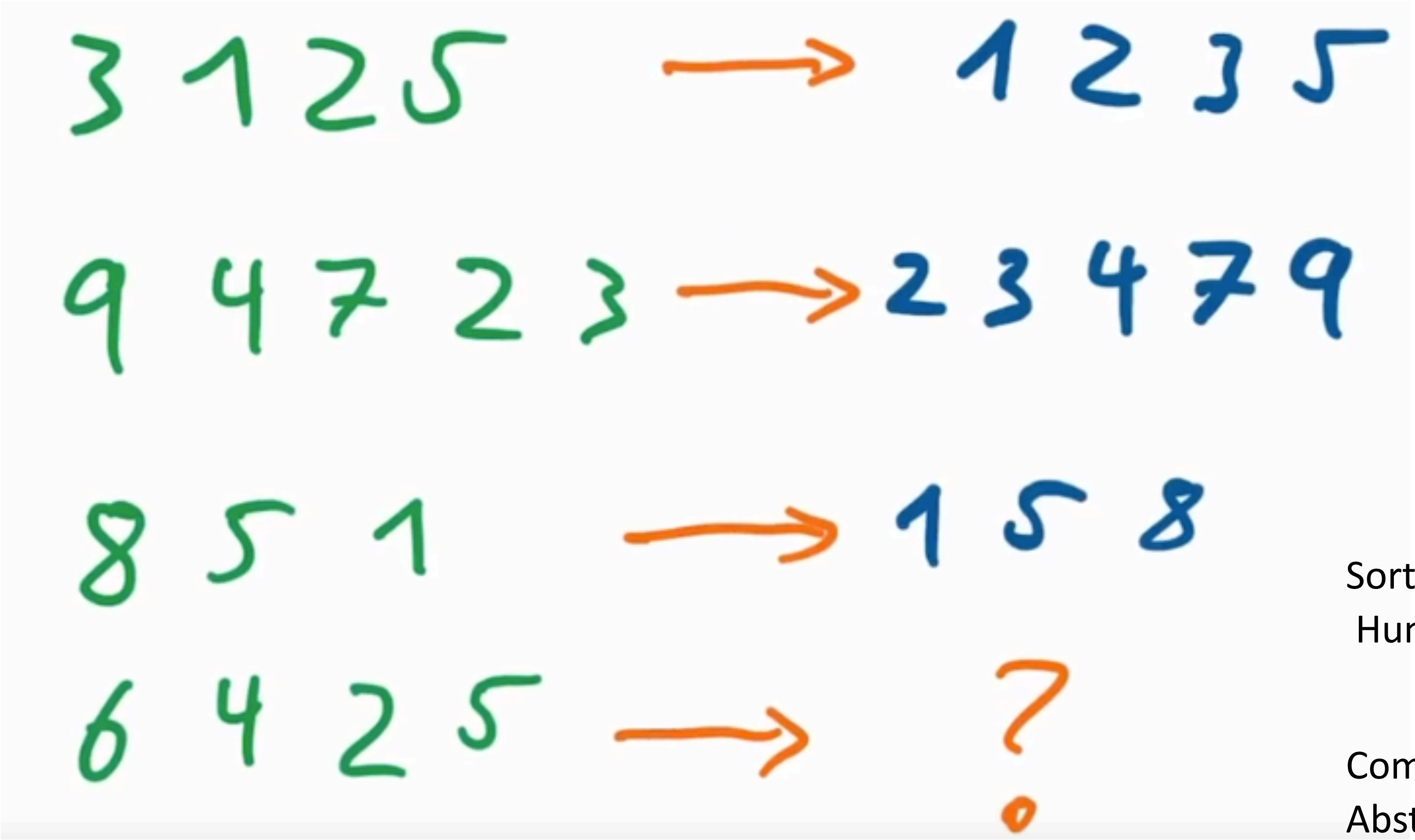
- DreamCoder : Bootstrapping Inductive Program Synthesis with Wake-Sleep Library Learning.
    - Ellis et al. PLDI 2021

DreamCoder

Kevin Ellis, Catherine Wong, Mexwell Nye, Mathias Sable-Meyer, Luc Cary, Lucas Morales, Like Hewitt, Armando Solar-Lezama, Joshua B. Tenenbaum

Kevin Ellis et al. DreamCoder: bootstrapping inductive program synthesis with wake-sleep library learning. PLDI 21

Whats happening here?



Sorting: This core feature of
Humans is hard for ML

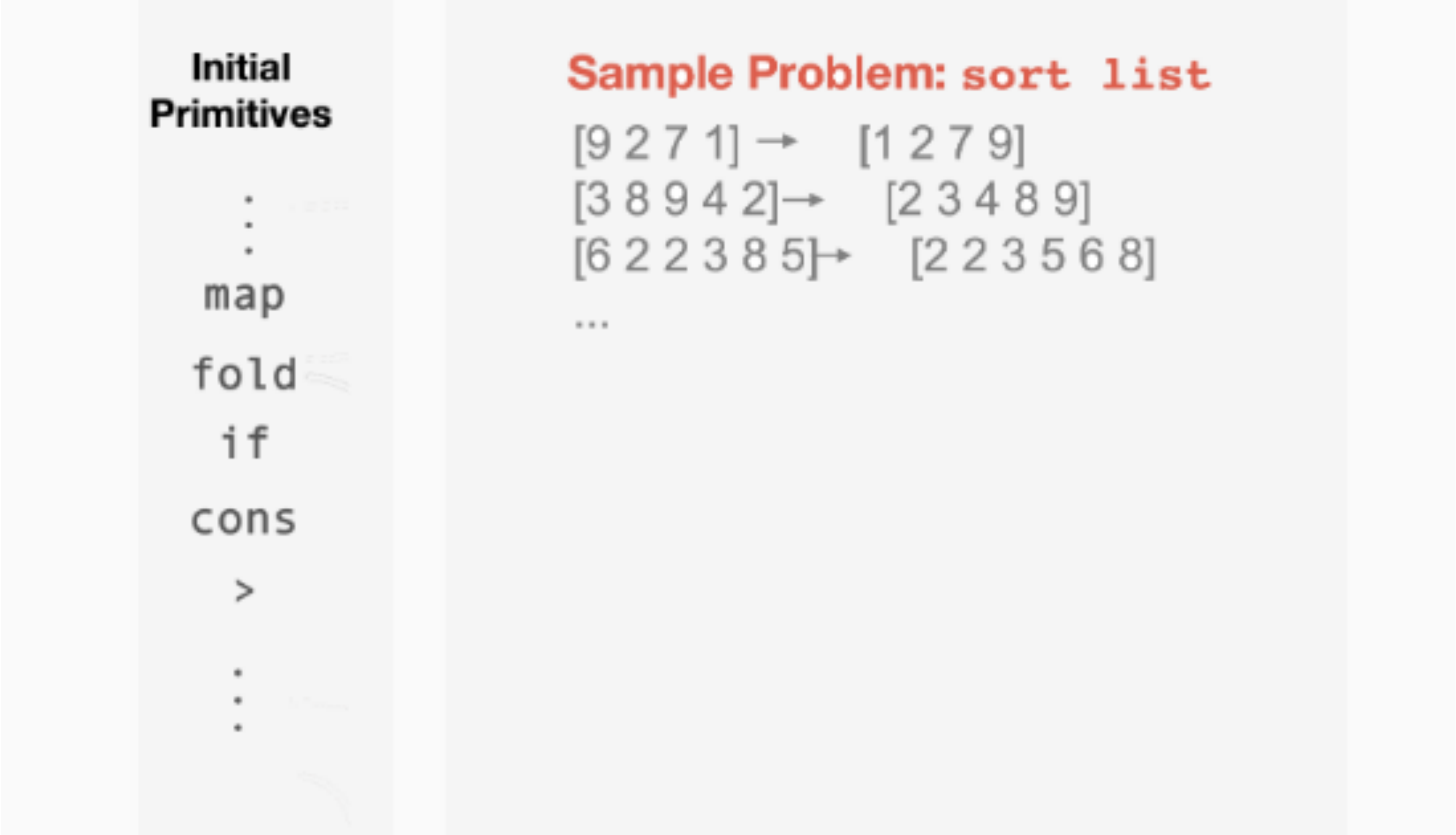
Combine ML techniques with
Abstration Learning

Program Synthesis (Inductive Program Synthesis)

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

- Library of abstractions (domain specific language)   Assumed this is given
- Inference strategy (synthesis algorithm)

Library Learning



Library Learning

Initial Primitives

⋮

`map`

`fold`

`if`

`cons`

`>`

⋮

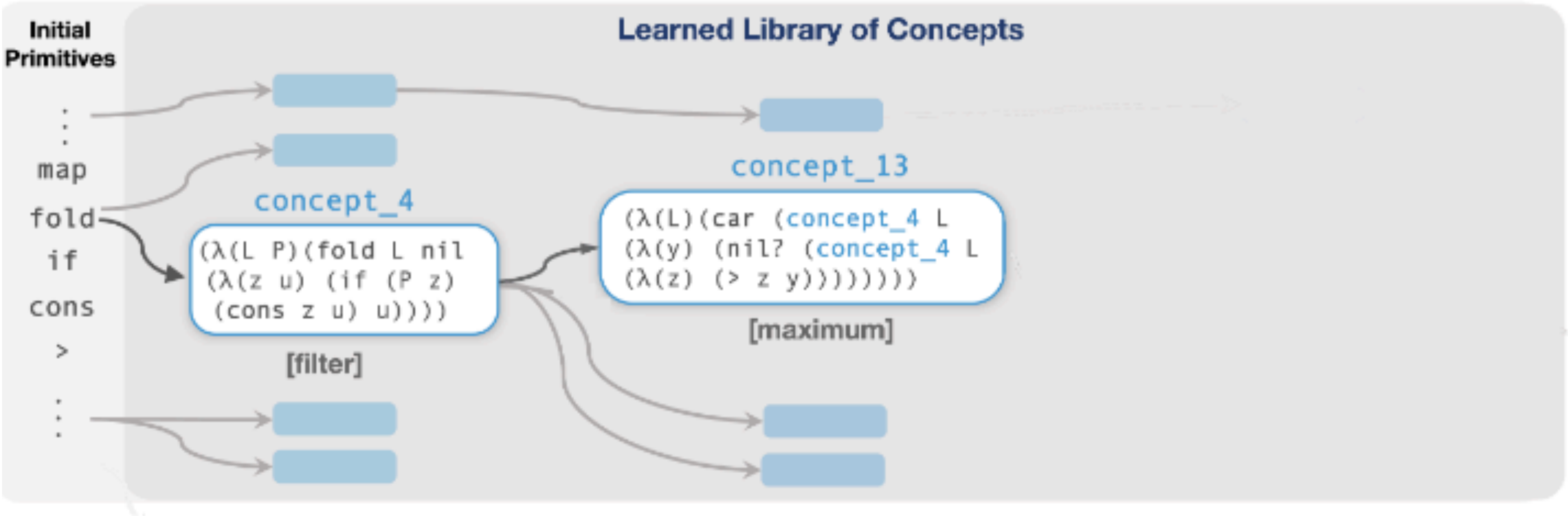
Sample Problem: `sort list`

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

Library Learning

Library Learning

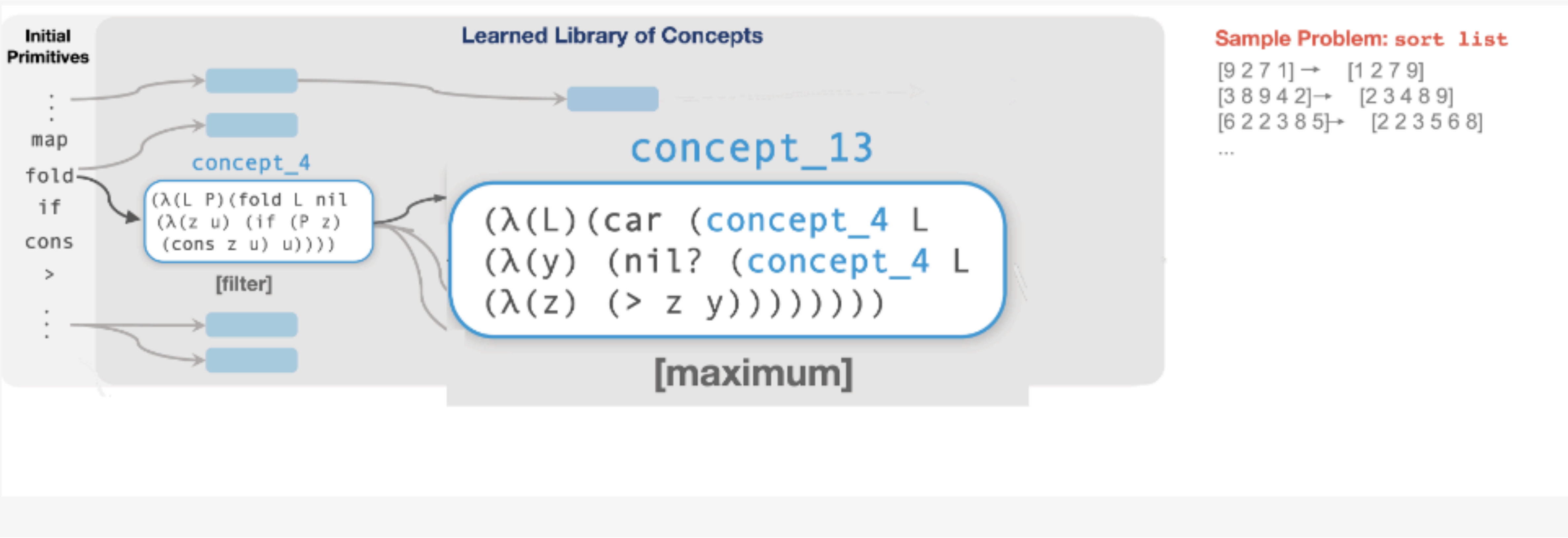
Library Learning



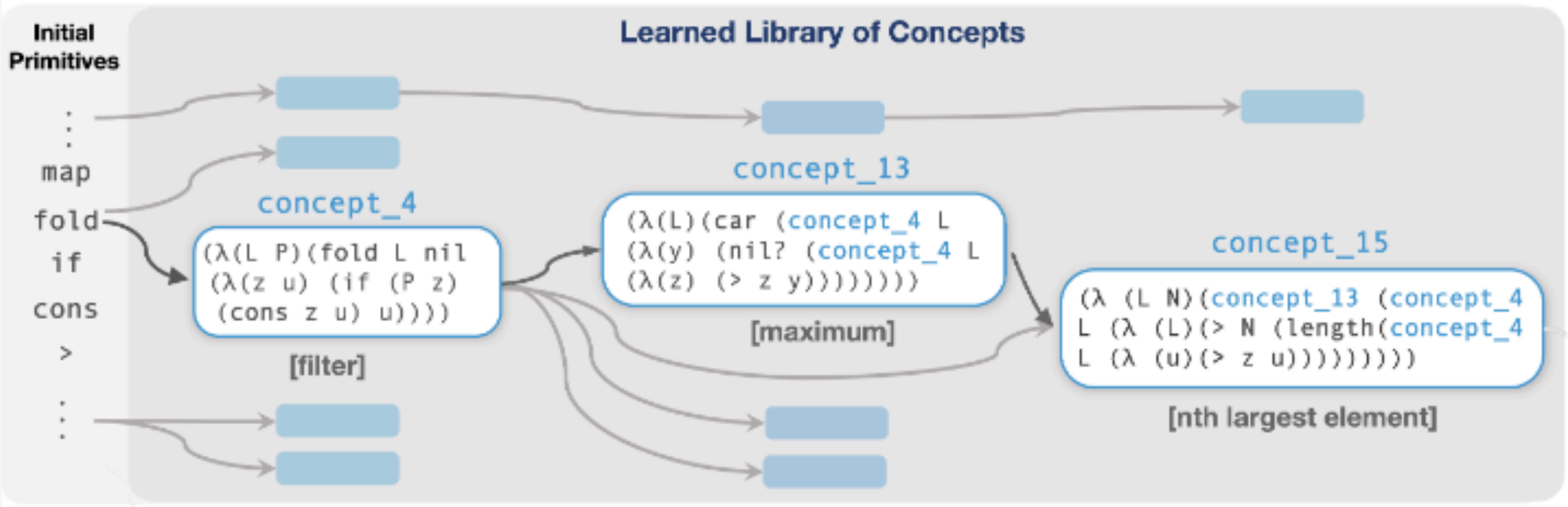
Sample Problem: `sort list`

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

Library Learning



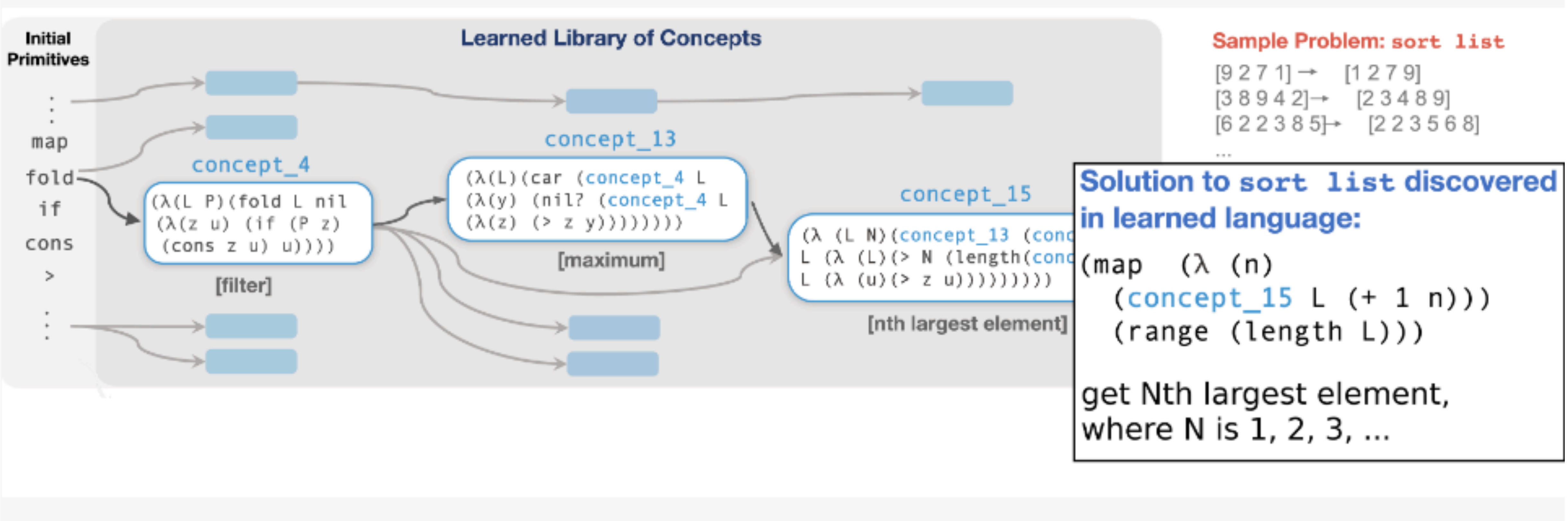
Library Learning



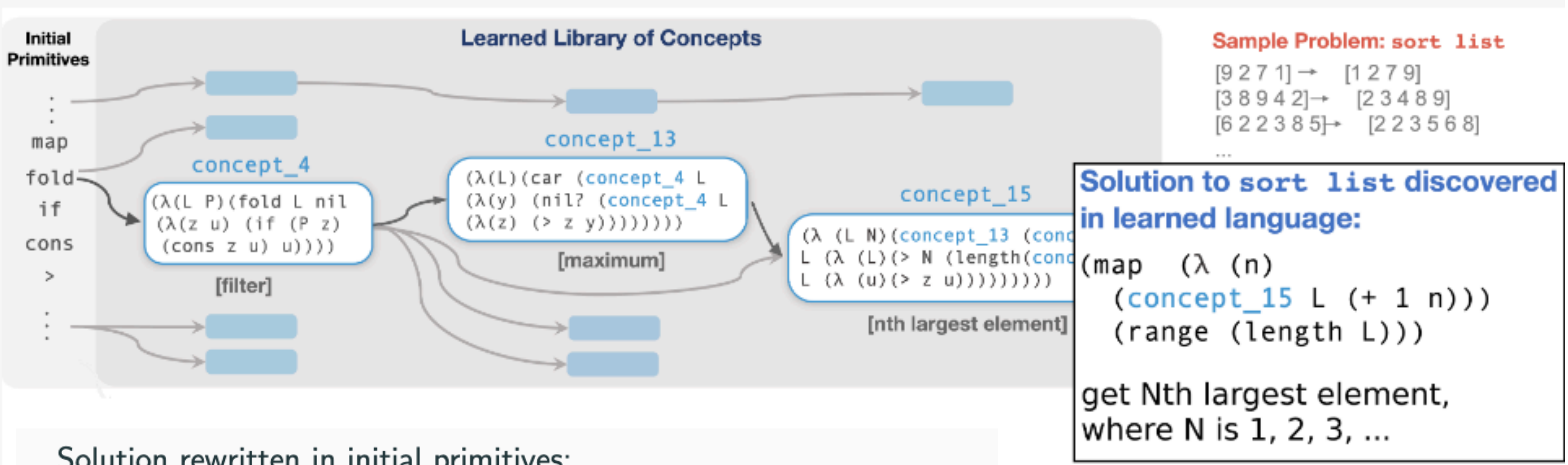
Sample Problem: `sort list`

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

Library Learning



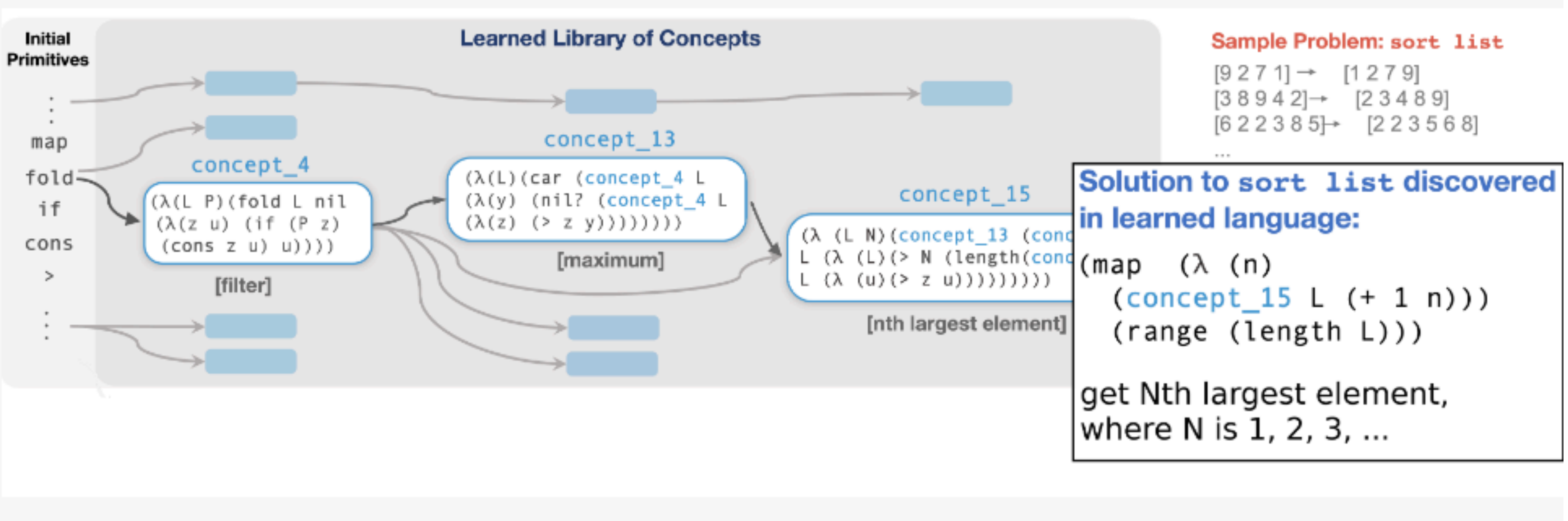
Library Learning



Solution rewritten in initial primitives:

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

Library Learning



Solution rewritten in initial primitives:

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

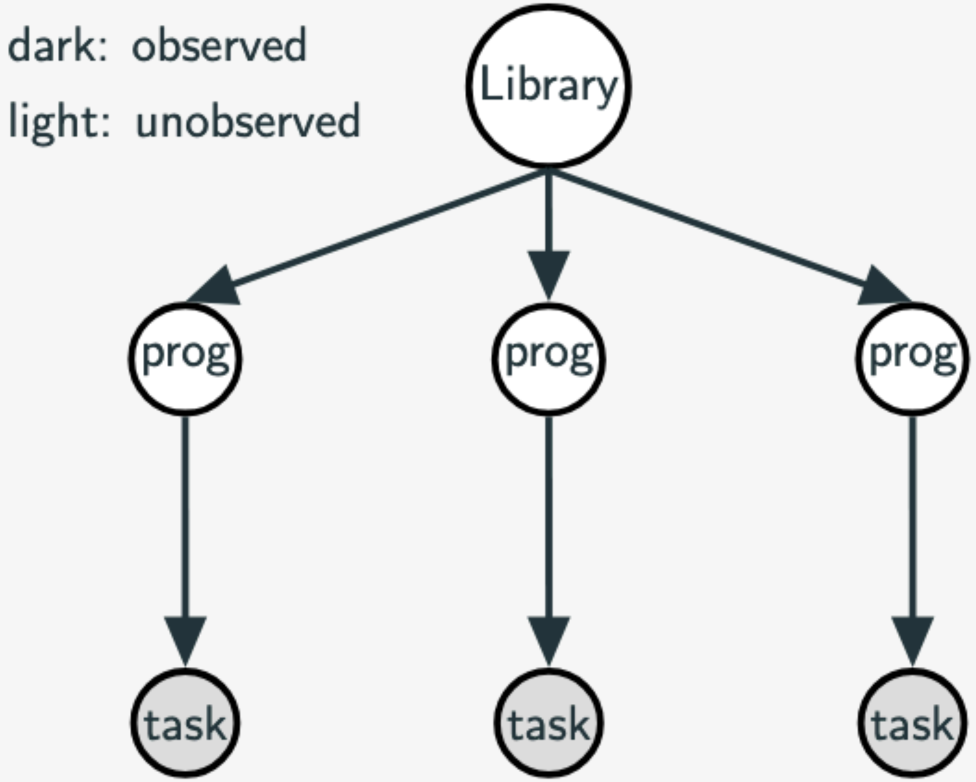
induced sort program found in $\leq$ 10min. Brute-force search without learned library would take $\approx 10^{73}$ years

DreamCoder

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

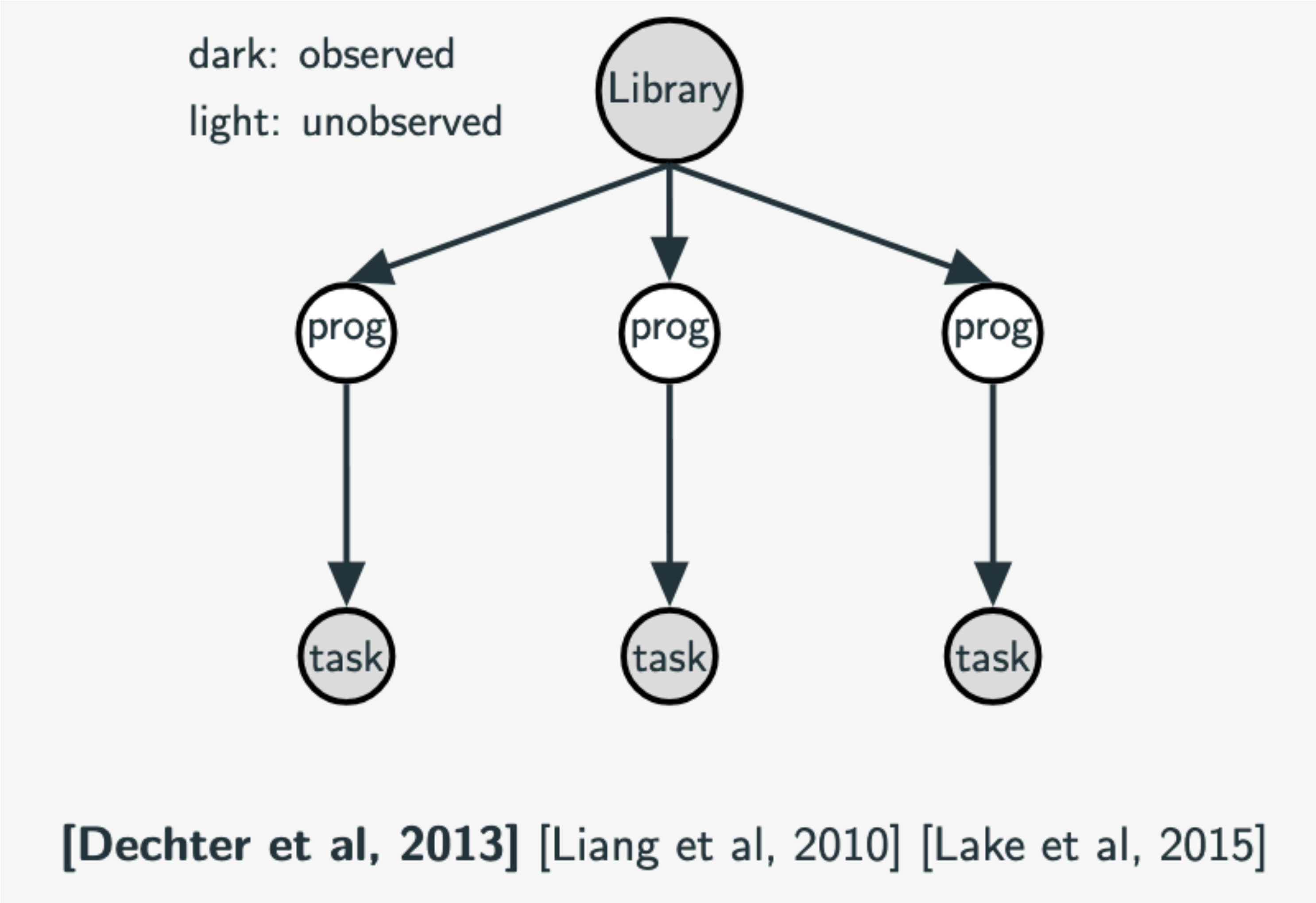
cf. Helmholtz machine, wake/sleep neural network training algorithms

Library Learning as Bayesian inferences

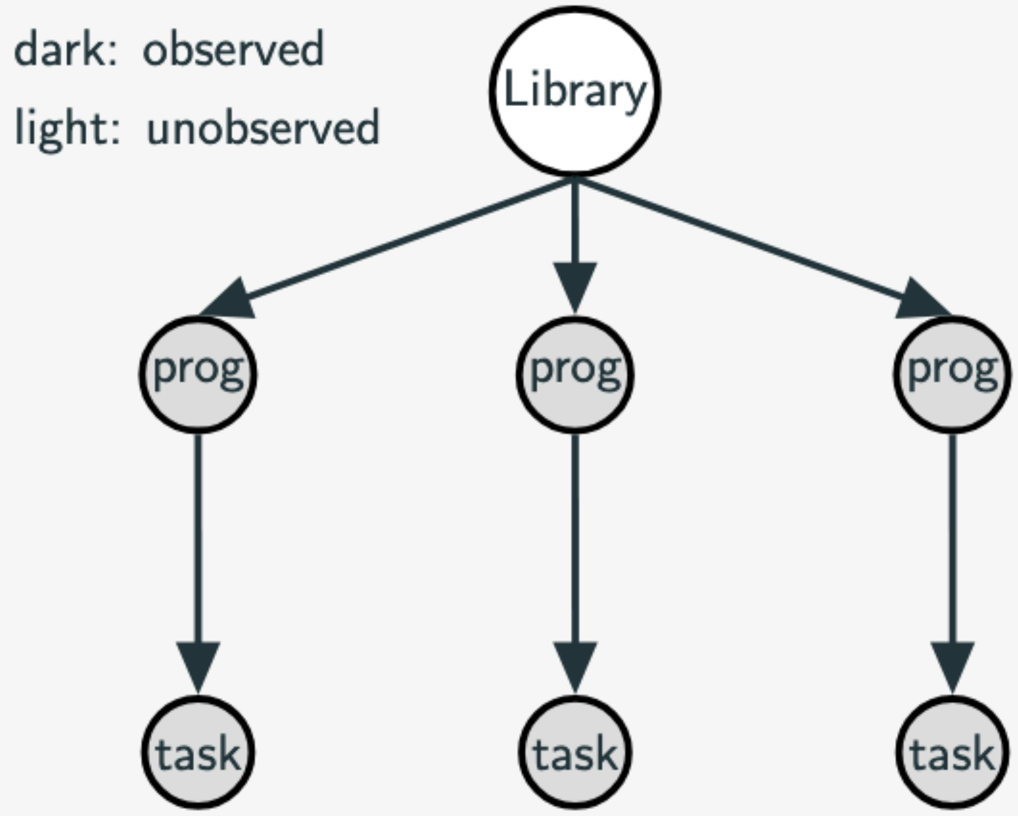


[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

Library Learning as Bayesian inferences

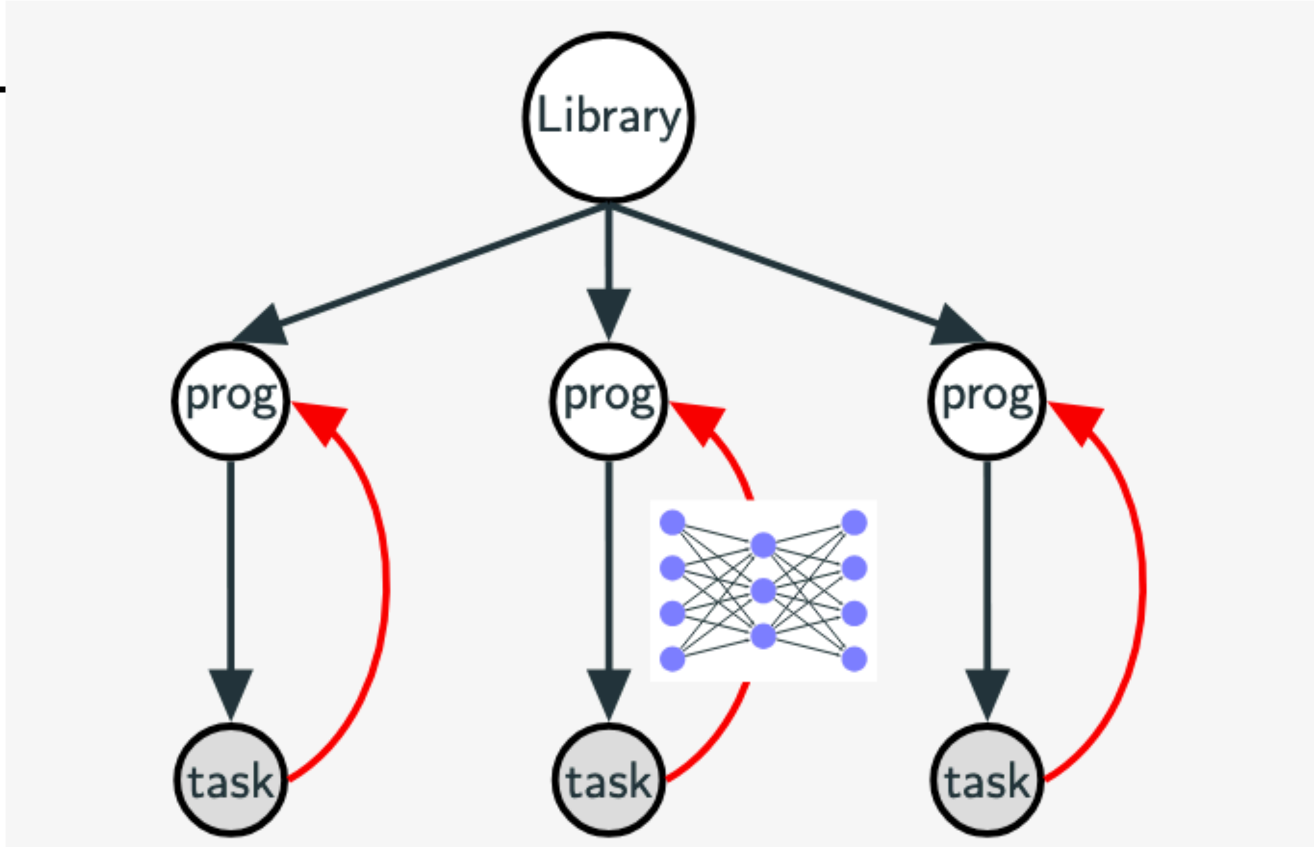


Library Learning as Bayesian inferences



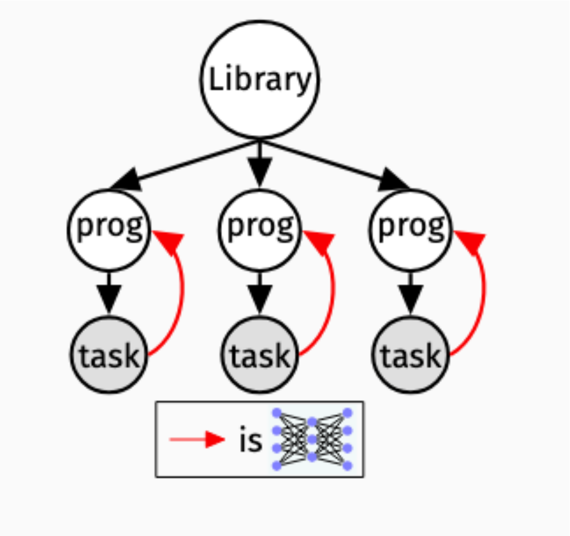
[Dechter et al, 2013] [Liang et al, 2010] [Lake et al, 2015]

Library Learning as neurally-guided Bayesian inference



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

Kevin Ellis et al. Library Learning for Neurally- Guided Bayesian Program Induction. In NeurIPS 2018.


Library
prog
prog
prog
task
task
task
is

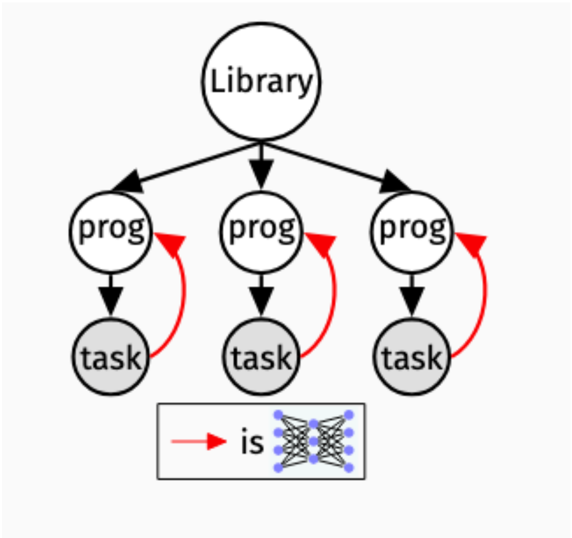
Wake

Synthesis

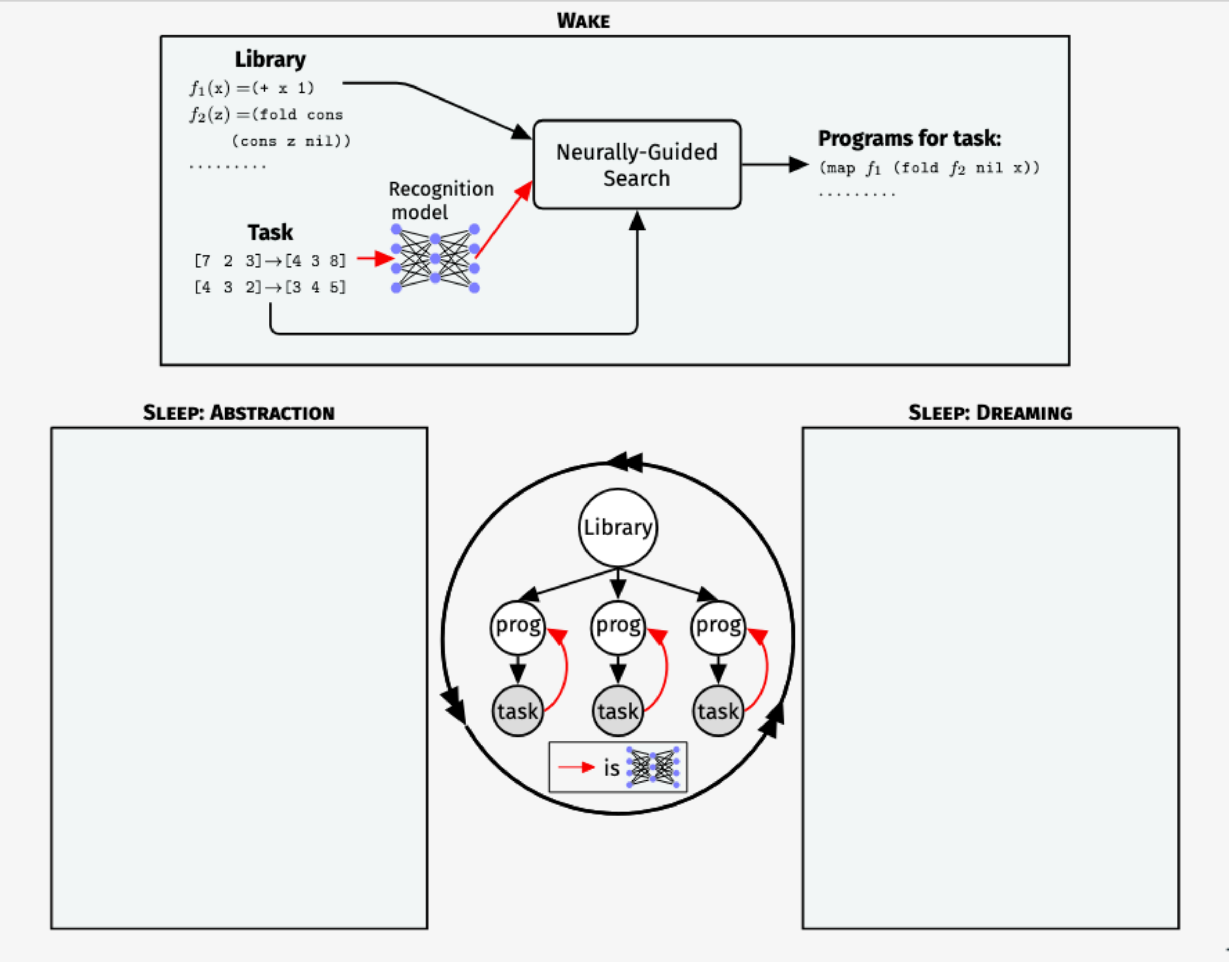
Train the recognition model

Sleep: Abstraction

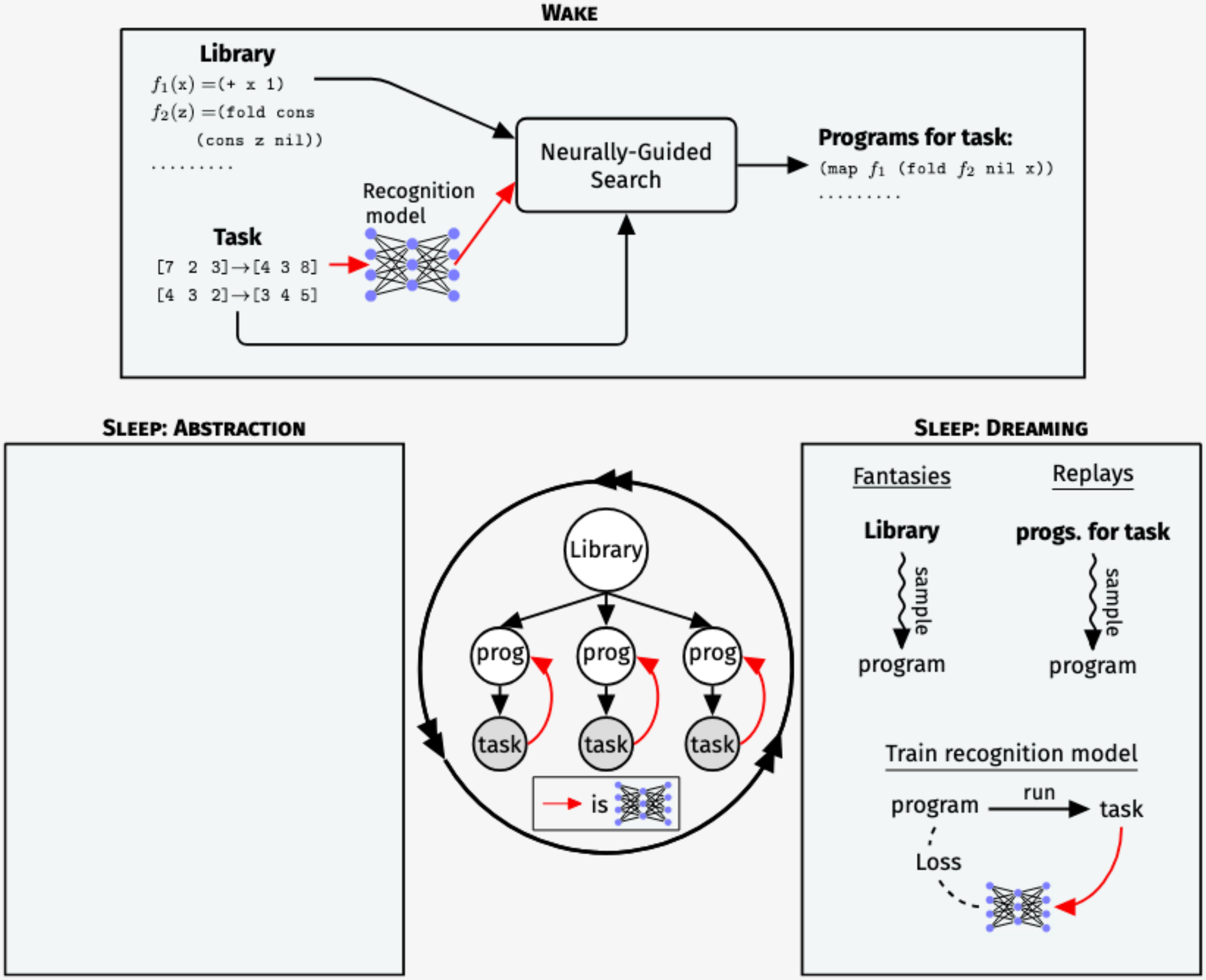
Identify re-used programs

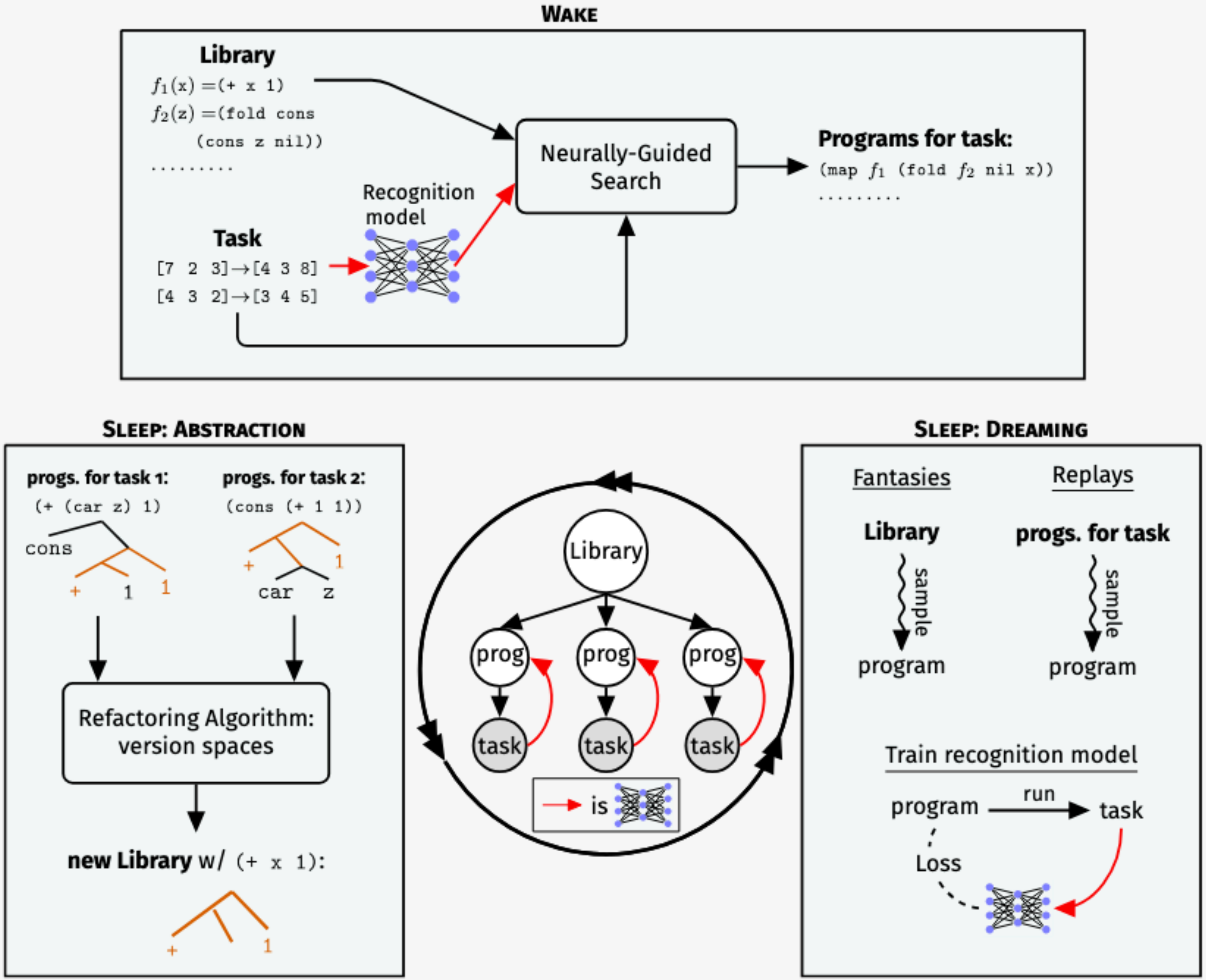


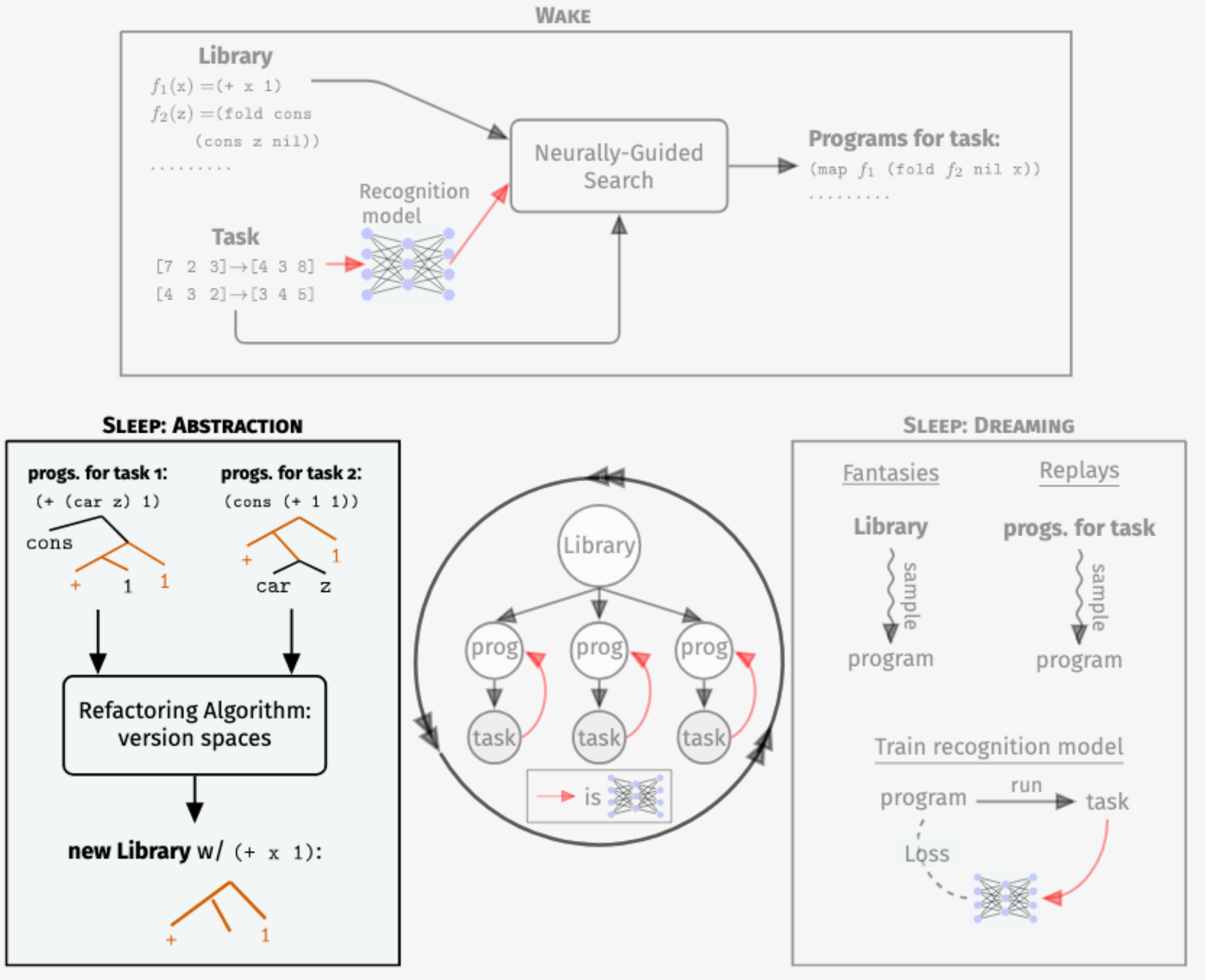
Sleep: Dreaming



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


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

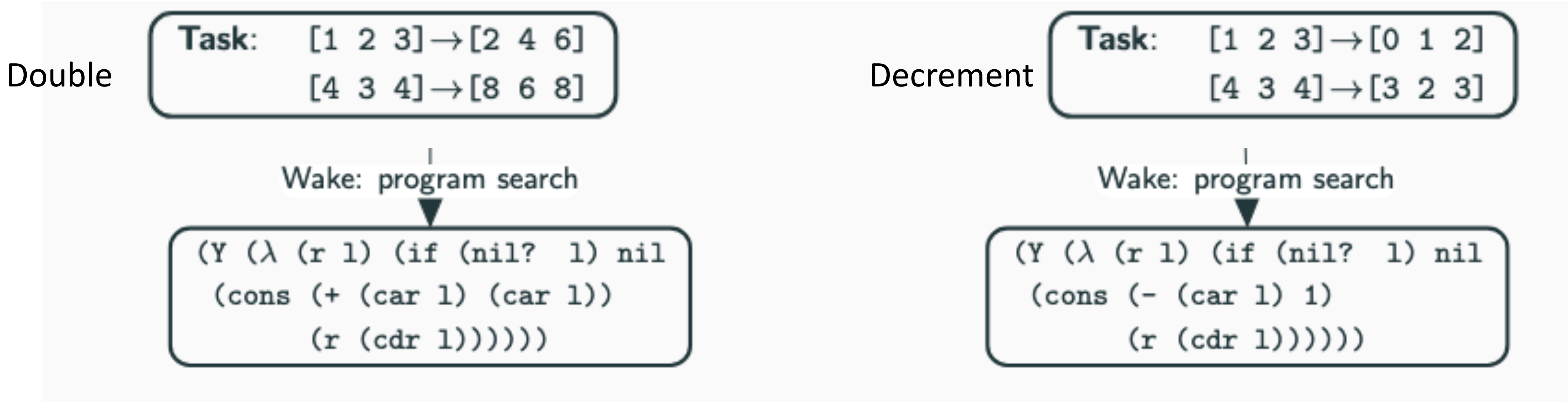

WAKE
Library
f1(x) =(+ x 1)
f2(z) =(fold cons (cons z nil))
Recognition model
Task
[7 2 3]→[4 3 8]
[4 3 2]→[3 4 5]
Neurally-Guided Search
Programs for task:
(map f1 (fold f2 nil x))
SLEEP: ABSTRACTION
progs. for task 1:
(+ (car z) 1)
cons
progs. for task 2:
(cons (+ 1 1))
car
z
Refactoring Algorithm: version spaces
new Library w/ (+ x 1):
Library
prog
task
is
SLEEP: DREAMING
Fantasies
Replays
Library
sample
program
progs. for task
Train recognition model
run
task
Loss



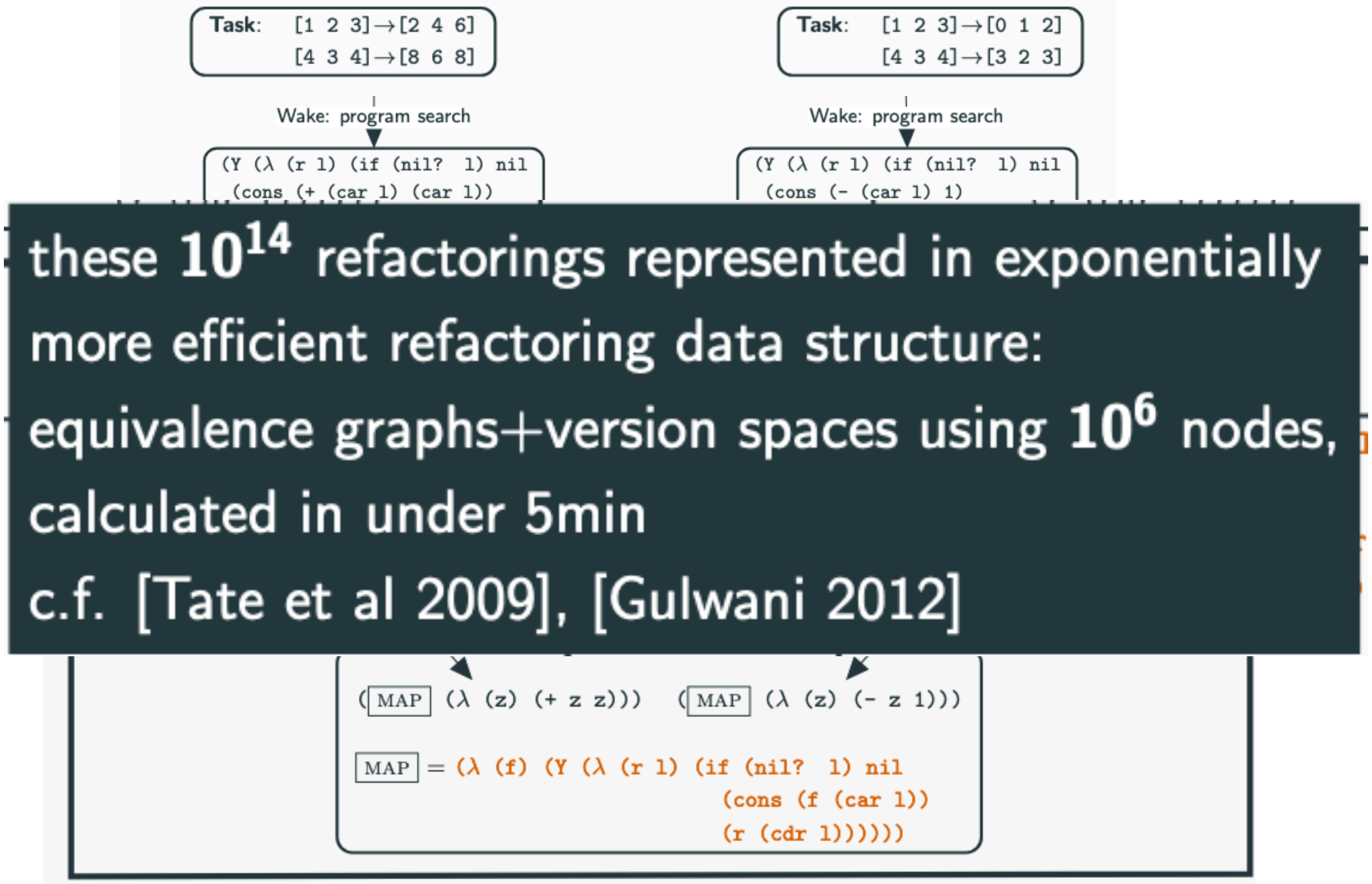
WAKE
Library
f_1(x) =(+ x 1)
f_2(z) =(fold cons (cons z nil))
....
Recognition model
Task
[7 2 3]→[4 3 8]
[4 3 2]→[3 4 5]
Neurally-Guided Search
Programs for task:
(map f_1 (fold f_2 nil x))
....
SLEEP: ABSTRACTION
progs. for task 1:
(+ (car z) 1)
cons
+
1
1
progs. for task 2:
(cons (+ 1 1))
+
1
car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

Program Induction and learning to learn
learning a DSL
learning to synthesize
synergy between DSL+learned synthesizer

Abstraction Sleep: Growing the library via refactoring



Abstraction Sleep: Growing the library via refactoring



these 10^{14} refactorings represented in exponentially more efficient refactoring data structure: equivalence graphs+version spaces using 10^6 nodes, calculated in under 5min

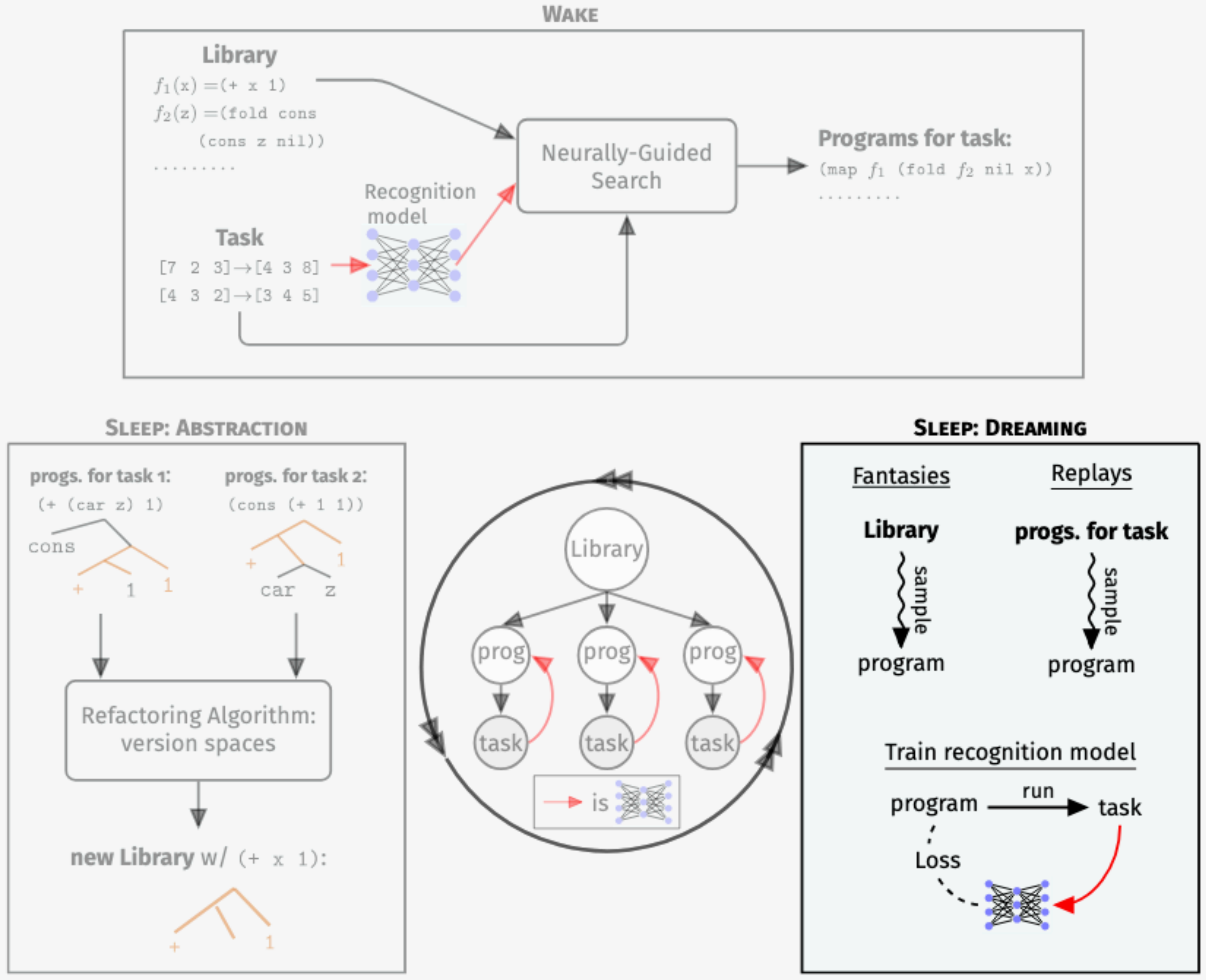
c.f. [Tate et al 2009], [Gulwani 2012]

Program Induction and learning to learn

learning a DSL

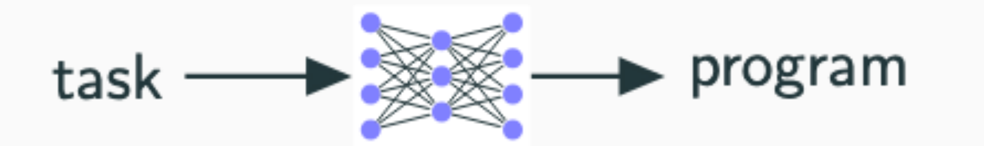
learning to synthesize

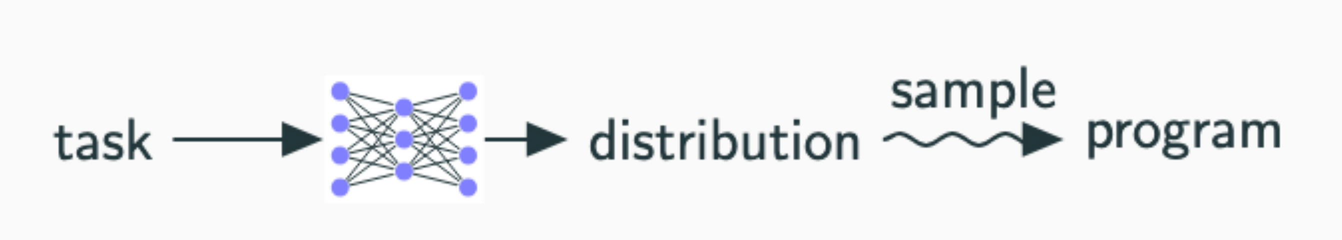
synergy between DSL+learned synthesizer

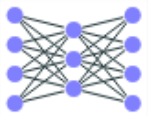


Wake
Library
f_1(x) =(+ x 1)
f_2(z) =(fold cons (cons z nil))
Recognition model
Task
[7 2 3]→[4 3 8]
[4 3 2]→[3 4 5]
Neurally-Guided Search
Programs for task:
(map f_1 (fold f_2 nil x))
Sleep: Abstraction
progs. for task 1:
(+ (car z) 1)
progs. for task 2:
(cons (+ 1 1))
cons
car
z
Refactoring Algorithm: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

Neural recognition model guides search



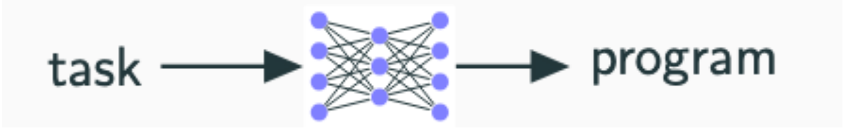


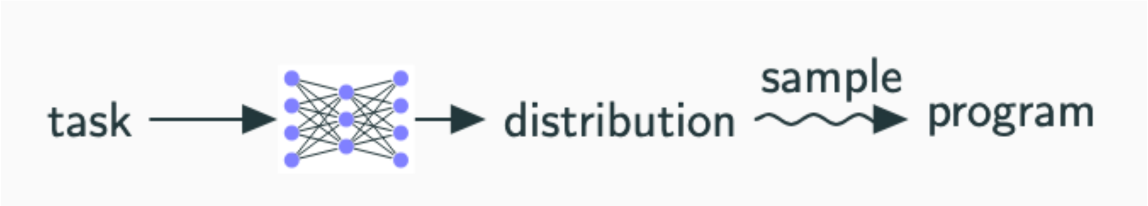
 is a...

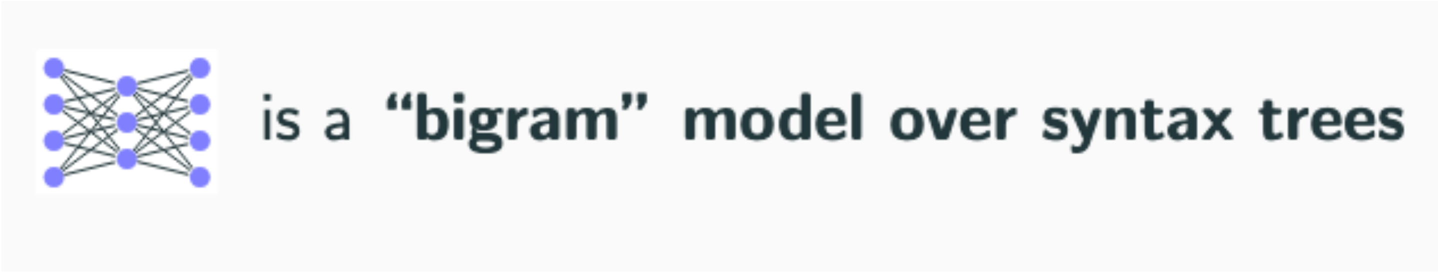
recurrent network (Devlin et al 2017)

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

Neural recognition model guides search







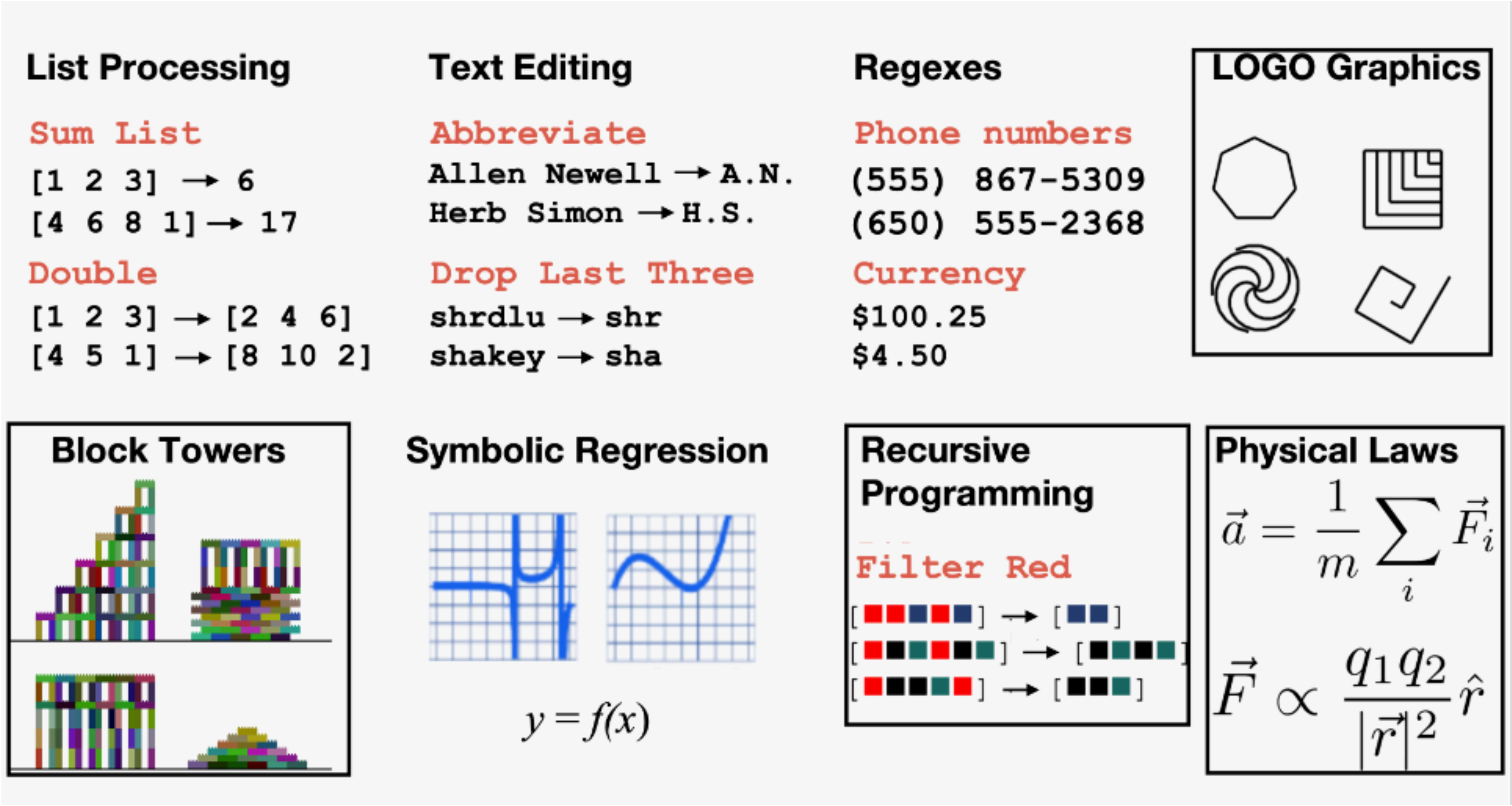
Program Induction and learning to learn

learning a DSL

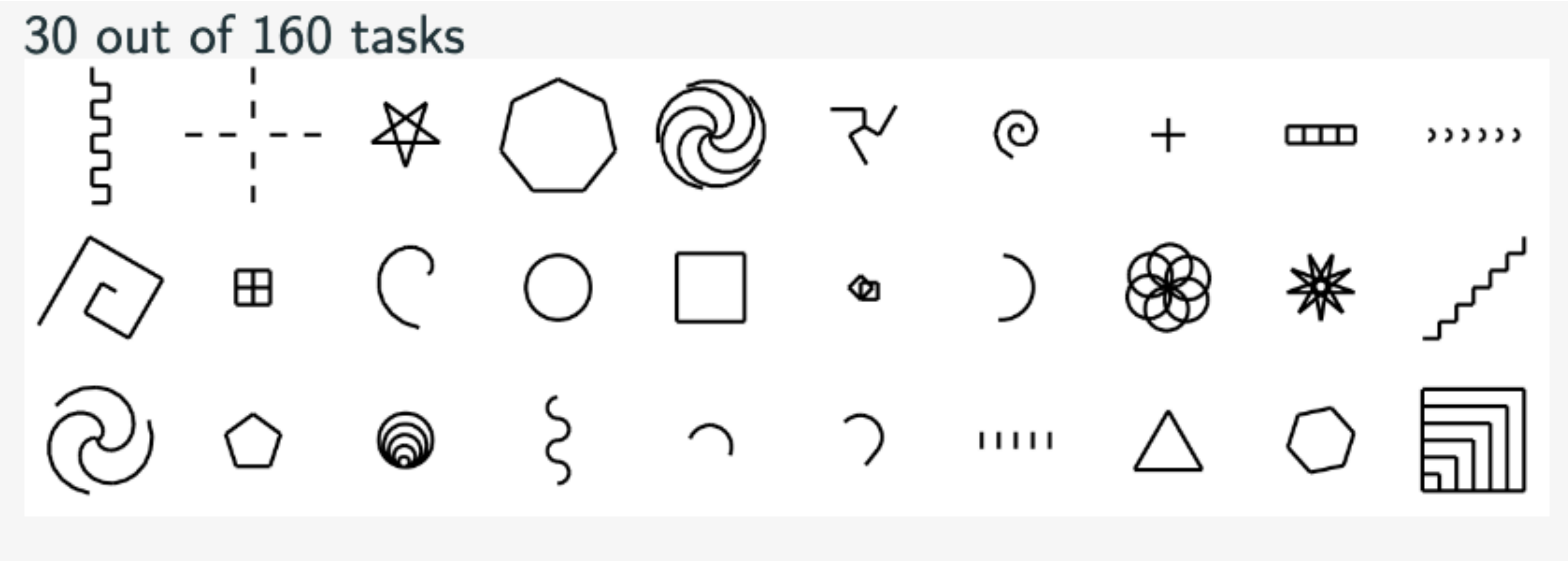
learning to synthesize

synergy between DSL+learned synthesizer

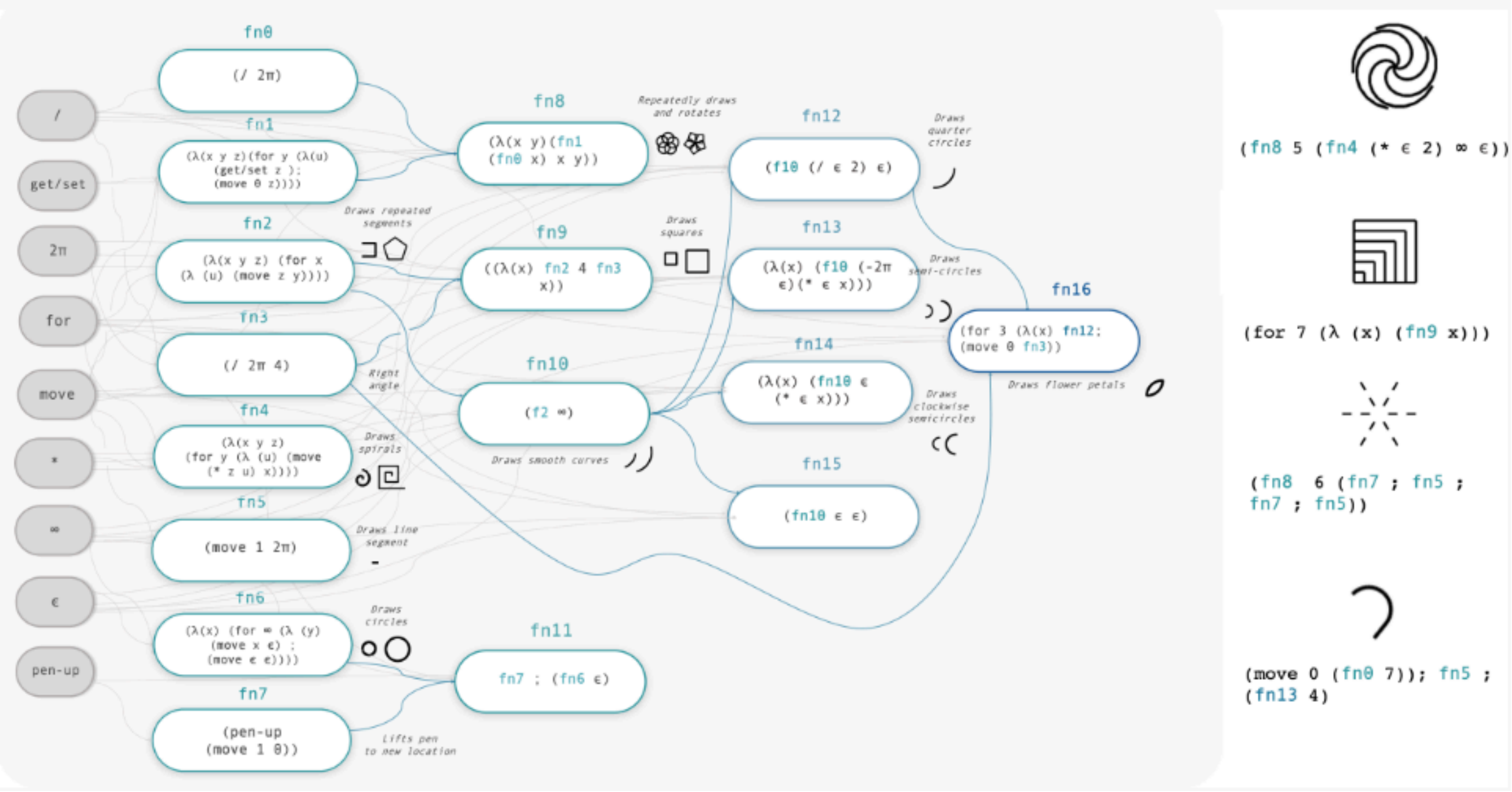
DreamCoder Domains



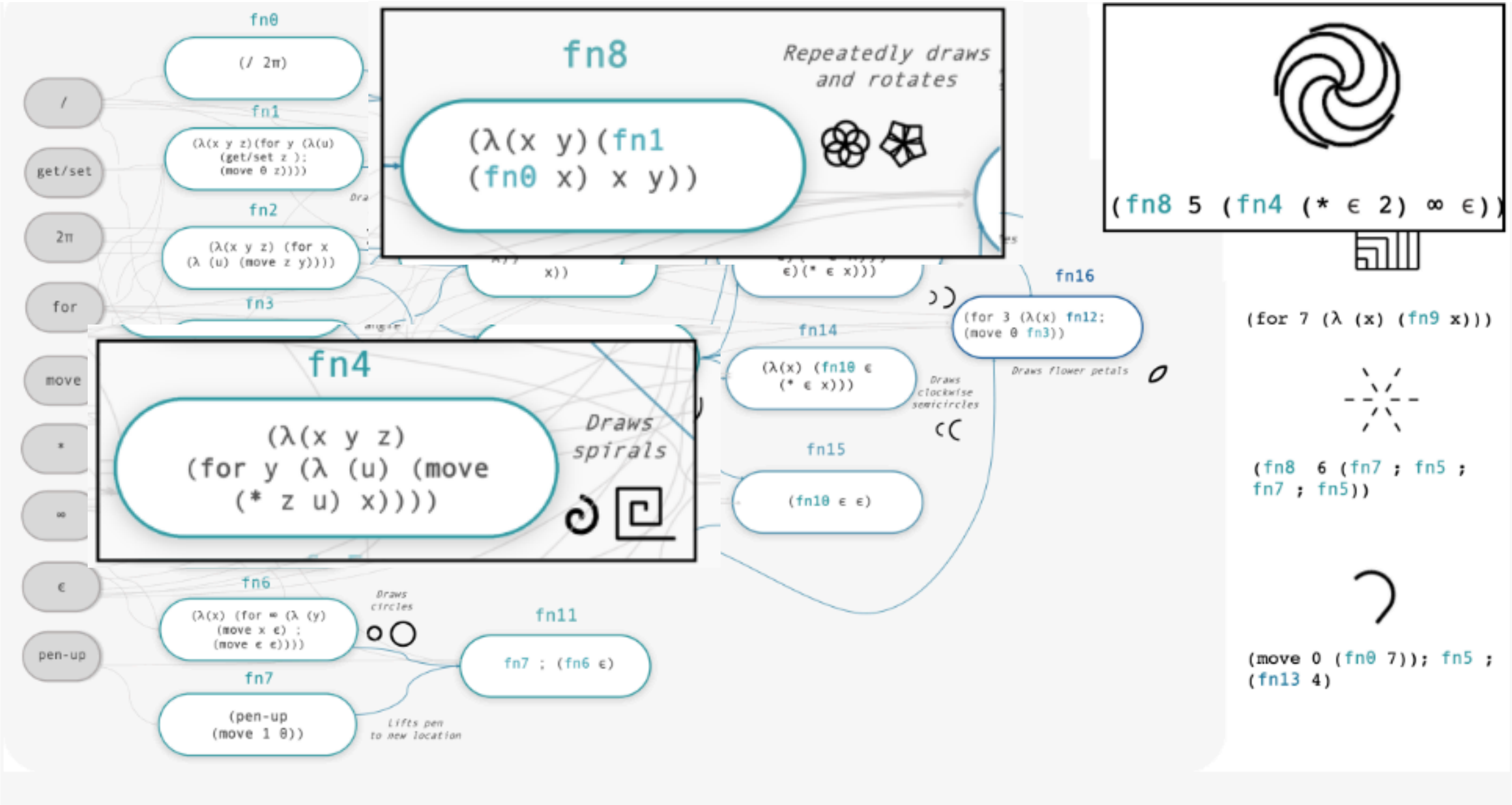
LOGO Turtle Graphics



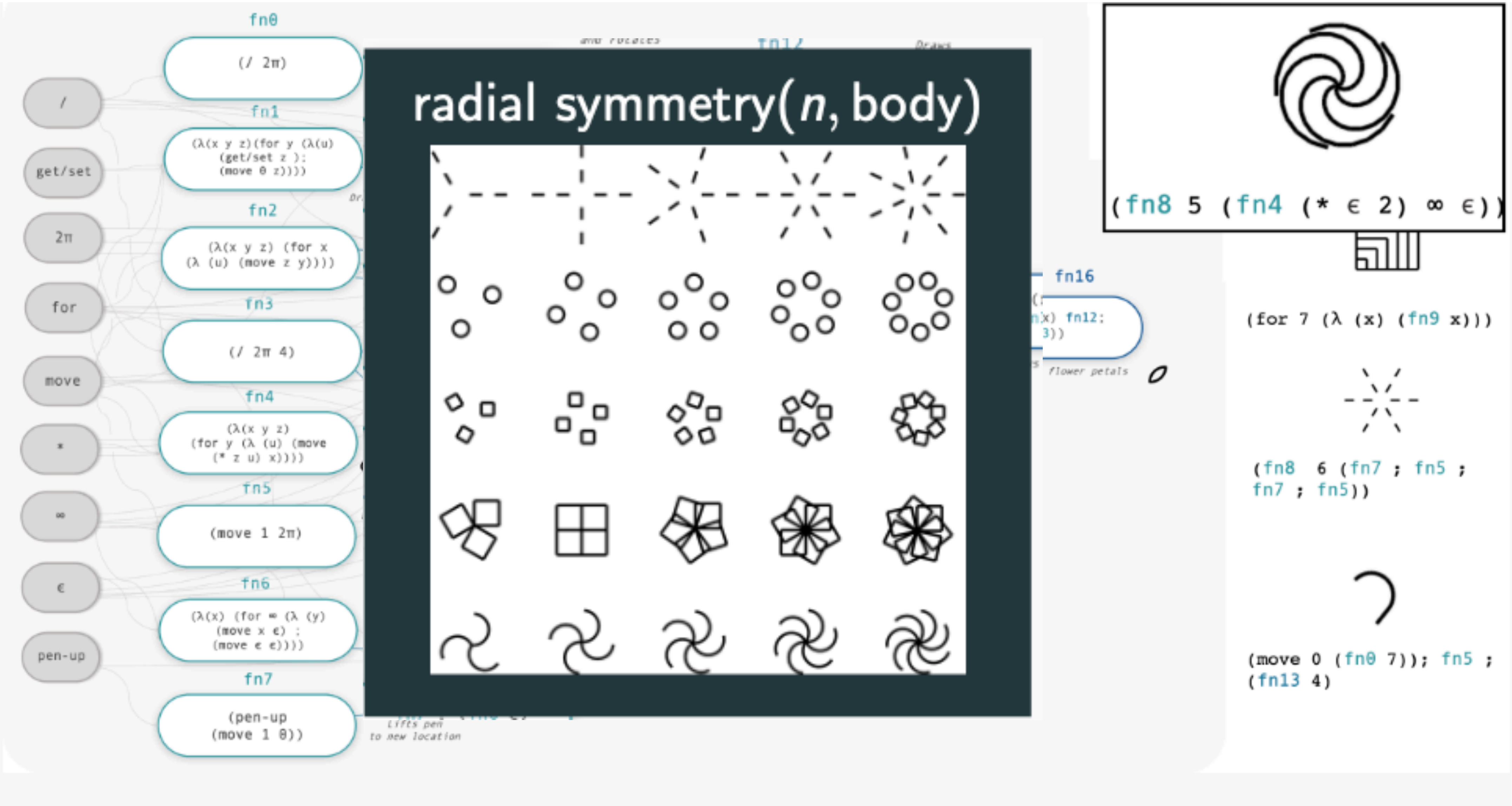
LOGO Turtle Graphics – learning an interpretable library



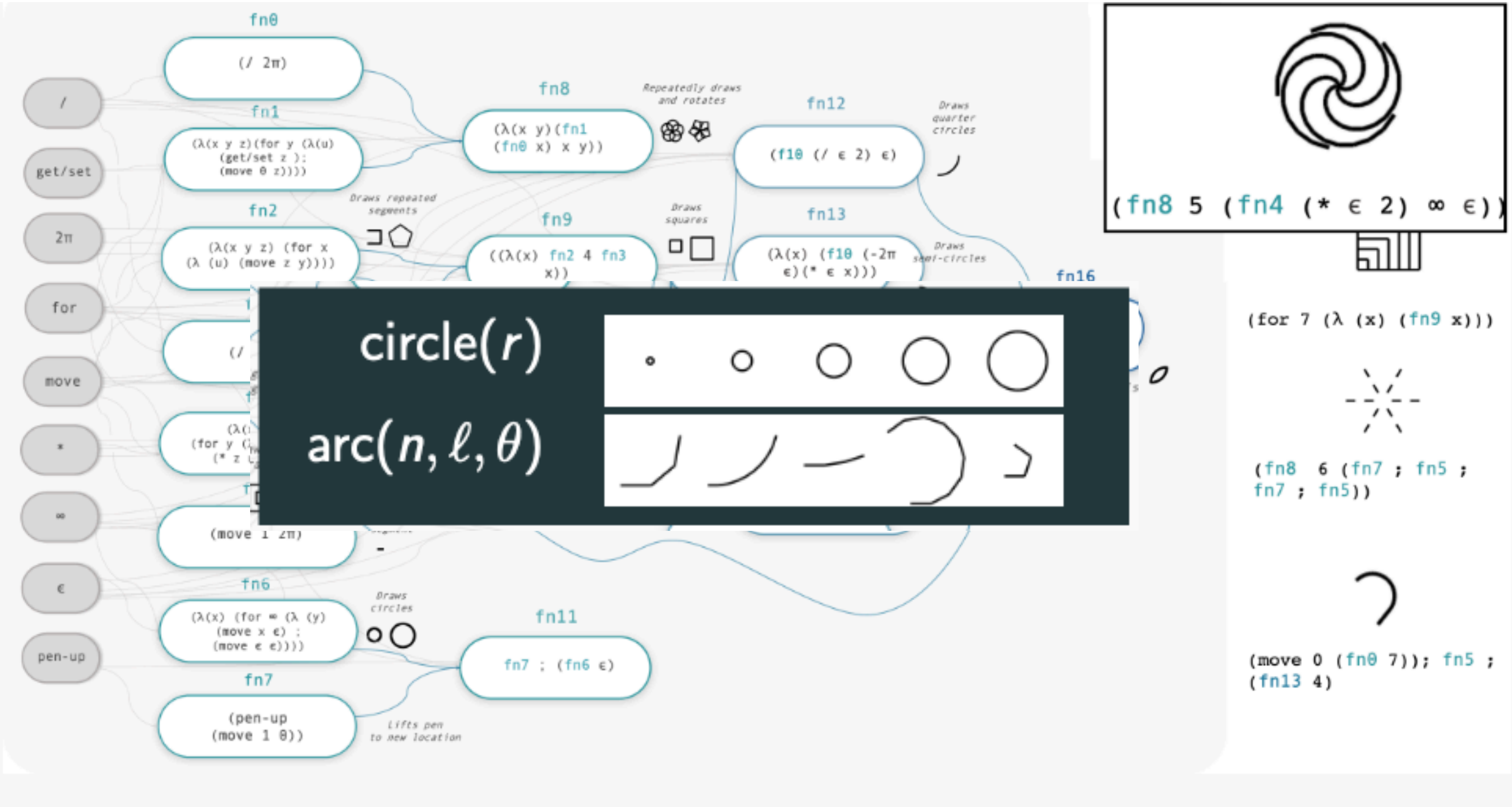
LOGO Turtle Graphics – learning an interpretable library



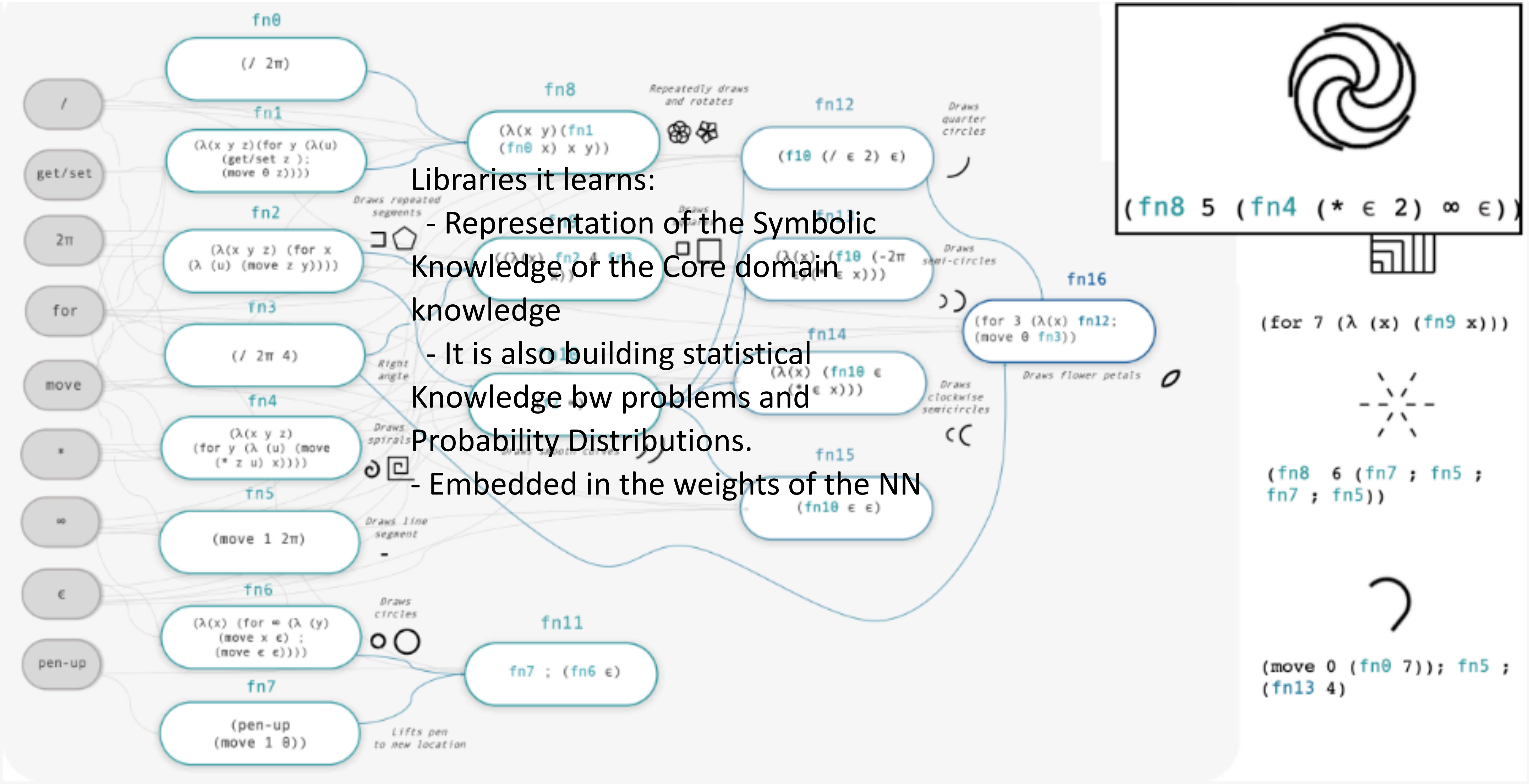
LOGO Turtle Graphics – learning an interpretable library



LOGO Turtle Graphics – learning an interpretable library



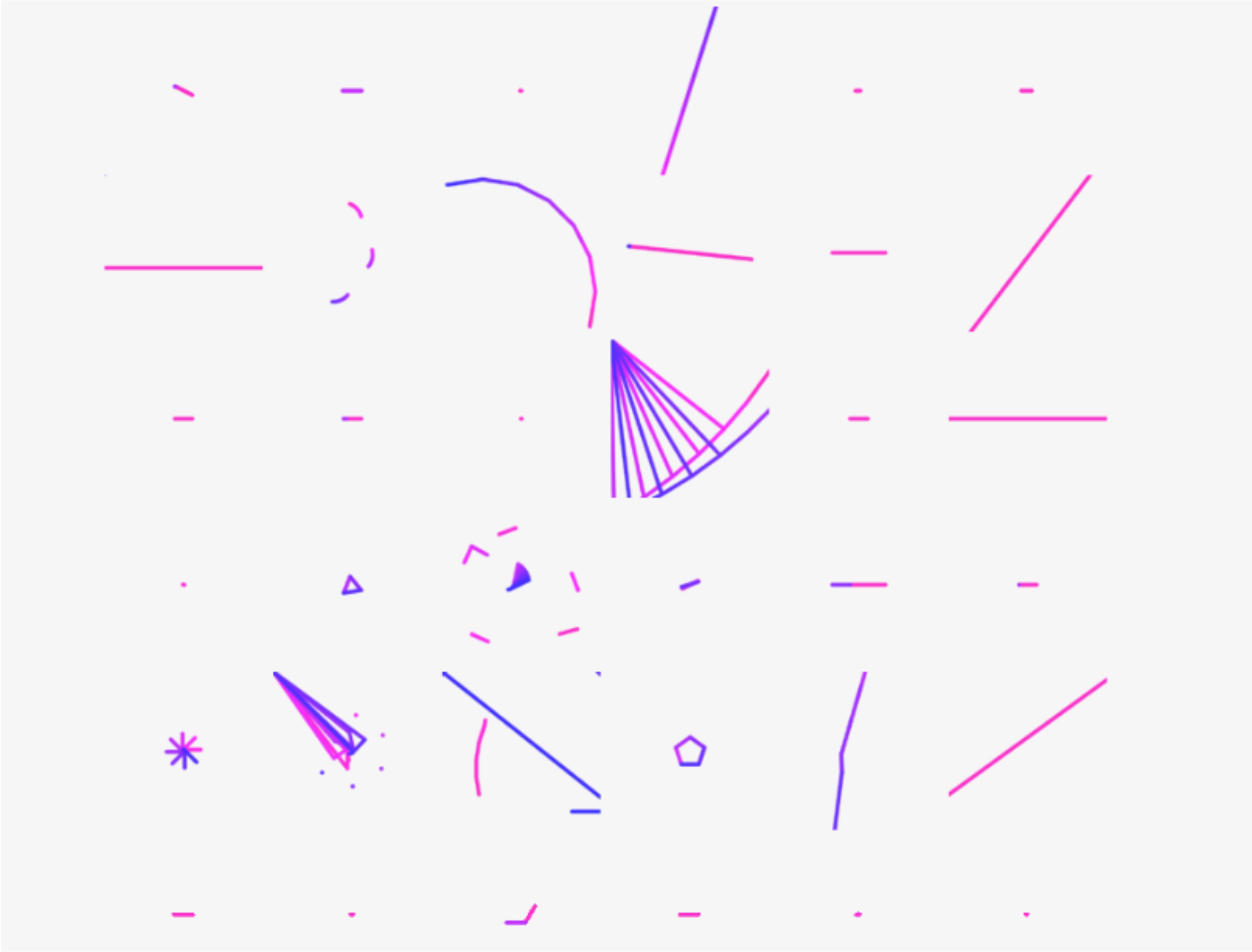
LOGO Turtle Graphics – learning an interpretable library



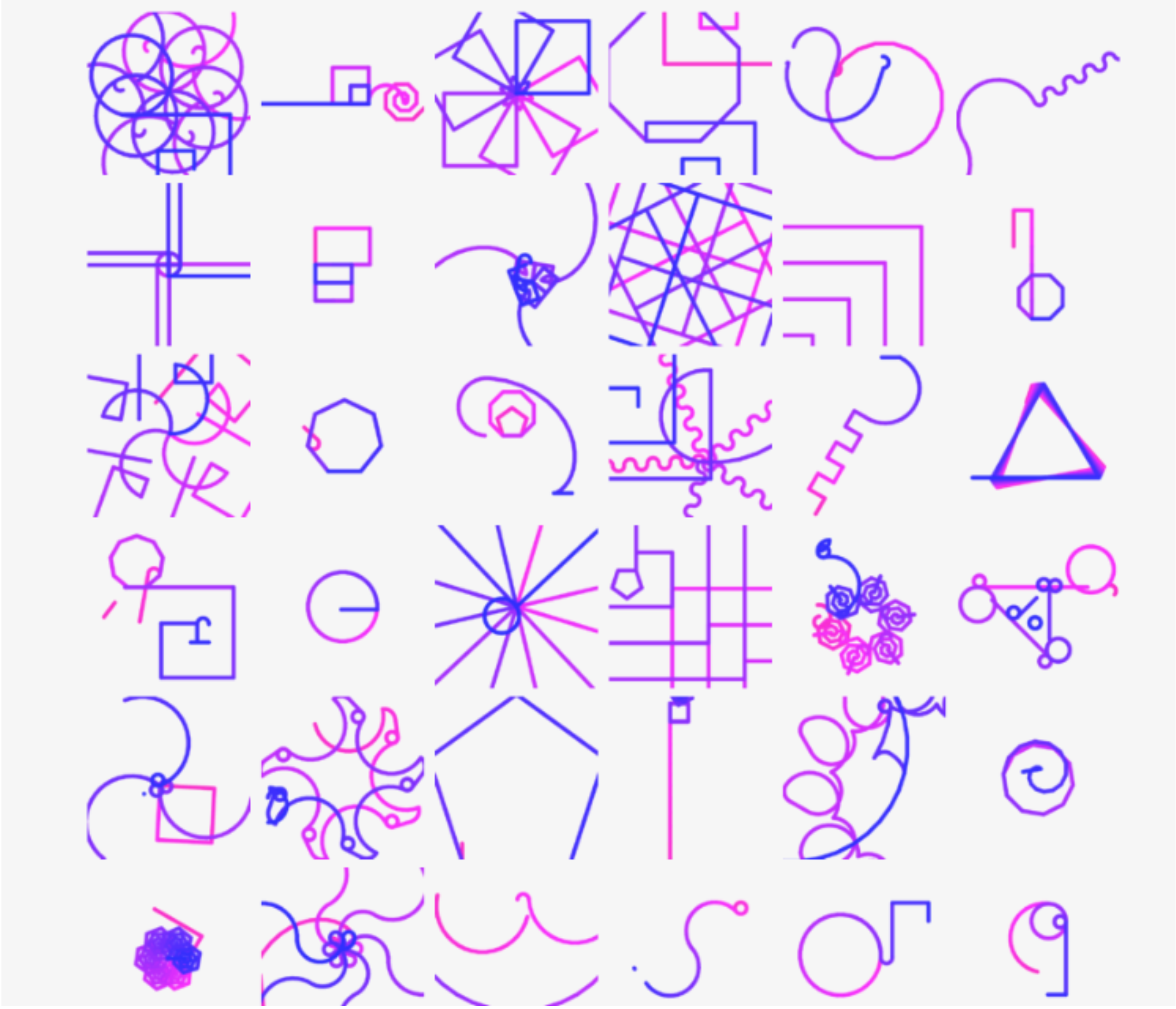
Libraries it learns:

- Representation of the Symbolic Knowledge or the Core domain knowledge
- It is also building statistical Knowledge bw problems and Probability Distributions.
- Embedded in the weights of the NN

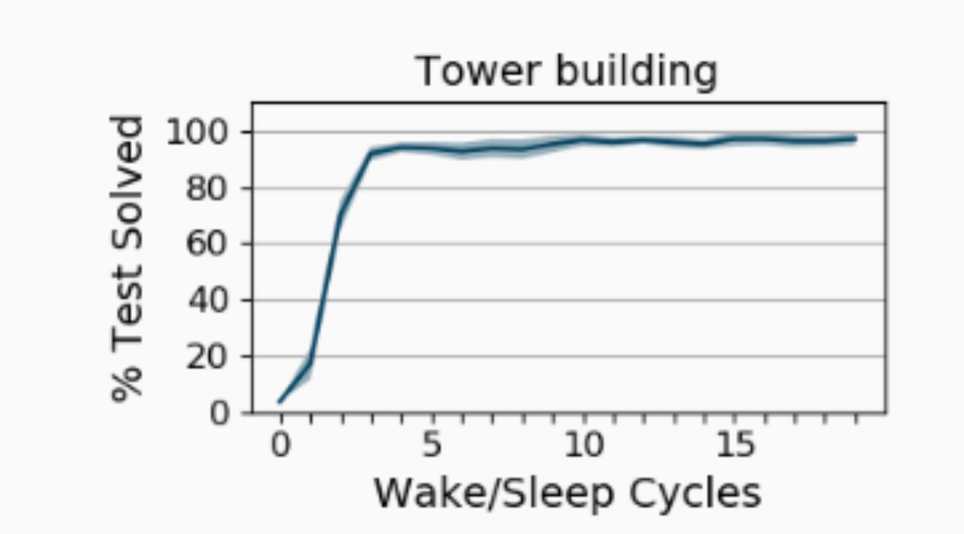
What does DreamCoder dream of? (before learning)



What does DreamCoder dream of? (after learning)

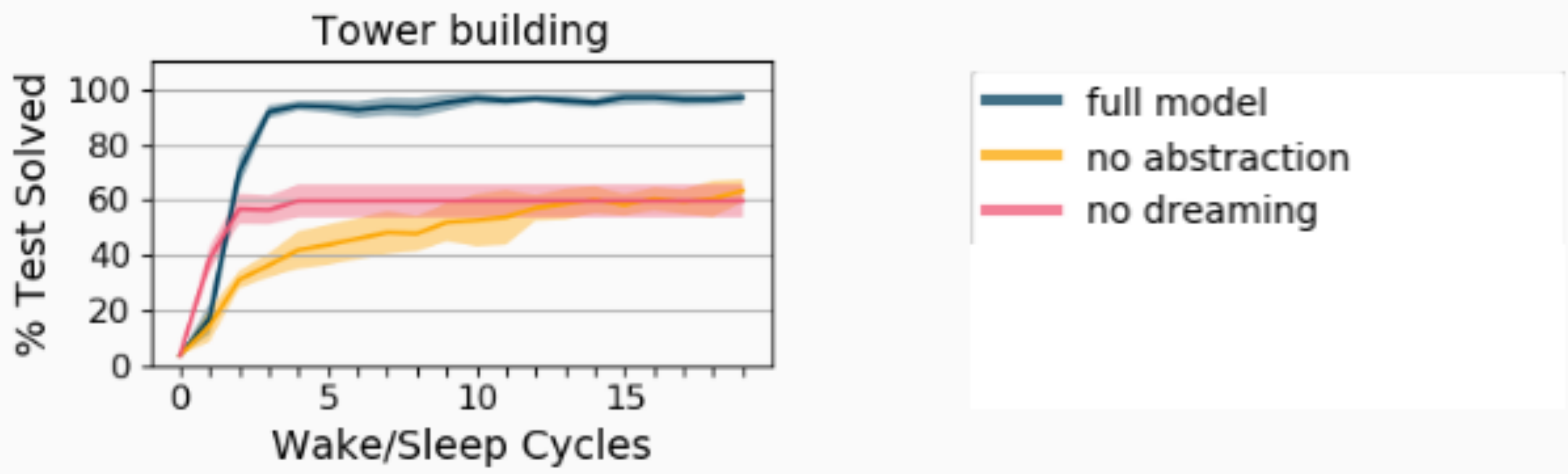


Learning dynamics

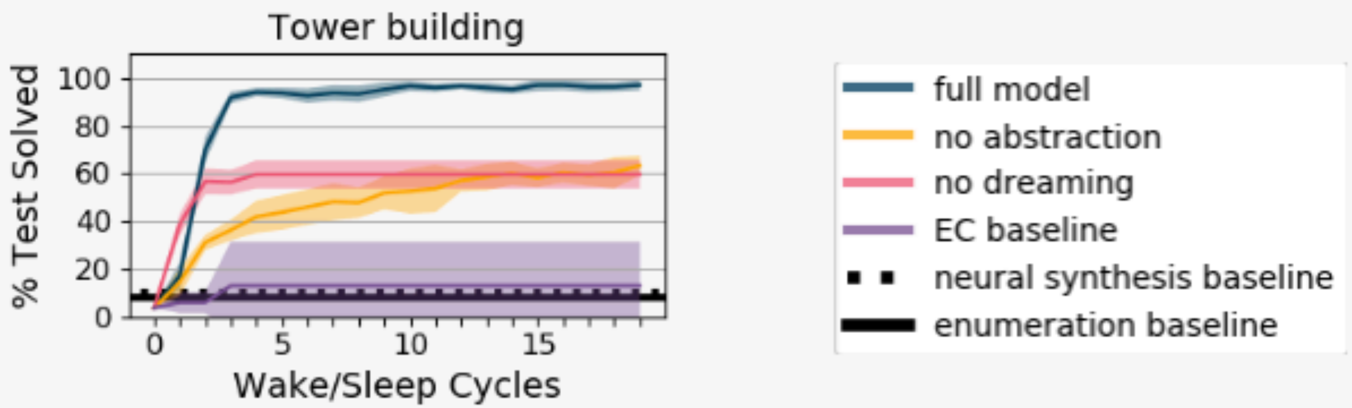


Bootstrapping action

Learning dynamics

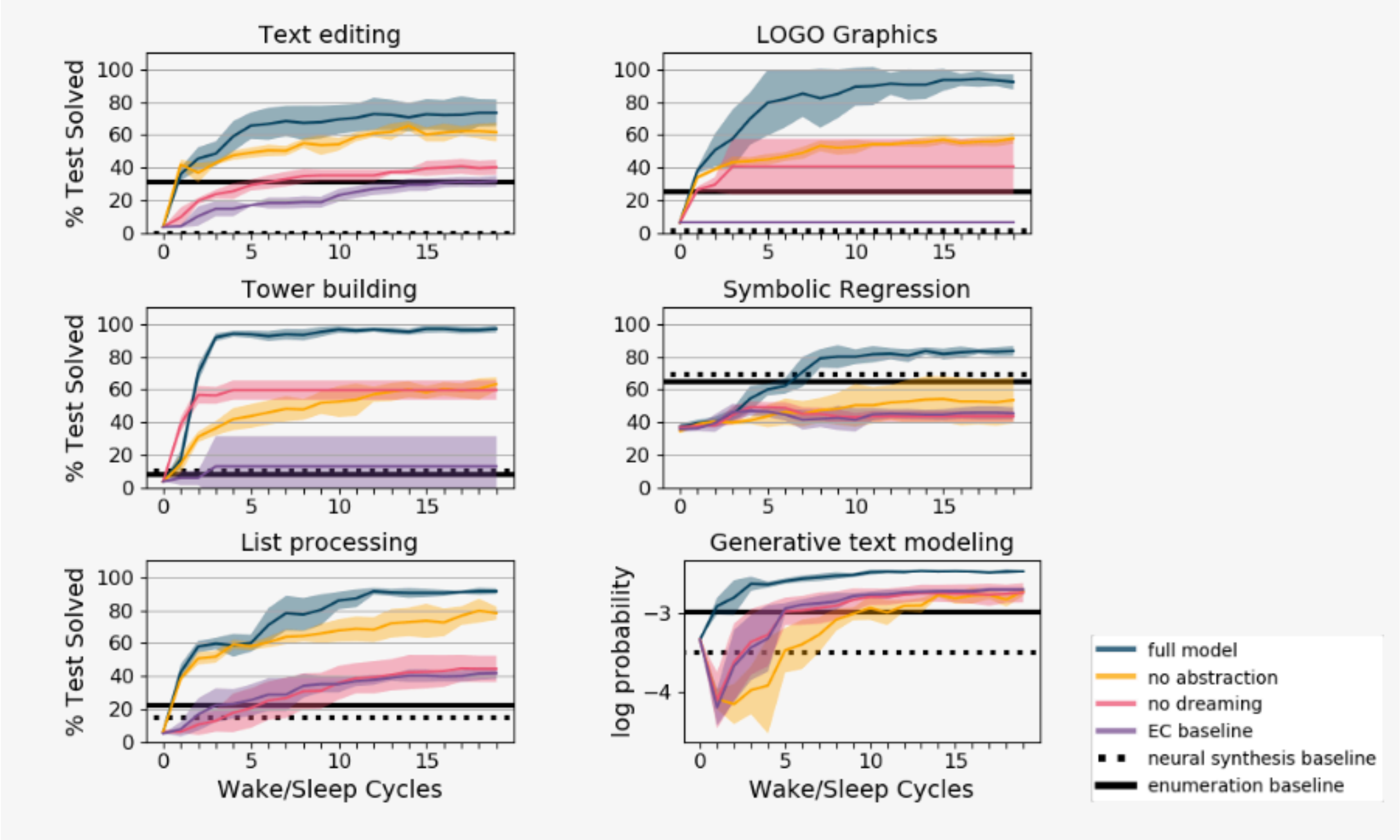


Learning dynamics



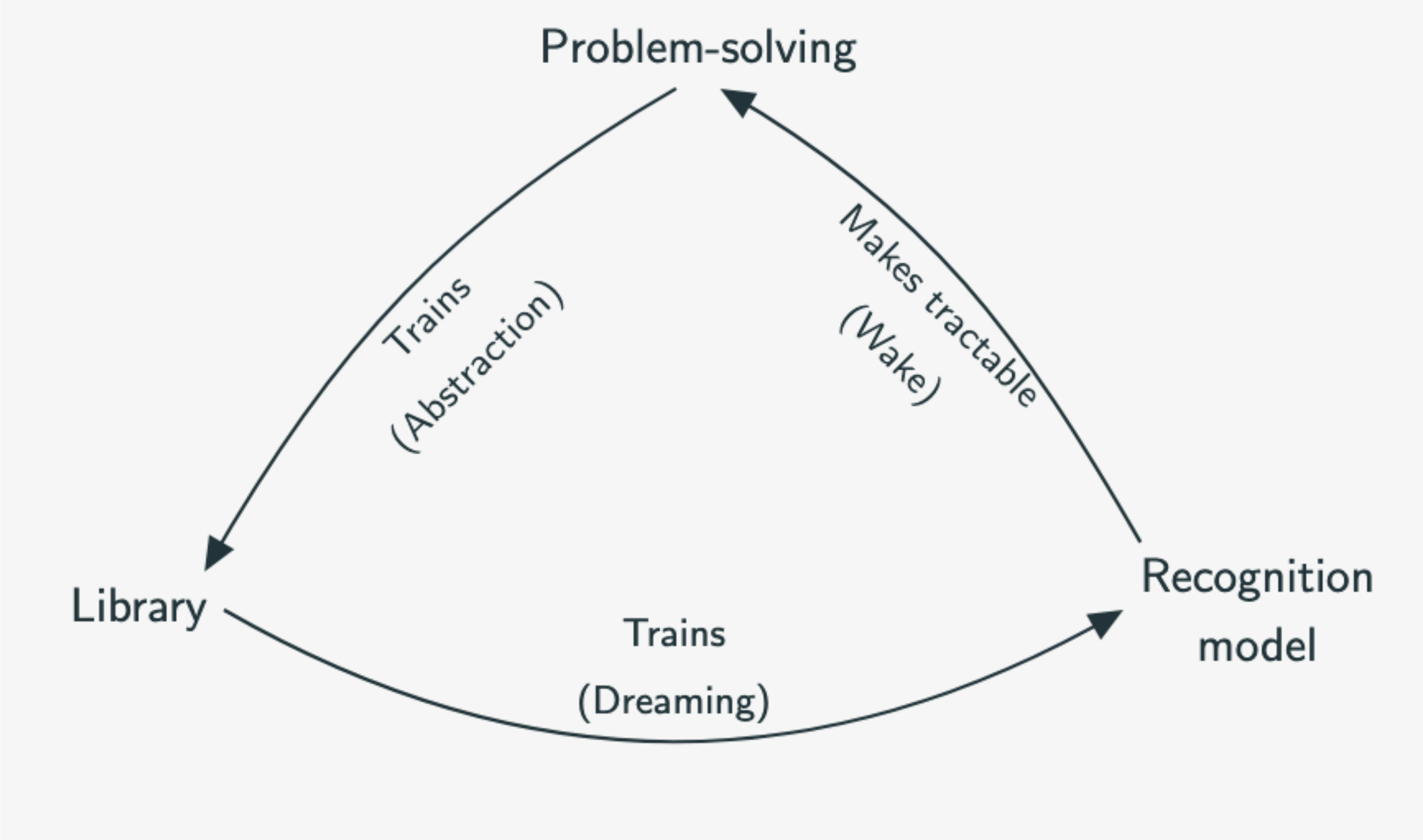
baselines: Exploration-Compression, EC [Dechter et al. 2013]
neural program synthesis, RobustFill [Devlin et al. 2017]
24 hours of brute-force enumeration

Learning dynamics



Synergy between recognition model and library learning

Synergy between recognition model and library learning



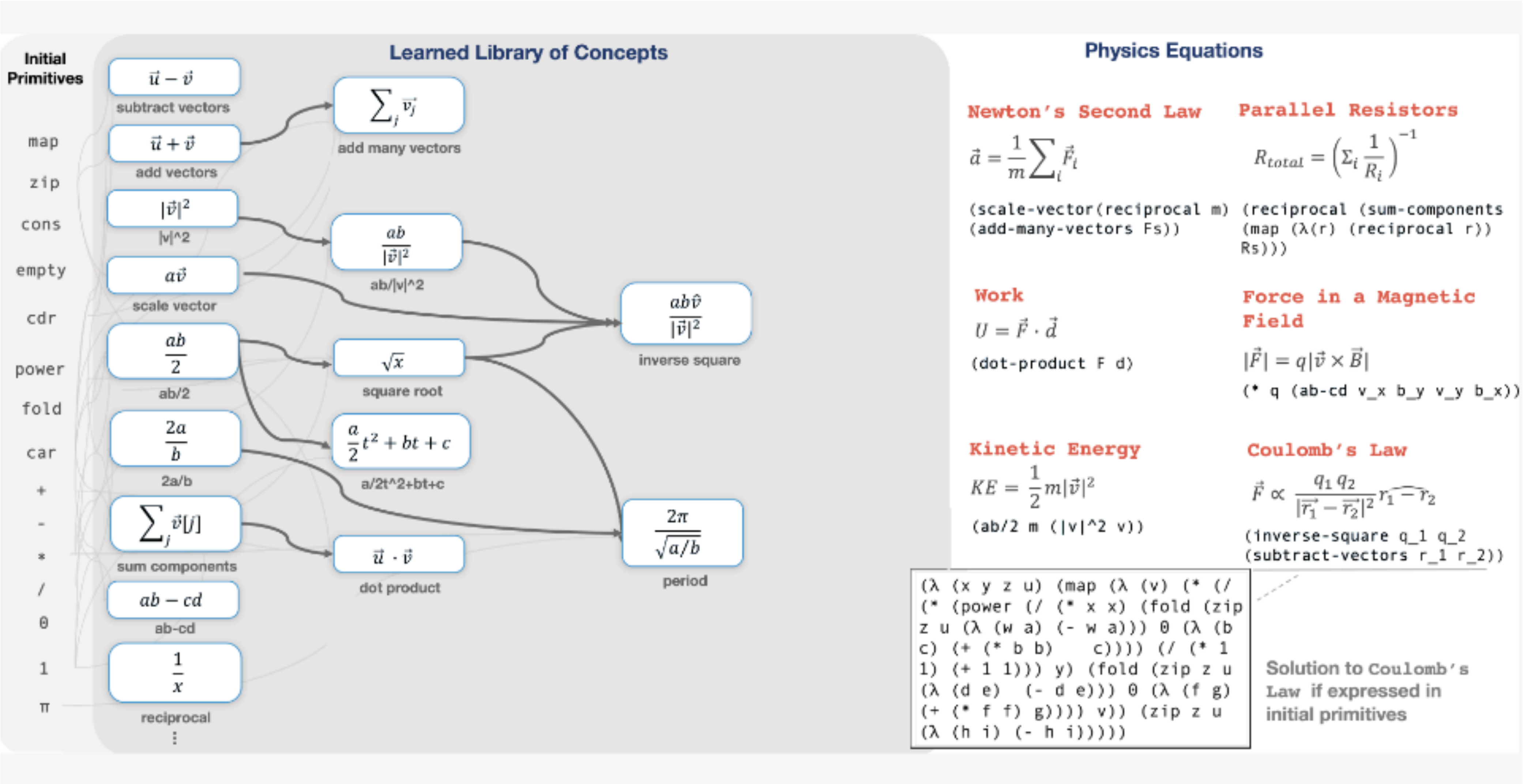
From learning libraries,
to learning languages

modern functional programming $\to$ physics

From learning libraries, to learning languages

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

Growing languages for vector algebra and physics



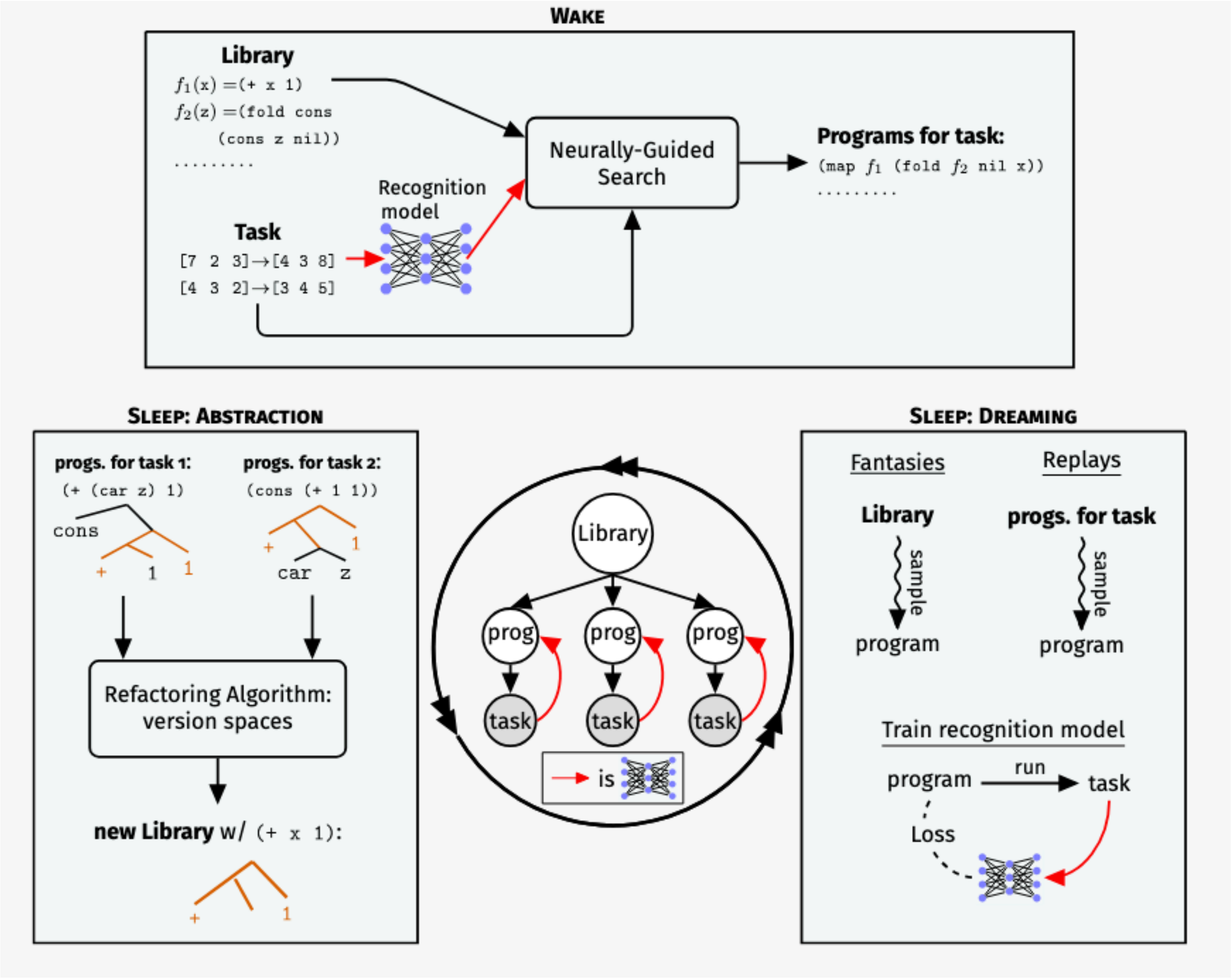
Lessons

Library learning interacts synergistically with neural synthesis: bootstrapping, more than sum of parts

Symbols aren't necessarily interpretable. Grow the language based on experience to make it more powerful *and* more human understandable

Learning-from-scratch is possible in principle. Don't do it. But program induction makes it convenient to build in what we know how to build in, and then learn on top of that

end.



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

Logistical

- Next week mid-progress meeting for the projects.
- Re-scheduling Fridays class.
- New-paper