Chair of Aerodynamics and Fluid Mechanics
Department of Mechanical Engineering
Technical University of Munich

Probabilistic Programming for Scientific Discovery

Lecture 1

Ludger Paehler

Lviv Data Science Summer School

July 29, 2020

Table of Contents

Course Outline

Example Applications of Probabilistic Programming

- ETALUMIS: Bringing Probabilistic Programming to Scientific Simulators at Scale
- DreamCoder: Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning

Why Do We even Need Probabilistic Programming?

Underlying Theoretical Ideas

Outline

Course Outline

Example Applications of Probabilistic Programming

- ETALUMIS: Bringing Probabilistic Programming to Scientific Simulators at Scale
- DreamCoder: Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning

Why Do We even Need Probabilistic Programming?

Underlying Theoretical Ideas

Course Outline

- 4 Lectures
    1. Foundational Knowledge
    2. Inference Engines & Introduction to Turing.jl
    3. Hierarchical Bayesian Approaches & Bayesian Deep Learning
    4. The Connection to Scientific Problems
- 3 Tutorials for Self-Paced Consumption
    1. In-Depth Introduction to Probabilistic Programming Systems with Turing.jl
    2. Bayesian Approaches in Probabilistic Programming
        - ▷ Bayesian Deep Learning
        - ▷ Hierarchical Bayesian Modelling
    3. Machine-Learning Based Design with Probabilistic Programming

Lecture 1

- Example Applications of Probabilistic Programming
  1. *ETALUMIS: Bringing Probabilistic Programming to Scientific Simulators at Scale*
  2. *DreamCoder: Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning*
- Why do we even need Probabilistic Programming?
- Underlying Theoretical Ideas

Lecture 2

- Approaches to Inference - the Inference Engine
- Probabilistic Programming Frameworks
- Practical Introduction to a Probabilistic Programming Framework
- Extending the ideas to a more complex example

Lecture 3

- Bayesian Hierarchical Approaches
- Bayesian Deep Learning, including but not limited to
  - Inference Networks
  - Uncertainty Quantification
- Marrying Deep Learning Frameworks with Probabilistic Programming for Type 2 Machine Learning

Lecture 4

- Interaction with Scientific Simulators
  - What types of simulators would I want to link to?
  - What are the hidden pitfalls?
- Areas of application
  - Robotics
  - Physics
  - Engineering
  - Machine-Learning Based Design

Outline

Course Outline

Example Applications of Probabilistic Programming

- ETALUMIS: Bringing Probabilistic Programming to Scientific Simulators at Scale
- DreamCoder: Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning

Why Do We even Need Probabilistic Programming?

Underlying Theoretical Ideas

Etalumis

Bringing Probabilistic Programming to Scientific Simulators at Scale [2]

- Large scale inverse problem, where a particle simulator is inverted by probabilistically inferring all choices in the simulator given the desired outputs
  - Developed in the context of particle simulations at CERN
- First large-scale application of probabilistic programming to physical simulators in the quest to potentially unearth new physics [1]
- Largest-scale posterior inference with 25000 latent variables at the time
- Amount of compute required highly dependent on the specific approach to inference and the nature of the simulator, i.e. latent dimensionality and intensity of the compute routine
  - Only set to improve with the impeding exascale-era

[1] Cranmer, K., Brehmer, J. and Louppe, G., 2020. The frontier of simulation-based inference. Proceedings of the National Academy of Sciences.

[2] Baydin, A.G., Shao, L., Bhimji, W., Heinrich, L., Meadows, L., Liu, J., Munk, A., Naderiparizi, S., Gram-Hansen, B., Louppe, G. and Ma, M., 2019, November. Etalumis: Bringing probabilistic programming to scientific simulators at scale. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-24).

Etalumis

- Proposes a direct linking of particle physics simulators with probabilistic programming systems to trace the internal structure of the simulator
  - Probabilistic programming system controls the random number draws of the simulator akin to samples from prior distributions in Bayesian statistics
- Utilizing inference compilation with three-dimensional convolutional LSTMs to guide the inference procedure and amortize the high computational costs of training
  - Dynamic compilation, in which a core gets expanded with further neural network components as inference compilation proceeds
- Utilizes importance sampling in conjunction with inference compilation as approach to inference

Etalumis

Simulators as Probabilistic Programs

- A simulator execution is viewed as an execution trace, a single sample to the probabilistic programming system
    - I.e. sampling is taking place in the space of execution traces
- Abstracting the simulator in this way enables the following analysis
    - Compute likelihoods
    - Learn/construct surrogate models
    - Generate training data for inference compilation
    - Introduce other generative approaches into the loop
- Enables us to guide the simulation in an intelligent fashion, using the inference network, which acts as a kind of oracle

Etalumis

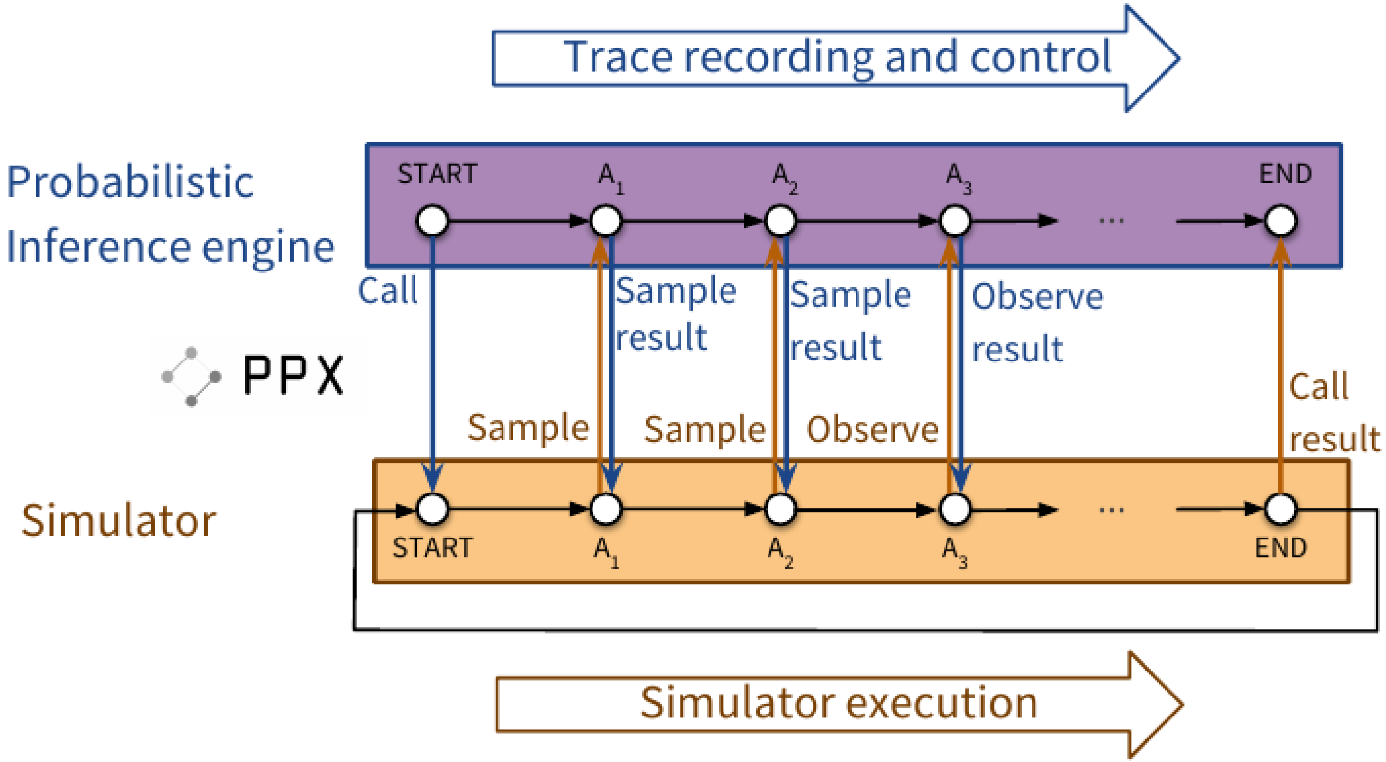


Figure: High-level view from the perspective of the probabilistic programming execution protocol (PPX).

Etalumis

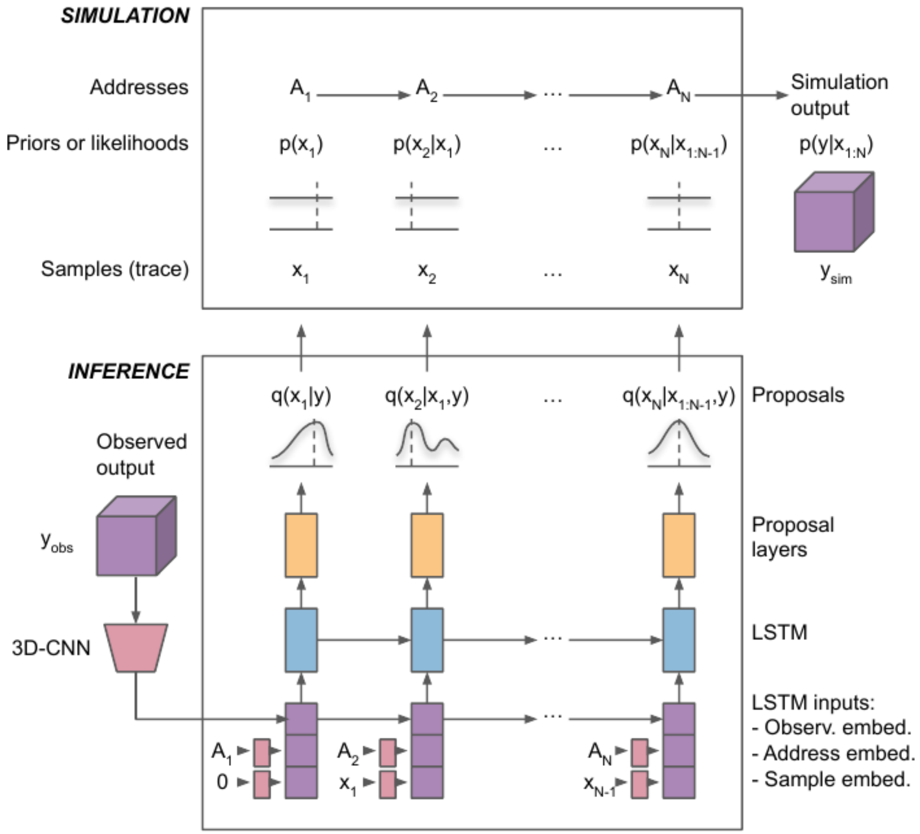


Figure: Detailed connection between simulation and inference in the probabilistic programming-based approach.

Etalumis

Recap: Amortized Inference [3] [4]

- Blub 1

[3] Gershman, S. and Goodman, N., 2014. Amortized inference in probabilistic reasoning. In Proceedings of the annual meeting of the cognitive science society (Vol. 36, No. 36).

[4] Ritchie, D., Horsfall, P. and Goodman, N.D., 2016. Deep amortized inference for probabilistic programs. arXiv preprint arXiv:1610.05735.

Etalumis

Recap: Amortized Inference

- Blub 2

Etalumis

Recap: Amortized Inference

- Blub 3

Etalumis

Outlook: Inference Compilation [5]

- Uses neural networks to construct a surrogate model for the probabilistic generative model, which is subsequently used at inference time as a custom proposal distribution to avoid sampling from the actual generative model
- Intuition is that the cost of constructing the surrogate can be amortized at inference time and be lower inference from the underlying generative model
  - Need to watch out for sample diversity and out-of-distribution samples
- Proposes adaptive neural network architecture with a recurrent core and embedding and proposal layers specified by the probabilistic program
- Approach is model-agnostic

[5] Le, T.A., Baydin, A.G. and Wood, F., 2017, April. Inference compilation and universal probabilistic programming. In Artificial Intelligence and Statistics (pp. 1338-1348).

Etalumis

Outlook: Inference Compilation

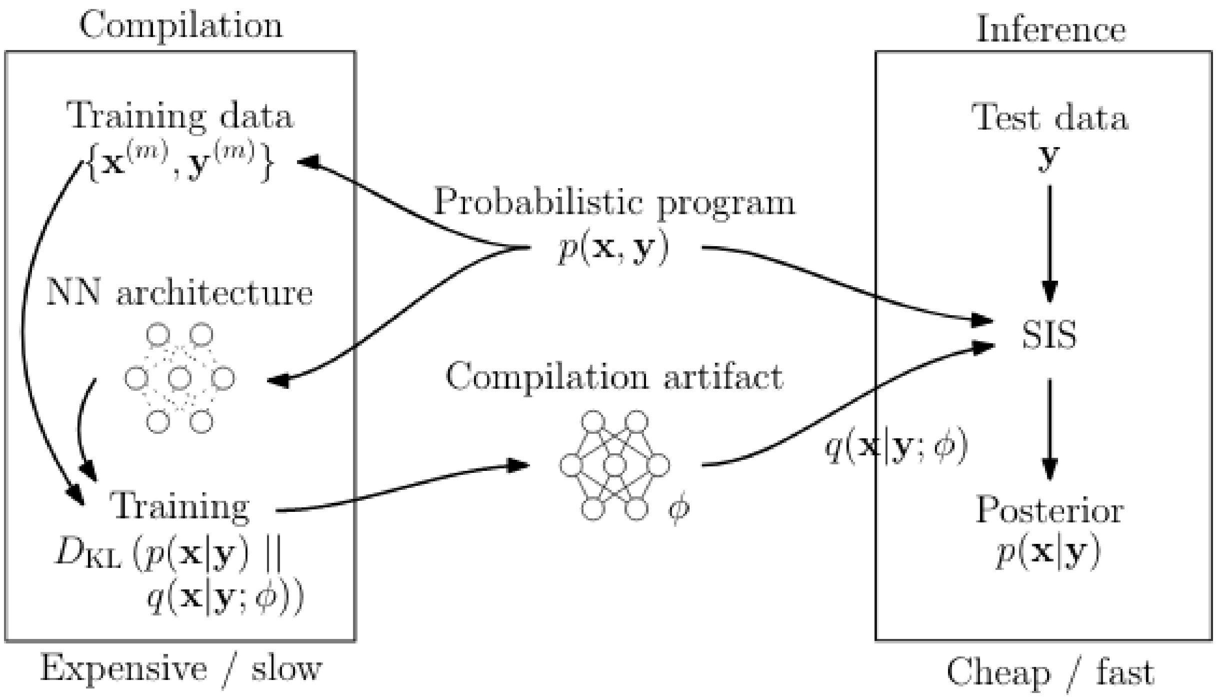

Compilation
Training data
$\{\mathbf{x}^{(m)}, \mathbf{y}^{(m)}\}$
NN architecture
Training
$D_{\mathrm{KL}}\left(p(\mathbf{x}|\mathbf{y}) \,||\, q(\mathbf{x}|\mathbf{y};\phi)\right)$
Expensive / slow
Probabilistic program
$p(\mathbf{x},\mathbf{y})$
Compilation artifact
ϕ
$q(\mathbf{x}|\mathbf{y};\phi)$
Inference
Test data
$\mathbf{y}$
SIS
Posterior
$p(\mathbf{x}|\mathbf{y})$
Cheap / fast

Figure: Automatic construction of a neural network surrogate, which is then trained with data generated by the probabilistic program to eventually act as the proposal distribution at inference time.

Etalumis

Outlook: Inference Compilation

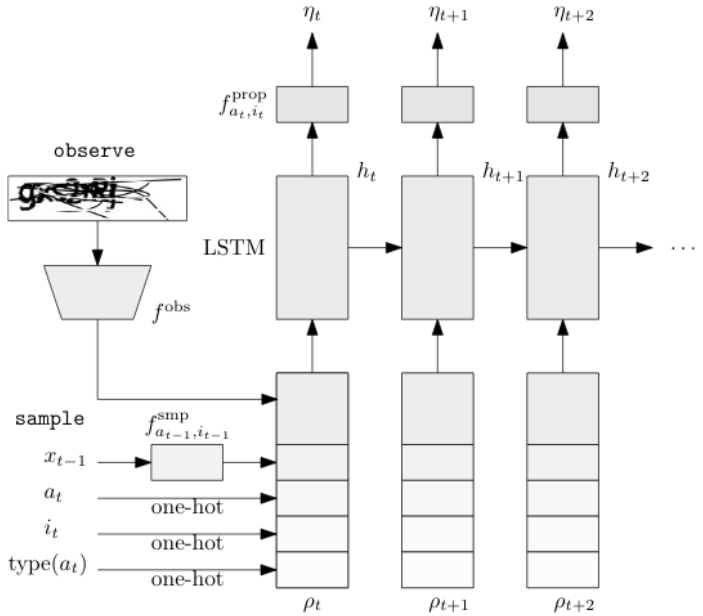
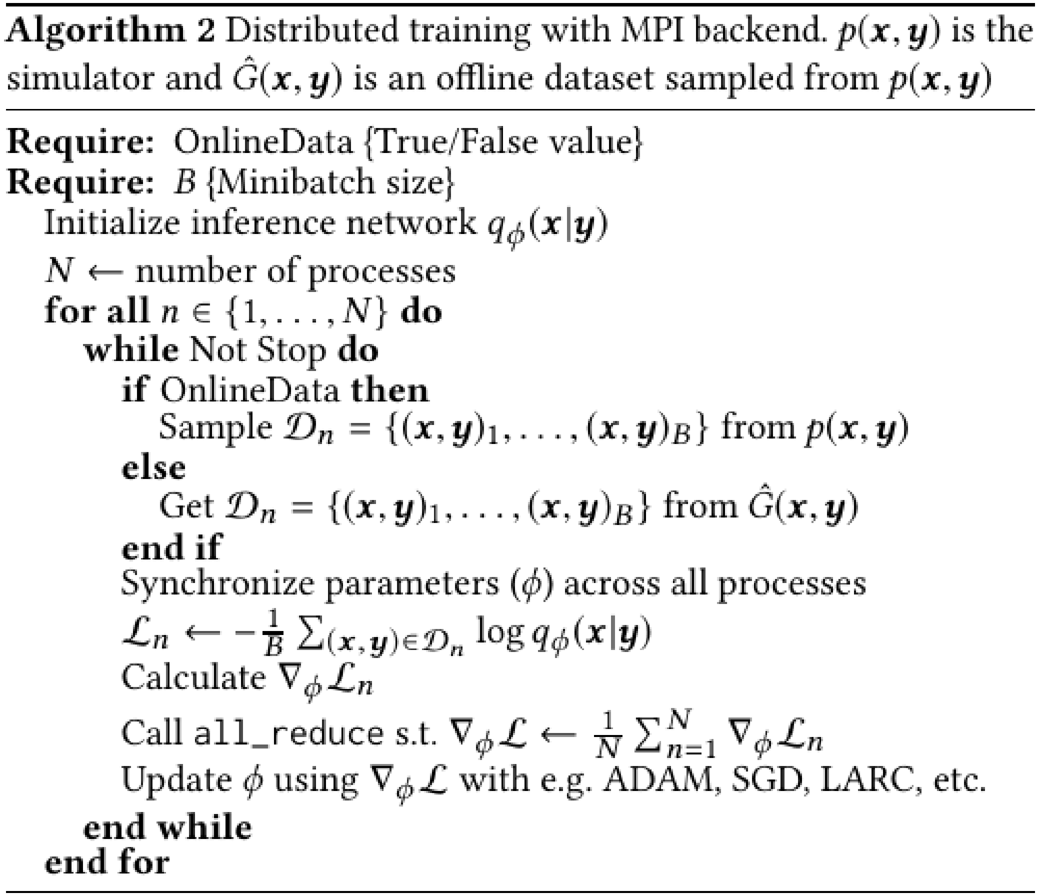


Figure: Example application for captcha solving based on probabilistic generative models for the captchas. With the LSTM at its core, required embeddings for the respective are attached adaptively.

Etalumis



Algorithm 2 Distributed training with MPI backend. $p(\boldsymbol{x}, \boldsymbol{y})$ is the simulator and $\hat{G}(\boldsymbol{x}, \boldsymbol{y})$ is an offline dataset sampled from $p(\boldsymbol{x}, \boldsymbol{y})$

```
Require: OnlineData {True/False value}
Require: $B$ {Minibatch size}
  Initialize inference network $q_\phi(\boldsymbol{x}|\boldsymbol{y})$
  $N \leftarrow$ number of processes
  for all $n \in \{1, \ldots, N\}$ do
    while Not Stop do
      if OnlineData then
        Sample $\mathcal{D}_n = \{(\boldsymbol{x}, \boldsymbol{y})_1, \ldots, (\boldsymbol{x}, \boldsymbol{y})_B\}$ from $p(\boldsymbol{x}, \boldsymbol{y})$
      else
        Get $\mathcal{D}_n = \{(\boldsymbol{x}, \boldsymbol{y})_1, \ldots, (\boldsymbol{x}, \boldsymbol{y})_B\}$ from $\hat{G}(\boldsymbol{x}, \boldsymbol{y})$
      end if
      Synchronize parameters ($\phi$) across all processes
      $\mathcal{L}_n \leftarrow -\frac{1}{B} \sum_{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}_n} \log q_\phi(\boldsymbol{x}|\boldsymbol{y})$
      Calculate $\nabla_\phi \mathcal{L}_n$
      Call all_reduce s.t. $\nabla_\phi \mathcal{L} \leftarrow \frac{1}{N} \sum_{n=1}^{N} \nabla_\phi \mathcal{L}_n$
      Update $\phi$ using $\nabla_\phi \mathcal{L}$ with e.g. ADAM, SGD, LARC, etc.
    end while
  end for
```

Etalumis

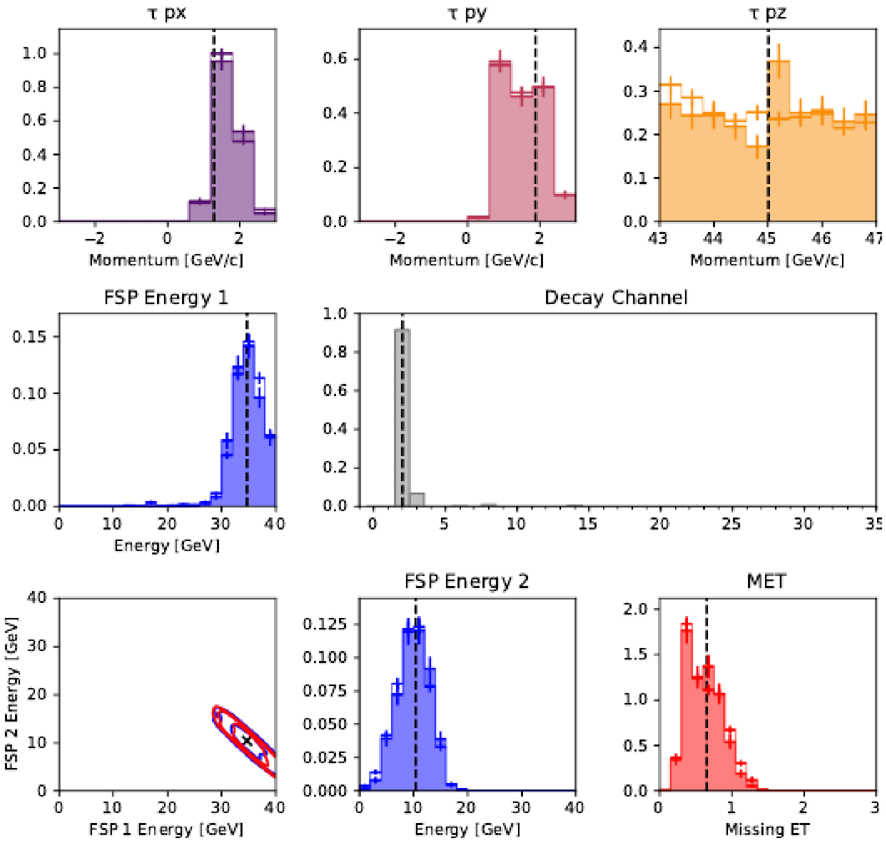
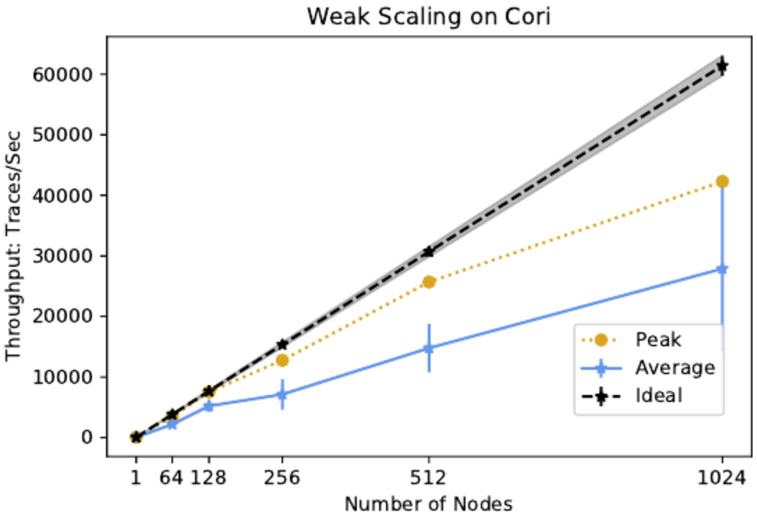
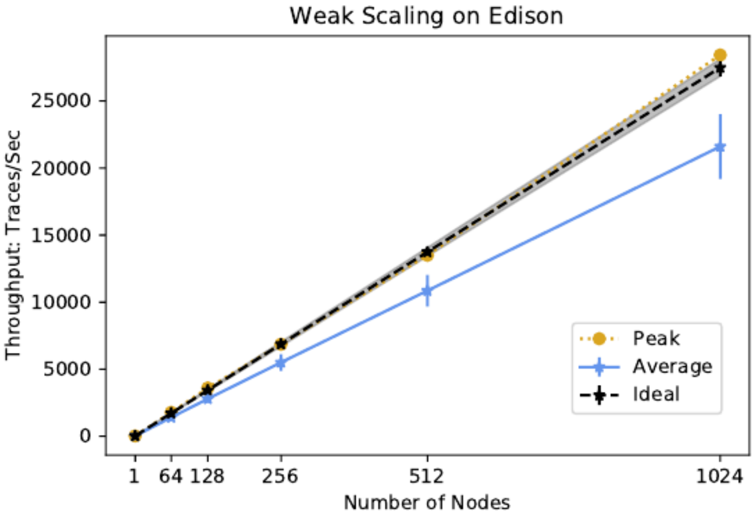


Figure: Posteriors obtained with random-walk Metropolis Hasting (filled histograms) and inference compilation (outline histograms) and ground truth values (dashed vertical lines).

Etalumis





- Moving to supercomputer scale for massive-scale inference to be able to generate the necessary number of simulations for the inference compilation to be successful.

Etalumis

- First probabilistic programming system to link to scientific simulators at scale to enable large-scale posterior inference on a supercomputing-scale
  - Can only ever capture the processes encapsulated in the simulator
- Introduces a probabilistic programming execution protocol to link to scientific simulators
- To make inference tractable it needs to rely on techniques, such as amortization through inference compilation, which essentially constructs an oracle
- Pushes the frameworks to the extreme with communication requirements and data exchange between computing instances

DreamCoder [6]

Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning

- Constructs domain-specific languages (DSLs) for scientific problems combined with a neural network, which embodies a learned domain-specific search strategy
  - Learns both the system prior and the needed inference algorithm
- Practically constructs a library of symbolic abstractions in a wake-sleep manner and applies said library to the solving of the chosen problem at hand
- Wake-sleep learning
  - During *sleep* the system consolidates its abstractions from the programs found during *wake* and improves upon the neural network recognition model by imagining new samples
  - During *wake* the generative model is exploited on the problem domain to find the programs with the highest posterior probability

[6] Ellis, K., Wong, C., Nye, M., Sable-Meyer, M., Cary, L., Morales, L., Hewitt, L., Solar-Lezama, A. and Tenenbaum, J.B., 2020. DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning. arXiv preprint arXiv:2006.08381.

DreamCoder

- Knowledge is accumulated in a multilayered hierarchy with knowledge and skills being successively learned over time, i.e. the knowledge is bootstrapped from very simple examples to ever more complex cases
- Can be broken down to a probabilistic inference procedure, i.e. observing task X and inferring program ρ_x to solve task $x \in X$ combined with a prior distribution over program, which migh solve tasks in the domain

$$\rho_x = \underset{\substack{\rho: \\ Q(\rho|x) \text{ is large}}}{\arg\max}\; P[\rho|x,L] \propto P[x|\rho]P[\rho|L], \text{ for each task } x \in X \qquad \textit{Wake}$$

$$L = \underset{L}{\arg\max} P[L] \prod_{x \in X} \max_{\rho \text{ a refactoring of } \rho_x} P[x|\rho]P[\rho|L] \qquad \textit{Sleep : Abstraction}$$

$$\text{Train } Q(\rho|x) \approx P[\rho|x,L], \text{ where } x \sim X \text{ ('replay') or } x \sim L \text{ ('fantasy')} \qquad \textit{Sleep : Dreaming}$$

DreamCoder

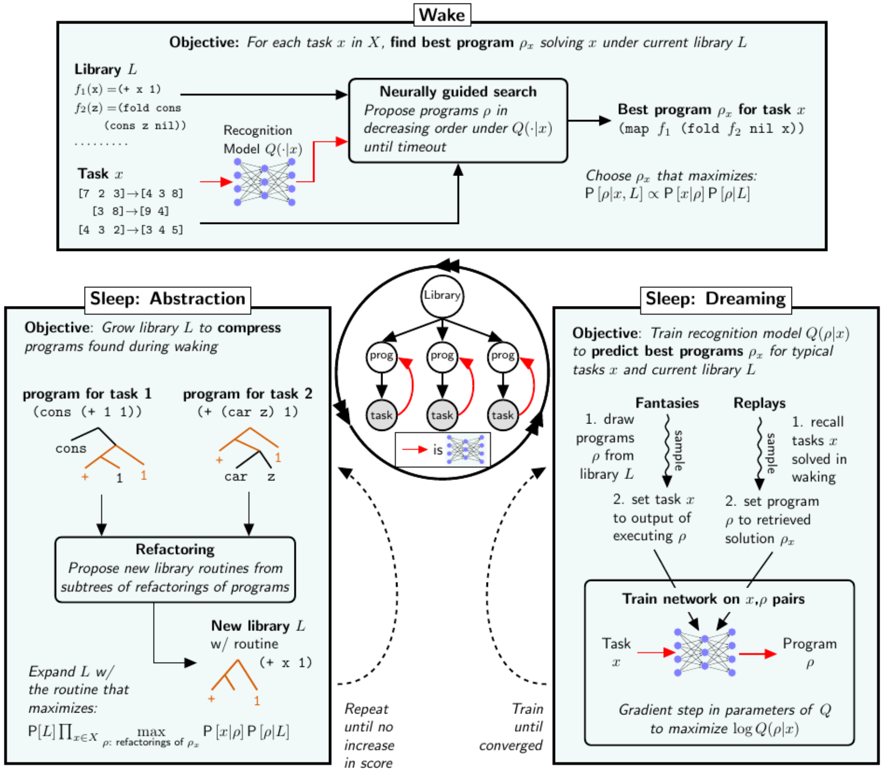
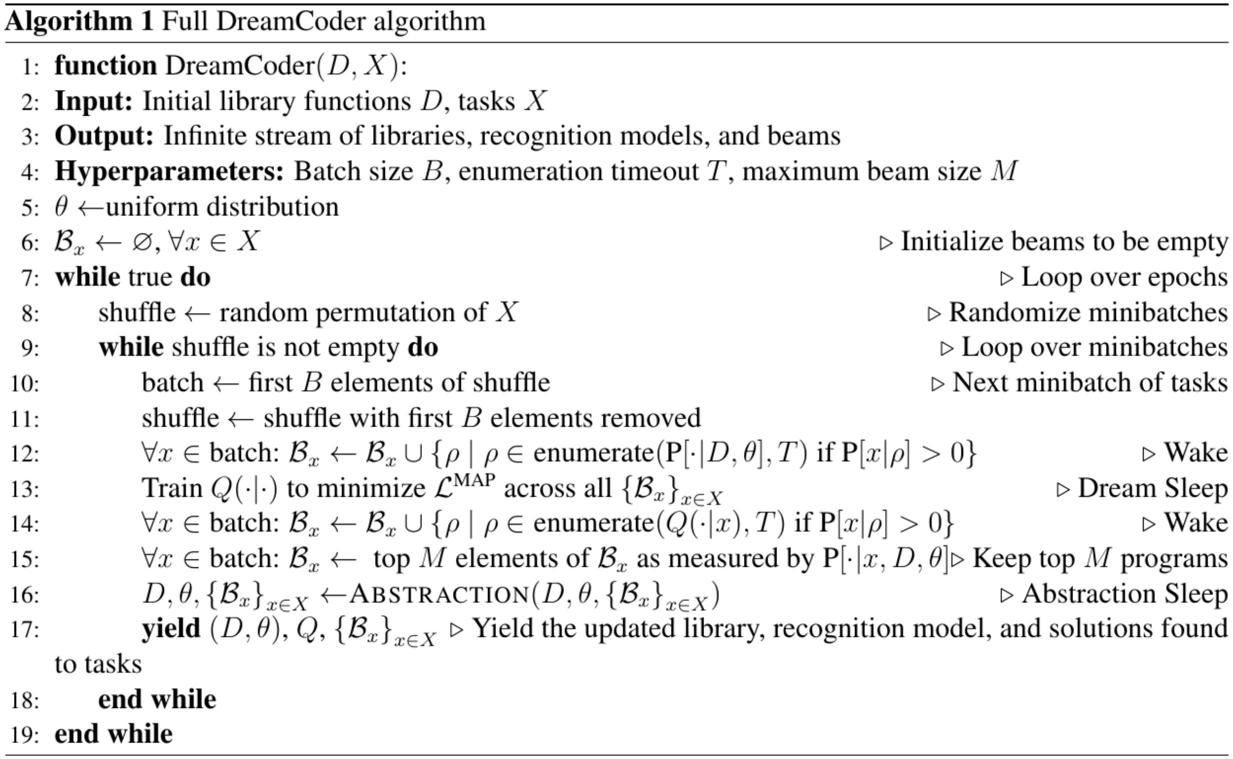

Wake
Objective: For each task x in X, find best program ρ_x solving x under current library L
Library L
$f_1(x) =$(+ x 1)
$f_2(z) =$(fold cons (cons z nil))
....
Task x
[7 2 3]→[4 3 8]
[3 8]→[9 4]
[4 3 2]→[3 4 5]
Recognition Model $Q(\cdot|x)$
Neurally guided search
Propose programs ρ in decreasing order under $Q(\cdot|x)$ until timeout
Best program ρ_x for task x
(map f_1 (fold f_2 nil x))
Choose ρ_x that maximizes:
$P[\rho|x, L] \propto P[x|\rho] P[\rho|L]$
Library
prog
prog
prog
task
task
task
is
Sleep: Abstraction
Objective: Grow library L to compress programs found during waking
program for task 1
(cons (+ 1 1))
cons
+
1
1
program for task 2
(+ (car z) 1)
+
1
car
z
Refactoring
Propose new library routines from subtrees of refactorings of programs
New library L
w/ routine
(+ x 1)
+
1
Expand L w/ the routine that maximizes:
$P[L] \prod_{x \in X} \max_{\rho:\text{ refactorings of }\rho_x} P[x|\rho] P[\rho|L]$
Repeat until no increase in score
Train until converged
Sleep: Dreaming
Objective: Train recognition model $Q(\rho|x)$ to predict best programs ρ_x for typical tasks x and current library L
Fantasies
1. draw programs ρ from library L
sample
2. set task x to output of executing ρ
Replays
1. recall tasks x solved in waking
sample
2. set program ρ to retrieved solution ρ_x
Train network on x,ρ pairs
Task x
Program ρ
Gradient step in parameters of Q to maximize $\log Q(\rho|x)$

Figure: Algorithm Cycle of DreamCoder

DreamCoder



Algorithm 1 Full DreamCoder algorithm

```
function DreamCoder($(D, X)$):
Input: Initial library functions $D$, tasks $X$
Output: Infinite stream of libraries, recognition models, and beams
Hyperparameters: Batch size $B$, enumeration timeout $T$, maximum beam size $M$
$\theta \leftarrow$uniform distribution
$\mathcal{B}_x \leftarrow \varnothing, \forall x \in X$                                ▷ Initialize beams to be empty
while true do                                                                          ▷ Loop over epochs
    shuffle $\leftarrow$ random permutation of $X$                                     ▷ Randomize minibatches
    while shuffle is not empty do                                                      ▷ Loop over minibatches
        batch $\leftarrow$ first $B$ elements of shuffle                               ▷ Next minibatch of tasks
        shuffle $\leftarrow$ shuffle with first $B$ elements removed
        $\forall x \in$ batch: $\mathcal{B}_x \leftarrow \mathcal{B}_x \cup \{\rho \mid \rho \in$ enumerate$(\mathrm{P}[\cdot|D, \theta], T)$ if $\mathrm{P}[x|\rho] > 0\}$    ▷ Wake
        Train $Q(\cdot|\cdot)$ to minimize $\mathcal{L}^{\mathrm{MAP}}$ across all $\{\mathcal{B}_x\}_{x \in X}$    ▷ Dream Sleep
        $\forall x \in$ batch: $\mathcal{B}_x \leftarrow \mathcal{B}_x \cup \{\rho \mid \rho \in$ enumerate$(Q(\cdot|x), T)$ if $\mathrm{P}[x|\rho] > 0\}$    ▷ Wake
        $\forall x \in$ batch: $\mathcal{B}_x \leftarrow$ top $M$ elements of $\mathcal{B}_x$ as measured by $\mathrm{P}[\cdot|x, D, \theta]$▷ Keep top $M$ programs
        $D, \theta, \{\mathcal{B}_x\}_{x \in X} \leftarrow$ABSTRACTION$(D, \theta, \{\mathcal{B}_x\}_{x \in X})$    ▷ Abstraction Sleep
        yield $(D, \theta)$, $Q$, $\{\mathcal{B}_x\}_{x \in X}$ ▷ Yield the updated library, recognition model, and solutions found to tasks
    end while
end while
```

DreamCoder

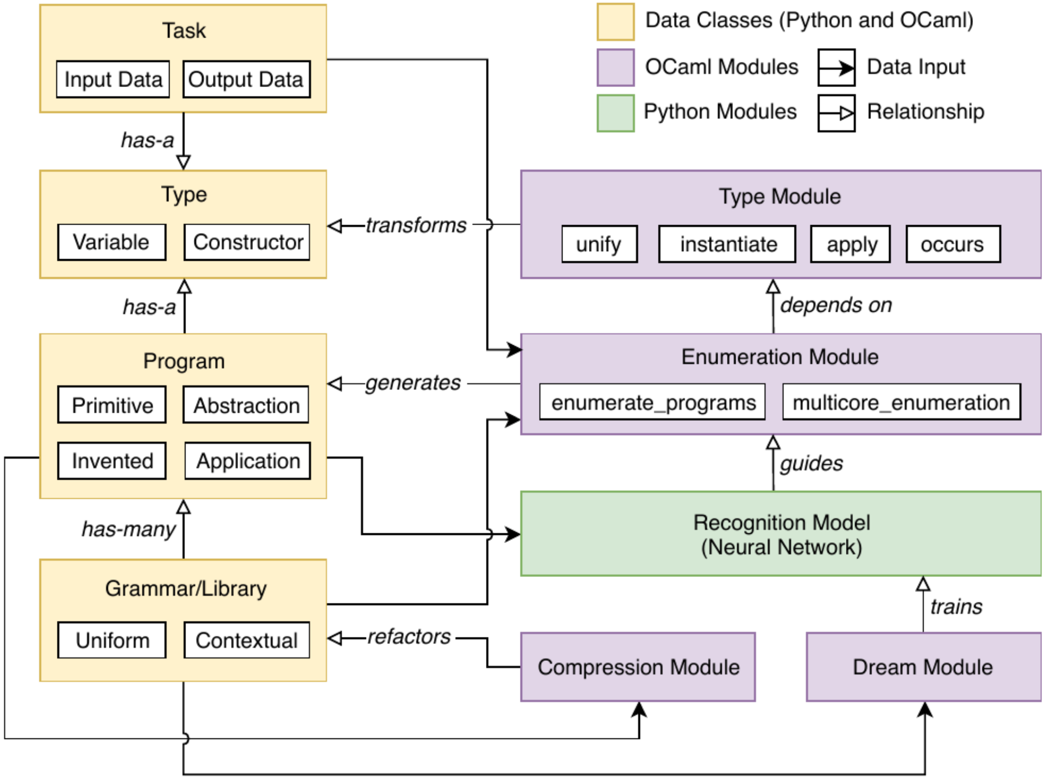


Figure: Different data-classes in DreamCoder.

DreamCoder

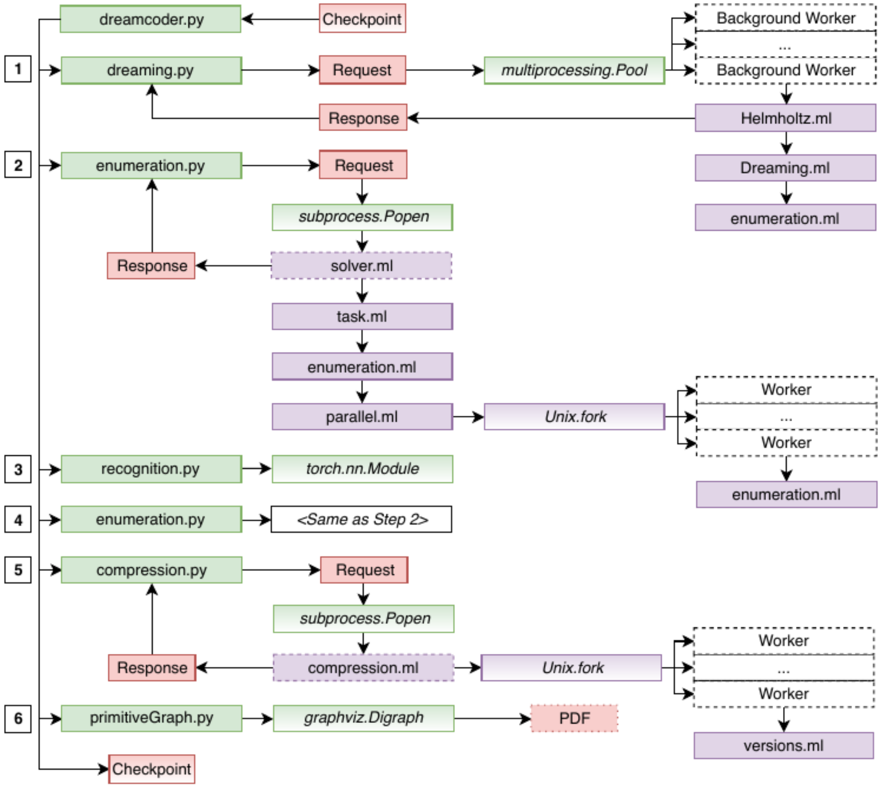
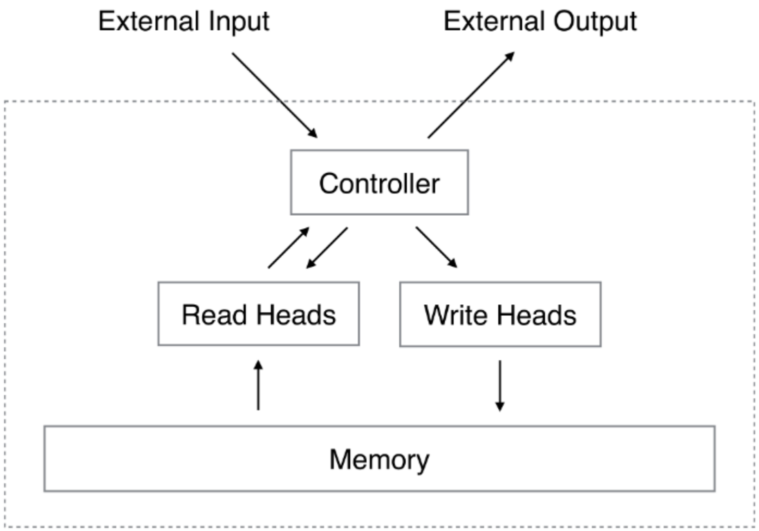


Figure: Program Flowchart: Phase 1, Dreaming. Phase 2, 1st Program Enumeration. Phase 3, Recognition Model Training. Phase 4, 2nd Program Enumeration. Phase 5, Abstraction (Compression). Phase 6, Library Visualization.

Recap: Helmholtz Machine [7] [8]

- Couples neural networks with external memory, which is accessed through an internal attention mechanism
  - Iteratively modify the state of the network through its memory mechanism
- Can infer simple algorithms from data
- Structure
  - Controller is a neural network
  - Heads select the part of the memory to access
  - Memory is essentially a large matrix



[7] Graves, A., Wayne, G. and Danihelka, I., 2014. Neural turing machines. arXiv preprint arXiv:1410.5401.
[8] YouTube: DeepMind x UCL | Deep Learning Lectures | 8/12 | Attention and Memory in Deep Learning

Recap: Helmholtz Machine

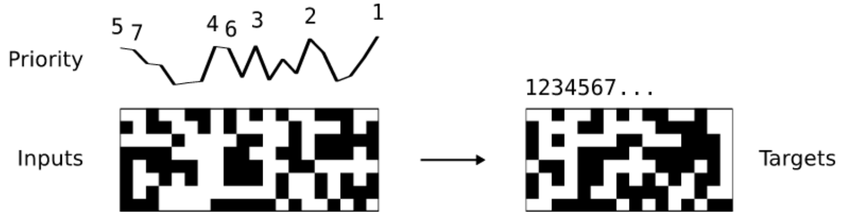
- Heavily relies on selection attention, whee the controller emits a distribution over the memory matrix, which then defines content- and location-based attention mechanisms [9]
- Content-based:
  - A key vector is compared to each memory location
- Location-based:
  - Use a shift kernel in conjunction with the weighting to shift to a new location in memory
- Results in three different interaction modes:
  1. Content key only
  2. Content and location
  3. Location only

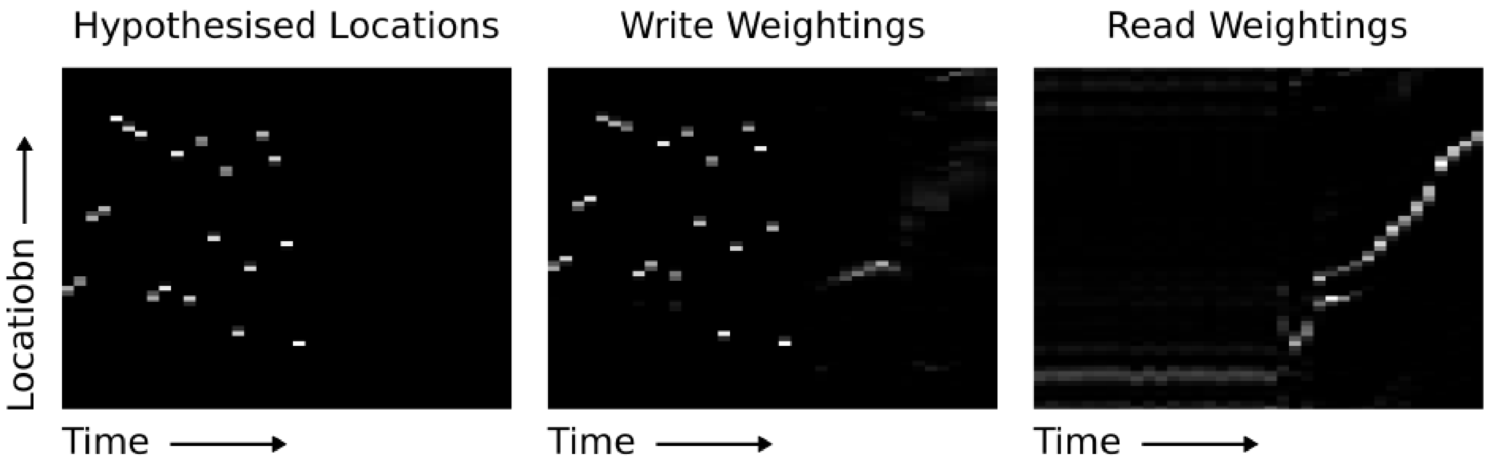
[9] Lilian Weng's review of attention: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Recap: Helmholtz Machine

Example: Priority Sort

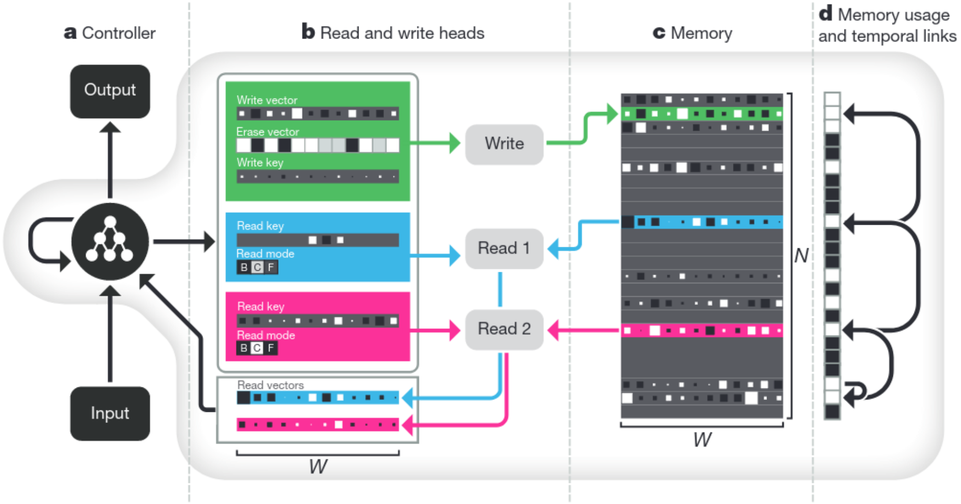
- Learns algorithm to sort data from a sequence of random binary vectors and their respective scalar priority rating





Outlook: Differentiable Neural Computer [10]

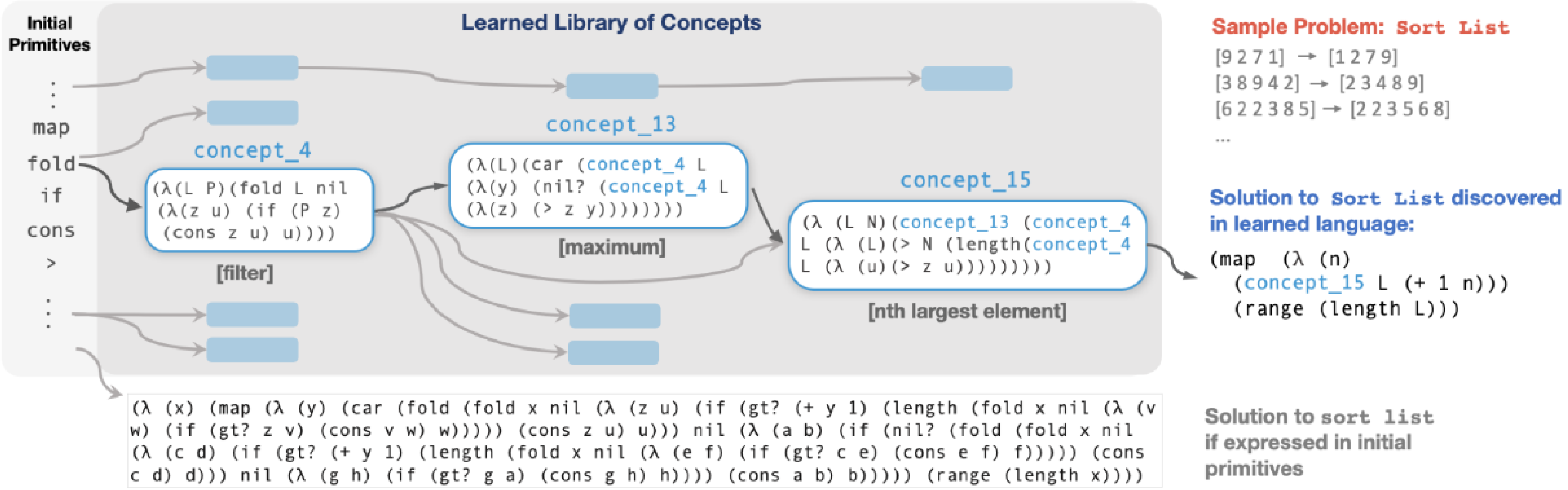
- Successor architecture to the neural turing machine with new attention mechanisms
- Specifically geared towards applications in graphs ⟶ more on this later!



[10] Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S.G., Grefenstette, E., Ramalho, T., Agapiou, J. and Badia, A.P., 2016. Hybrid computing using a neural network with dynamic external memory. Nature, 538(7626), pp.471-476.

DreamCoder

- Due to its compositional nature, representations of problems can be bootstrapped from earlier, simpler version of the scientific task to more and more complex settings



```
(λ(L P)(fold L nil
 (λ(z u) (if (P z)
  (cons z u) u))))
```

```
(λ(L)(car (concept_4 L
(λ(y) (nil? (concept_4 L
(λ(z) (> z y))))))))
```

```
(λ (L N)(concept_13 (concept_4
L (λ (L)(> N (length(concept_4
L (λ (u)(> z u)))))))))
```

Sample Problem: `Sort List`

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

Solution to `Sort List` discovered in learned language:

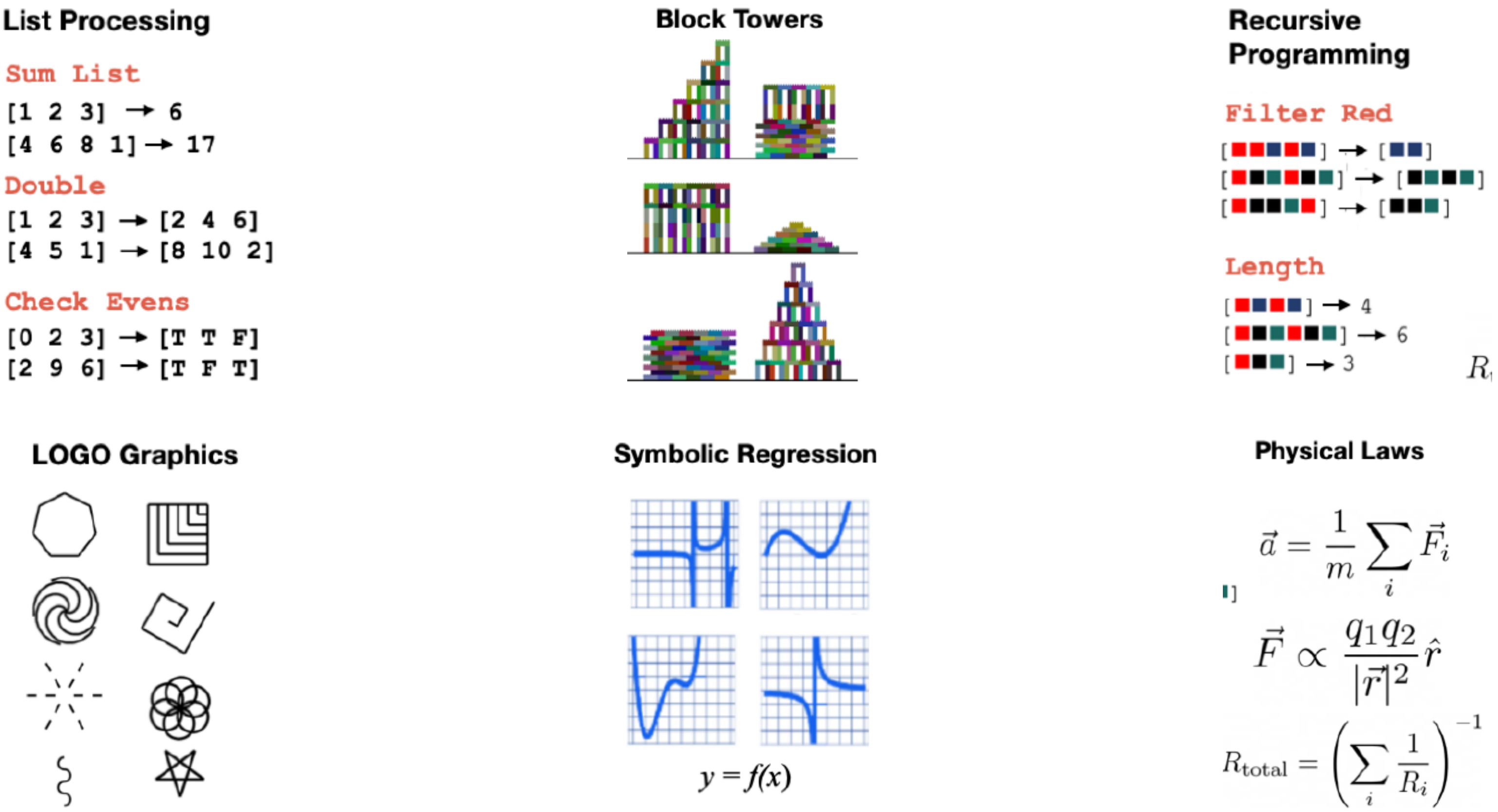
```
(map  (λ (n)
  (concept_15 L (+ 1 n)))
  (range (length L)))
```

```
(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)))
```

Solution to `sort list` if expressed in initial primitives

DreamCoder

Applications



List Processing

Sum List

[1 2 3] → 6
[4 6 8 1] → 17

Double

[1 2 3] → [2 4 6]
[4 5 1] → [8 10 2]

Check Evens

[0 2 3] → [T T F]
[2 9 6] → [T F T]

Block Towers

Recursive Programming

Filter Red

[🟥🟥🟦🟥🟦] → [🟦🟦]
[🟥⬛🟩🟥⬛🟩] → [⬛🟩⬛🟩]
[🟥⬛⬛🟩🟥] → [⬛⬛🟩]

Length

[🟥🟦🟥🟦] → 4
[🟥⬛🟩🟥⬛🟩] → 6
[🟥⬛🟩] → 3

LOGO Graphics

Symbolic Regression

$y = f(x)$

Physical Laws

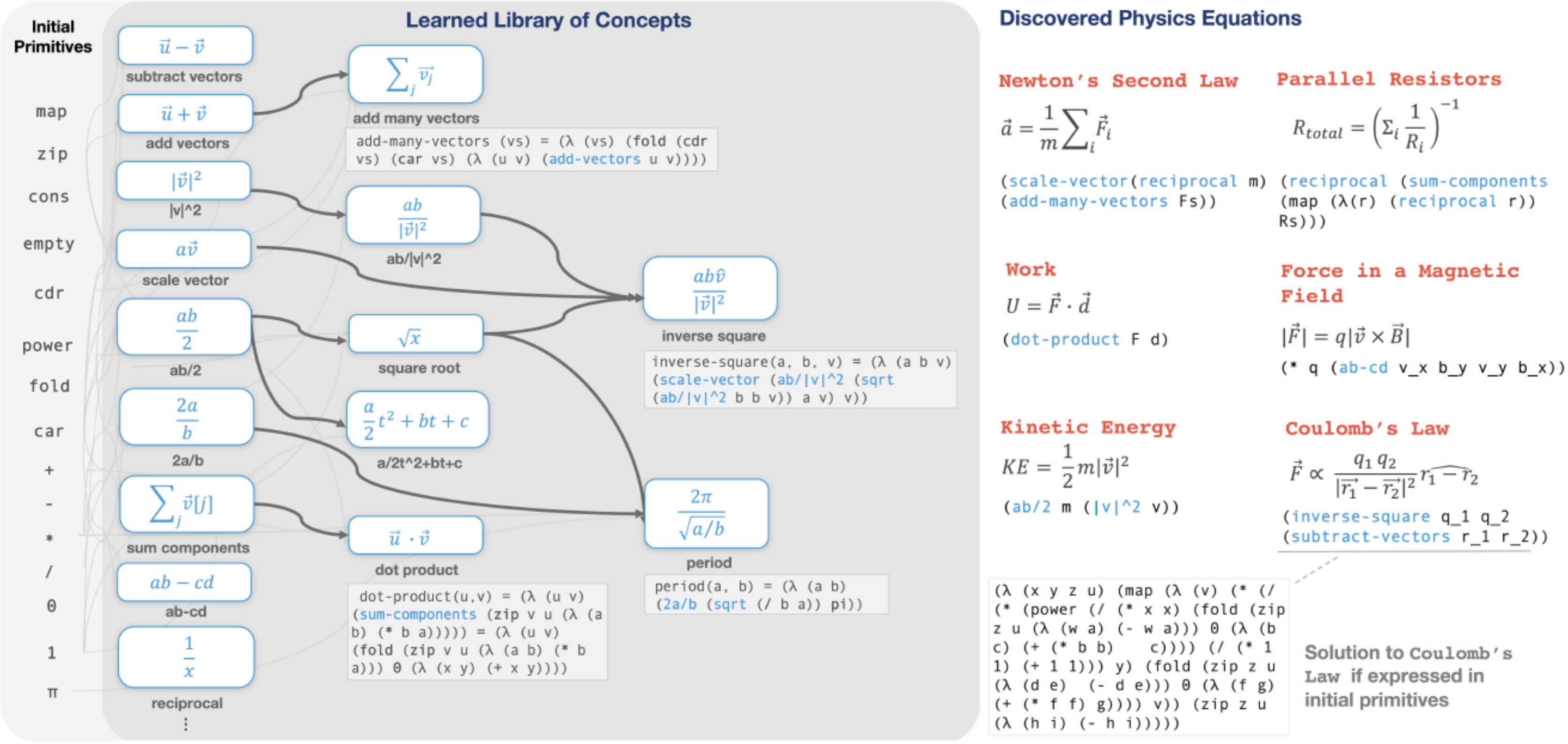
$$\vec{a} = \frac{1}{m}\sum_i \vec{F}_i$$

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

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

DreamCoder

Applications



```
add-many-vectors (vs) = (λ (vs) (fold (cdr
vs) (car vs) (λ (u v) (add-vectors u v))))
```

```
dot-product(u,v) = (λ (u v)
(sum-components (zip v u (λ (a
b) (* b a))))) = (λ (u v)
(fold (zip v u (λ (a b) (* b
a))) 0 (λ (x y) (+ x y))))
```

```
inverse-square(a, b, v) = (λ (a b v)
(scale-vector (ab/|v|^2 (sqrt
(ab/|v|^2 b b v)) a v) v))
```

```
period(a, b) = (λ (a b)
(2a/b (sqrt (/ b a)) pi))
```

Newton's Second Law

$$\vec{a}=\frac{1}{m}\sum_i \vec{F_i}$$

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

Parallel Resistors

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

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

Work

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

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

Force in a Magnetic Field

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

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

Kinetic Energy

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

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

Coulomb's Law

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

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

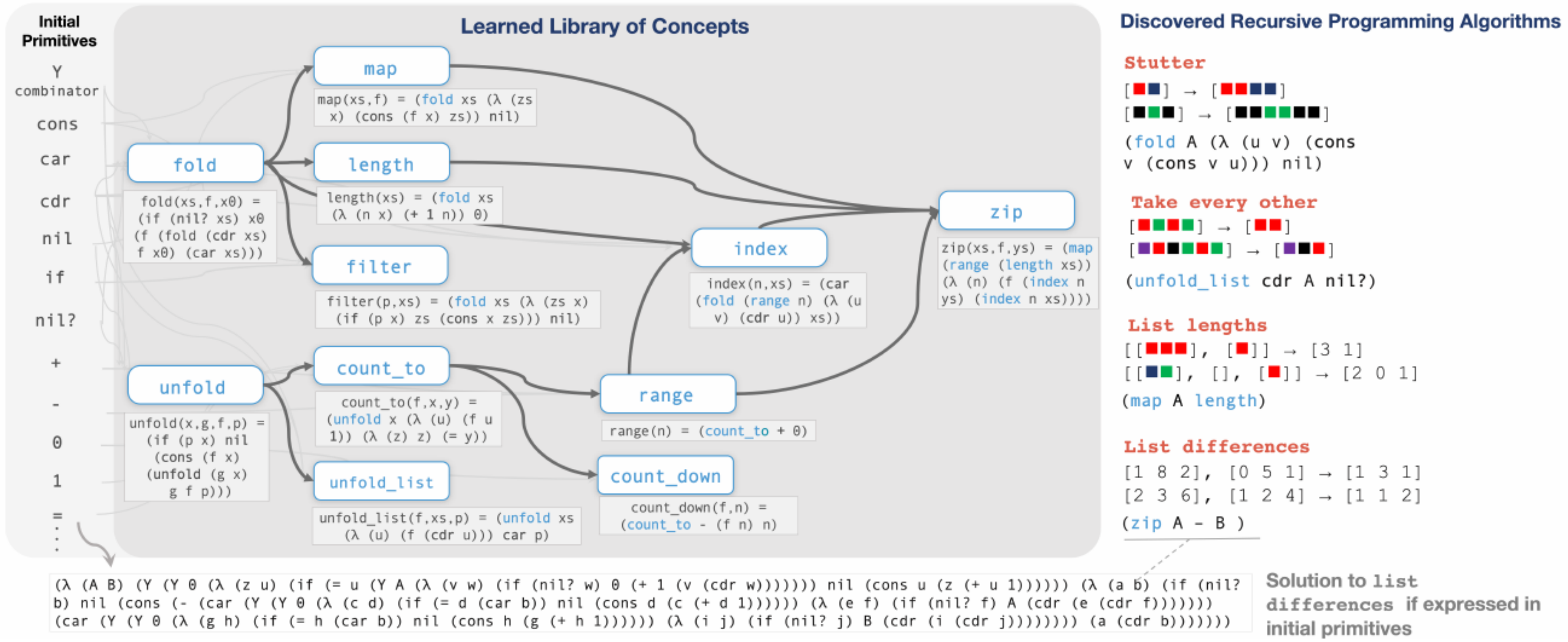
```
(λ (x y z u) (map (λ (v) (* (/
(* (power (/ (* x x) (fold (zip
z u (λ (w a) (- w a))) 0 (λ (b
c) (+ (* b b)    c)))) (/ (* 1
1) (+ 1 1))) y) (fold (zip z u
(λ (d e)  (- d e))) 0 (λ (f g)
(+ (* f f) g)))) v)) (zip z u
(λ (h i) (- h i)))))
```

Solution to Coulomb's Law if expressed in initial primitives

Figure: Learned library for physics equations.

DreamCoder

Applications



```
fold(xs,f,x0) = (if (nil? xs) x0 (f (fold (cdr xs) f x0) (car xs)))
map(xs,f) = (fold xs (λ (zs x) (cons (f x) zs)) nil)
length(xs) = (fold xs (λ (n x) (+ 1 n)) 0)
filter(p,xs) = (fold xs (λ (zs x) (if (p x) zs (cons x zs))) nil)
index(n,xs) = (car (fold (range n) (λ (u v) (cdr u)) xs))
zip(xs,f,ys) = (map (range (length xs)) (λ (n) (f (index n ys) (index n xs))))
unfold(x,g,f,p) = (if (p x) nil (cons (f x) (unfold (g x) g f p)))
count_to(f,x,y) = (unfold x (λ (u) (f u 1)) (λ (z) z) (= y))
range(n) = (count_to + 0)
unfold_list(f,xs,p) = (unfold xs (λ (u) (f (cdr u))) car p)
count_down(f,n) = (count_to - (f n) n)
```

Discovered Recursive Programming Algorithms

Stutter

[🟥🟦] → [🟥🟥🟦🟦]
[⬛🟩⬛] → [⬛⬛🟩🟩⬛⬛]

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

Take every other

[🟥🟩🟥🟩] → [🟥🟥]
[🟪🟥⬛🟩🟥🟩] → [🟪⬛🟥]

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

List lengths

[[🟥🟥🟥], [🟥]] → [3 1]
[[🟦🟩], [], [🟥]] → [2 0 1]

```
(map A length)
```

List differences

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

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

```
(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)))))))
```

Solution to list differences if expressed in initial primitives

Figure: Learned library for recursive programming algorithm.

DreamCoder

Applications

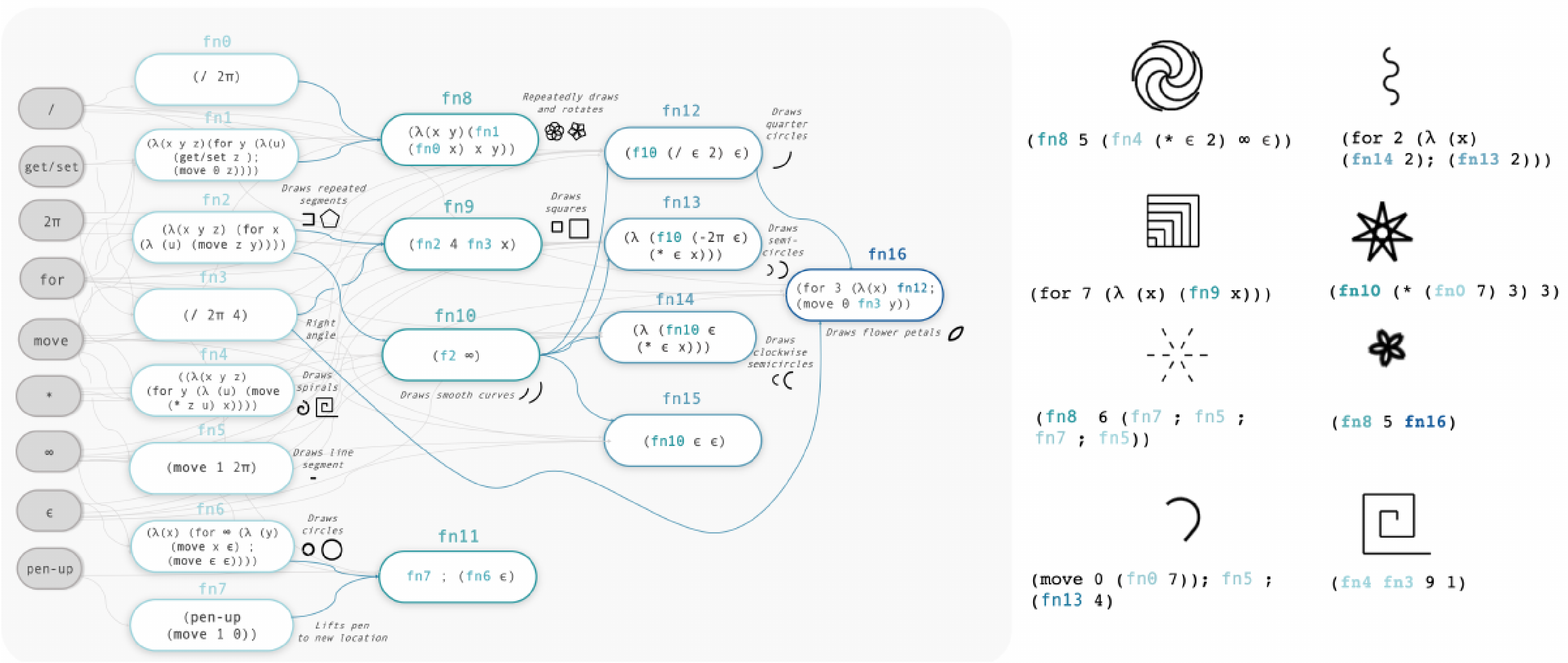
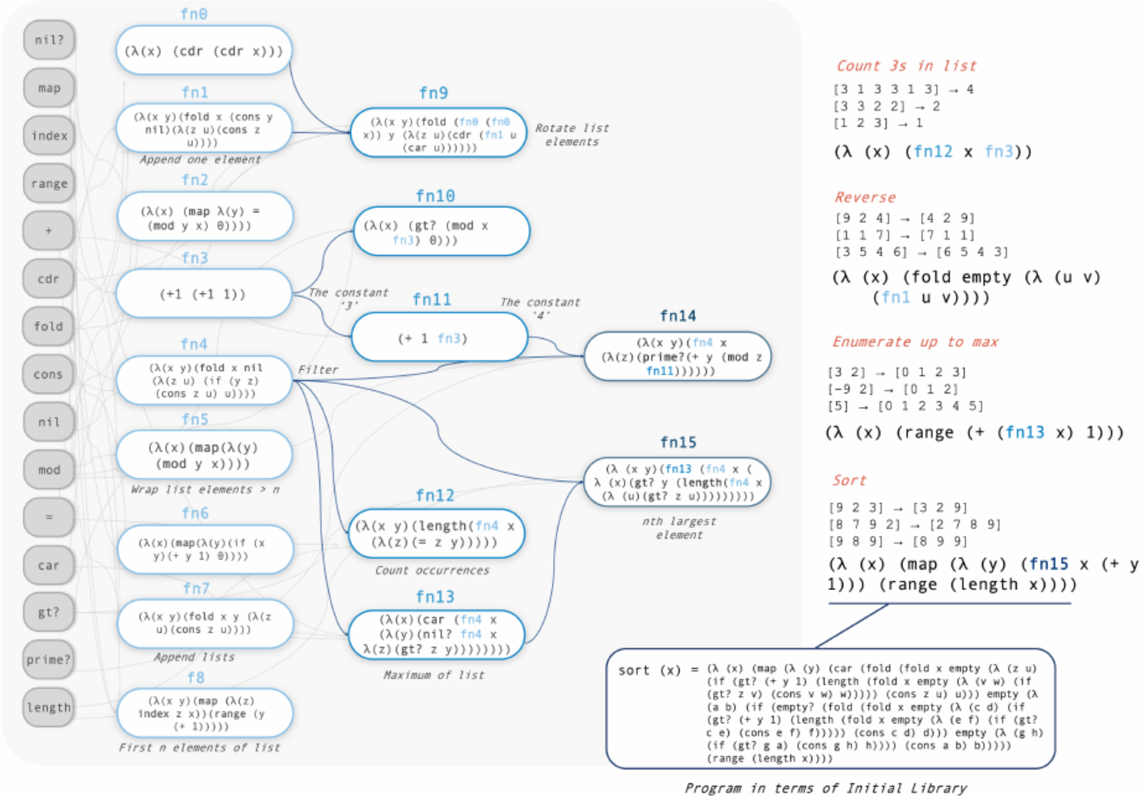


Figure: Learned library for LOGO graphics.

DreamCoder

Applications



```
fn0   (λ(x) (cdr (cdr x)))
fn1   (λ(x y)(fold x (cons y nil)(λ(z u)(cons z u))))
fn2   (λ(x) (map λ(y) = (mod y x) 0))))
fn3   (+1 (+1 1))
fn4   (λ(x y)(fold x nil (λ(z u) (if (y z) (cons z u) u))))
fn5   (λ(x)(map(λ(y) (mod y x))))
fn6   (λ(x)(map(λ(y)(if (x y)(+ y 1) 0))))
fn7   (λ(x y)(fold x y (λ(z u)(cons z u))))
f8    (λ(x y)(map (λ(z) index z x))(range (y (+ 1)))))
fn9   (λ(x y)(fold (fn0 (fn0 x)) y (λ(z u)(cdr (fn1 u (car u))))))
fn10  (λ(x) (gt? (mod x fn3) 0)))
fn11  (+ 1 fn3)
fn12  (λ(x y)(length(fn4 x (λ(z)(= z y)))))
fn13  (λ(x)(car (fn4 x (λ(y)(nil? fn4 x λ(z)(gt? z y))))))))
fn14  (λ(x y)(fn4 x (λ(z)(prime?(+ y (mod z fn11))))))
fn15  (λ (x y)(fn13 (fn4 x ( λ (x)(gt? y (length(fn4 x (λ (u)(gt? z u)))))))))
```

Count 3s in list

```
[3 1 3 3 1 3] → 4
[3 3 2 2] → 2
[1 2 3] → 1
(λ (x) (fn12 x fn3))
```

Reverse

```
[9 2 4] → [4 2 9]
[1 1 7] → [7 1 1]
[3 5 4 6] → [6 5 4 3]
(λ (x) (fold empty (λ (u v)
    (fn1 u v))))
```

Enumerate up to max

```
[3 2] → [0 1 2 3]
[-9 2] → [0 1 2]
[5] → [0 1 2 3 4 5]
(λ (x) (range (+ (fn13 x) 1)))
```

Sort

```
[9 2 3] → [3 2 9]
[8 7 9 2] → [2 7 8 9]
[9 8 9] → [8 9 9]
(λ (x) (map (λ (y) (fn15 x (+ y
1))) (range (length x))))
```

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

Program in terms of Initial Library

Figure: Learned library for list processing.

DreamCoder

Take-Aways

- Combining probabilistic programming with a DSL-learning procedure and novel probabilistic inference procedure to iteratively learn to represent a problem's domain allows one to gain the ability to solve a problem
- Such applications require highly complex codebase structures across multiple languages
- For more complex examples reliant on a highly efficient inference procedure

Outline

Course Outline

Example Applications of Probabilistic Programming

- ETALUMIS: Bringing Probabilistic Programming to Scientific Simulators at Scale
- DreamCoder: Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning

Why Do We even Need Probabilistic Programming?

Underlying Theoretical Ideas

Why Do We Need Probabilistic Programming?

- Blub 1

Why Do We Need Probabilistic Programming?

- Blub 2

Why Do We Need Probabilistic Programming?

- Blub 3

Why Do We Need Probabilistic Programming?

- Blub 4

Why Do We Need Probabilistic Programming?

- Blub 5

Why Do We Need Probabilistic Programming?

- Blub 6

Why Do We Need Probabilistic Programming?

- Blub 7

Why Do We Need Probabilistic Programming?

- Blub 8

Outline

Course Outline

Example Applications of Probabilistic Programming

- ETALUMIS: Bringing Probabilistic Programming to Scientific Simulators at Scale
- DreamCoder: Growing Generalizable, Interpretable Knowledge with Wake-Sleep Bayesian Program Learning

Why Do We even Need Probabilistic Programming?

Underlying Theoretical Ideas