

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

Different Types of Probabilistic Programming Systems

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

Different Types of Probabilistic Programming Systems

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
- Extensive Machine-Learning Based Design Example

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

Different Types of Probabilistic Programming Systems

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 impending exascale-era

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

²Baydin, 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

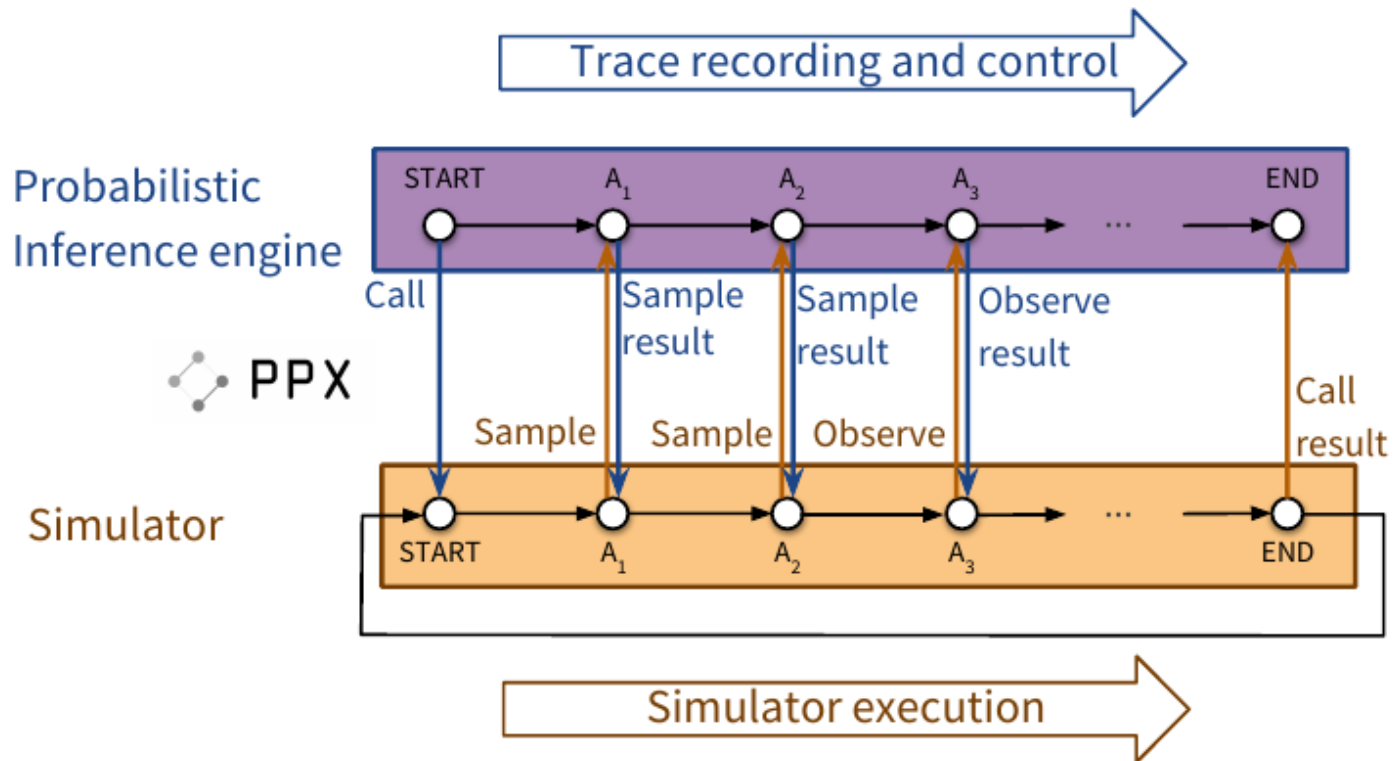


Figure: High-level view from the perspective of the probabilistic programming execution protocol (PPX).

Etalumis

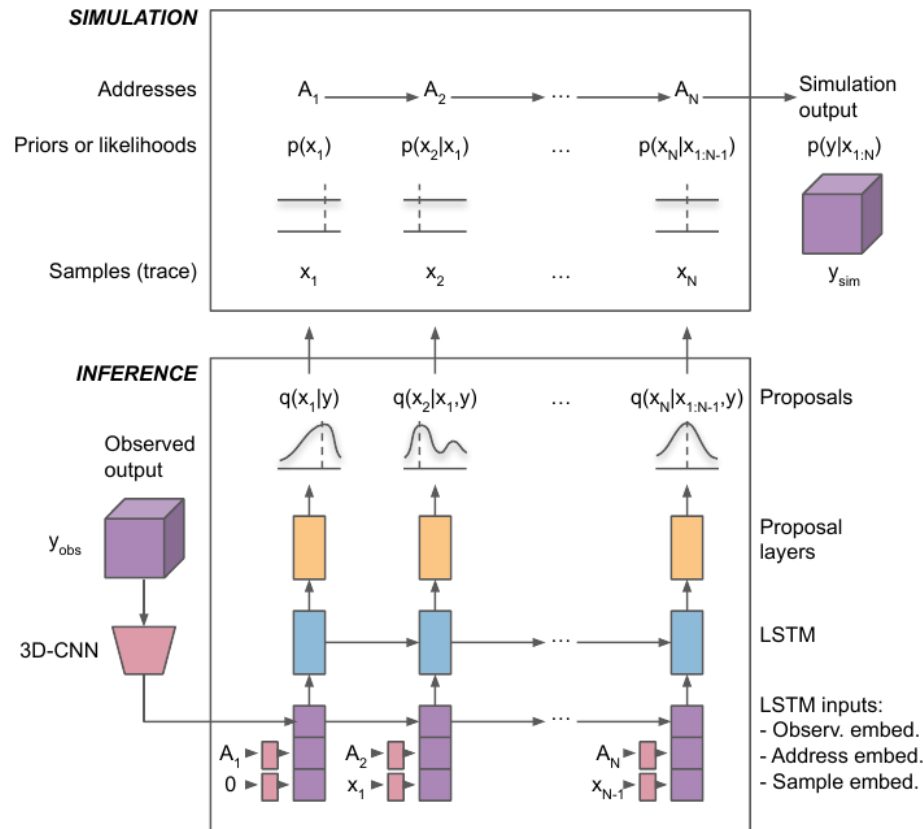


Figure: Detailed connection between simulation and inference in the probabilistic programming-based approach.

Etalumis

Recap: Amortized Inference^{3 4}

- Blub 1

³Gershman, 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).

⁴Ritchie, 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 Compilation⁵

- 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

⁵Le, 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

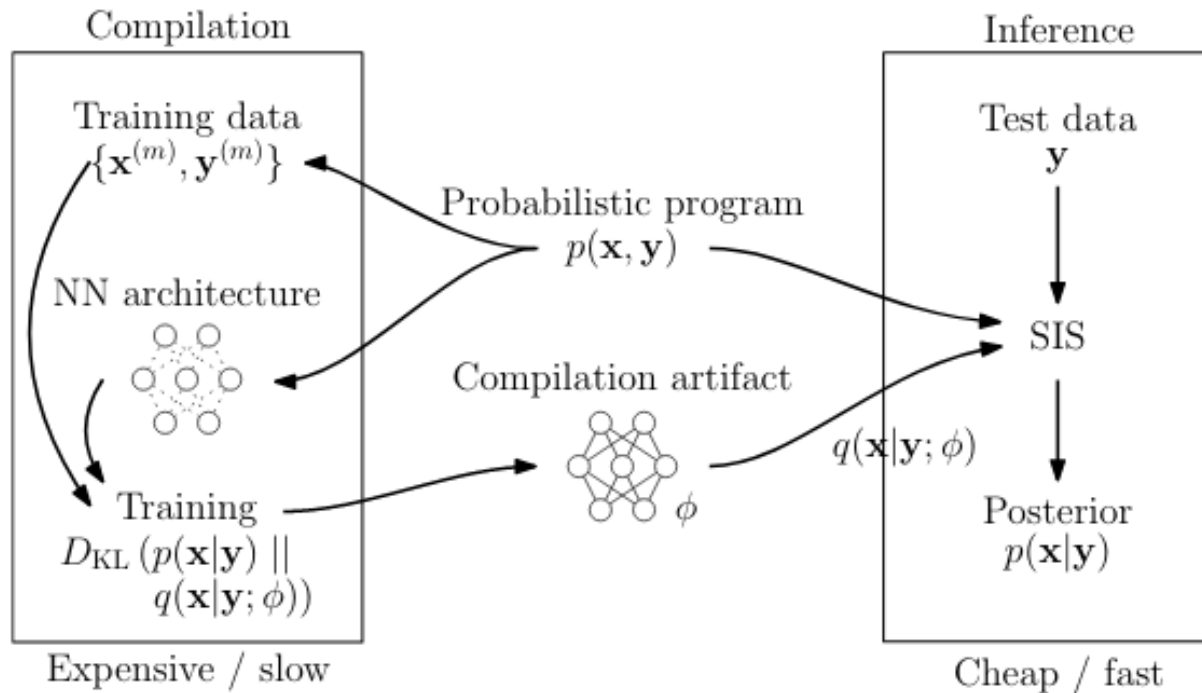


Figure: Automatic construction of a neural network surrogate, which is then trained with data generated by the probabilistic program to eventually act as the proposal distribution at inference time.

Etalumis

Outlook: Inference Compilation

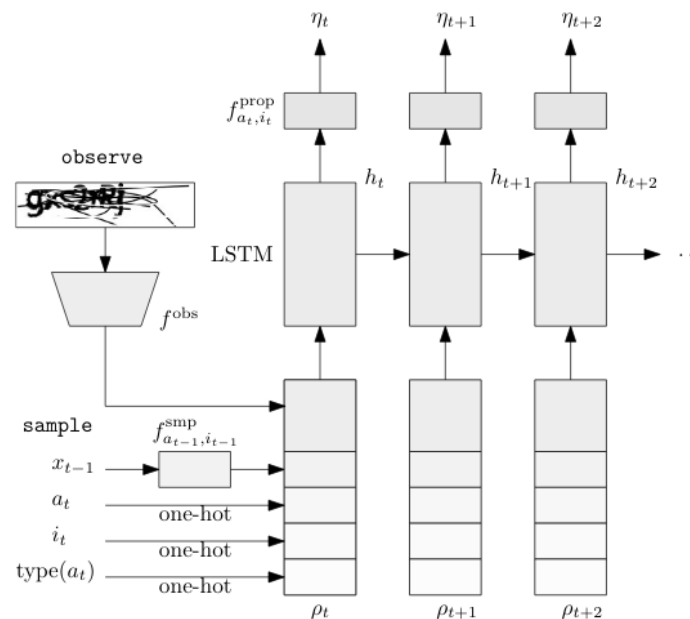


Figure: Example application for captcha solving based on probabilistic generative models for the captchas. With the LSTM at its core, required embeddings for the respective are attached adaptively.

Etalumis

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

Require: OnlineData {True/False value}

Require: B {Minibatch size}

Initialize inference network $q_\phi(\mathbf{x}|\mathbf{y})$

$N \leftarrow$ number of processes

for all $n \in \{1, \dots, N\}$ **do**

while Not Stop **do**

if OnlineData **then**

 Sample $\mathcal{D}_n = \{(\mathbf{x}, \mathbf{y})_1, \dots, (\mathbf{x}, \mathbf{y})_B\}$ from $p(\mathbf{x}, \mathbf{y})$

else

 Get $\mathcal{D}_n = \{(\mathbf{x}, \mathbf{y})_1, \dots, (\mathbf{x}, \mathbf{y})_B\}$ from $\hat{G}(\mathbf{x}, \mathbf{y})$

end if

 Synchronize parameters (ϕ) across all processes

$\mathcal{L}_n \leftarrow -\frac{1}{B} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_n} \log q_\phi(\mathbf{x}|\mathbf{y})$

 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 ϕ using $\nabla_\phi \mathcal{L}$ with e.g. ADAM, SGD, LARC, etc.

end while

end for

Etalumis

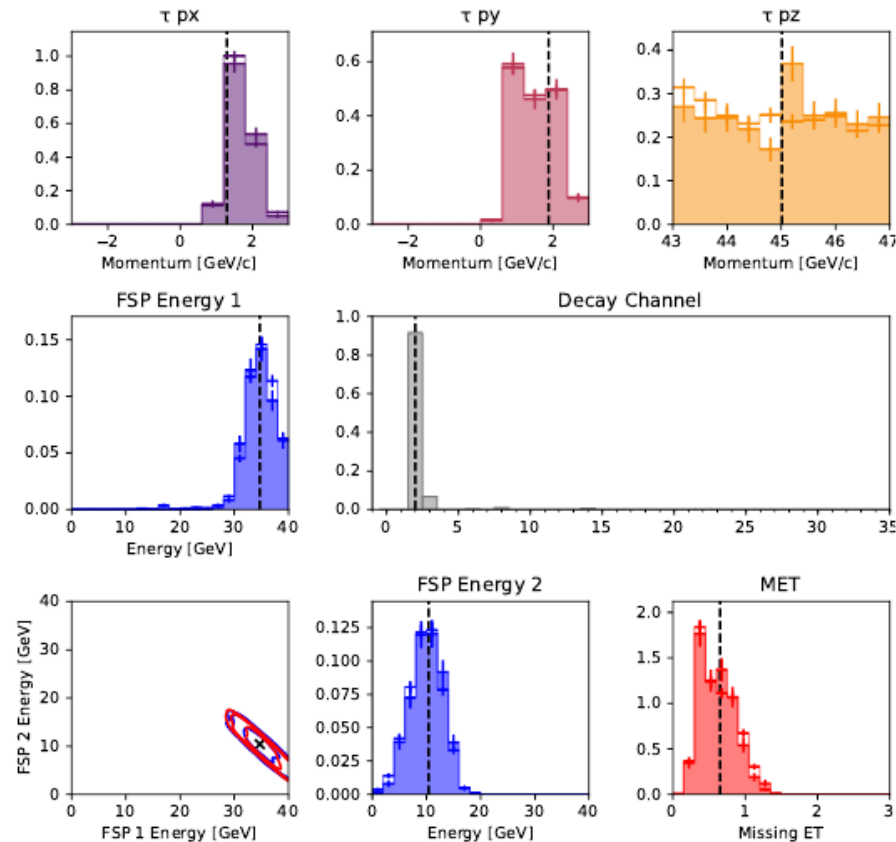
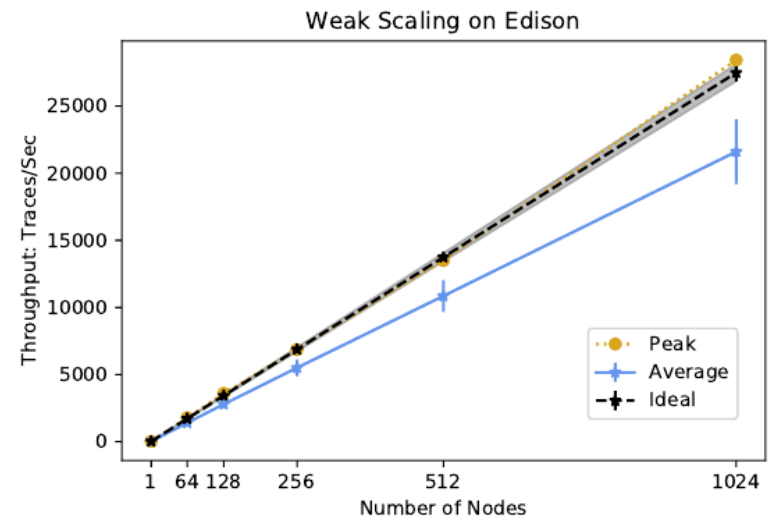
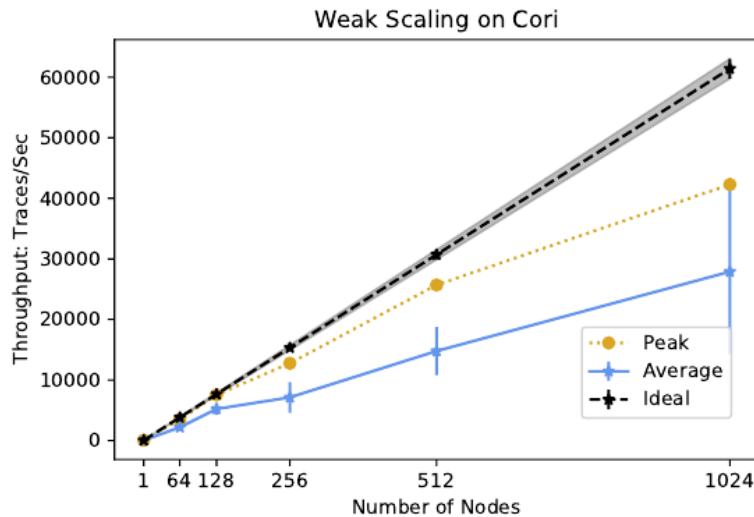


Figure: Posteriors obtained with random-walk Metropolis Hasting (filled histograms) and inference compilation (outline histograms) and ground truth values (dashed vertical lines).

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

DreamCoder⁶

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

⁶Ellis, 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 might solve tasks in the domain

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

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

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

DreamCoder

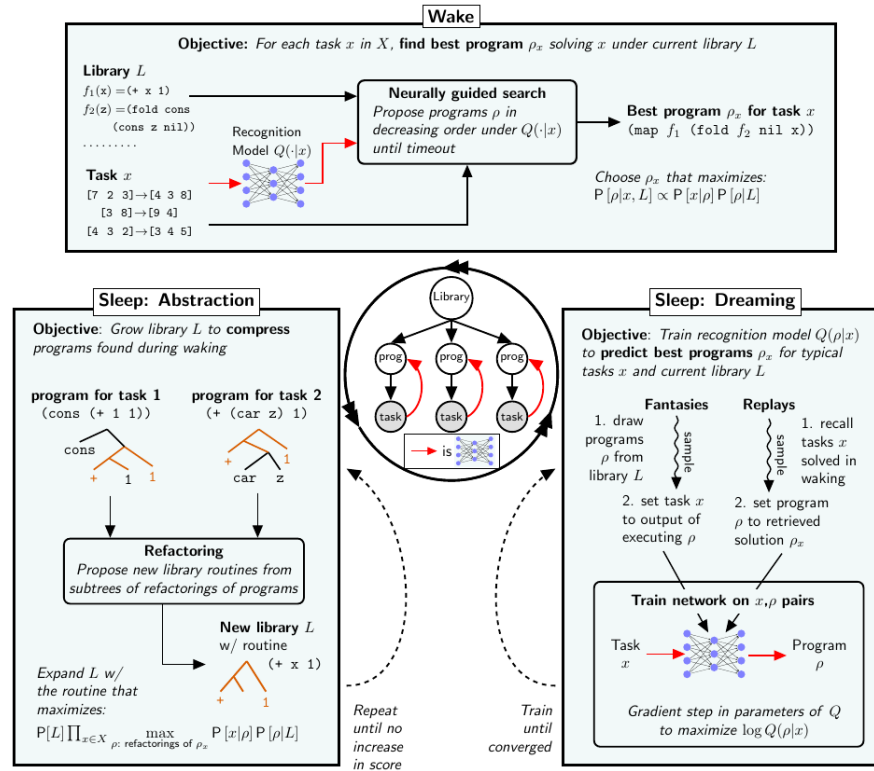


Figure: Algorithm Cycle of DreamCoder

DreamCoder

Algorithm 1 Full DreamCoder algorithm

```

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

```

DreamCoder

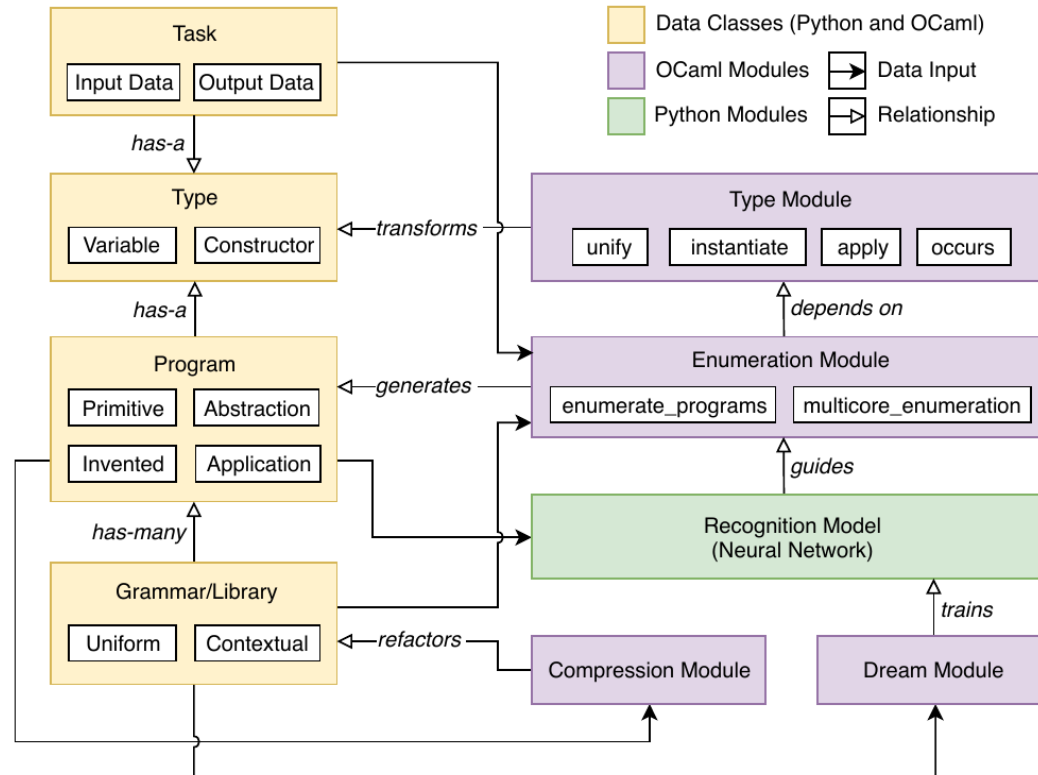


Figure: Different data-classes in DreamCoder.

DreamCoder

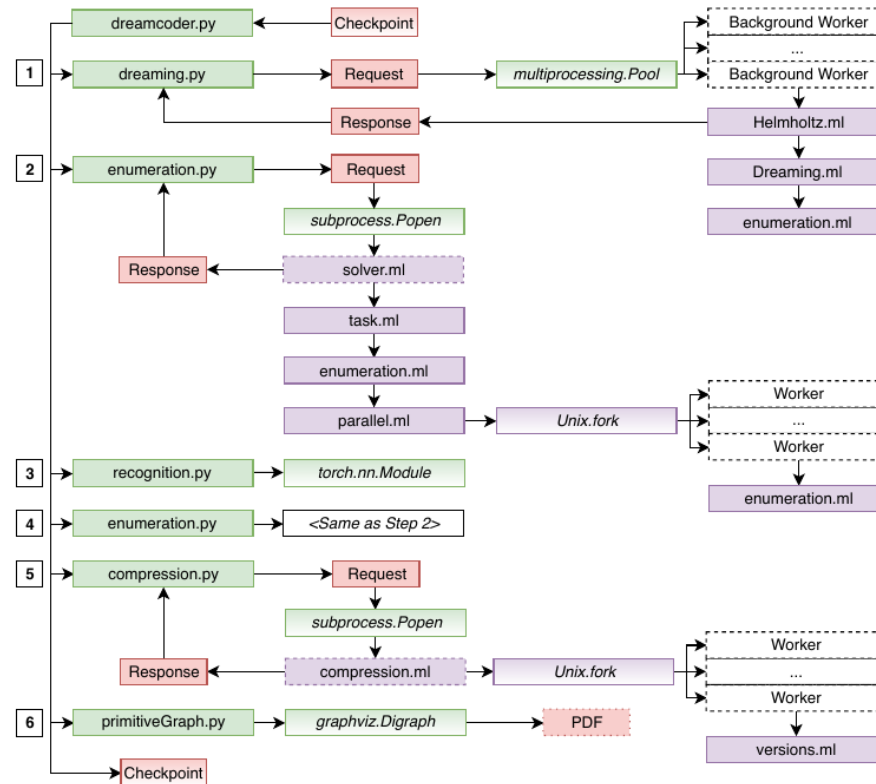
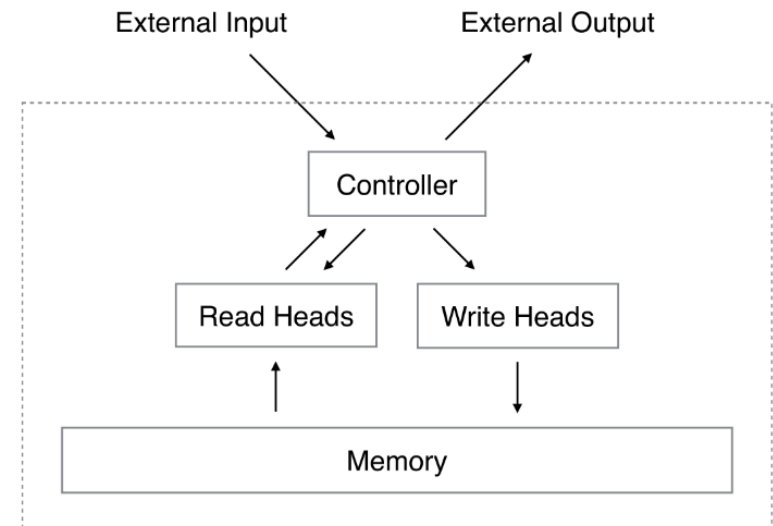


Figure: Program Flowchart: Phase 1, Dreaming. Phase 2, 1st Program Enumeration. Phase 3, Recognition Model Training. Phase 4, 2nd Program Enumeration. Phase 5, Abstraction (Compression). Phase 6, Library Visualization.

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



⁷Graves, A., Wayne, G. and Danihelka, I., 2014. Neural turing machines. arXiv preprint arXiv:1410.5401.

⁸YouTube: DeepMind x UCL | Deep Learning Lectures | 8/12 | Attention and Memory in Deep Learning

Recap: Helmholtz Machine

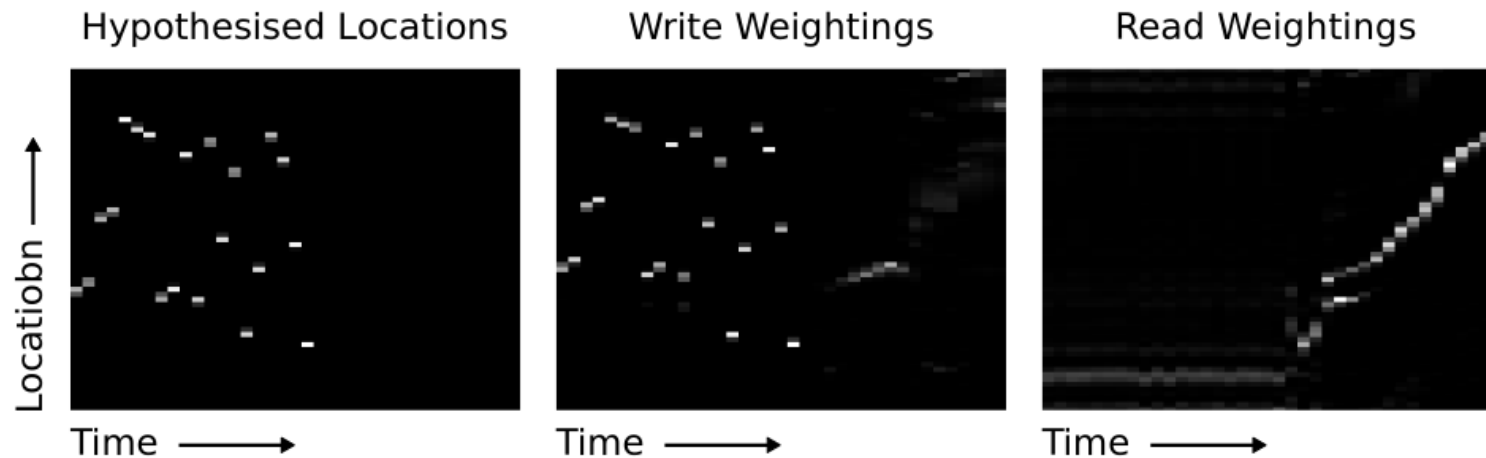
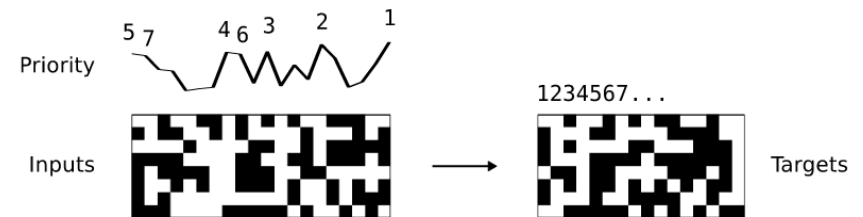
- Heavily relies on selection attention, where the controller emits a distribution over the memory matrix, which then defines content- and location-based attention mechanisms⁹
- Content-based:
 - A key vector is compared to each memory location
- Location-based:
 - Use a shift kernel in conjunction with 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

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

Recap: Helmholtz Machine

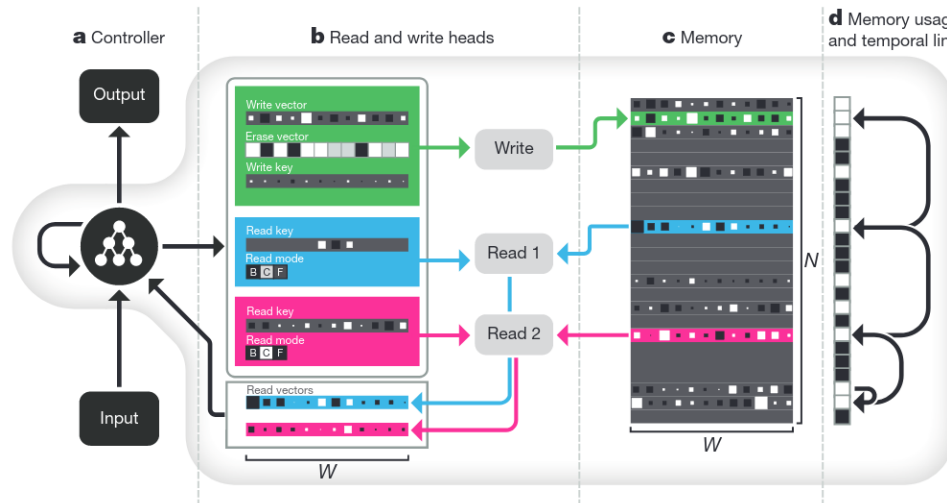
Example: Priority Sort

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



Outlook: Differentiable Neural Computer¹⁰

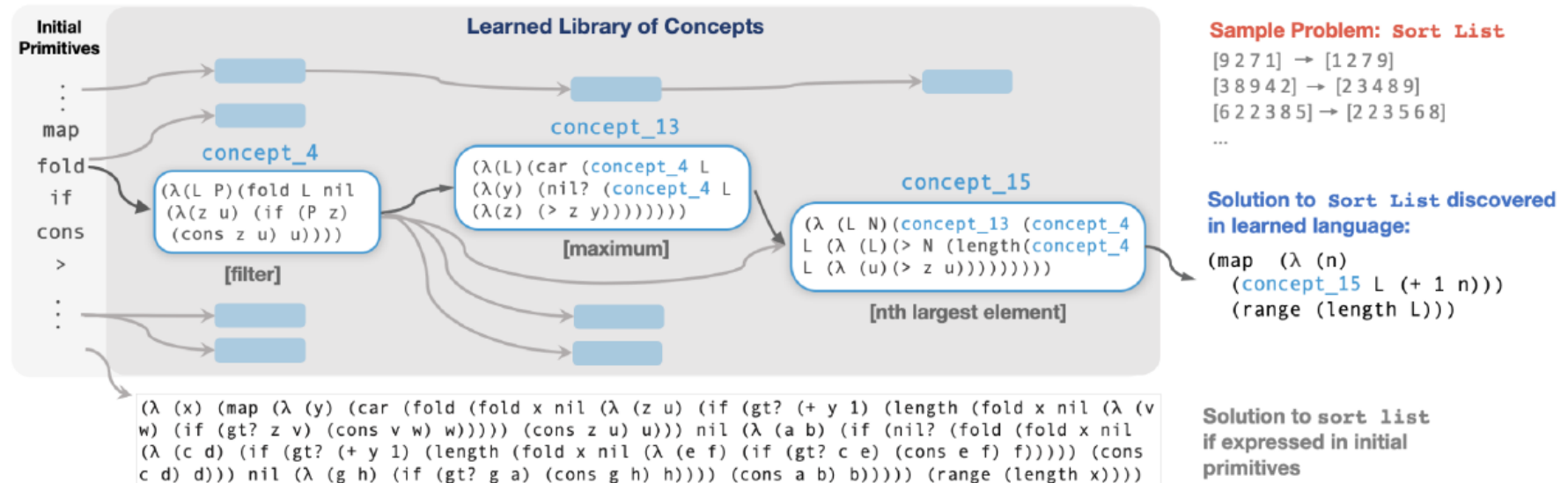
- Successor architecture to the neural turing machine with new attention mechanisms
- Specifically geared towards applications in graphs → more on this later!



¹⁰Graves, 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



DreamCoder

Applications

List Processing

Sum List

[1 2 3] → 6
[4 6 8 1] → 17

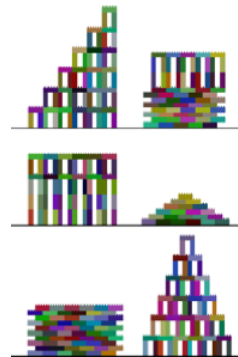
Double

[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

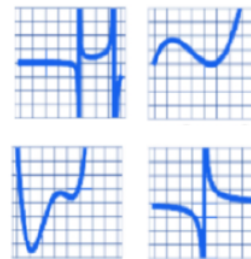
[■ ■ ■ ■] → 4
[■ ■ ■ ■ ■] → 6
[■ ■ ■] → 3

R_i

LOGO Graphics



Symbolic Regression



$$y = f(x)$$

Physical Laws

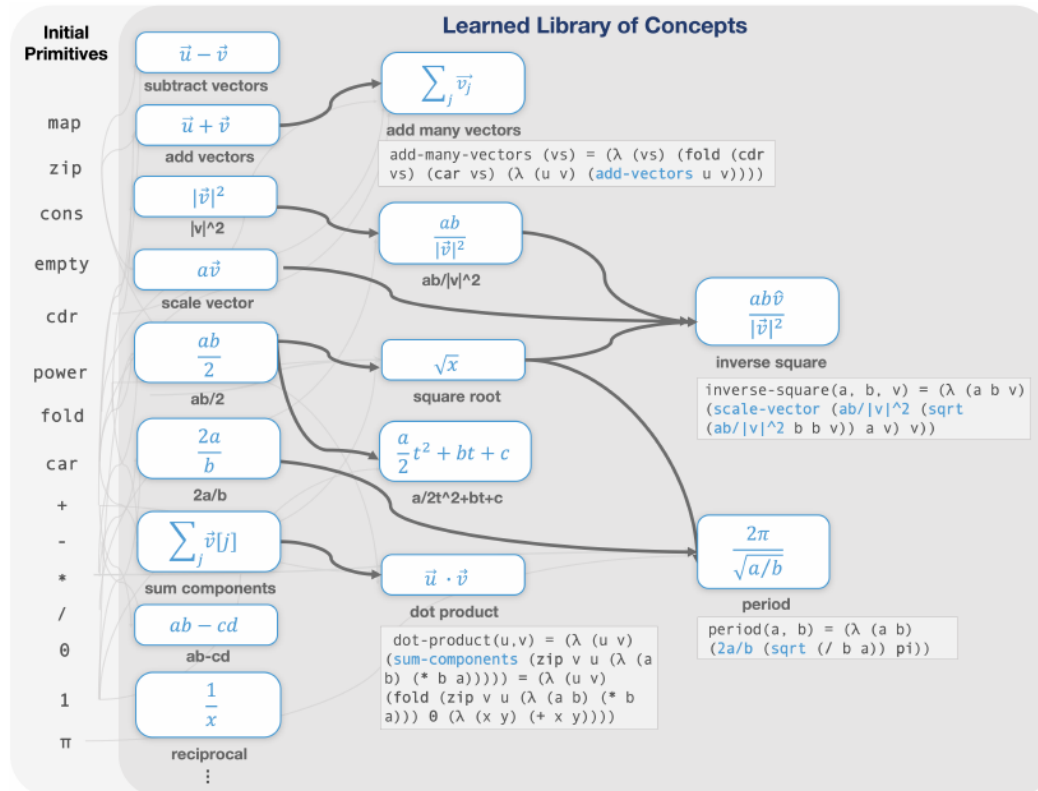
$$\vec{a} = \frac{1}{m} \sum_i \vec{F}_i$$

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

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

DreamCoder

Applications



Discovered Physics Equations

Newton's Second Law

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

`(scale-vector (reciprocal m) (reciprocal (sum-components (add-many-vectors Fs)) (map (λ(r) (reciprocal r)) Rs)))`

Parallel Resistors

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

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

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

`(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

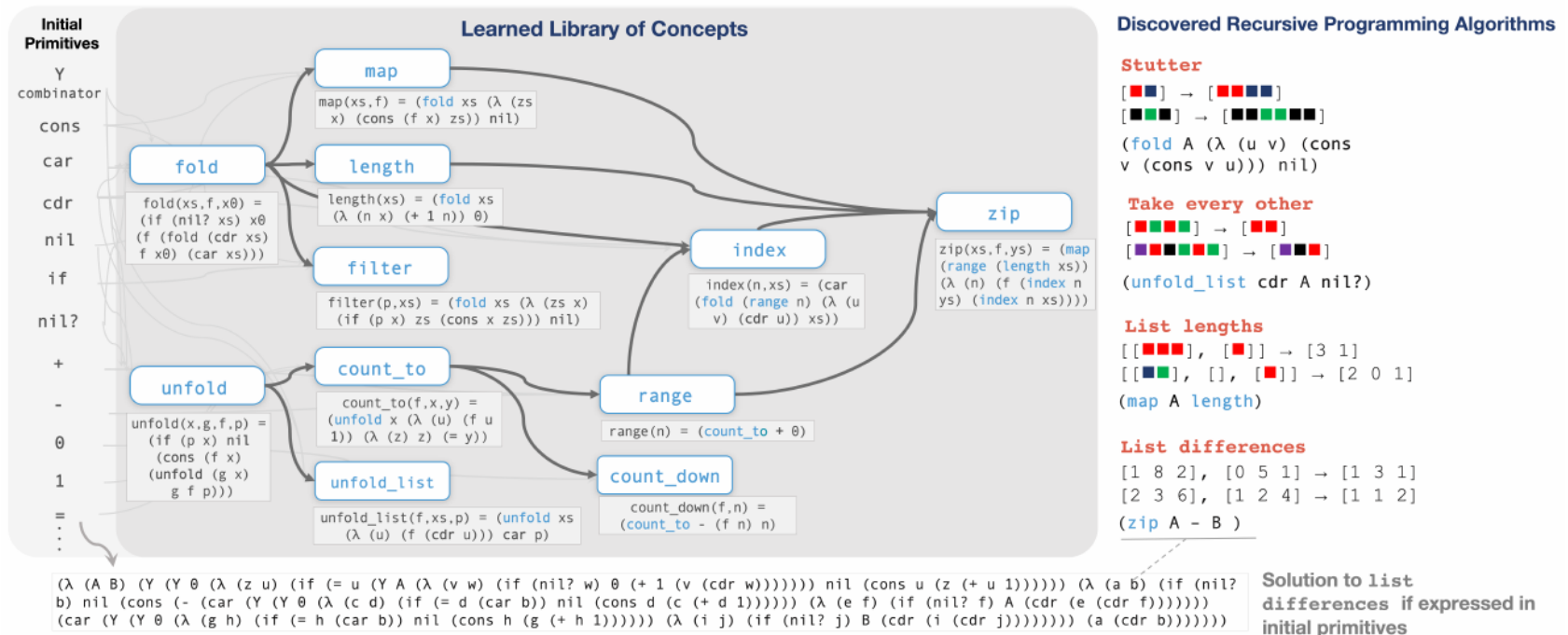


Figure: Learned library for recursive programming algorithm.

DreamCoder

Applications

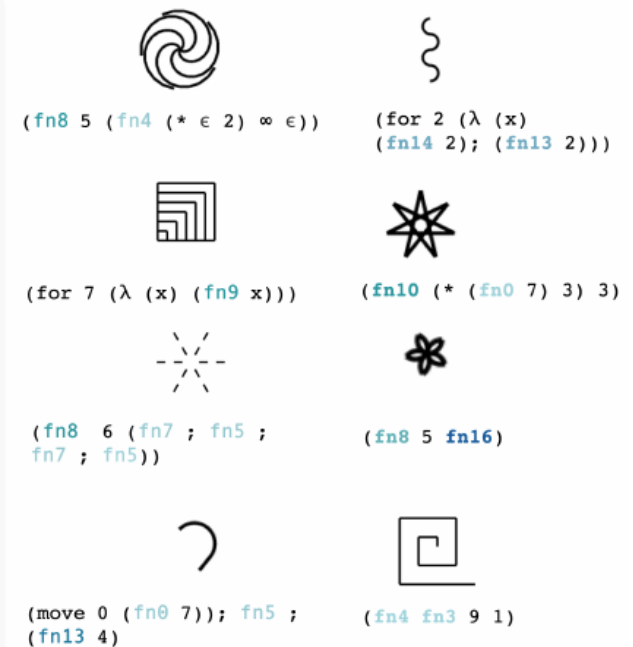
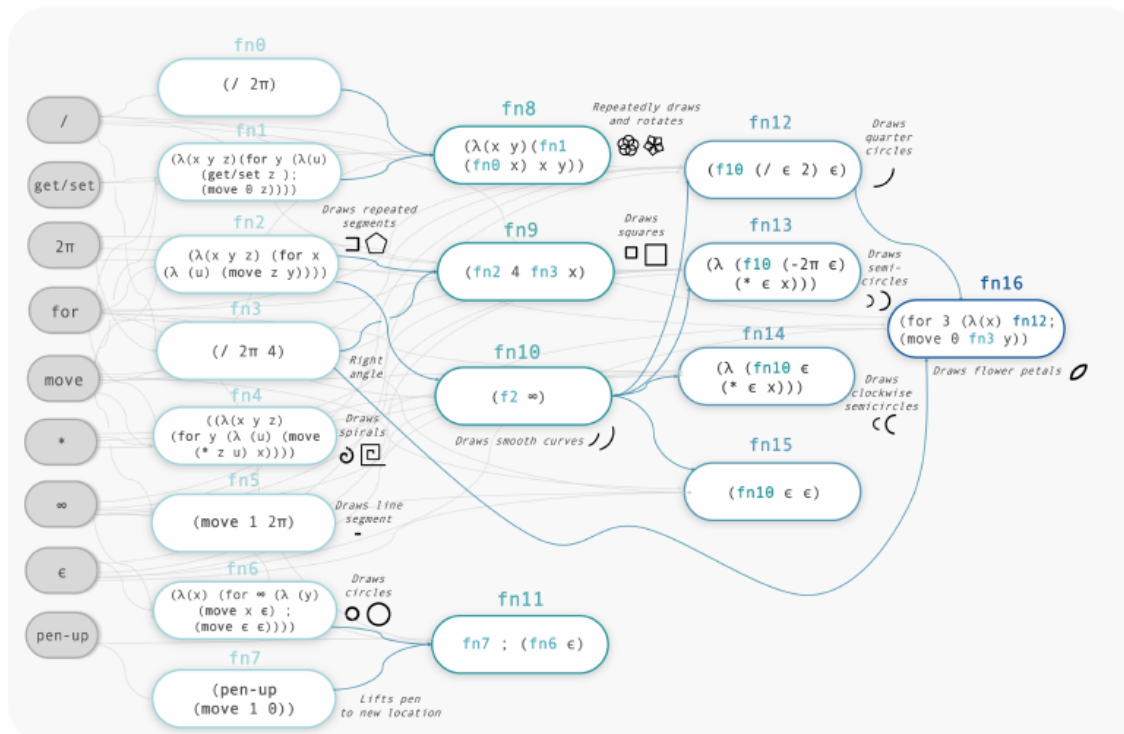


Figure: Learned library for LOGO graphics.

DreamCoder

Applications

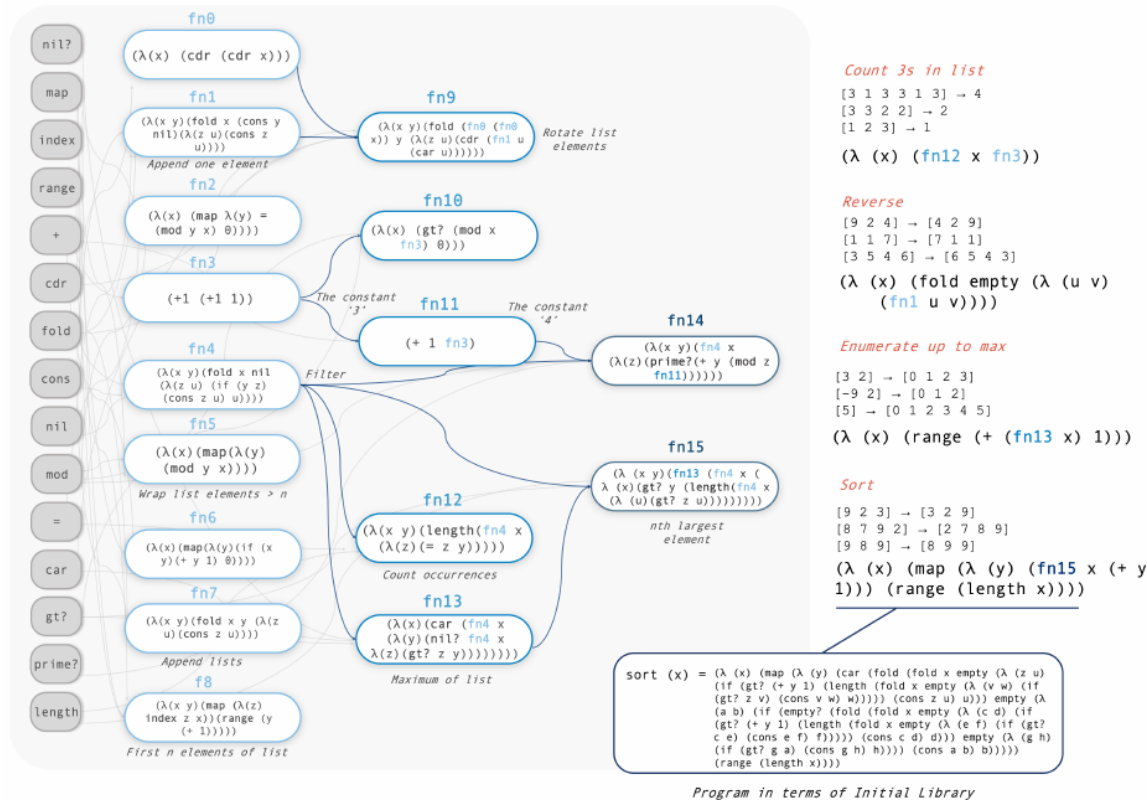


Figure: Learned library for list processing.

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

Different Types of Probabilistic Programming Systems

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

Different Types of Probabilistic Programming Systems

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

Different Types of Probabilistic Programming Systems