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

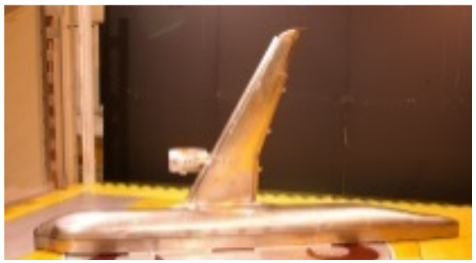
July 29, 2020

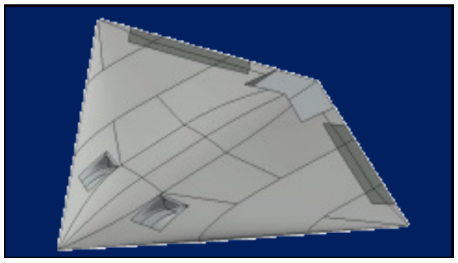
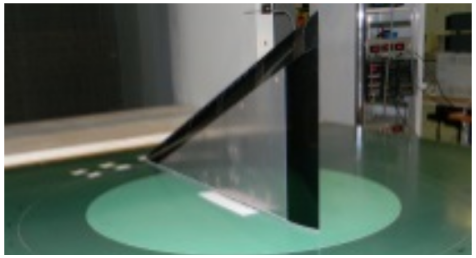
Table of Contents

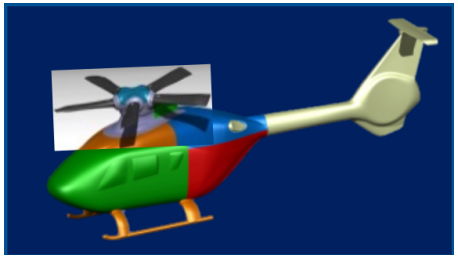
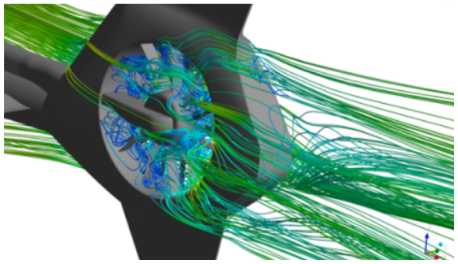
Outline

Overview

Aircraft- and Helicopter Aerodynamics, Prof.Dr. C. Breitsamter

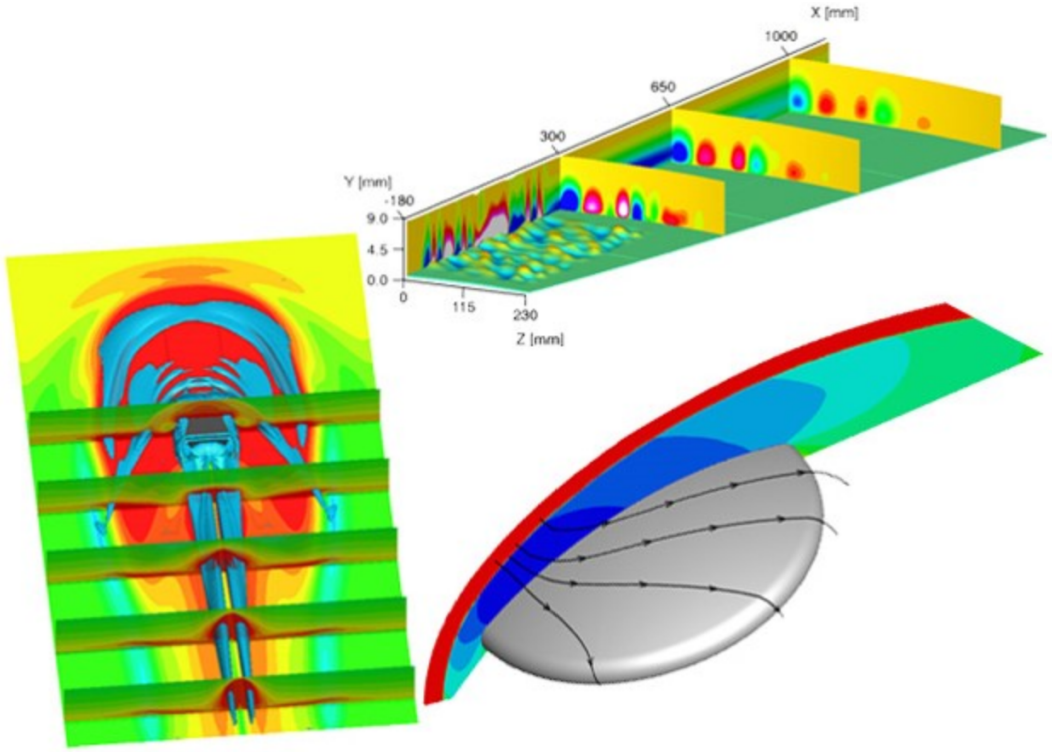


Overview

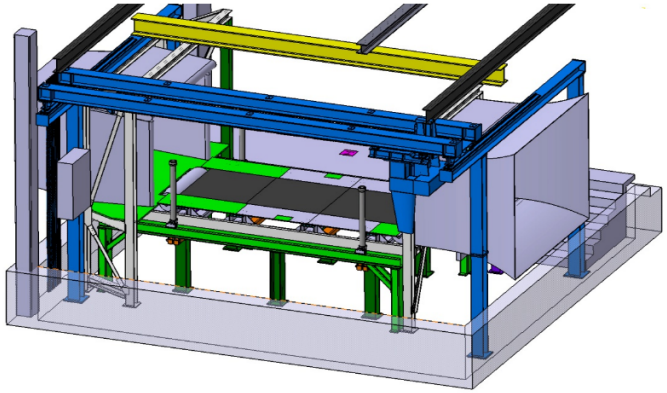
High-Speed Aerodynamics, apl.Prof.Dr. C. Stemmer

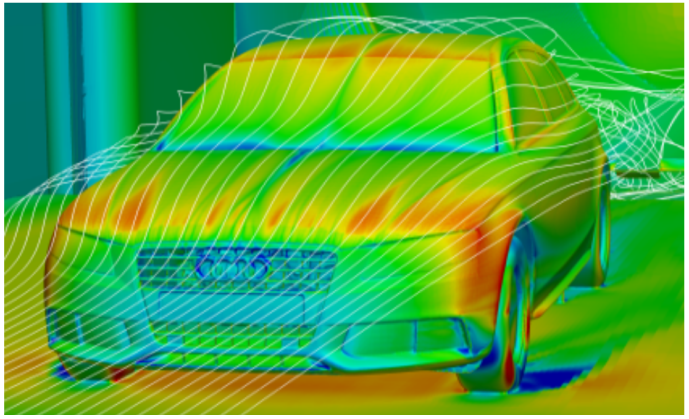


Overview

Automotive Aerodynamics, PD Dr. T. Indinger

- DrivAER
- Experimental investigation using a moving floor
- Numerical investigations using the lattice boltzmann method

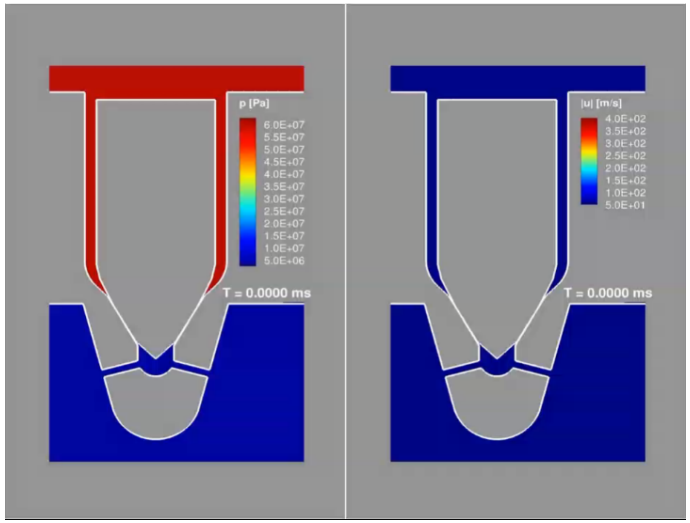


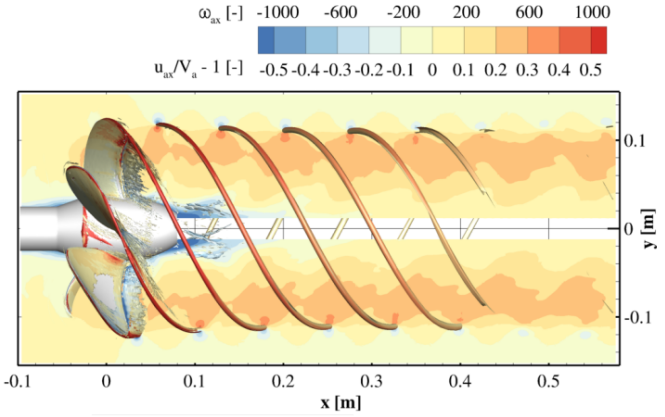


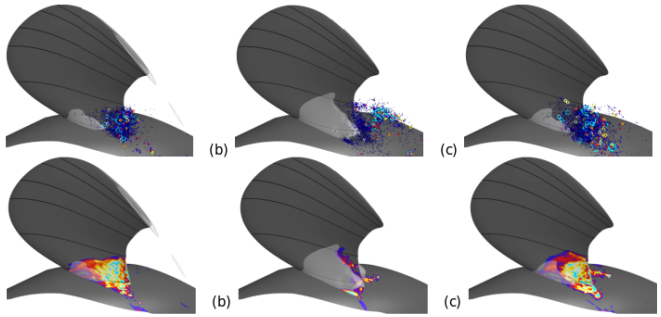
Overview

Compressible Flows, Dr. S. Schmidt

- Investigation of cavitation phenomena in multiphase-flows
- Interaction of turbulence and cavities
- Effects of primary jet breakup



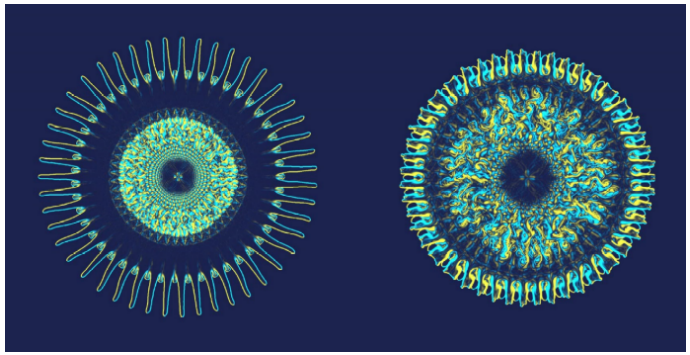


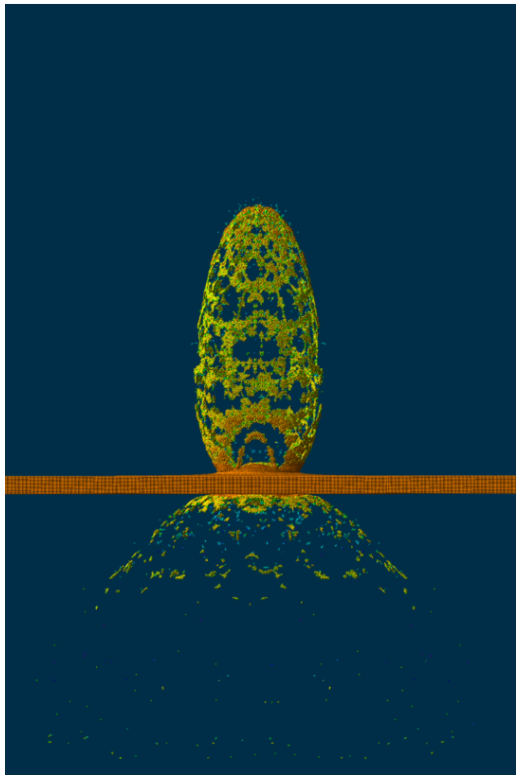


Overview

Complex Fluids, PD Dr. XY Hu, Dr. S. Adami

- Smoother particle hydrodynamics models for incompressible turbulent flows
- Numerical simulations of multiphase-flows using particle methods
- Microfluidics, Polymer and DNA solutions
- Waterjet- and Spray-dynamics

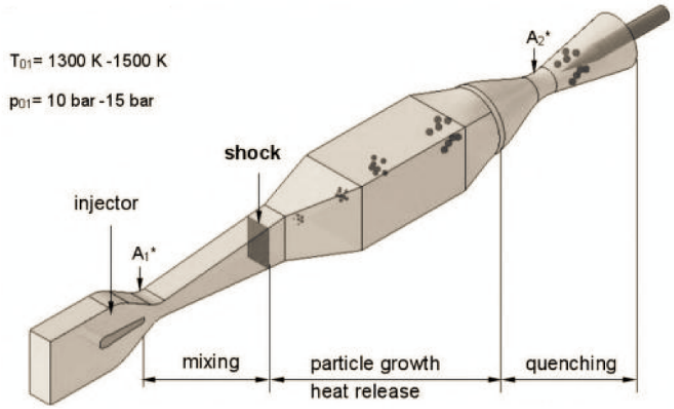


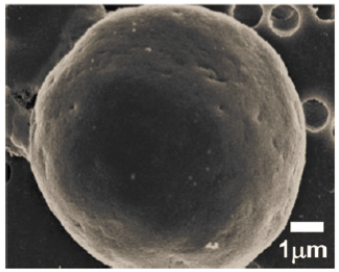


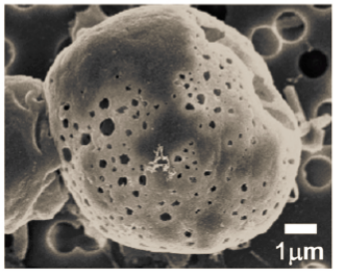
Overview

NANOSHOCK - ERC Advanced Grant, Dr. S. Adami, Dr. M. Giglmaier

- Shock-induced processes in multiphase-flows
- Exploration of contactless flow-manipulation in processing, nanotech and medtech





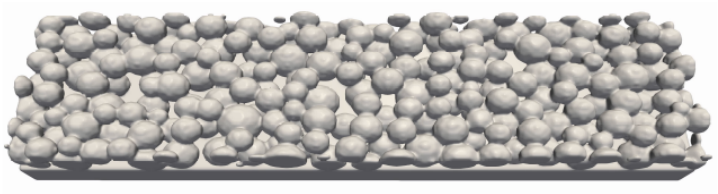


Overview

Digital Manufacturing - 3D Printing, Dr. S. Adami, Dr. M. Giglmaier

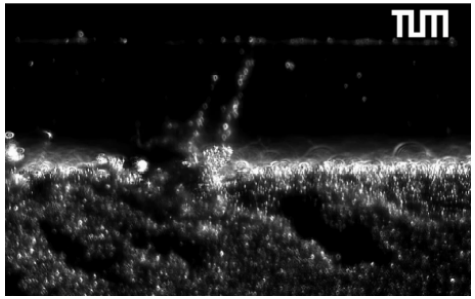
Example of a typical SPH melt-pool simulation

- Laser: P=70W, v=4m/s
- SPH particle: 805.000, x = 2.5μm
- Powder particles: 25μm, 5μm
- $\sim$ 1 CPUd on simple desktop workstation



Recording of lab-size experiment

- Inconel 718 Powder
- Laser: P=125W, v=2m/s
- Shielding gas: Argon



Outline

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

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

Lecture 3

- Bayesian Deep Learning
- Marrying Deep Learning Frameworks with Probabilistic Programming for Type 2 Machine Learning
  - Generative Adversarial Networks
  - Variational Autoencoder

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

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

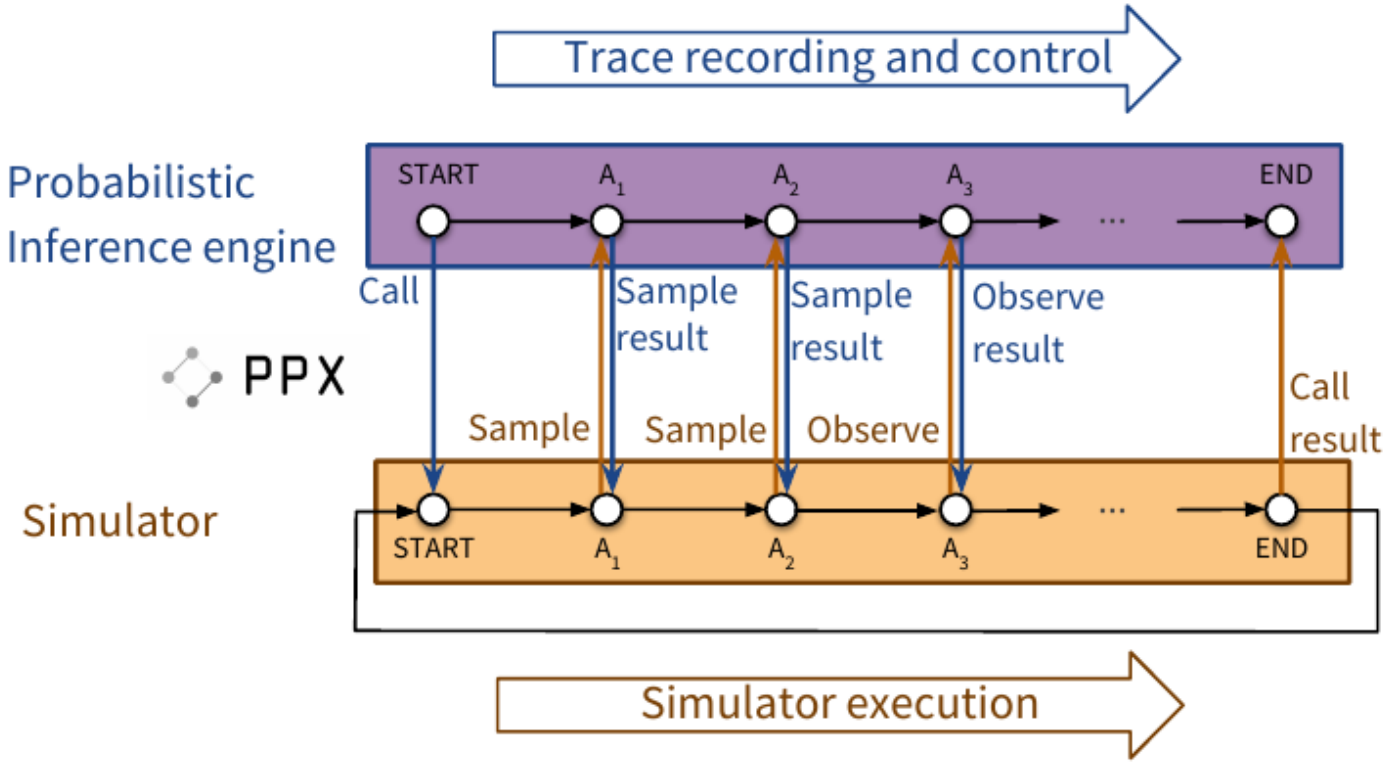


Figure: High-level view from the perspective of the probabilistic programming execution protocol (PPX).

Etalumis

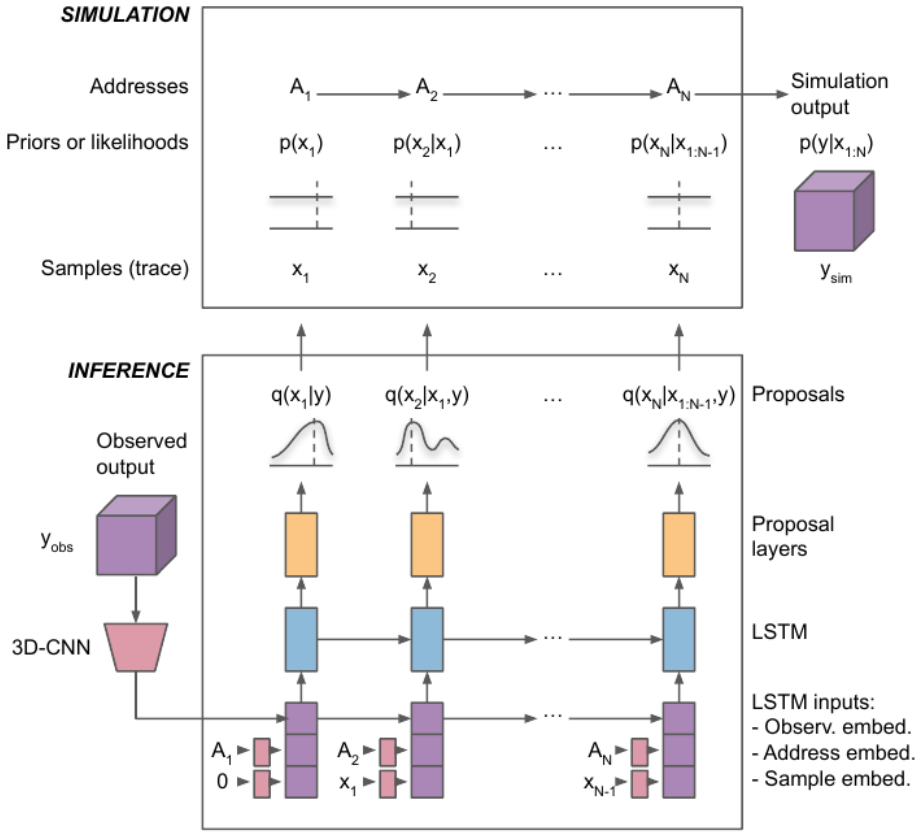


Figure: Detailed connection between simulation and inference in the probabilistic programming-based approach.

Etalumis

Recap: Amortized Inference [3] [4]

- Key idea: Learn from past inferences, to make future inferences run faster
  - Motivated by the brain's processing of information in context related information
- Constructs a parameterized guide program, which does not necessarily need to be a neural network
- Traditionally probabilistic programming systems utilize a form of inference, such as Monte Carlo, dynamic programming, or analytic computation to approximately solve an intractable integral from scratch on every invocation
- Simplified by constraining the control flow of the guide to the one of the original, in opposition to previous program induction approaches
- Manual construction at the stage of the mentioned papers. → see inference compilation

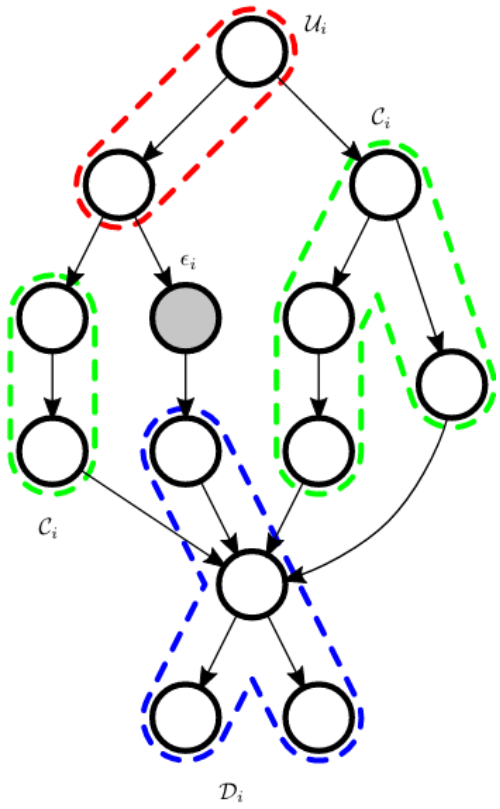
[3] Gershman, S. and Goodman, N., 2014. Amortized inference in probabilistic reasoning. In Proceedings of the annual meeting of the cognitive science society (Vol. 36, No. 36).

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

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

Etalumis

Recap: Amortized Inference




```
var graph = loadQMRGraph('qmr_graph.json'),
var data = loadData('qmr_data.json'),

var noisyOrProb = function(symptomNode, diseases) {
   var cp = product(map(function(parent) {
      return diseases[parent.index] ? (1 - parent.prob) . 1,
   }, symptomNode.parents)),
   return 1 - (1-symptomNode.leakProb)*cp,
},

var guideNet = nn.mlp(graph.numSymptoms, [
   {nOut. graph.numDiseases, activation. sigmoid}
], 'guideNet'),
var predictDiseaseProbs = function(symptoms) {
   return nneval(guideNet, Vector(symptoms)),
},

var model = function() {
   mapData({data. data, batchSize. 20}, function(symptoms) {
      var predictedProbs = predictDiseaseProbs(symptoms),
      var diseases = mapIndexed(function(i, disease) {
         return sample(Bernoulli({p. disease.priorProb}), {
            guide. Bernoulli({p. T.get(predictedProbs, i)})
         }),
      }, graph.diseaseNodes),

      mapData({data. symptoms}, function(symptom, symptomIndex) {
         var symptomNode = graph.symptomNodes[symptomIndex],
         observe(Bernoulli({p. noisyOrProb(symptomNode, diseases)}), symptom),
      }),
   }),
},
```

Etalumis

Recap: Amortized Inference

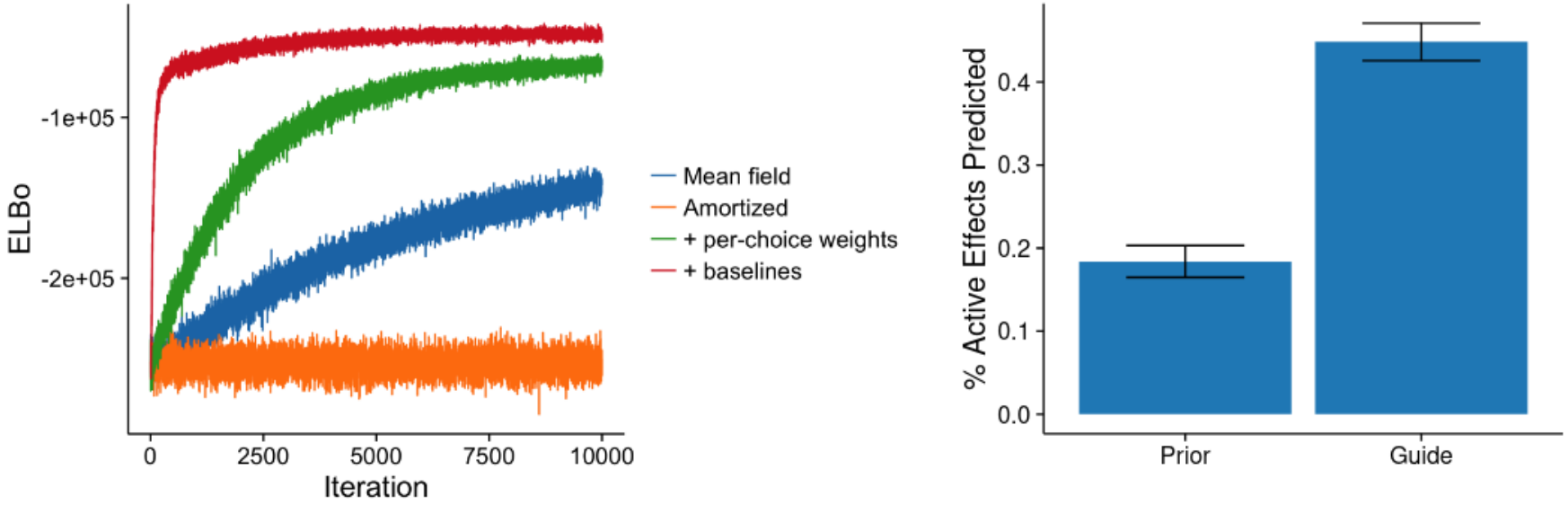


Figure: (Left) ELBo optimization progress. (Right) Percentage of test set active effects correctly predicted using latent causes from either the prior or the guide program.

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

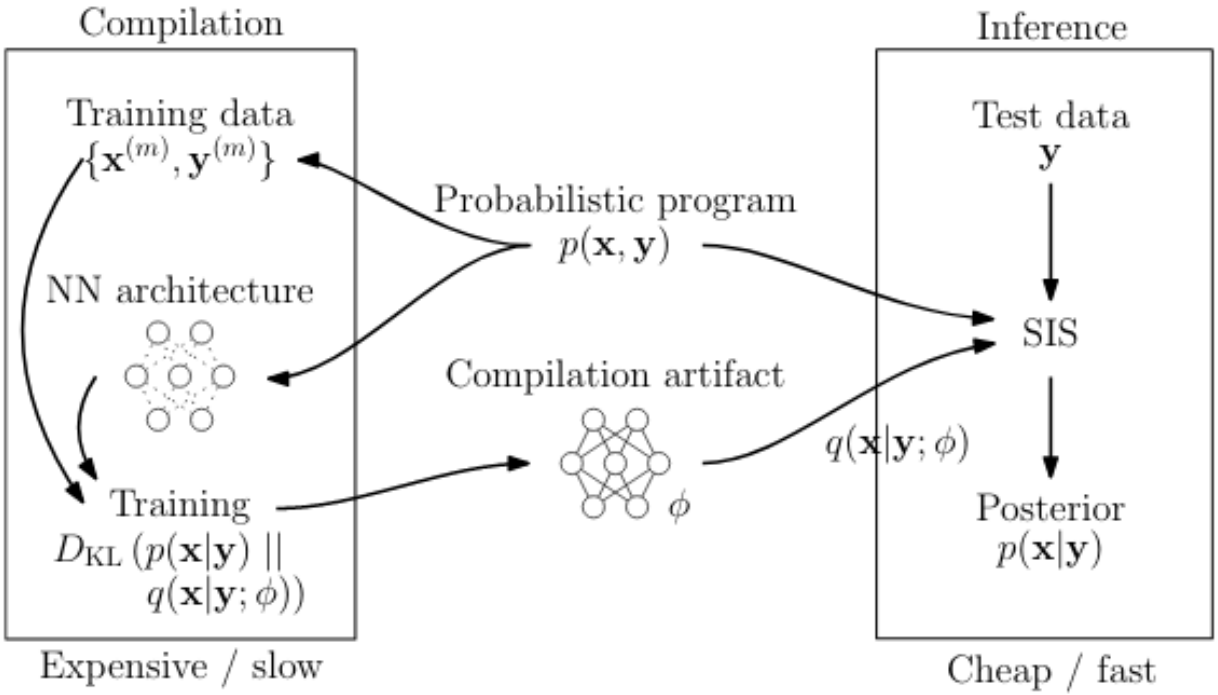


Figure: Automatic construction of a neural network surrogate, which is then trained with data generated by the probabilistic program to eventually act as the proposal distribution at inference time.

Etalumis

Outlook: Inference Compilation

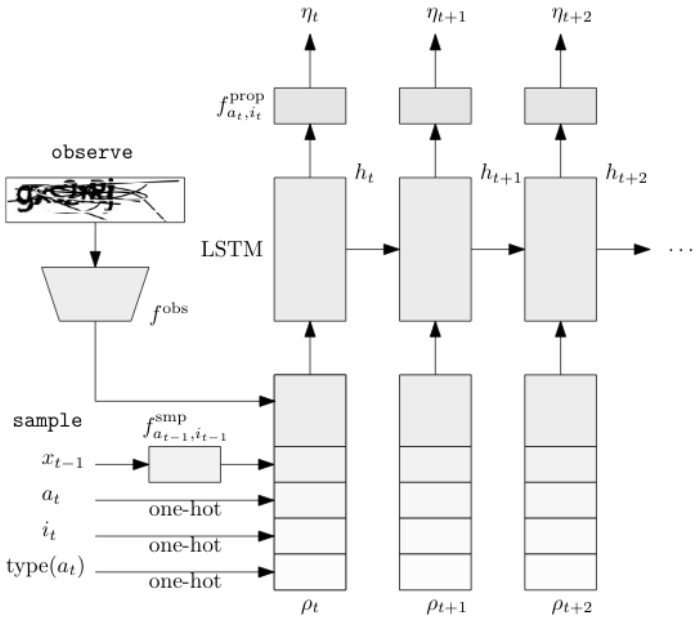


Figure: Example application for captcha solving based on probabilistic generative models for the captchas. With the LSTM at its core, required embeddings for the respective are attached adaptively.

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 (ϕ) 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 ϕ using $\nabla_\phi \mathcal{L}$ with e.g. ADAM, SGD, LARC, etc.
    end while
  end for

Etalumis

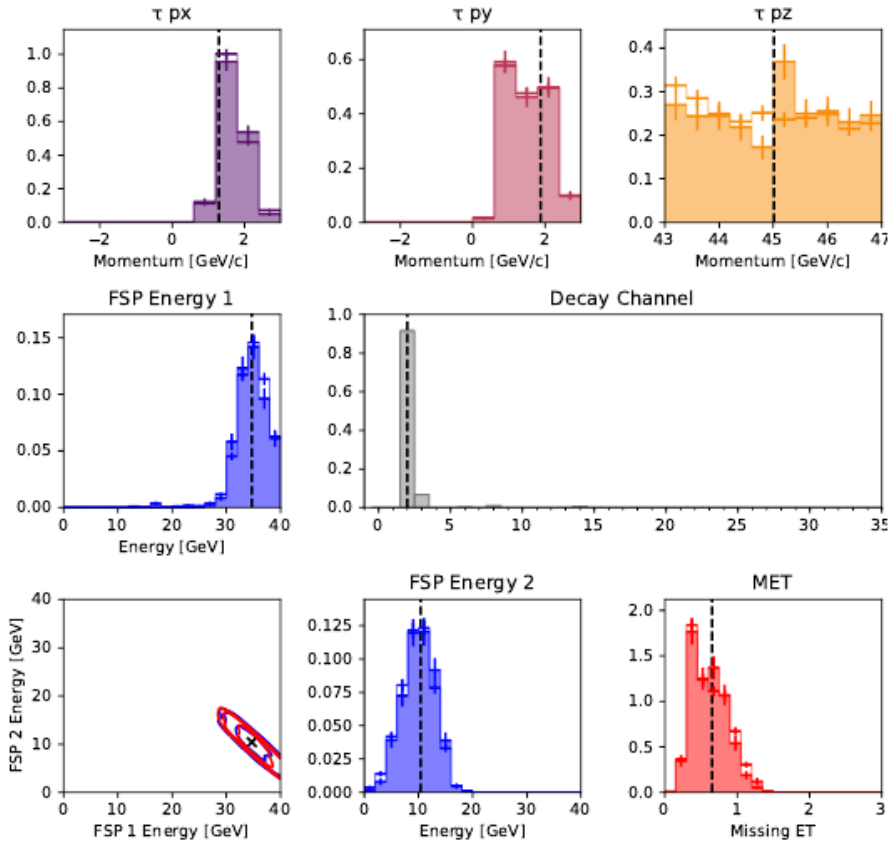
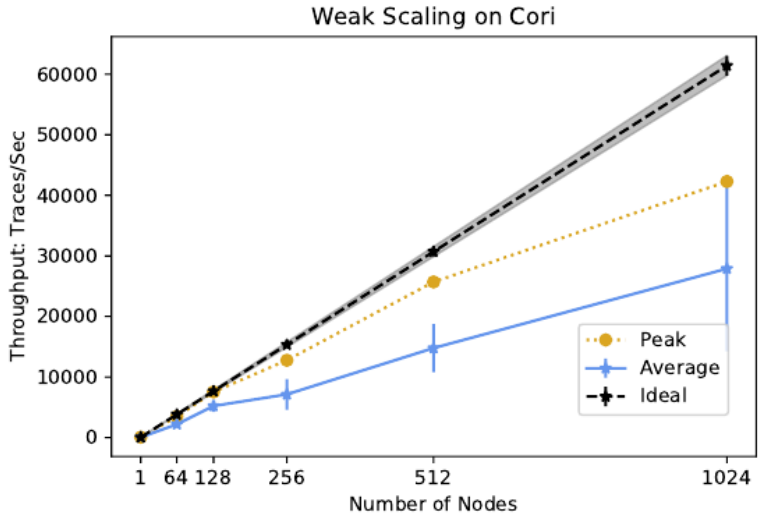
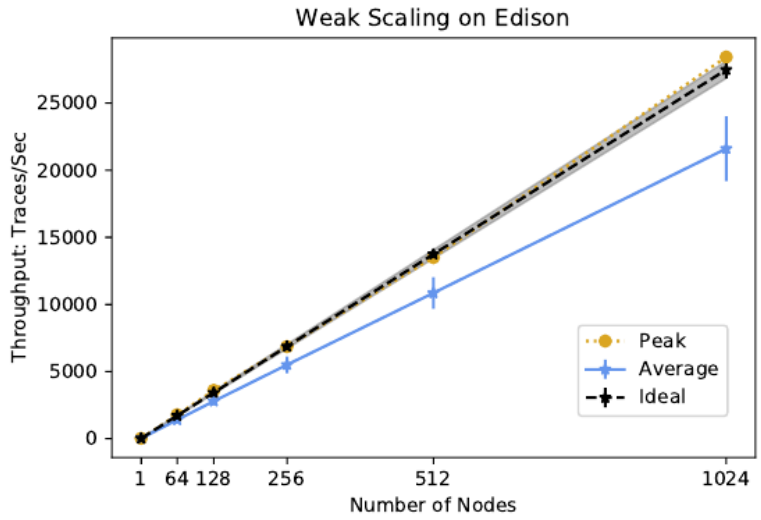


Figure: Posteriors obtained with random-walk Metropolis Hasting (filled histograms) and inference compilation (outline histograms) and ground truth values (dashed vertical lines).

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

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

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

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

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

DreamCoder

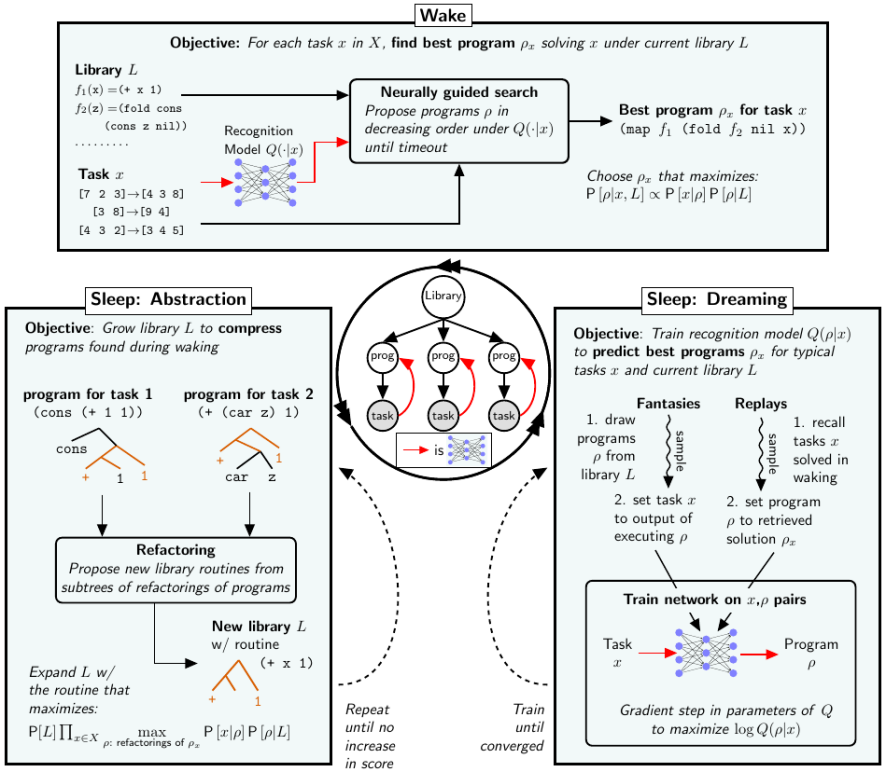


Figure: Algorithm Cycle of DreamCoder

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: $\mathcal{B}_x \leftarrow \mathcal{B}_x \cup \{\rho \mid \rho \in \text{enumerate}(\text{P}[\cdot|D,\theta], T) \text{ if } \text{P}[x|\rho] > 0\}$ ▷ Wake
        Train $Q(\cdot|\cdot)$ to minimize $\mathcal{L}^{\text{MAP}}$ across all $\{\mathcal{B}_x\}_{x \in X}$ ▷ Dream Sleep
        $\forall x \in$ batch: $\mathcal{B}_x \leftarrow \mathcal{B}_x \cup \{\rho \mid \rho \in \text{enumerate}(Q(\cdot|x), T) \text{ if } \text{P}[x|\rho] > 0\}$ ▷ Wake
        $\forall x \in$ batch: $\mathcal{B}_x \leftarrow$ top M elements of $\mathcal{B}_x$ as measured by $\text{P}[\cdot|x, D, \theta]$ ▷ Keep top M programs
        $D, \theta, \{\mathcal{B}_x\}_{x \in X} \leftarrow \textsc{Abstraction}(D, \theta, \{\mathcal{B}_x\}_{x \in X})$ ▷ Abstraction Sleep
        yield (D, θ), Q, $\{\mathcal{B}_x\}_{x \in X}$ ▷ Yield the updated library, recognition model, and solutions found to tasks
    end while
end while

DreamCoder

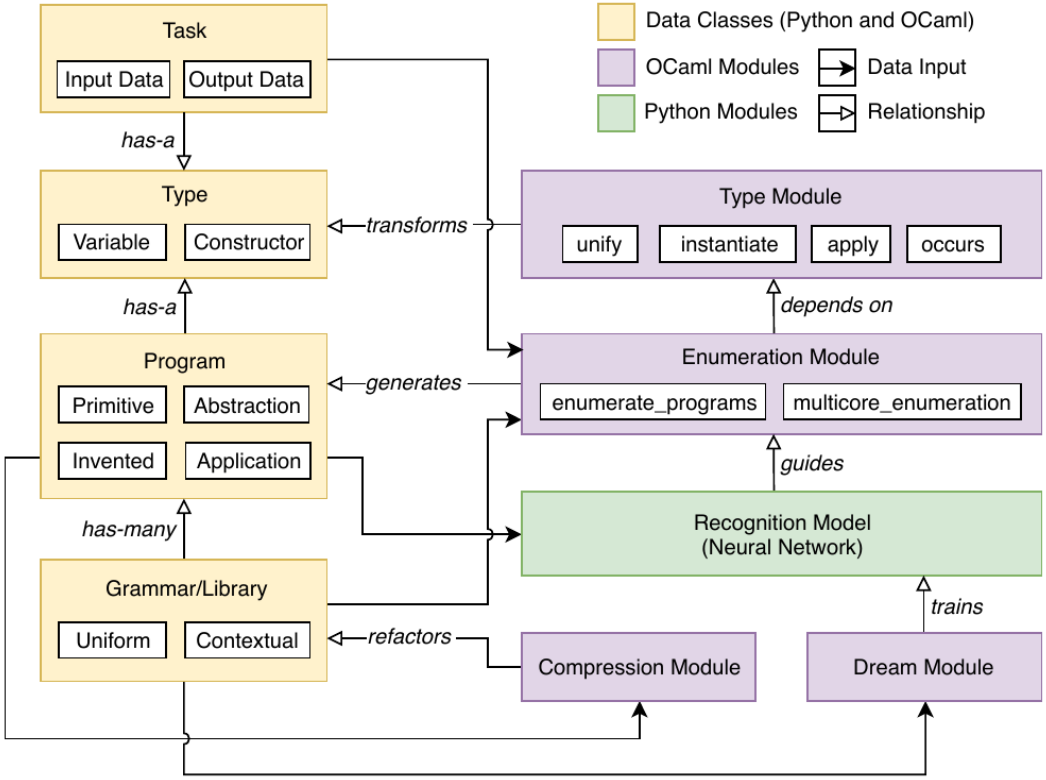


Figure: Different data-classes in DreamCoder.

DreamCoder

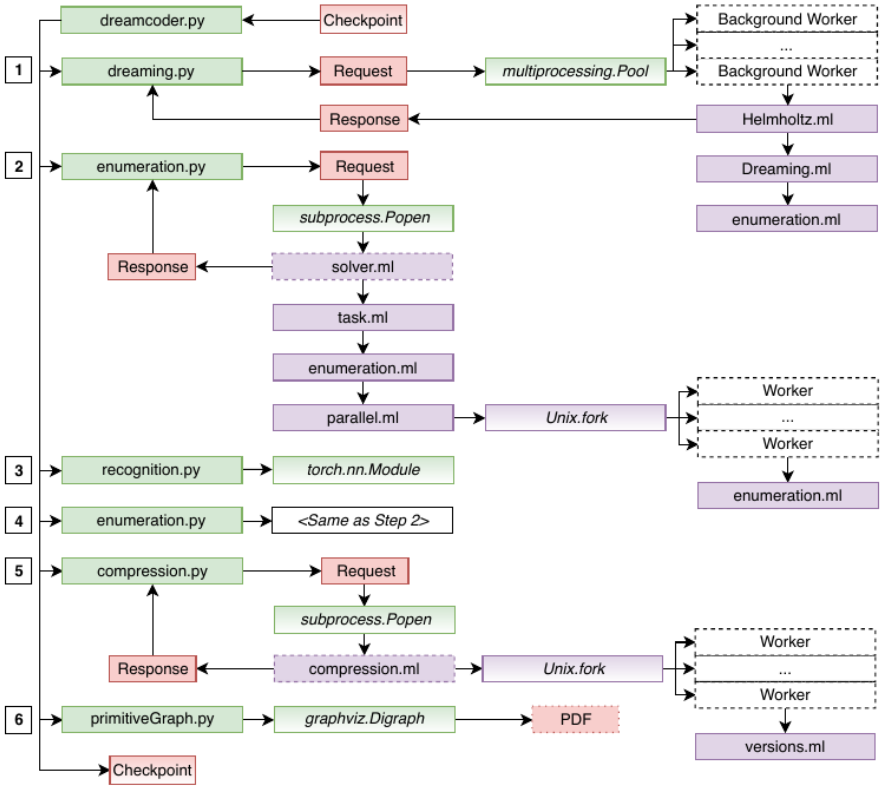
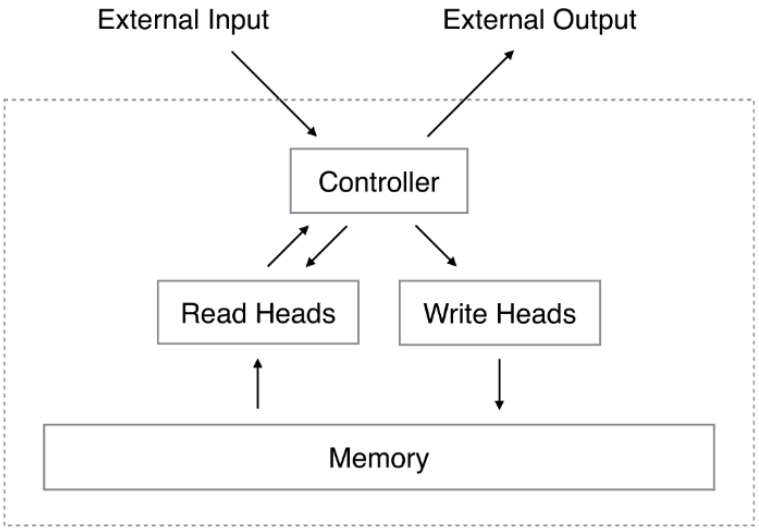


Figure: Program Flowchart: Phase 1, Dreaming. Phase 2, 1st Program Enumeration. Phase 3, Recognition Model Training. Phase 4, 2nd Program Enumeration. Phase 5, Abstraction (Compression). Phase 6, Library Visualization.

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

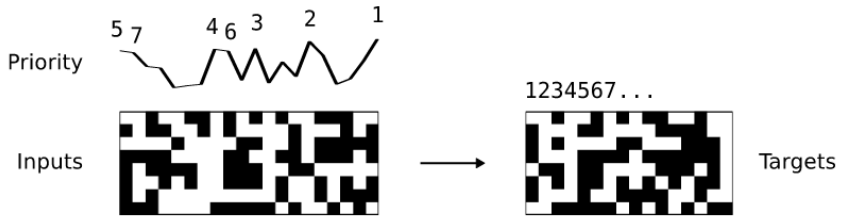
[9] Lilian Weng's review of attention: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

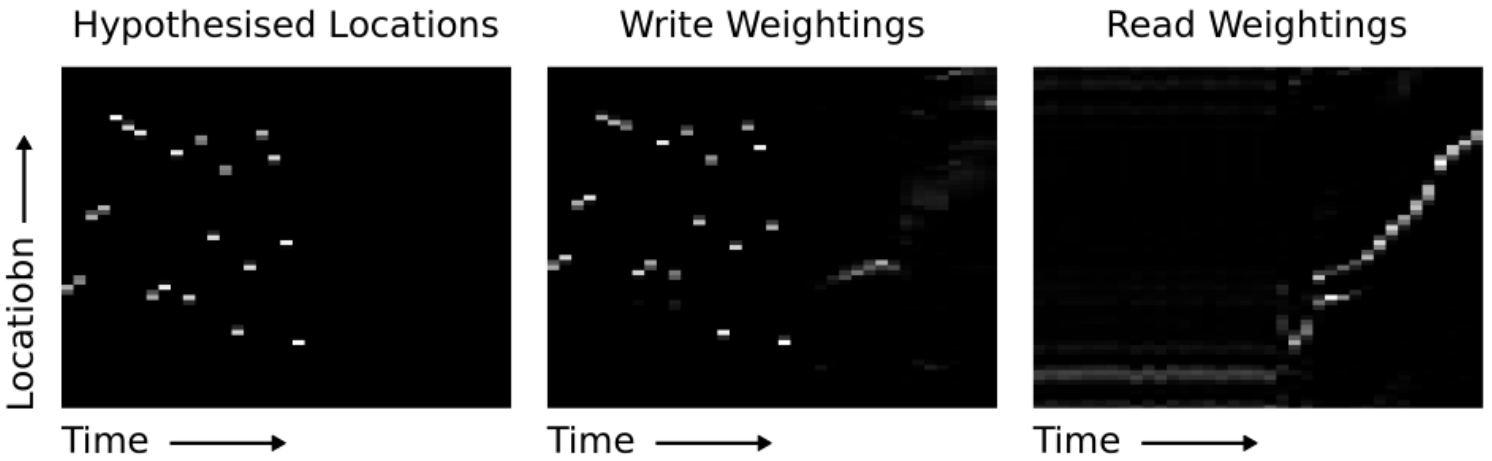
Chair of Aerodynamics and Fluid Mechanics
Department of Mechanical Engineering
Technical University of Munich

Recap: Helmholtz Machine

Example: Priority Sort

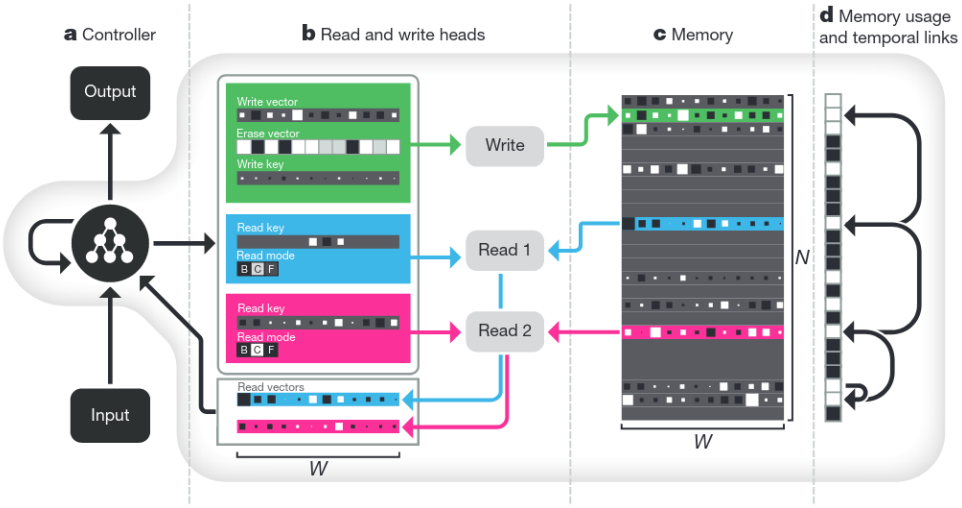
- Learns algorithm to sort data from a sequence of random binary vectors and their respective scalar priority rating





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!

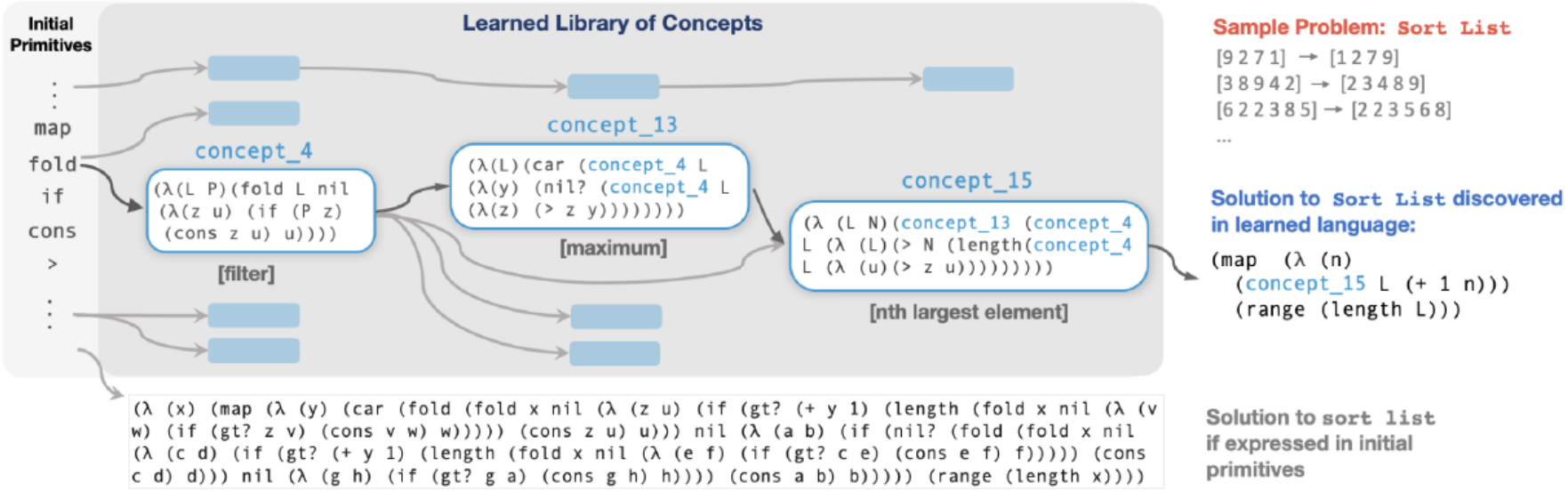


[10] Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S.G., Grefenstette, E., Ramalho, T., Agapiou, J. and Badia, A.P., 2016. Hybrid computing using a neural network with dynamic external memory. Nature, 538(7626), pp.471-476.

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

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
```

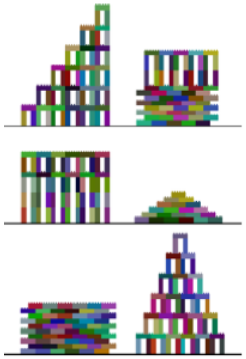
Double

```
[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]
```

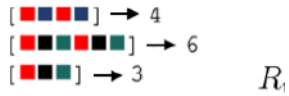
Block Towers



Recursive Programming

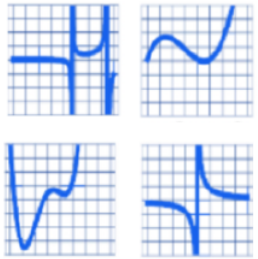
Filter Red

Length



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

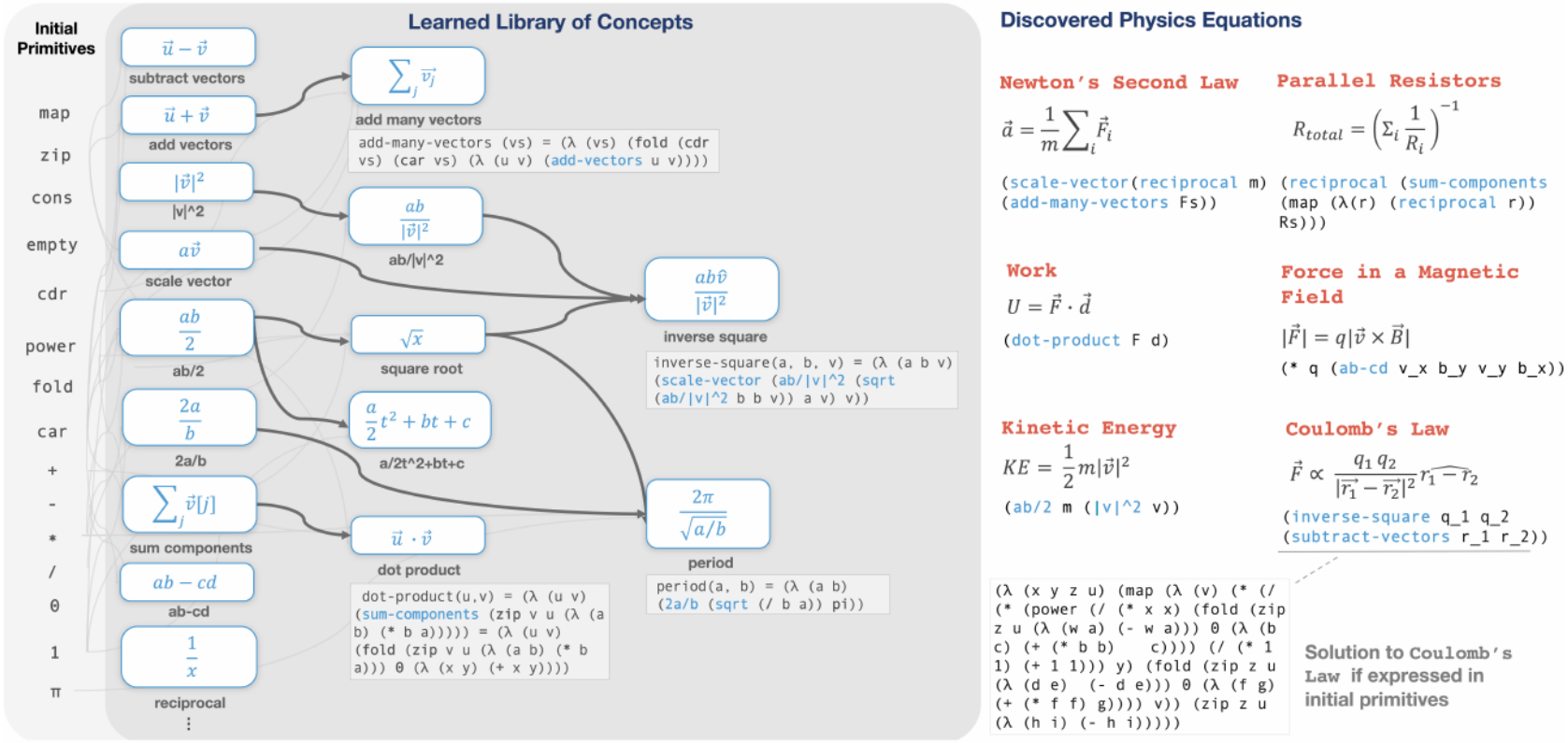


Figure: Learned library for physics equations.

DreamCoder

Applications

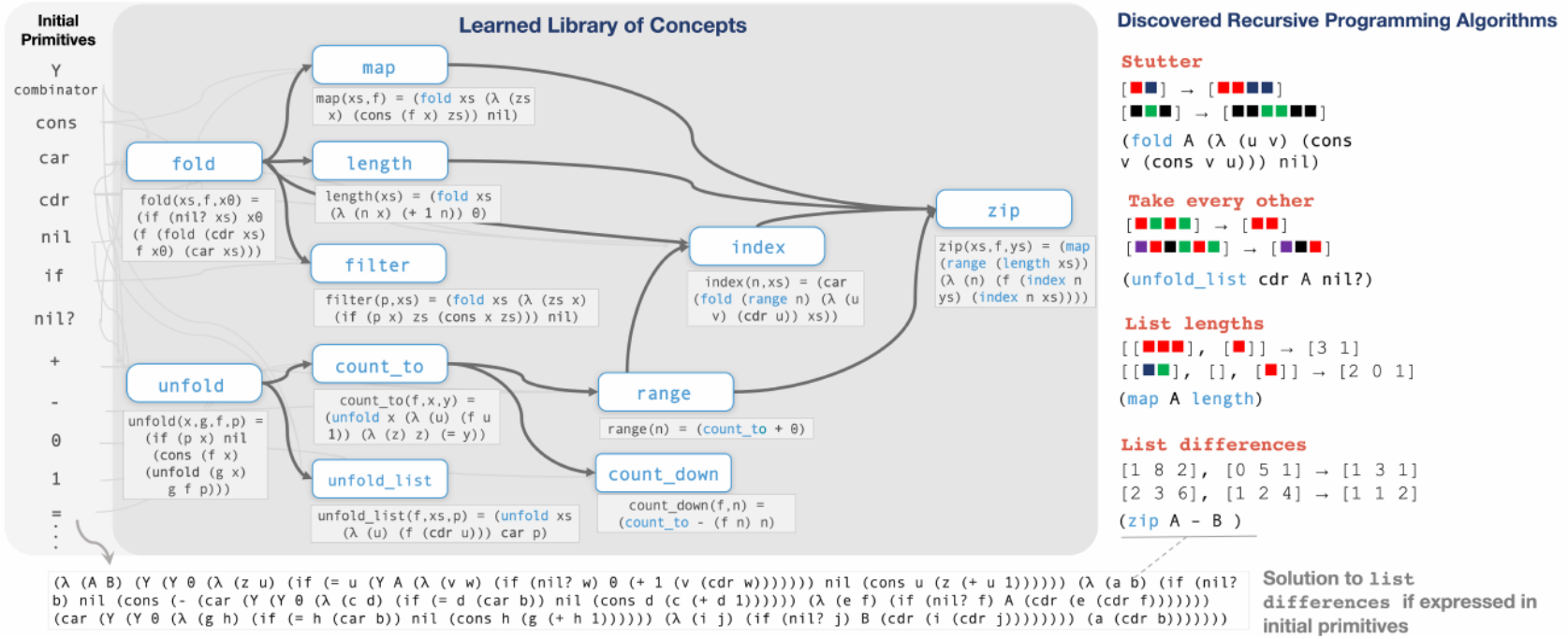


Figure: Learned library for recursive programming algorithm.

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

DreamCoder

Applications

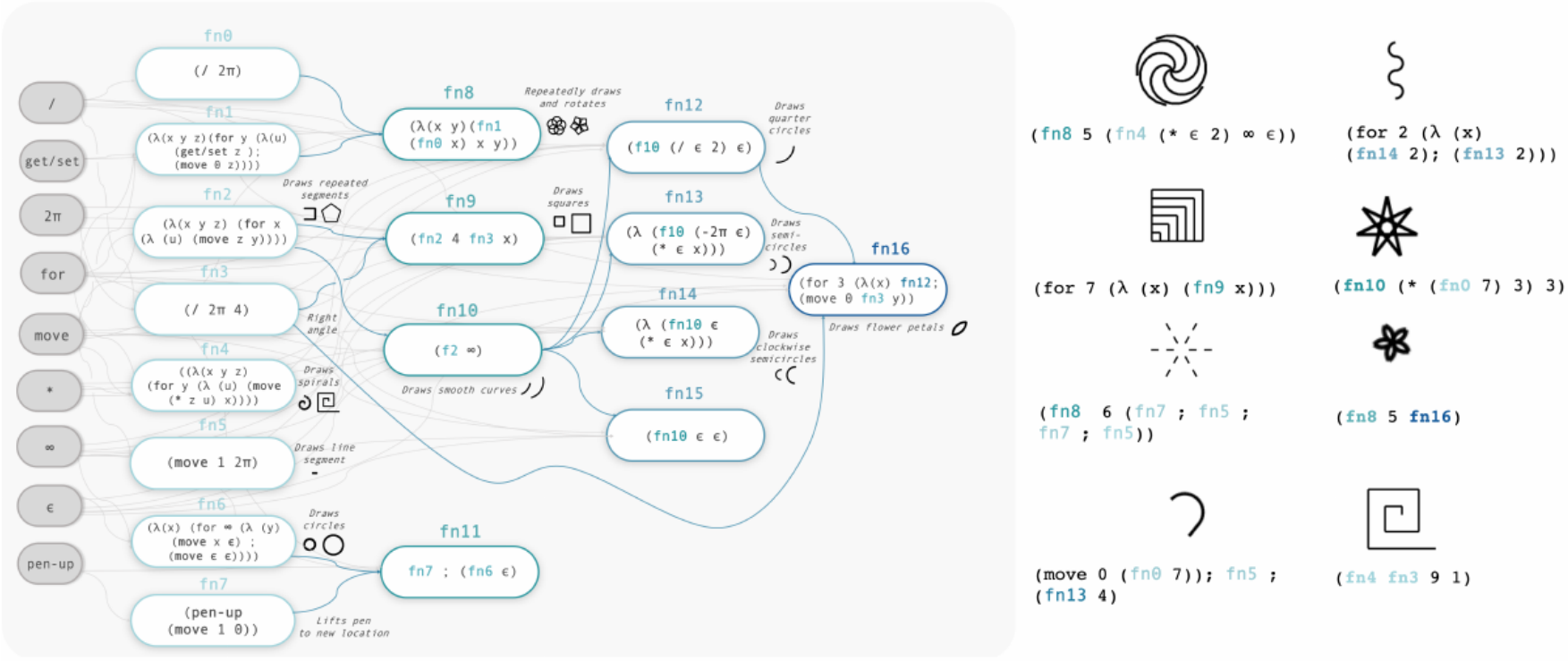


Figure: Learned library for LOGO graphics.

DreamCoder

Applications

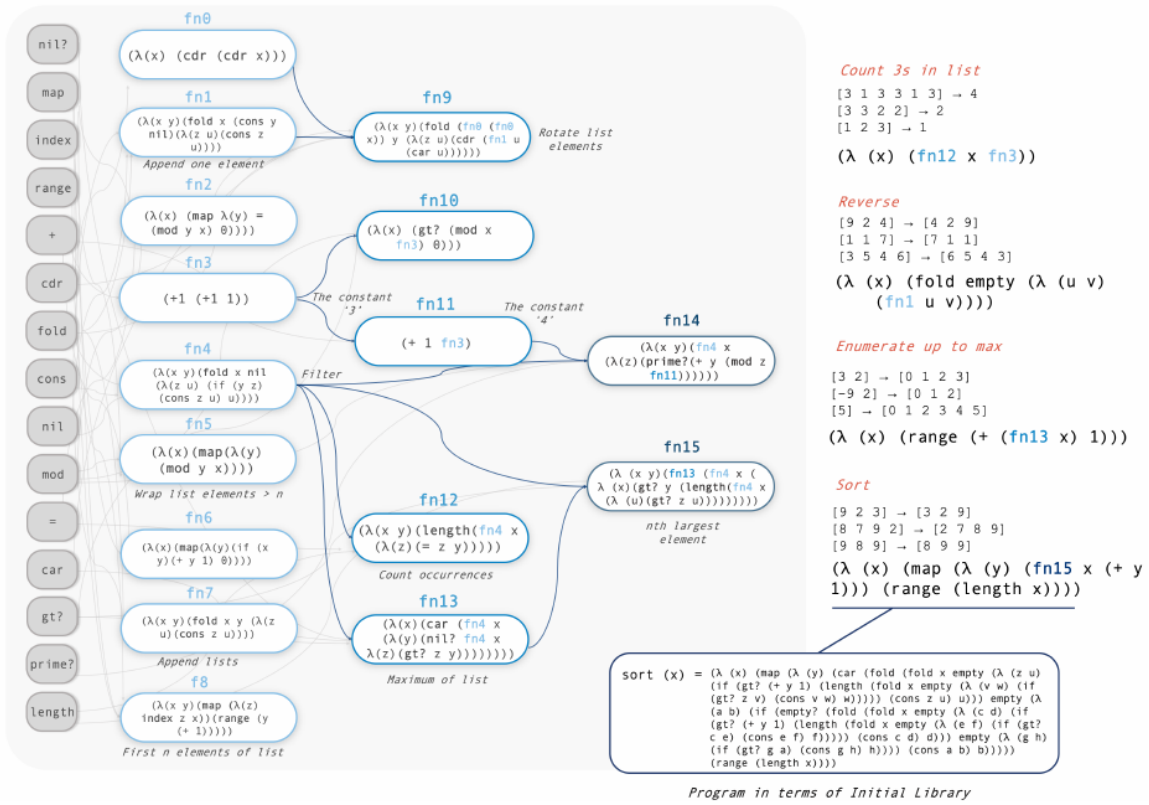


Figure: Learned library for list processing.

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

Why Do We Need Probabilistic Programming?

The Myth of Probabilistic Programming

Automated Inference

Not solved for high-dimensions and modern probabilistic models

Why Do We Need Probabilistic Programming?

Move away from handcoded inference routines to fast, verified ones for **reproducibility** and **focus on science**

Why Do We Need Probabilistic Programming?

Accelerate the probabilistic modelling research cycle

Why Do We Need Probabilistic Programming?

Natively express **uncertainties** over our model and workflow.

Why Do We Need Probabilistic Programming?

Express ever more **complex probabilistic models**

Why Do We Need Probabilistic Programming?

Express probabilistic models **as succinctly as possible**

Why Do We Need Probabilistic Programming?

Lower the barrier of entry to Bayesian analysis for data-scientists, engineers, etc.

Outline

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

Probabilistic Programming

Research Perspective

- Designing languages, which have expressive semantics
- Interpreters
- Compiler to translate inference problems into formal mathematical objects
- Integration with modern inference frameworks
  - Aiming to develop automated inference systems and/or general-purpose inference algorithms

Model-Based Reasoning [11]

- Random variables and probabilistic calculations are a core requirement → probabilistic machine learning
- All computational models we formulate are essentially approximations of reality
- Numerical models emulate stochasticity through the use of pseudorandom number generators, which are subsequently responsible for random phenomena and other uncertainties
- This leads to an explosion of the possible states of individual nodes in the graph of the program
- A good probabilistic model should then be able to represent all possible states of the real world

[11] van de Meent, J.W., Paige, B., Yang, H. and Wood, F., 2018. An introduction to probabilistic programming. arXiv preprint arXiv:1809.10756.

Model-Based Reasoning

- To really explore the space of possibilities for our model, we need a large population of model executions for statistical significance
- Our models allow us to measure value, for phenomena which also take place in the real world, these are our *observations*
- But we are unable to capture every detail of our model be it for physical reasons, or numerical constraints
- A welcome use for our probabilistic programming system is hence the statistical validation of theories, where inability to reproduce experimental results exposes weaknesses in our theoretical understanding
- Statistical decision-making under uncertainty does also constitute a direct application

Model-Based Reasoning

- What does usefulness mean in this context?
  - Reusable, interpretable abstraction
  - Reusable, non-interpretable but accurately generating data
  - Problem-specific simulators in engineering and the natural sciences
- But all models are parameterized
  - Amenable to model-fitting

Model-Based Reasoning

- How are such models denoted, and how are they manipulated to compute quantities of interest?
- Condition on sets of observation to arrive at a joint distribution
- Use Bayes' rule [12] [13]

$$p(X|Y) = \frac{p(Y|X)p(X)}{p(Y)} = \frac{p(X,Y)}{p(Y)} = \frac{p(X,Y)}{\int p(X,Y)dX}$$

- A model is then the joint distribution $p(Y,X) = p(Y|X)p(X)$ of the observations Y and the ranom choices made in the generative model X, which we know as the latent variables
- Setup encapsulates a large number of problems in the sciences

[12] Davidson-Pilon, C., 2015. Bayesian methods for hackers: probabilistic programming and Bayesian inference. Addison-Wesley Professional.

[13] Pfeffer, A., 2016. Practical probabilistic programming. Manning Publications Co..

Model-Based Reasoning

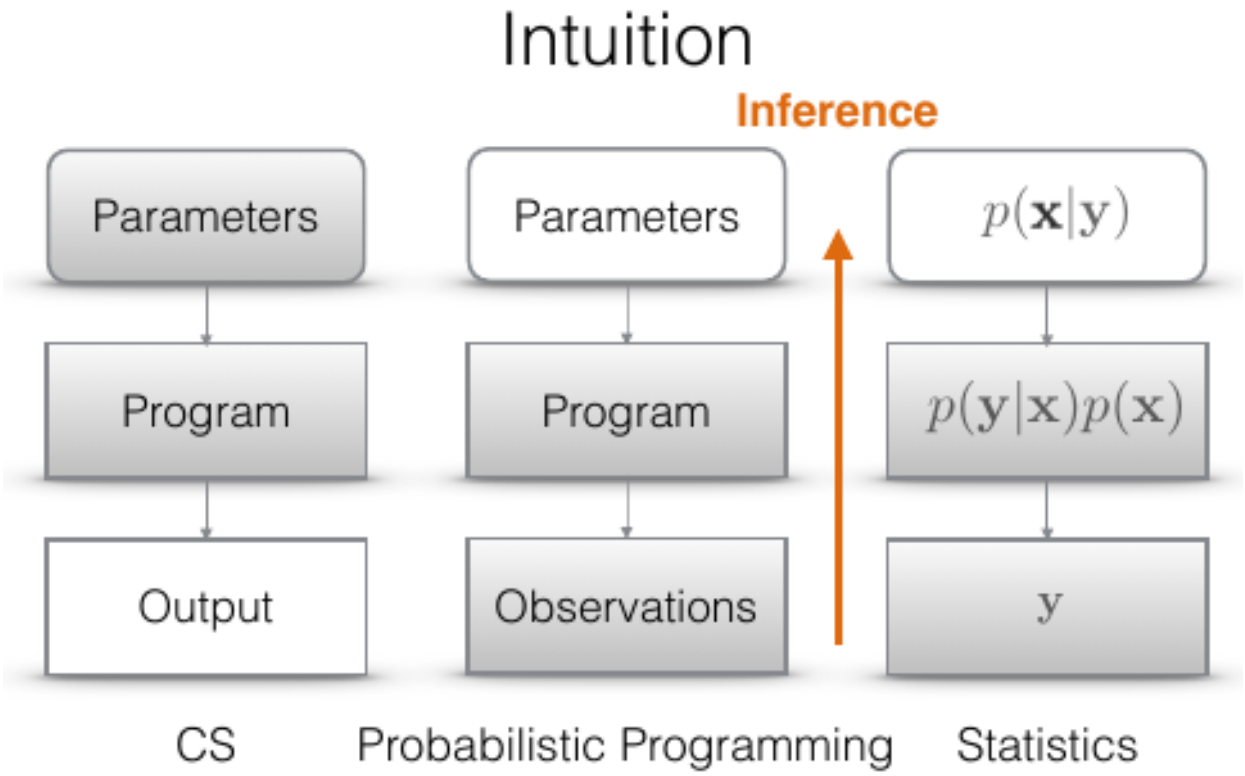
- Probabilistic programming system scale type of problems pairs:

X	Y
scene description	image
simulation	simulator output
program source code	program return value
policy prior and world simulator	rewards
cognitive decision making process	observed behavior

Model-Based Reasoning

- The first such pair, was already successfull used to solve Captcha using general-purpose inference
- All that was needed was a probabilistic program, which could generate similar Captchas
- As there are a great number of distributions we are unable to perform exact inference on, we have to rely on approximate inference algorithms
- We seek to characterize the posterior distribution of all random choices made during the forward exection, given that the model produces a specific output.
- Key here is always the ability of the probabilistic programming system to condition on existing values
- **Expressivity is not limited to traditional statistical models!**

Model-Based Reasoning



Model-Based Reasoning

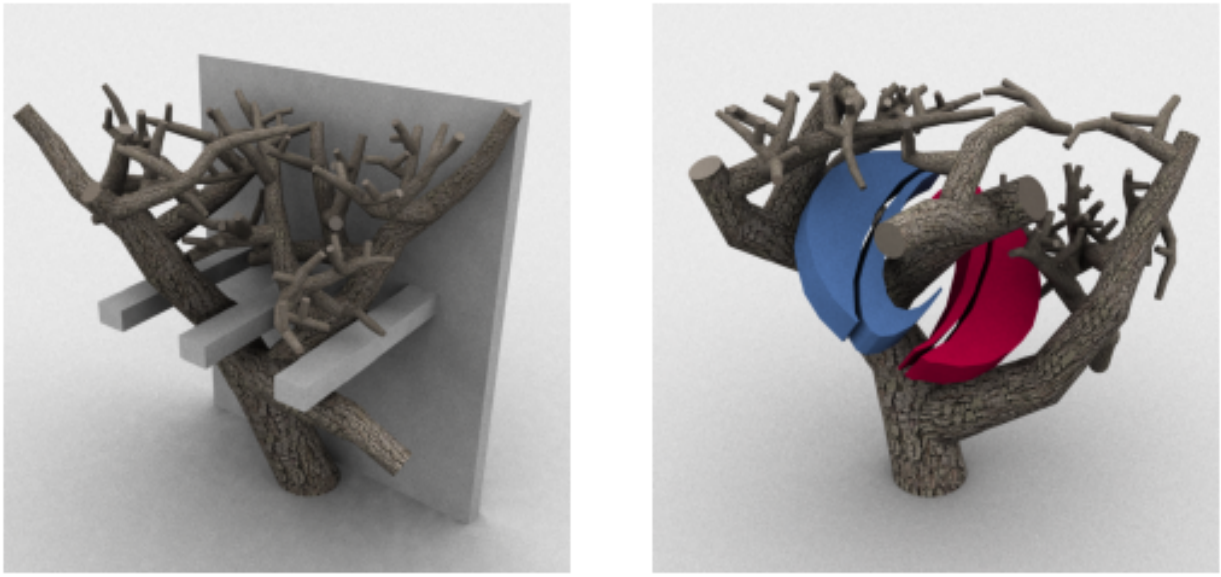


Figure: Constrained Simulation

Models with Static Computation Graphs

- A first-order probabilistic programming language (FOPPL) already includes the main syntax of a general-purpose programming language
- First-order means:
  - Functions cannot accept other functions as arguments
  - No recursivity
  - Ensures there to be only be a finite number of random variables in any model
- Any program expressable in the FOPPL can be compiled into the respective graphical model [14]
- Highly-suited to express any graphical-model data-structures

[14] Koller, D. and Friedman, N., 2009. Probabilistic graphical models: principles and techniques. MIT press.

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

Models with Static Computation Graphs

- The reduced FOPPL is defined by the following eight rules:
  1. A constant c can be a vlue of primitive data type
  2. A variable v is a symbol that references the value of another expression in the program.
  3. A let form (let [v e_1] e_2) binds the value of the expression e_1 to the variable v
  4. An if form (if e_1 e_2 e_3) takes the value of e_2 when the value of e_1 is logically true and the value of e_3 when e_1 is logically false
  5. A function application (f e_1 ... e_n) call the user-defined function f
  6. A primitive procedure applications (c e_1 ... e_n) calls a built-in function c
- Probabilistic programming language specific features
  1. A sample form (sample e) represents an unobserved random variable. It accepts a single expression e, which must evaluate to a distribution object, and returns a value that is a sample from this distribution
  2. An observe form (observe e_1 e_2) represents an unobserved random variable. It accepts an argument e_1, which must evaluate to a distribution, and conditions on the next argument e_2, which is the value of the random variable

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

Models with Static Computation Graphs

```
(defn hmm-step [t states data trans-dists likes]
  (let [z (sample (get trans-dists
                       (last states)))]
    (observe (get likes z)
             (get data t))
    (append states z)))

(let [data [0.9 0.8 0.7 0.0 -0.025 -5.0 -2.0 -0.1
            0.0 0.13 0.45 6 0.2 0.3 -1 -1]
      trans-dists [(discrete [0.10 0.50 0.40])
                   (discrete [0.20 0.20 0.60])
                   (discrete [0.15 0.15 0.70])]
      likes [(normal -1.0 1.0)
             (normal 1.0 1.0)
             (normal 0.0 1.0)]
      states [(sample (discrete [0.33 0.33 0.34]))]]
  (loop 16 states hmm-step
    data trans-dists likes))
```

Figure: A hidden Markov model expressed in the FOPPL.

Models with Dynamic Computation Graphs

- Higher-order probabilistic programming languages (HOPPL) extend upon FOPPLs by providing higher-order procedures and general recursion
- Can theoretically denote models with an unbounded number of random variables
  - The end for graph-based inference strategies
- Evaluation-based inference is still a viable option, when we constrain ourselves to a finite number of random variables at any one time
- Removal of a static loop depth requires function recursion → HOPPL-feature
- As stochastic recursion can easily implement an infinite number of random variables, we need to be aware of this potential trap

Models with Dynamic Computation Graphs

```
(defn geometric [p]
 (let [dist (flip p)]
    (if (sample dist)
      0
      (if (sample dist)
        1
        (if (sample dist)
          2
          ⋮
            (if (sample dist)
               ∞
               (geometric-helper (+ ∞ 1))))))))
```

Figure: Example with infinite stochastic recursion, which hence needs constraining

Models with Dynamic Computation Graphs

- Expanding to the HOPPL now allows for the embedding in any programming language as the modelling language
- Functions can be declared locally in the HOPPL
- Languages like Church, Venture, Anglican, WebPPL, Probabilistic-C, Turing, and CPProb all have these features or require containment of these features in the language to reason about them.
- If the HOPPL is enriched with a desugared *let* and *foreach* it becomes a superset of previous FOPPL
- Are able to define open-universe models, which are able to adjust their number of clusters/mixtures according to the size of the dataset
- Are able to define constrained sampling algorithms

Models with Dynamic Computation Graphs

- Using the HOPPL we are also able to construct a generative model for mathematical functions akin to what DreamCoder does in part
- The system will write its own code for the function bodies
- Inference problem becomes one of samping from the space of all function $f(x)$
- Funtion returns the source call for a new function
- The number of combinators supplied to the function controls the size of the space of all functions
- If we seek to invoke the generated function, we have to use $eval$, which can the be used to evaluate the validity of the results for specific target data
- To construct inference routines in the HOPPL we still need to develop run, $interrupt$, $resume$, and $fork$.

Advanced Topics

Model Learning

- So far the history of machine learning has had one prevailing constant: Data-driven discriminative techniques are superior to hand-designed algorithms if enough labeled training data is available
- Goal in probabilistic programming: **learn** the generative model itself
- To do this in any general kind of fashion one has to follow a bottom-up approach in which one starts with the observations and learns the distribution over the latent parameters
- To do this generally we have to rely on neural networks as transformation approximators

Advanced Topics

Model Learning

- As writing a good generative model is a hard task in itself learning the model with intelligent agents is the obvious next step
- This is in line with the views on agents of many cognitive scientists:
  - An agent constructs and reasons in models of its own worls and uses its world-model to compute the expected utility of its actions
- The direct choice for this is the Helmholtz machine [15]
  - Posits an intertwined forward- and backwards model with an internal representation learning algorithm, which writes directly onto the latent space of the model
- A further development of this idea is the variational autoencoder [16]
- The neural Turing machine, or differentiable neural computer as developed by Graves are also highly interesting candidates

[15] Dayan, P., Hinton, G.E., Neal, R.M. and Zemel, R.S., 1995. The helmholtz machine. Neural computation, 7(5), pp.889-904.

[16] Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114.

Advanced Topics

Model Learning

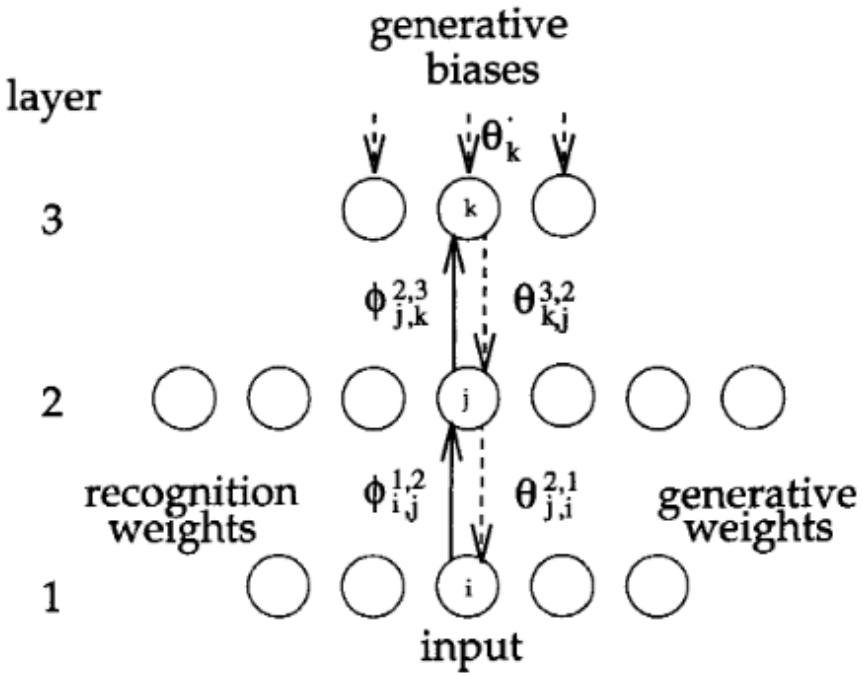


Figure: A 3-layer Helmholtz machine modelling its inputs in a two-stage hierarchical model.

Advanced Topics

Model Learning

- As the decoder has no prior structure it needs to learn the complete program structure from data
  - Program induction research suggests it a good idea to impose structure
- No general proof of the impeding supremacy of the bottom-up approach on the horizon yet

Advanced Topics

Nesting

- Nesting probabilistic programs is an obvious extension of current ideas, i.e. treating a higher-order probabilistic program as a distribution and using it inside of another higher-order probabilistic program
  - Could directly result in a doubly intractable inference problem
- Requires more language-features for a clean inclusion
- The meaning of such a program would also not be mathematically clear
- Inference in such a structure must be performed with the utmost care
  - Pay close attention to convergence rates
  - Utilize sampling methods for nested cases
- If it were to work it would, amongst other models, enable intelligent multi-agent application

Outline

Summary

- Probabilistic programming enables highly thought-provoking applications, which require interaction between statistical inference routines and machine-learning building blocks
  - More on this tomorrow!
- The concept of conditioning enables many interesting applications across science (if inference can be made tractable)
  - More on inference routines later today!
  - Outlook onto science applications tomorrow!
- Probabilistic programming systems need to be able to express our desired class of models
  - More on the different types of probabilistic programming systems later today!