

Models and Programs: Better Together

Sriram Rajamani
Microsoft Research, India

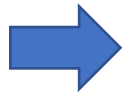
November 2020

Two ways to construct computer systems

Programming

Given a specification over input and output, construct a program that satisfies the specification

Sort a sequence of numbers in ascending order

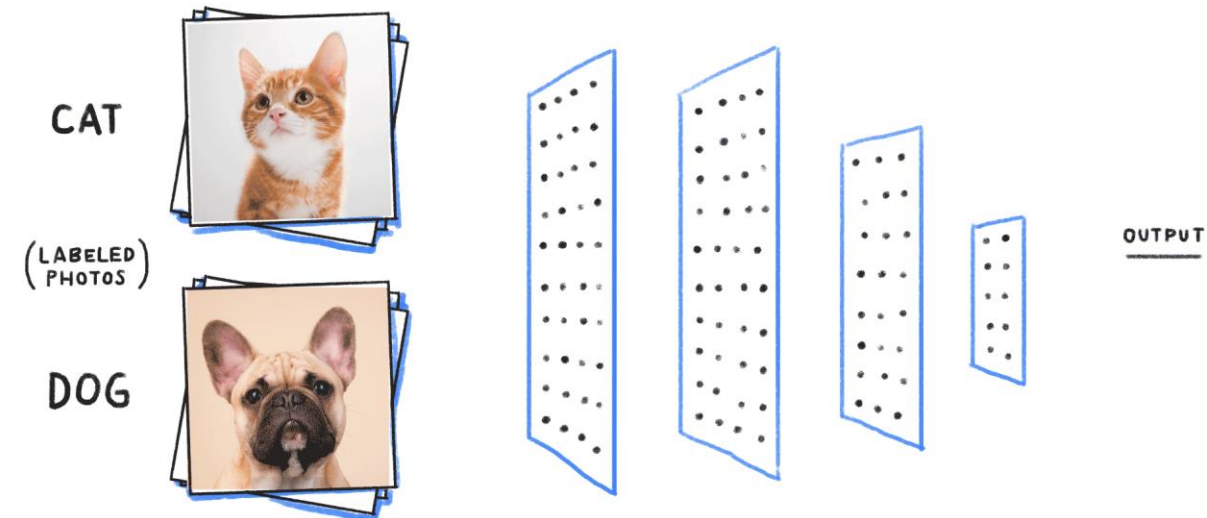


```
QuickSort(A,p,q):  
if p < q then  
  r := Partition(A, p, q)  
  QuickSort(A, p, r-1)  
  QuickSort(A, r+1, q)
```

Initial call:
QuickSort(A, 0, n-1)

(Supervised) Learning

Given a set of training examples (input-output pairs) learn a model that generalizes and learns the transformation from input to output



Two ways to construct computer systems

Programming

Given a specification $\varphi(x, y)$ over input and output, construct a program P such that $\forall x. \varphi(x, P(x))$

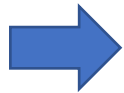
(Supervised) Learning

Given a set of training examples

$$T = \{ (x_i, y_i) \mid i = 1 \dots N \}$$

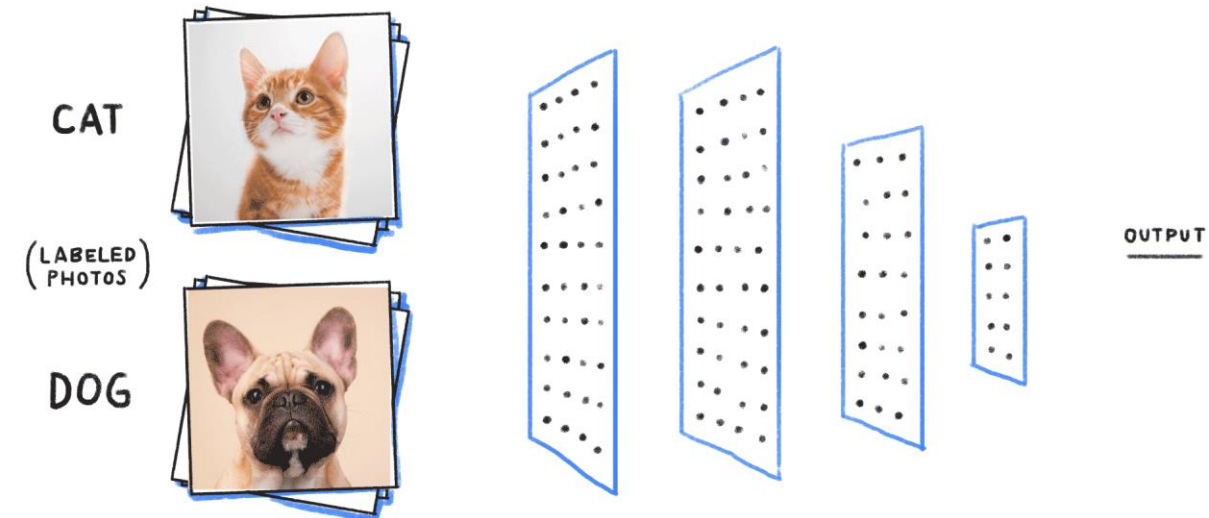
learn a model M that minimizes the loss $\sum_{1 \leq i \leq N} (L(M(x_i), y_i))$

Sort a sequence of numbers in ascending order



QuickSort(A,p,q):
if p < q then
 r := Partition(A, p, q)
 QuickSort(A, p, r-1)
 QuickSort(A, r+1, q)

Initial call:
QuickSort(A, 0, n-1)



When programming, and when learning?

Programming makes sense when there exists

- precise requirements
- a provably correct program to satisfy the requirements

Even if we don't write these down formally!

E.g. Database, operating system, device driver, payroll processing, tax calculations

Learning makes sense when it is hard to write

- precise requirements
- or provably correct implementation

Even if we were to spend time and energy to write these down formally!

E.g. Image classification, NLP, sentiment understanding, language translation, search

Is there value in combining Programs and Models?

Why bother?

Programs and Models: Serving Each Other

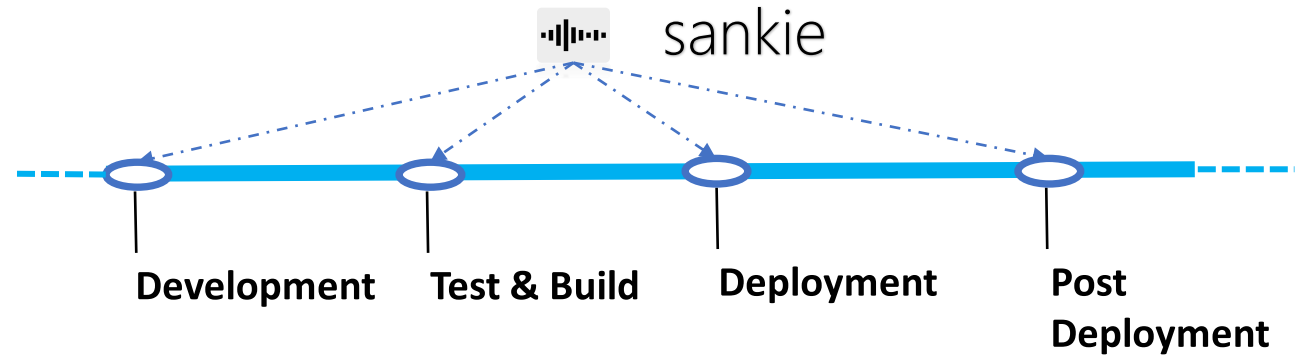
- We can instrument the software development process (coding, code reviews, testing, deployment, debugging, etc) collect data, and use ML models to make the process more efficient.
- We can use programming tools to make learning more efficient.

Programs and Models: Serving Each Other

- Large scale Programming (Software Engineering) can benefit from using ML to provide recommendations during software life cycle

Programs and Models: Serving Each Other

- Large scale Programming (Software Engineering) can benefit from using ML to provide recommendations during software life cycle



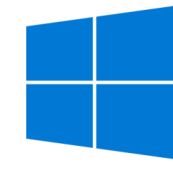
Office



Azure



Bing



Windows



Dynamics

[OSDI 18 (best paper), ICSE 19, FSE19, NSDI 20]

Widely deployed and used inside Microsoft

More information:

<https://www.microsoft.com/en-us/research/project/sankie/>

Programs and Models: Serving Each Other

- Large scale Programming (Software Engineering) can benefit from using ML to provide recommendations during software life cycle

Getafix: How Facebook tools learn to fix bugs automatically

```
1 { cat.meow(volume); }
   { if (cat == null)
     { return;
       cat.meow(volume);
     }
}

{ dog.drink(h0); }
  { if (dog == null)
    { return;
      dog.drink(h0);
    }
}
errorVar: dog

1 { dog.drink(milk); }
  { if (dog == null)
    { return;
      dog.drink(milk);
    }
}
errorVar: dog

1 { bowl.fill();
  dog.drink(water); }
  { bowl.fill();
    if (dog == null)
    { return;
      dog.drink(water);
    }
}
errorVar: dog

dog.eat(h0);
  { if (dog == null)
    { return;
      dog.eat(h0);
    }
}
errorVar: dog
```

Merge edits which check `dog` for null before calling `dog.drink()` (either `milk` or `water`)

By Satish Chandra, Johannes Bader, Eric Lippert, Andrew Scott



Programs and Models: Serving Each Other

- Large scale Programming (Software Engineering) can benefit from using ML to provide recommendations during software life cycle
- Programming language and compiler techniques play a key role in making ML systems flexible and efficient

Automatic differentiation in PyTorch

Adam Paszke
University of Warsaw
adam.paszke@gmail.com

Sam Gross
Facebook AI Research

Soumith Chintala
Facebook AI Research

Gregory Chanan
Facebook AI Research

Edward Yang
Facebook AI Research

Zachary DeVito
Facebook AI Research

Zeming Lin
Facebook AI Research

Alban Desmaison
University of Oxford

Luca Antiga
OROBIX Srl

Adam Lerer
Facebook AI Research

Journal of Machine Learning Research 18 (2018) 1-43

Submitted 8/17; Published 4/18

Automatic Differentiation in Machine Learning: a Survey

Atılım Güneş Baydin
*Department of Engineering Science
University of Oxford
Oxford OX1 3PJ, United Kingdom*

GUNES@ROBOTS.OX.AC.UK

Barak A. Pearlmutter
*Department of Computer Science
National University of Ireland Maynooth
Maynooth, Co. Kildare, Ireland*

BARAK@PEARLMUTTER.NET

Alexey Andreyevich Radul
*Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139, United States*

AXCH@MIT.EDU

Jeffrey Mark Siskind
*School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907, United States*

QOBI@PURDUE.EDU

Editor: Léon Bottou

Abstract

Automatic differentiation module of PyTorch — a
differentiation module of PyTorch — a
on machine learning models. It builds
PyTorch, Chainer, and HIPS Autograd [4],
environment with easy access to automatic
different devices (CPU and GPU). To make
use the symbolic approach used in many
systems on differentiation of purely imperative
code with low overhead. Note that this preprint is
a working paper covering all PyTorch features.

Programs and Models: Serving Each Other

- Large scale Programming (Software Engineering) can benefit from using ML to provide recommendations during software life cycle
- Programming language and compiler techniques play a key role in making ML systems flexible and efficient



2nd C4ML workshop, at [CGO 2020](#)

Sunday, February 23, 2020

San Diego Mission Bay Resort - **Terrazza Ballroom**

Presentation slides linked from the abstracts below

Previous workshops: C4ML [2019](#)

Scope

Machine learning applications are becoming ubiquitous in large-scale production systems. With that growth and the scaling in data volume and model complexity, the focus on efficiently executing machine learning models has become even greater. The push for increased energy efficiency has led to the emergence of diverse heterogeneous system and accelerator architectures. In parallel, model complexity and diversity pushed for higher productivity systems, more powerful programming abstractions, type systems, language embeddings, frameworks and libraries. Compilers have historically been the bridge between programmer efficiency and high performance code, allowing the expression of code that remains understandable and productive to port and extend, while producing high-performance code for diverse architectures. As such, compiler techniques have been increasingly incorporated into machine learning frameworks. This goes both ways: given the broadening gap between high-level constructs and hardware accelerators, compilers in machine learning frameworks also emerged as natural clients of machine learning techniques, from domain-specific heuristics to autotuning.

This workshop aims to highlight cutting edge work and research that incorporate compiler techniques and algorithms in optimizing machine learning workloads. Compiler techniques affect a large part of the machine learning stack. The workshop topics span from high-level abstract representations to code generation for accelerators. The list of invited speakers are similarly experts across the different levels of the stack. The workshop does not have formal proceedings, and presentations will include ample time for interaction.

Is there value in combining programs and models more deeply?

When programming, and when learning?

Programming makes sense when there exists

- precise requirements
- a provably correct program to satisfy the requirements

Even if we don't write these down formally!

E.g. Database, operating system, device driver, payroll processing, tax calculations

Learning makes sense when it is hard to write

- precise requirements
- or provably correct implementation


Even if we were to spend time and energy to write these down formally!

E.g. Image classification, NLP, sentiment understanding, language translation, search


Characteristics of programs and models

Programs are intended to work for all inputs satisfying a precondition

If the specification or environment changes, programs typically fail!

Programs are succinct ways to specify domain knowledge 

Learning works well “on average” when the test distribution is similar to training distribution

ML models can generalize and work on unforeseen inputs 

ML models can be opaque sets of floating-point numbers, and hard to interpret

What if we want programs to be adaptive?

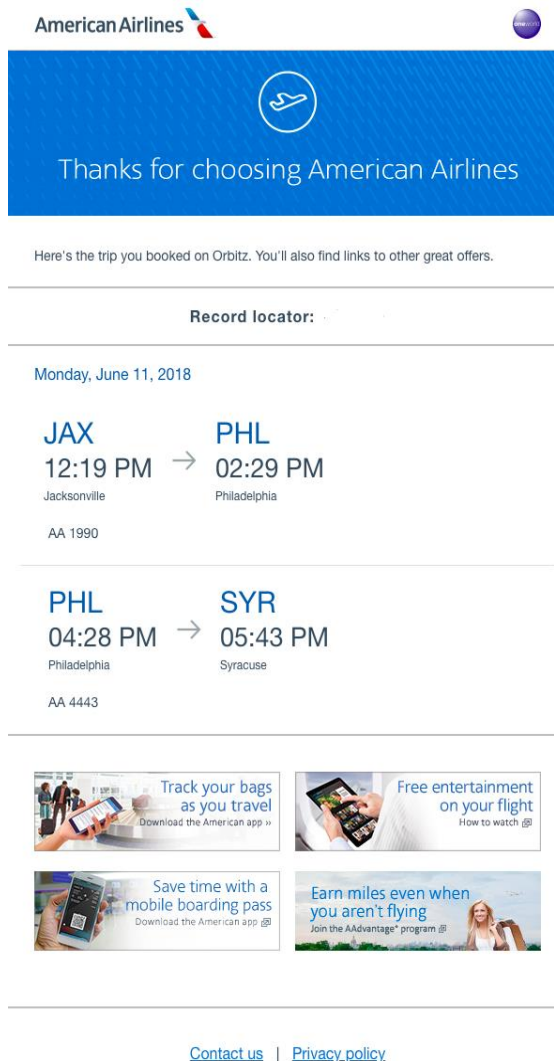
- What if a mathematical specification exists, but it keeps changing and evolving over time? Can we have the system evolve and “adapt” without programmer intervention?
- What if the environment of the program changes, and we want the program to “self-tune” itself in response to the environment changes?



Examples of changing specifications

Changing data formats: shopping web pages, machine generated formats

Customization by each entity in an industry: health data formats, financial data formats,....



American Airlines logo and Orbitz logo.

Thanks for choosing American Airlines

Here's the trip you booked on Orbitz. You'll also find links to other great offers.

Record locator:

Monday, June 11, 2018

JAX 12:19 PM → **PHL** 02:29 PM
Jacksonville Philadelphia
AA 1990

PHL 04:28 PM → **SYR** 05:43 PM
Philadelphia Syracuse
AA 4443

Track your bags as you travel. Download the American app.

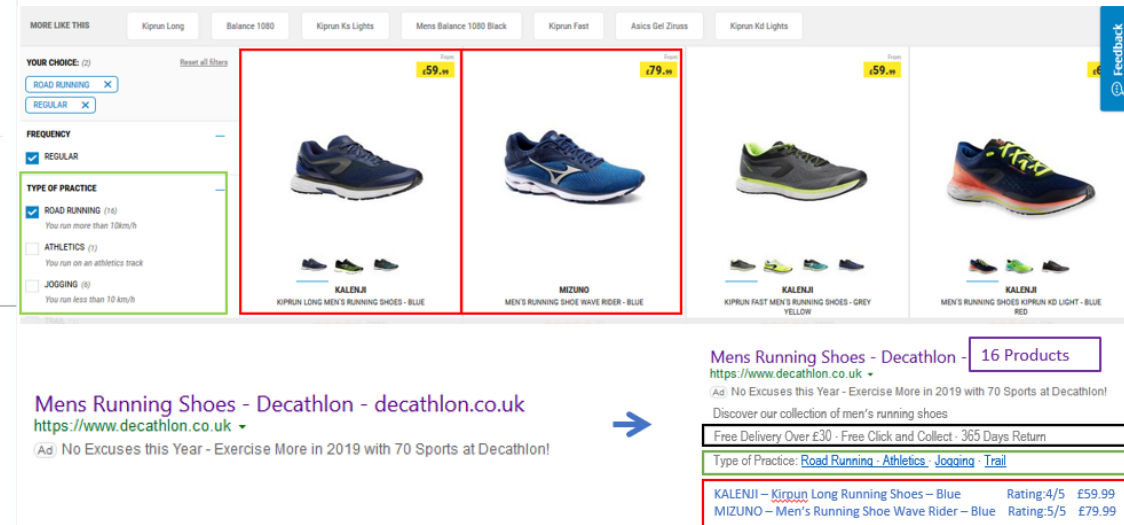
Free entertainment on your flight. How to watch.

Save time with a mobile boarding pass. Download the American app.

Earn miles even when you aren't flying. Join the AAdvantage program.

[Contact us](#) | [Privacy policy](#)

```
MSH | ^~\& | ADT1 | MCM | LABADT | MCM | 198808181126 | SECURITY | ADT ^ A01 | MSG00001- | P | 2.4
EVN | A01 | 198808181123
PID | | PATID1234 ^ 5 ^ M11 | | JONES ^ WILLIAM ^ A ^ III | | 19610615 | M- | | C
PV1 | 1 | I | 2000 ^ 2012 ^ 01 | | | 004777 ^ LEBAUER ^ SIDNEY ^ J. | | | SUR | | - | | ADM | AO
AL1 | 1 | | ^ PENICILLIN | | PRODUCES HIVES ~ RASH ~ LOSS OF APPETITE
DG1 | 001 | I9 | 1550 | MAL NEO LIVER, PRIMARY | 198805011103005 | F
PR1 | 2234 | M11 | 111 ^ CODE151 | COMMON PROCEDURES | 198809081123
```



MORE LIKE THIS: Kiprun Long, Balance 1080, Kiprun Ks Lights, Mens Balance 1080 Black, Kiprun Fast, Asics Gel Zinros, Kiprun Kd Lights

YOUR CHOICE (2): ROAD RUNNING, REGULAR

FREQUENCY: REGULAR

TYPE OF PRACTICE: ROAD RUNNING (14), ATHLETICS (1), JOGGING (8)

Products shown: KALENJI KIPRUN LONG MEN'S RUNNING SHOES - BLUE (\$59.00), MIZUNO MEN'S RUNNING SHOE WAVE RIDER - BLUE (\$79.00), KALENJI KIPRUN FAST MEN'S RUNNING SHOES - GREY YELLOW (\$59.00), KALENJI MEN'S RUNNING SHOES KIPRUN KD LIGHT - BLUE RED (\$59.00)

Mens Running Shoes - Decathlon - 16 Products
<https://www.decathlon.co.uk>

Free Delivery Over £30 - Free Click and Collect - 365 Days Return

Type of Practice: Road Running - Athletics - Jogging - Trail

KALENJI - Kiprun Long Running Shoes - Blue Rating: 4/5 £59.99

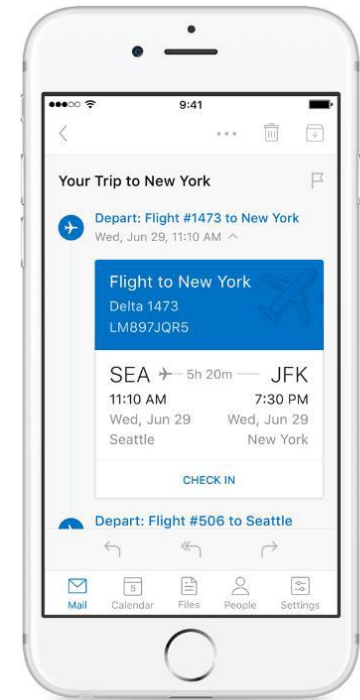
MIZUNO - Men's Running Shoe Wave Rider - Blue Rating: 5/5 £79.99

AIR Confirmation: **AXYB12** — Booking Id Confirmation Date: 02/21/2013

| Passenger(s) | Rapid Rewards # | Ticket # | Expiration | Est. Points Earned |
|--------------|------------------|---------------|--------------|--------------------|
| Person Name | - None Entered - | 5611324256781 | Feb 21, 2014 | 540 |

Rapid Rewards points earned are only estimates. Not a member - visit <http://www.southwest.com/rapidrewards> and sign up today!

| Date | Flight | Departure/Arrival | Airport | Time |
|-----------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|------------|---------------------|
| Fri Mar 8 | 3216 | Depart SEATTLE TACOMA WA (SEA) on Southwest Airlines at 8:10 PM Arrive in SAN JOSE CA (SJC) at 10:20 PM Travel Time 2 hrs 10 mins Wanna Get Away | SEA SJC | 8:10 PM 10:20 PM |



Extract from machine-to-human (M2H) emails

Millions of emails/day

Heterogeneity: 100s of **ever evolving** formats

Some rare formats with very few emails

10s of data annotators write **100s of hard-crafted templates**

Every breakage fixed manually

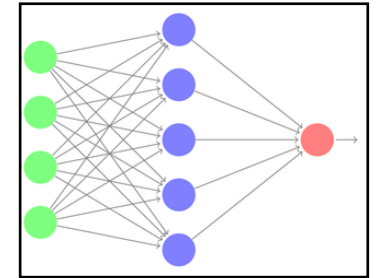
- Goals:
1. Self repair when formats change
 2. When new airlines and travel aggregators come online, handle them as automatically as possible
 3. Predictability

Two approaches to entity extraction from emails

- Train ML models using labeled data
- Write or automatically synthesize programs from labeled data (using systems such as PROSE)



DNN training



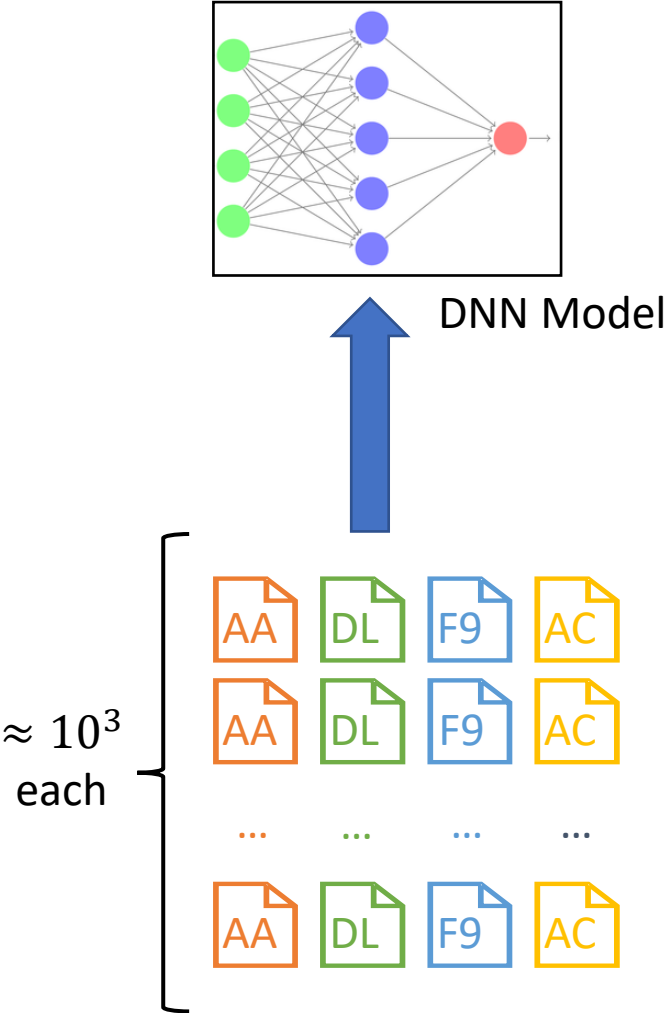
PROSE

Programs in a
Domain
Specific Language

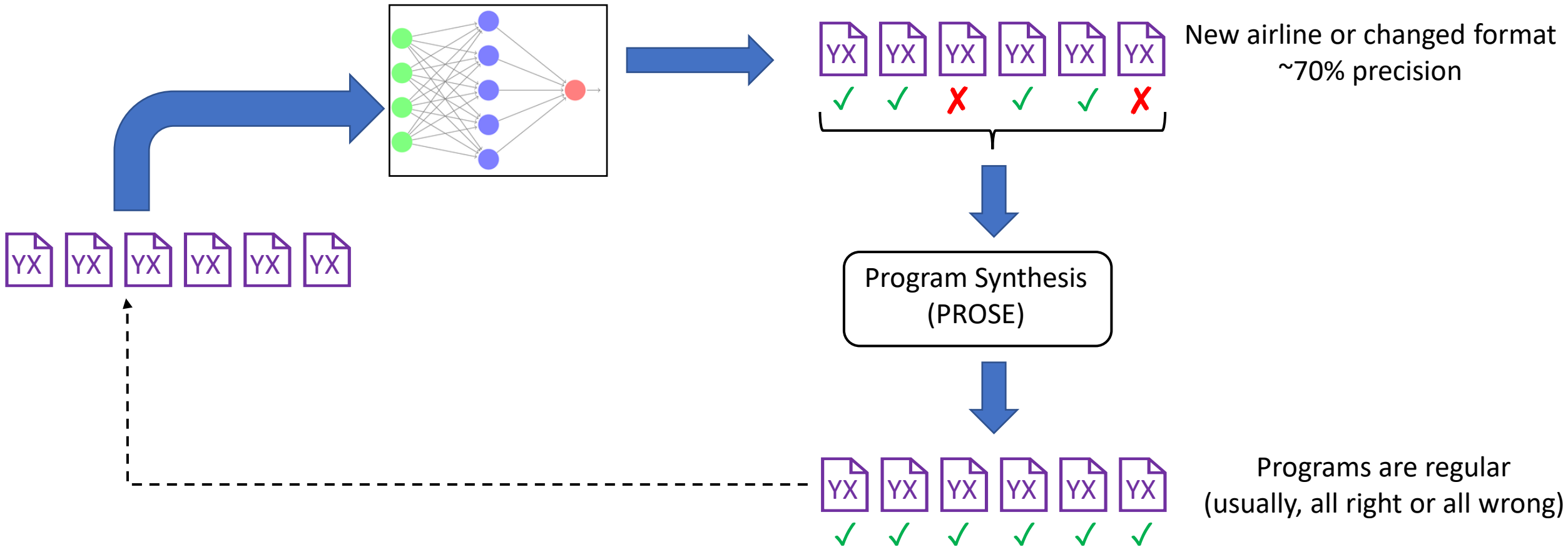
What if the input format changes?

- Programs work well when formats are stable, but just fall flat when format's change
- ML Models generalize somewhat, but don't get to 100%
- Combining both produces better results than either one in isolation!

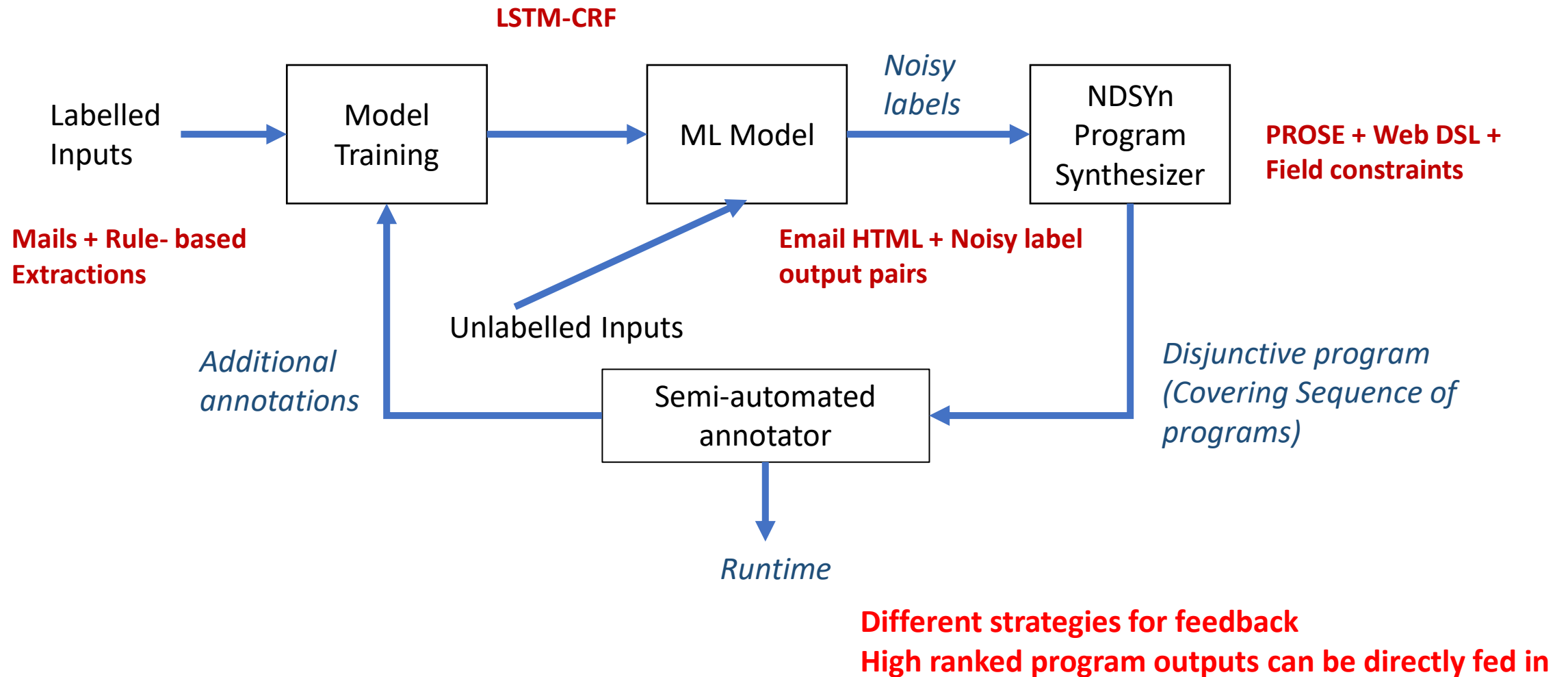
Models for Generalization, Programs for Predictability



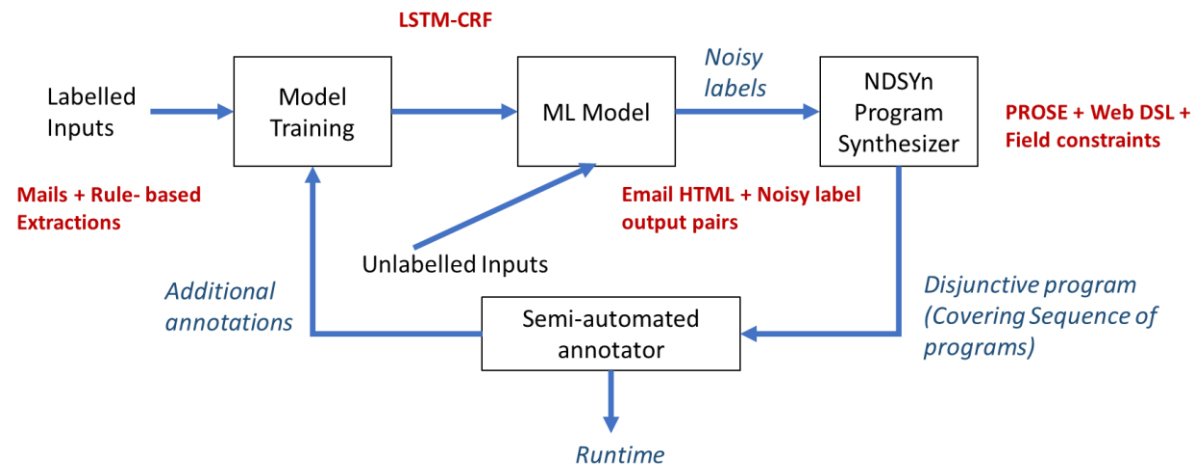
Models for Generalization, Programs for Predictability



Heterogeneous Data Extraction Framework



Models for Generalization, Programs for Predictability



```
Prog := map(λ node . FFProg(node), Nodes)
Nodes := AllNodes(input) | Descendants(Nodes)
       | filter(Selector, Nodes) | Children(Nodes)
Selector := tag = c | class = c | id = c | nth-child(n) | ...
FFProg := Substring | Concat(SubString, FFProg)
SubString := node.TextValue
           | Extract(RegexPos, RegexPos, SubString)
RegexPos := RegexSearch(regex, k)
```

- Design of “Domain Specific Language” (DSL) is key for useful functioning of the combined system
- With a well-designed DSL, program synthesis can act as a “regularizer” and make the system predictable, whereas ML models enable the system to be generalizable and robust to format changes

Deployment results:

”saves us nearly 100-120 Hrs of flight model maintenance time from data annotation per week ... 50% of data annotator bandwidth”

Programs are cheaper to execute, so they are used at runtime. ML models are used offline for self-healing and robustness when formats change

Synthesis and Machine Learning for Heterogeneous Extraction

Arun Iyer
Microsoft Research, Bangalore
ariy@microsoft.com

Manohar Jonnalagedda*
Inpher, Lausanne, Switzerland
manohar.jonnalagedda@gmail.com

Suresh Parthasarathy
Microsoft Research, Bangalore, India
supartha@microsoft.com

Arjun Radhakrishna
Microsoft, Bellevue, United States
arradha@microsoft.com

Sriram K. Rajamani
Microsoft Research, Bangalore, India
sriram@microsoft.com

Abstract

We present a way to combine techniques from the program synthesis and machine learning communities to extract structured information from heterogeneous data. Such problems arise in several situations such as extracting attributes from web pages, machine-generated emails, or from data obtained from multiple sources. Our goal is to extract a set of structured attributes from such data.

We use machine learning models (“ML models”) such as conditional random fields to get an initial labeling of potential attribute values. However, such models are typically not interpretable, and the noise produced by such models is hard to manage or debug. We use (noisy) labels produced by such ML models as inputs to program synthesis, and generate interpretable programs that cover the input space. We also employ type specifications (called “field constraints”) to certify well-formedness of extracted values. Using synthesized programs and field constraints, we re-train the ML models with improved confidence on the labels. We then use these improved labels to re-synthesize a better set of programs. We iterate the process of re-synthesizing the programs and re-training the ML models, and find that such an iterative process improves the quality of the extraction process. This iterative approach, called HDEF, is novel, not only in the way it combines the ML models with program synthesis, but also in the way it adapts program synthesis to deal with noise and heterogeneity.

*This work was done when the author was at Microsoft Research, Bangalore, India.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PLDI '19, June 22–26, 2019, Phoenix, AZ, USA

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6712-7/19/06...\$15.00
<https://doi.org/10.1145/3314221.3322485>

More broadly, our approach points to ways by which machine learning and programming language techniques can be combined to get the best of both worlds – handling noise, transferring signals from one context to another using ML, producing interpretable programs using PL, and minimizing user intervention.

CCS Concepts • Software and its engineering → Automatic programming; • Computing methodologies → Machine learning.

Keywords Data extraction, Program synthesis, Machine Learning, Heterogeneous data

ACM Reference Format:

Arun Iyer, Manohar Jonnalagedda, Suresh Parthasarathy, Arjun Radhakrishna, and Sriram K. Rajamani. 2019. Synthesis and Machine Learning for Heterogeneous Extraction. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '19)*, June 22–26, 2019, Phoenix, AZ, USA. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3314221.3322485>

1 Introduction

Extracting structured attributes from heterogeneous unstructured or semi-structured data is an important problem, which arises in many situations. One example is processing websites in domains such as travel, shopping, and news and extracting specific attributes from them. Another example is in processing machine generated emails in such domains, and extracting specific attributes. A third example is data wrangling where the goal is to transform and map raw data to a more structured format, with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.

In the ML community, these problems have been handled by training ML models. While impressive progress has been made in making use of signals from noisy and large scale data [12, 23, 24, 26], the models produced are not interpretable and hence hard to maintain, debug and evolve. In the PL community, program synthesis has been used to generate programs, such as Excel macros, from a small number of training examples [7, 13, 18]. If the data-sets are large and heterogeneous, and training data is small and noisy, neither

Related work

“Programmatically Interpretable Reinforcement Learning”, Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri, In ICML 2018

“Verifiable reinforcement learning via policy extraction”, Osbert Bastani, Yewen Pu, Armando Solar-Lezama, NIPS 2019

“An Inductive Synthesis Framework for Verifiable Reinforcement Learning”, He Zhu, Zikang Xiong, Stephen Magill, Suresh Jagannathan, PLDI 2019

What if we want programs to be adaptive?

- What if a mathematical specification exists, but it keeps changing and evolving over time? Can we have the system evolve and “adapt” without programmer intervention?
- What if the environment of the program changes, and we want the program to “self-tune” itself in response to the environment changes?



Configuration settings are ubiquitous in software!

[Resource1]

```
RefreshInterval=00:01:00
PriorityHighUnderloaded=97
PriorityHighOverloaded=98
PriorityLowUnderloaded=88
PriorityLowOverloaded=90
...
```

[Resource2]

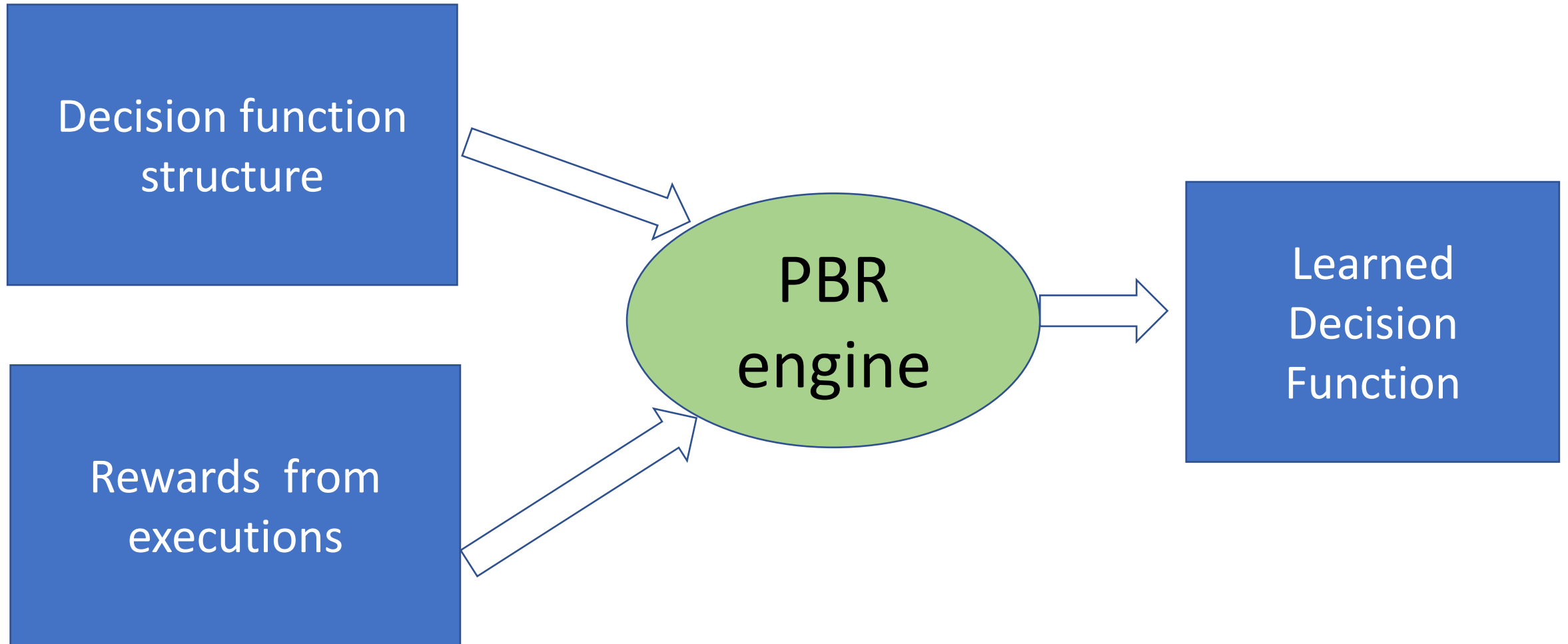
```
RefreshInterval=00:01:00
PriorityHighUnderloaded=80
PriorityHighOverloaded=90
...
```

```
double ScoreLinesMap(double sel, double lines) {
    double minScore = 100.0;
    double alpha = 1.0; double beta = 1.0;
    return alpha * sel + beta * lines + minScore * 20;
}

bool IsLikelyDataRatio(int dataCount, int totalCount) {
    if (totalCount < 10) return dataCount >= 6;
    if (totalCount < 20) return dataCount >= 15;
    if (totalCount < 50) return dataCount >= 30;
    return dataCount / (double) totalCount >= 0.6;
}
```

Can we learn such “structured” decision functions automatically?

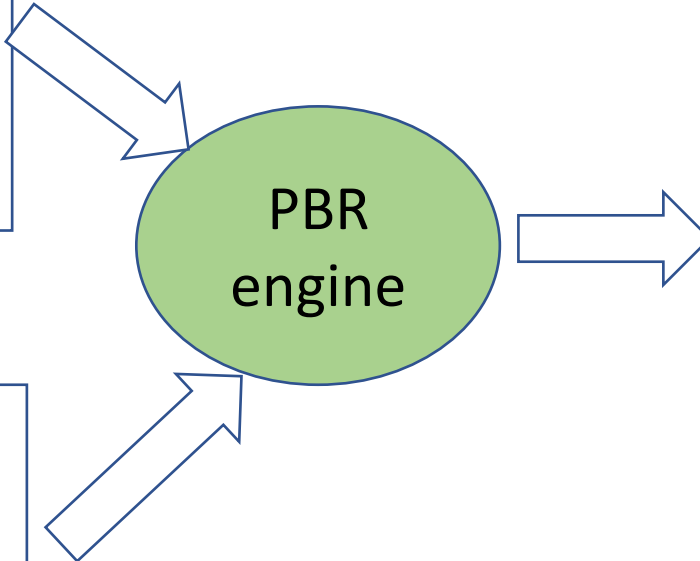
Programming By Rewards (PBR)



Programming By Rewards (PBR)

```
int[2] counts = getCounts(contents);  
...  
if  
(PBR.DecisionFunction(PBRID_IsLikelyDataRatio,  
                      count[0], count[1]))  
  preProcess(fileName);  
...
```

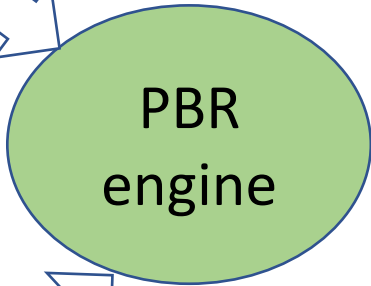
```
...  
double reward =  
  (success/Config.Benchmarks.Length) -  
  (sw.ElapsedMilliseconds/1000);  
PBR.AssignReward(reward);
```



Learned
Decision
Function

Programming By Rewards (PBR)

```
int[2] counts = getCounts(contents);  
...  
if  
(PBR.DecisionFunction(PBRID_IsLikelyDataRatio,  
                      count[0], count[1]))  
  preProcess(fileName);  
...
```



```
bool IsLikelyDataRatio  
  (int dataCount, int totalCount) {  
    if (totalCount < 10)  
      return dataCount >= 6;  
    if (totalCount < 20)  
      return dataCount >= 15;  
    if (totalCount < 50)  
      return dataCount >= 30;  
  
    return (dataCount /totalCount >= 0.6);  
  }
```

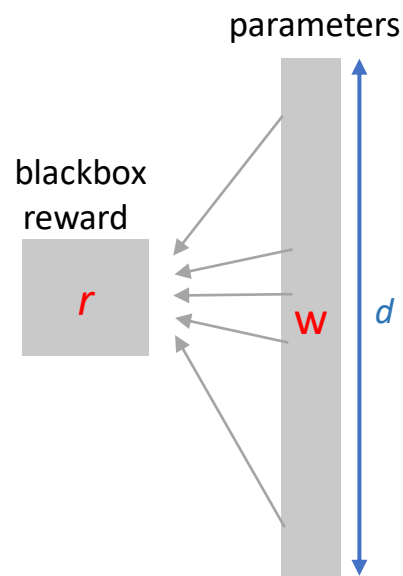
```
...  
double reward =  
  (success/Config.Benchmarks.Length) -  
  (sw.ElapsedMilliseconds/1000);  
PBR.AssignReward(reward);
```

Learning with black-box rewards

For a given **unknown (black-box)** reward function r , and a **known** code template for the decision (e.g., linear), the goal is to solve:

$$\max_{w \in \mathbb{R}^d} \sum_i r_i(\underbrace{w^T x_i}_{\text{linear}})$$

$$w_1 * \text{latency} + \\ w_2 * \text{load} + \\ w_3 * \text{min} + w_4$$



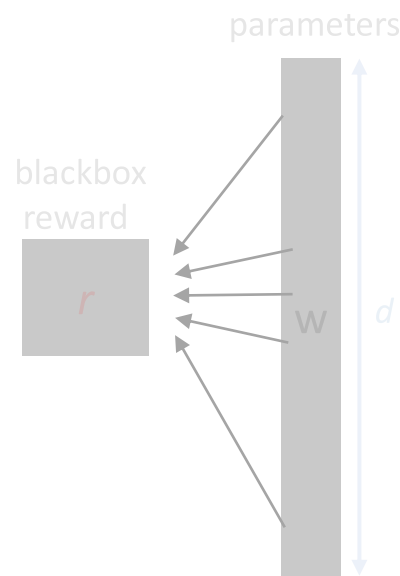
RL, online learning, black-box optimizers are expensive in terms of #reward calls needed (prop. to d)

Learning with black-box rewards

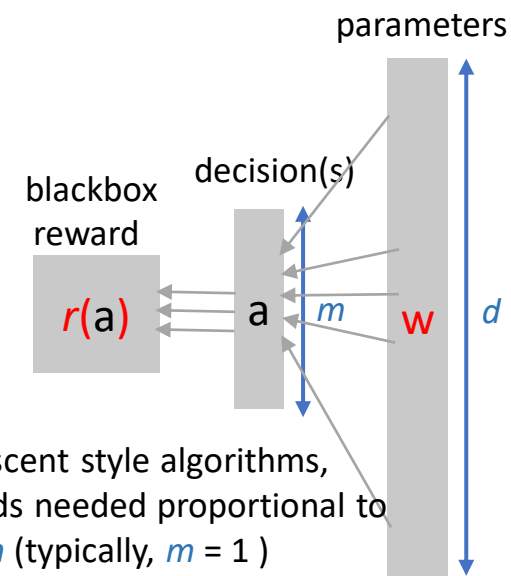
For a given **unknown (black-box)** reward function r , and a **known** code template for the decision (e.g., linear), the goal is to solve:

$$\max_{w \in \mathbb{R}^d} \sum_i r_i(\underbrace{w^T x_i}_{\text{decision}})$$

w_1 * latency +
 w_2 * load +
 w_3 * min + w_4



RL, online learning, black-box optimizers are expensive in terms of #reward calls needed (prop. to d)



Gradient-descent style algorithms, with #rewards needed proportional to #decisions m (typically, $m = 1$)

Case study: PROSE codebase

- We applied Self-Tune to simultaneously learn ~70 ranking heuristics in PROSE
- Reward: # tasks where PROSE synthesizes a correct program
- Each reward query is expensive (~20 minutes)
- In ~100 hours of training, PBR improves over state-of-the-art ML-ranker by ~8% in terms of accuracy
- Competitive with the manually-tuned heuristics that took 2+ years of effort

| Ranker | Accuracy |
|--------------------------|----------|
| ML-PROSE [2019**] | 606/740 |
| PROSE + SelfTune [2020*] | 668/740 |

* [Programming by Rewards, 2020](#). N., Ajaykrishna Karthikeyan, Prateek Jain, Ivan Radicek, Sriram Rajamani, Sumit Gulwani, Johannes Gehrke. <https://arxiv.org/pdf/2007.06835.pdf>

** [Learning natural programs from a few examples in real-time](#). N., Dany Simmons, Naren Datha, Prateek Jain, Sumit Gulwani. AISTATS, 2019.

Learning algorithms can exploit the structure of the decision function to get better sample complexity

Programming by Rewards

Synthesizing programs using black-box rewards

NAGARAJAN NATARAJAN, Microsoft Research, IN
AJAYKRISHNA KARTHIKEYAN, Microsoft Research, IN
PRATEEK JAIN, Microsoft Research, IN
IVAN RADIČEK, Microsoft, Austria
SRIRAM RAJAMANI, Microsoft Research, IN
SUMIT GULWANI, Microsoft, USA
JOHANNES GEHRKE, Microsoft Research, USA

We formalize and study “programming by rewards” (PBR), a new approach for specifying and synthesizing subroutines for optimizing some quantitative metric such as performance, resource utilization, or correctness over a benchmark. A PBR specification consists of (1) input features \mathbf{x} , and (2) a reward function r , modeled as a black-box component (which we can only run), that assigns a reward for each execution. The goal of the synthesizer is to synthesize a *decision function* f which transforms the features to a decision value for the black-box component so as to maximize the expected reward $E[r \circ f(\mathbf{x})]$ for executing decisions $f(\mathbf{x})$ for various values of \mathbf{x} .

We consider a space of decision functions in a DSL of loop-free if-then-else programs, which can branch on linear functions of the input features in a tree-structure and compute a linear function of the inputs in the leaves of the tree. We find that this DSL captures decision functions that are manually written in practice by programmers. Our technical contribution is the use of continuous-optimization techniques to perform synthesis of such decision functions as if-then-else programs. We also show that the framework is theoretically-founded—in cases when the rewards satisfy nice properties, the synthesized code is optimal in a precise sense.

PBR hits a sweet-spot between program synthesis techniques that require the entire system $r \circ f$ as a white-box, and reinforcement learning (RL) techniques that treat the entire system $r \circ f$ as a black-box. PBR takes a middle path treating f as a white-box, thereby exploiting the structure of f to get better accuracy and faster convergence, and treating r as a black-box, thereby scaling to large real-world systems. Our algorithms are provably more accurate and sample efficient than existing synthesis-based and reinforcement learning-based techniques under certain assumptions.

We have leveraged PBR to synthesize non-trivial decision functions related to search and ranking heuristics in the PROSE codebase (an industrial strength program synthesis framework) and achieve competitive results to manually written procedures over multiple man years of tuning. We present empirical evaluation against other baseline techniques over real-world case studies (including PROSE) as well on simple synthetic benchmarks.

Additional Key Words and Phrases: AI driven software engineering, sketching, online learning

arXiv:2007.06835v1 [cs.LG] 14 Jul 2020

Story so far: Programs + Models

Using ML to provide recommendations during the software life cycle

Using compilers and runtimes to make ML systems flexible and efficient

Having programs adapt when specifications (eg, formats) change

Having programs “self-tune” when environments change

**Probabilistic
Programming**



RESEARCH ARTICLES

COGNITIVE SCIENCE

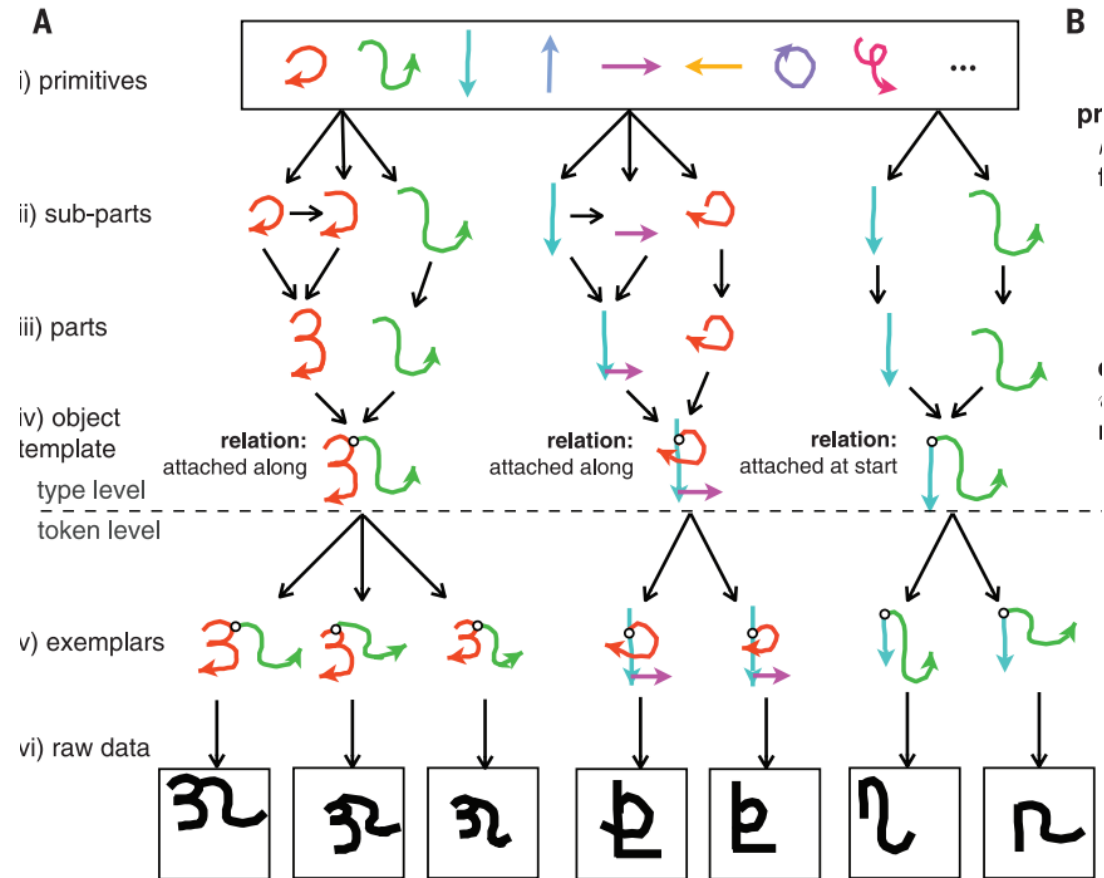
Human-level concept learning through probabilistic program induction

Brenden M. Lake,^{1*} Ruslan Salakhutdinov,² Joshua B. Tenenbaum³

[Science 2015]

Bayesian programming language framework (BPL)

- Capable of learning visual concepts from a single example
- Programmer specifies primitives, parts and subparts as domain knowledge
- System infers knowledge representation as probabilistic programs using Bayesian inference



Tutorial on probabilistic programs (1)

```
bool c1, c2;  
c1 = Bernoulli(0.5);  
c2 = Bernoulli(0.5);  
return(c1, c2);
```

| $c1$ | $c2$ | $P(c1, c2)$ |
|-------|-------|-------------|
| false | false | 1/4 |
| false | true | 1/4 |
| true | false | 1/4 |
| true | true | 1/4 |

Tutorial on probabilistic programs (2)

```
bool c1, c2;  
c1 = Bernoulli(0.5);  
c2 = Bernoulli(0.5);  
observe(c1 || c2);  
return(c1, c2);
```

| $c1$ | $c2$ | $P(c1, c2)$ |
|-------|-------|-------------|
| false | false | 0 |
| false | true | 1/3 |
| true | false | 1/3 |
| true | true | 1/3 |

Tutorial on probabilistic programs (3)



“TrueSkill” from Infer.Net
@MSR Cambridge

- Player A beats Player B, if A performs better than B during the game
- Performance is a stochastic function of skill

```
float skillA, skillB, skillC;
float perfA1, perfB1, perfB2,
      perfC2, perfA3, perfC3;
skillA = Gaussian(100, 10);
skillB = Gaussian(100, 10);
skillC = Gaussian(100, 10);

// first game: A vs B, A won
perfA1 = Gaussian(skillA, 15);
perfB1 = Gaussian(skillB, 15);
observe(perfA1 > perfB1);

// second game: B vs C, B won
perfB2 = Gaussian(skillA, 15);
perfC2 = Gaussian(skillB, 15);
observe(perfB2 > perfC2);

// third game: A vs C, A won
perfA3 = Gaussian(skillA, 15);
perfC3 = Gaussian(skillB, 15);
observe(perfA3 > perfC3);

return(skillA, skillB, skillC);
```

- Sample *perfA* from a noisy *skillA* distribution
- Sample *perfB* from a noisy *skillB* distribution
- if *perfA* > *perfB* then A wins else B wins

skillA = Gaussian(102.1,7.8)
skillB = Gaussian(100.0,7.6)
skillC = Gaussian(97.9,7.8)

Kidney Disease Estimation

```
double logScr, age;
bool isFemale, isAA;

double f1 =
    estimateLogEGFR(logScr, age,
                    isFemale, isAA);
double nLogScr, nAge;
bool nIsFemale, nisAA;

nLogScr = logScr +
    Uniform(-0.1, 0,1);
nAge = age +
    Uniform(-1, 1);

nIsFemale = isFemale;
if(Bernoulli(0.01))
    nIsFemale = !isFemale;

nIsAA = isAA;
if(Bernoulli(0.01))
    nIsAA = !isAA;

double f2 =
    estimateLogEGFR(nLogScr, nAge,
                    nIsFemale, nIsAA);

bool bigChange = 0;
if(f1 - f2 >= 0.1)
    bigChange = 1;
if(f2 - f1 >= 0.1)
    bigChange = 1;

return(bigChange);
```

```
double estimateLogEGFR(
    double logScr, double age,
    bool isFemale, bool isAA)
{
    double k, alpha;
    double f = 4.94;
    if(isFemale){
        k = -0.357;
        alpha = -0.328;
    }
    else{
        k = -0.105;
        alpha = -0.411;
    }

    if(logScr < k)
        f = alpha * (logscr-k);
    else
        f = -1.209 * (logscr-k);

    f = f - 0.007 * age;

    if(isFemale) f = f + 0.017;
    if(isAA) f = f + 0.148;

    return f;
}
```

S. Sankaranarayanan, A. Chakarov, and S. Gulwani. Static analysis of probabilistic programs: Inferring whole program properties from finitely many executions. In Programming Languages Design and Implementation (PLDI), 2013.

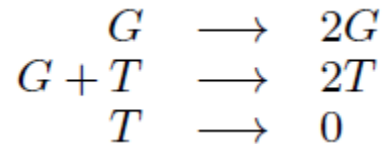
Lotka-Volterra Population Model

```
int goats, tigers;
double c1, c2, c3, curTime;
// initialize populations
goats = 100; tigers = 4;
// initialize reaction rates
c1 = 1; c2 = 5; c3 = 1;
//initialize time
curTime = 0;

while (curTime < TIMELIMIT)
{
  if (goats > 0 && tigers > 0)
  {
    double rate1, rate2, rate3,
           rate;
    rate1 = c1 * goats;
    rate2 = c2 * goats * tigers;
    rate3 = c3 * tigers;
    rate = rate1 + rate2 + rate3;

    double dwellTime =
      Exponential(rate);
    int discrete =
      Disc3(rate1/rate,rate2/rate);
    curTime += dwellTime;
    switch (discrete)
    {
      case 0: goats++; break;
      case 1: goats--; tigers++;
              break;
      case 2: tigers--; break;
    }
  }
}
```

```
else if (goats > 0)
{
  double rate;
  rate = c1 * goats;
  double dwellTime =
    Exponential(rate);
  curTime += dwellTime;
  goats++;
}
else if (tigers > 0)
{
  double rate;
  rate = c3 * tigers;
  double dwellTime =
    Exponential(rate);
  curTime += dwellTime;
  tigers--;
}
} //end while loop
return(goats,tigers);
}
```



Lotka, Elements of physical biology. Williams & Wilkins company, Baltimore, 1925.

V. Volterra. Fluctuations in the abundance of a species considered mathematically. Nature, 118:558–560, 1926.

Several more applications that can be modeled as probabilistic programs

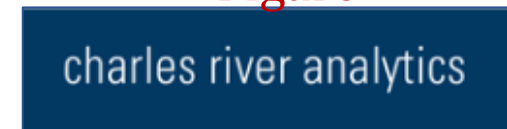
- Hidden Markov Models (eg. for speech recognition)
- Kalman Filters (eg. In computer vision)
- Markov Random Fields (eg. In image processing)
- Markov Chains
- Bayesian Networks
- And more applications:
 - Ecology & Biology (Carbon modeling, Evolutionary Genetics,...)
 - Security (quantitative information flow, inference attacks)

Probabilistic Inference

- Infer the distribution specified by a probabilistic program.
 - *Generate samples to test a machine learning algorithm*
 - *Calculate the expected value of a function wrt the distribution specified by the program*
 - *Calculate the mode of the distribution specified by the program*
- **Punchline:**
 - Inference *is* program analysis of probabilistic programs



Figaro



Stan



TensorFlow Probability



Pearl's Burglar alarm example

```
int alarm() {  
    char earthquake = Bernoulli(0.001);  
    char burglary = Bernoulli(0.01);  
    char alarm = earthquake || burglary;  
    char phoneWorking =  
        (earthquake)? Bernoulli(0.6) : Bernoulli(0.99);  
    char maryWakes;  
    if (alarm && earthquake)  
        maryWakes = Bernoulli(0.8);  
    else if (alarm)  
        maryWakes = Bernoulli(0.6);  
    else maryWakes = Bernoulli(0.2);  
    char called = maryWakes && phoneWorking;  
    observe(called);  
    return burglary;  
}
```

“called” is a low probability event, and causes large number of rejections during sampling

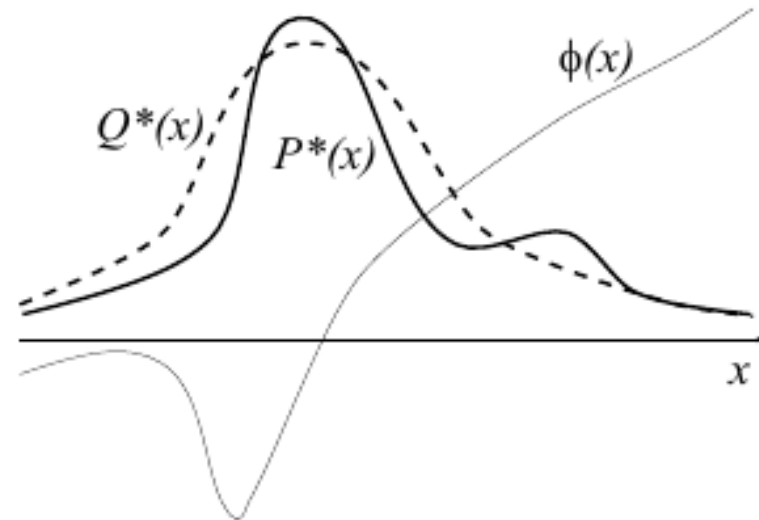
Pre transformation

- Let P be any program
- Let Pre(P) denote the program obtained by propagating observe statements immediately after sample statements

Theorem: $P = \text{Pre}(P)$

```
int alarm() {
  bool earthquake, burglary, alarm, phoneWorking,
  char earthquake = Bernoulli(0.001);
  maryWakes, called;
  char burglary = Bernoulli(0.01);
  earthquake = Bernoulli(0.001);
  burglary = Bernoulli(0.01);
  char alarm = earthquake || burglary;
  alarm = earthquake || burglary;
  char phoneWorking =
  if (earthquake) {
    (earthquake)?Bernoulli(0.6);:Bernoulli(0.99);
    phoneWorking = Bernoulli(0.6);
    observe(phoneWorking);
  }
  char maryWakes;
}
else {
  if (alarm && earthquake)
    maryWakes = Bernoulli(0.98);
    observe(phoneWorking);
  else if (alarm)
    maryWakes = Bernoulli(0.6);
  if (alarm && earthquake)
    maryWakes = Bernoulli(0.8);
  else maryWakes = Bernoulli(0.2);
  observe(maryWakes && phoneWorking);
}
char called = maryWakes && phoneWorking;
else if (alarm) {
  observe(called);
  maryWakes = Bernoulli(0.6);
  return burglary;
  observe(maryWakes && phoneWorking);
}
else {
  maryWakes = Bernoulli(0.2);
  observe(maryWakes && phoneWorking);
}
called = maryWakes && phoneWorking;
return burglary;
}
```

Background: Sampling



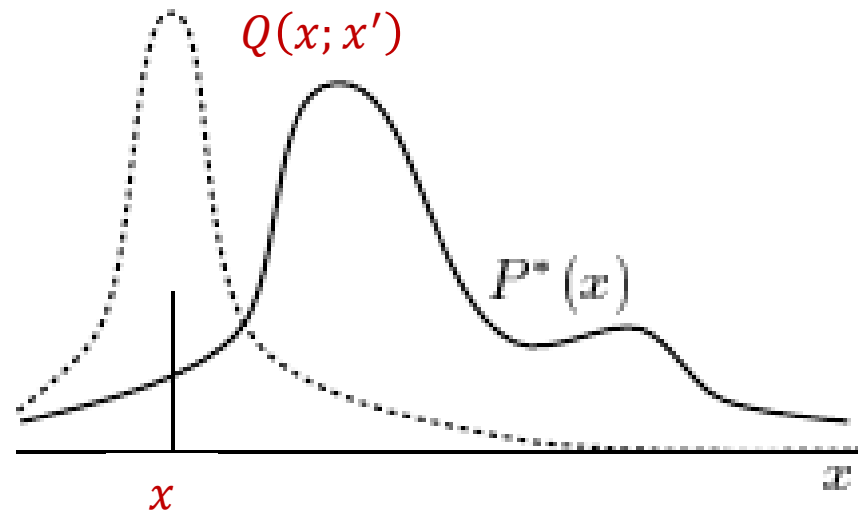
Problem. Estimate expectation of $\phi(x)$ wrt to the distribution

$$P^*(x): \int_x P^*(x) \times \phi(x) dx$$

If we can sample from $P^*(x)$ we can estimate the expectation as:

$$\frac{1}{N} \times (\phi(x_1) + \phi(x_2) \cdots + \phi(x_N))$$

Background: MH sampling



1. Draw samples for x' from a proposal $Q(x; x')$
2. Compute $a = \frac{P^*(x') \times Q(x; x')}{P^*(x) \times Q(x'; x)}$
3. If $a \geq 1$, accept x' else accept with probability a

MH without rejections

For each statement of the form:

$x_i = \text{Dist}(E); \text{observe}(\phi)$

Calculate

$$\beta_i = \frac{\text{Density}(\text{Dist}(E))(x') \times Q_{|\phi}(x^{(t)}; x')}{\text{Density}(\text{Dist}(E))(x^{(t)}) \times Q_{|\phi}(x'; x^{(t)})}$$

During each run of π_i , for each sample statement:

- Sample from proposal sub-distribution Q conditioned by ϕ
- $\beta = \beta_1 \times \beta_2 \times \dots \times \beta_n$

If $\beta \geq 1$, accept x' else accept with probability β

Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel.

[R2: An Efficient MCMC Sampler for Probabilistic Programs](#),

In *AAAI '14: AAAI Conference on Artificial Intelligence*, July 2014

Program Slicing

[Mark Weiser, 1981]

Reduce a program to a smaller program “slice” when interested in only some values of interest at a program point

Many applications:

- Debugging
- Optimization
- Maintenance

PROGRAM SLICING*

Mark Weiser

Computer Science Department
University of Maryland
College Park, MD 20742

Abstract

Program slicing is a method used by experienced computer programmers for abstracting from programs. Starting from a subset of a program's behavior, slicing reduces that program to a minimal form which still produces that behavior. The reduced program, called a "slice", is an independent program guaranteed to faithfully represent the original program within the domain of the specified subset of behavior.

Finding a slice is in general unsolvable. A dataflow algorithm is presented for approximating slices when the behavior subset is specified as the values of a set of variables at a statement. Experimental evidence is presented that these slices are used by programmers during debugging. Experience with two automatic slicing tools is summarized. New measures of program complexity are suggested based on the organization of a program's slices.

KEYWORDS: debugging, program maintenance, software tools, program metrics, human factors, data-flow analysis

Introduction

A large computer program is more easily constructed, understood, and maintained when broken into smaller pieces. Several different methods decompose programs during program design, such as information hiding (Parnas 1972), data abstraction

behavior is of interest. For instance, during debugging a subset of behavior is being corrected, and in program modification or maintenance a subset of behavior is being improved or replaced. In these cases, a programmer starts from the program behavior and proceeds to find and modify the corresponding portions of program code. Code not having to do with behavior of interest is ignored. Gould and Dronkowski (1974) report programmers behaving this way during debugging, and a further confirming experiment is presented below.

A programmer maintaining a large, unfamiliar program would almost have to use this behavior-first approach to the code. Understanding an entire system to change only a small piece would take too much time. Since most program maintenance is done by persons other than the program designers, and since 67 percent of programming effort goes into maintenance (Zelkowitz, Shaw, and Gannon 1979), decomposing programs by behavior must be a common occurrence.

Automatic slicing requires that behavior be specified in a certain form. If the behavior of interest can be expressed as the values of some sets of variables at some set of statements, then this specification is said to be a slicing criterion. Dataflow analysis (Hecht 1977) can find all the program code which might have influenced the specified behavior, and this code is called a slice of the program. A slice is itself an executable program, whose behavior must be identi-

Dependencies used by Slicing

S1: $A := B * C$

S2: $C := A * E + 1$

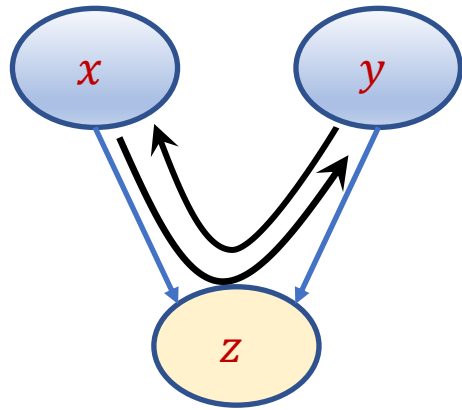
S2 is “Data Dependent”
on S

S1: if (A) then

S2: $B = C + D$

S2 is “Control Dependent” on S1

Probabilistic Programs have new dependences



- Figure represents

$$p(x, y, z) = p(z|x, y) \cdot p(x) \cdot p(y)$$

- There is no dependence between x and y
- On the other hand, if z (or some descendant of z) is observed, then x depends on y and vice versa
- This is called “observe dependence”

Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, Selva Samuel.

[Slicing Probabilistic Programs](#), In *PLDI '14: Programming Language Design and Implementation*, June 2014

Slicing Probabilistic Programs

Chung-Kil Hur*
Seoul National University
gil.hur@cse.snu.ac.kr

Aditya V. Nori
Microsoft Research
adityan@microsoft.com

Sriram K. Rajamani
Microsoft Research
sriram@microsoft.com

Selva Samuel
Microsoft Research
t-ssamue@microsoft.com

Abstract

Probabilistic programs use familiar notation of programming languages to specify probabilistic models. Suppose we are interested in estimating the distribution of the return expression r of a probabilistic program P . We are interested in *slicing* the probabilistic program P and obtaining a simpler program $SLI(P)$ which retains only those parts of P that are relevant to estimating r , and elides those parts of P that are not relevant to estimating r . We desire that the SLI transformation be both correct and efficient. By correct, we mean that P and $SLI(P)$ have identical estimates on r . By efficient, we mean that estimation over $SLI(P)$ be as fast as possible.

We show that the usual notion of program slicing, which traverses control and data dependencies backward from the return expression r , is unsatisfactory for probabilistic programs, since it produces incorrect slices on some programs and sub-optimal ones on others. Our key insight is that in addition to the usual notions of control dependence and data dependence that are used to slice non-probabilistic programs, a new kind of dependence called *observe*

1. Introduction

Probabilistic programs are “usual” programs (written in languages like C or Java or LISP or ML) with two added constructs: (1) the ability to draw values at random from distributions, and (2) the ability to condition values of variables in a program through observe statements (which allow data from real world observations to be incorporated into a probabilistic program). A variety of probabilistic programming languages and systems have been proposed [2, 10–12, 18, 20, 23, 26]. However, unlike “usual” programs which are written for the purpose of being executed, the purpose of a probabilistic program is to implicitly specify a probability distribution. Probabilistic programs can be used to represent *probabilistic graphical models* [19], which use graphs to denote conditional dependences between random variables. Probabilistic graphical models are widely used in statistics and machine learning, with diverse application areas including information extraction, speech recognition, computer vision, coding theory, biology and reliability analysis.

Probabilistic inference is the problem of computing an explicit

A Theory of Slicing for Imperative Probabilistic Programs

TORBEN AMTOFT, Kansas State University, USA
ANINDYA BANERJEE, IMDEA Software Institute, Spain

Dedicated to the memory of Sebastian Danicic.

We present a theory for slicing imperative probabilistic programs containing random assignments and “observe” statements for conditioning. We represent such programs as probabilistic control-flow graphs (pCFGs) whose nodes modify probability distributions. This allows direct adaptation of standard machinery such as data dependence, postdominators, relevant variables, and so on, to the probabilistic setting. We separate the specification of slicing from its implementation:

- (1) first, we develop syntactic conditions that a slice must satisfy (they involve the existence of another disjoint slice such that the variables of the two slices are *probabilistically independent* of each other);
- (2) next, we prove that any such slice is semantically correct;
- (3) finally, we give an algorithm to compute the least slice.

To generate smaller slices, we may in addition take advantage of knowledge that certain loops will terminate (almost) always.

Our results carry over to the slicing of *structured* imperative probabilistic programs, as handled in recent work by Hur et al. For such a program, we can define its slice, which has the same “normalized” semantics as the original program; the proof of this property is based on a result proving the adequacy of the semantics of pCFGs w.r.t. the standard semantics of structured imperative probabilistic programs.

CCS Concepts: • **Theory of computation** → **Probabilistic computation**; **Program semantics**; • **Software and its engineering** → **Correctness**; **Automated static analysis**;

Additional Key Words and Phrases: Probabilistic programming, program slicing, probabilistic control-flow graphs

ACM Reference format:

Torben Amtoft and Anindya Banerjee. 2020. A Theory of Slicing for Imperative Probabilistic Programs. *ACM Trans. Program. Lang. Syst.* 42, 2, Article 6 (April 2020), 71 pages.
<https://doi.org/10.1145/3372895>

Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, Selva Samuel.
[Slicing Probabilistic Programs](#), In *PLDI '14: Programming Language Design and Implementation*,
June 2014

Probabilistic Programming

Andrew D. Gordon

Microsoft Research
adg@microsoft.com

Thomas A. Henzinger

IST Austria
tah@ist.ac.at

Aditya V. Nori

Microsoft Research
adityan@microsoft.com

Sriram K. Rajamani

Microsoft Research
sriram@microsoft.com

Abstract

Probabilistic programs are usual functional or imperative programs with two added constructs: (1) the ability to draw values at random from distributions, and (2) the ability to condition values of variables in a program via observations. Models from diverse application areas such as computer vision, coding theory, cryptographic protocols, biology and reliability analysis can be written as probabilistic programs.

Probabilistic inference is the problem of computing an explicit representation of the probability distribution implicitly specified by a probabilistic program. Depending on the application, the desired output from inference may vary—we may want to estimate the expected value of some function f with respect to the distribution, or the mode of the distribution, or simply a set of samples drawn from the distribution.

In this paper, we describe connections this research area called “Probabilistic Programming” has with programming languages and software engineering, and this includes language design, and the static and dynamic analysis of programs. We survey current state of the art and speculate on promising directions for future research.

application areas including information extraction, speech recognition, computer vision, coding theory, biology and reliability analysis.

Probabilistic inference is the problem of computing an explicit representation of the probability distribution implicitly specified by a probabilistic program. If the probability distribution is over a large number of variables, an explicit representation of the joint probability distribution may be both difficult to obtain efficiently, and unnecessary in the context of specific application contexts. For example, we may want to compute the expected value of some function f with respect to the distribution (which may be more efficient to calculate without representing the entire joint distribution). Alternatively, we may want to calculate the most likely value of the variables, which is the mode of the distribution. Or we may want to simply draw a set of samples from the distribution, to test some other system which expects inputs to follow the modeled distribution.

The goal of probabilistic programming is to enable probabilistic modeling and machine learning to be accessible to the working programmer, who has sufficient domain expertise, but perhaps not enough expertise in probability theory or machine learning. We wish to hide the details of inference inside the compiler and run-

Andrew D. Gordon, Thomas A. Henzinger,
Aditya V. Nori, Sriram K. Rajamani,
Probabilistic programming, ICSE-FoSE 2014

Programs + Models

Using ML to provide recommendations during the software life cycle

Using compilers and runtimes to make ML systems flexible and efficient

Having programs adapt when specifications (eg, formats) change

Having programs “self-tune” when environments change

Probabilistic programs: General framework to express rules and examples

Deep combinations of program analysis ideas with ML ideas have the potential to scale probabilistic inference

Challenges and Opportunities (1)

Adaptive specifications and environments

- Need baselines and benchmarks, that capture evolution over time
- Need metrics to measure manual effort in updating annotations as well as precision and recall over time
- Need new user interaction models for involving annotators and users when the system needs human help

Possibility to develop a new field: “model and program engineering”, on how models and programs evolve over time.

The screenshot shows the American Airlines website interface. At the top, it says "Thanks for choosing American Airlines" and "Here's the trip you booked on Orbitz. You'll also find links to other great offers." Below this, it displays flight details for Monday, June 11, 2018, including routes from JAX to PHL and PHL to SYR. The bottom section features promotional banners for mobile boarding passes and earning miles. On the right side, there is a product recommendation section for "Mens Running Shoes - Decathlon" with a list of 16 products. A red box highlights two specific shoe models: "KALENIA KIRGUN LONG MEN'S RUNNING SHOES - BLUE" and "MIZUNO MEN'S RUNNING SHOE WAVE RIDER - BLUE".

```
MSH | ^~\& | ADT1 | MCM | LABADT | MCM | 198808181126 | SECURITY | ADT ^ A01 | MSG00001 - | P | 2, 4  
EWN | A01 | 198808181123  
PID | | | PATID1234 ^ S ^ M11 | | JONES ^ WILLIAM ^ A ^ III | | 19610615 | M - | | C  
PV1 | 1 | I | | 2000 ^ 2012 ^ 01 | | | 004777 ^ LEBAUER ^ SIDNEY ^ J. | | | SUR | | - | | ADM | AO  
ML1 | 1 | | | ^ PENICILLIN | | PRODUCE HIVES - RASH - LOSS OF APPETITE  
DG1 | 001 | I9 | 1550 | MAL NEO LIVER, PRIMARY | 19880501103005 | F  
PR1 | 2234 | M11 | 111 ^ CODE151 | COMMON PROCEDURES | 198809081123
```

MORE LIKE THIS: Kirgun Long, Balance 1000, Kirgun Ho Lights, Mens Balance 1000 Black, Kirgun Fast, Aico Gel Drive, Kirgun Ho Lights

YOUR CHOICE: (1)

FREQUENCY: REGULAR

TYPE OF PRACTICE: ROAD RUNNING (1) (You ran more than 10kms) ATHLETICS (1) (You ran on an athletics track) JOGGING (1) (You ran less than 10 km)

Mens Running Shoes - Decathlon - 16 Products
<https://www.decathlon.co.uk>
Ad: No Excuses this Year - Exercise More in 2019 with 70 Sports at Decathlon!
Discover our collection of men's running shoes
Free Delivery Over £30 - Free Click and Collect - 30 Days Return
<https://www.decathlon.co.uk>
Type of Practice: Road Running, Athletics, Jogging, Trail

| | | |
|-----------------------------------------------|-------------|--------|
| KALENIA - Kirgun Long Running Shoes - Blue | Rating: 4/5 | £59.99 |
| MIZUNO - Men's Running Shoe Wave Rider - Blue | Rating: 5/5 | £79.99 |

Challenges and Opportunities (2)

Assurance of ML/AI systems using verified monitoring

- Can we write partial specifications for safety critical ML/AI systems?
- Can we synthesize monitors to “safeguard” such systems even if the ML/AI algorithms are hard to verify?
- How do we evolve the specifications over time?

“An Inductive Synthesis Framework for Verifiable Reinforcement Learning”, He Zhu, Zikang Xiong, Stephen Magill, Suresh Jagannathan, PLDI 2019

Challenges and Opportunities (3)

Practical and usable frameworks to combine domain knowledge (rules) with empirical knowledge (examples and data)

- Technical challenge:
 - Scaling probabilistic inference
 - Usable programming languages and notation
- Industrial challenge: Development life cycle and tools for probabilistic programs
- Educational challenge:
 - Developing educational material and pedagogy for modeling rules and empirical knowledge together