CS4400/5400 Programming Languages

Spring 2024

Instructor: Steven Holtzen

s.holtzen@northeastern.edu

Northeastern University
Khoury College of Computer and Information Sciences

Final logistics

- Quiz tomorrow, homework due tonight
- Final grades should be done by next Wednesday
- Please complete trace evaluation
- Thanks for being the first class ☺

Questions

Why are new languages being made today?

Why are there so many programming languages?

Which language should I learn? Should I use?

Are some languages worse than others? Better? How can I compare them?

What distinguishes one language from another?

How are new languages made?

Why study programming languages?

- Be a more effective programmer
  - How to *choose* languages for your problems
  - How to *design and implement* languages when needed

- Become equipped to *learn new languages quickly*

- Be *prepared for an evolving world*
  - New languages are showing up all the time

- • Enjoy an *aesthetic journey* through this elegant field

Big course themes and topics

Language implementation

Type systems and type safety

Compilers

Growing languages

Control flow

The lambda calculus

Memory safety

Runtime/dynamic safety

What is a “programming language”?

“A programming language is a system of notation for writing computer programs.”

https://en.wikipedia.org/wiki/Programming_language
Accessed Friday, January 5

This course evolved as it went

Our original schedule

CS4400 Spring'24 Schedule : Sheet1

Module	Date	Lecture title	Topics	Resources
Introduction to Plait and Functional Programming	Monday Jan 8	Introduction and course overview	- Why programming languages? - A first look at Racket/Plait	https://docs.racket-lang.org/plait/Tutorial.html Sections 1.1 -- 1.5
	Wednesday Jan 10	Programming in Racket and Plait	- Solving problems functionally and recursively - How to write tests in Plait - How Plait's type system works	https://docs.racket-lang.org/plait/Tutorial.html Sections 1.1 -- 1.8
	Monday Jan 15	**No Class --- Martin Luther King Jr. Day**		
	Wednesday Jan 17	**No Class --- Steven Traveling for POPL**	- Optional in-class workshop on Plait	
Building a SMoL Interpreter	Monday Jan 22	Abstract Syntax	- Abstract syntax trees - A simple calculator language - Interpreting a language by hand	PLAI pg. 17 -- 27 See https://www.plai.org/3/2/PLAI%20Version%203.2.2%20printing.pdf
	Wednesday Jan 24	Evaluation	- Implementing and testing the evaluator - Parsing and s-expressions	PLAI pg. 28 -- 37
	Monday Jan 29	Conditionals	- Extending the syntax with \`if\` - Design space of \`if\` - Adding Booleans, the \`Value\` type	PLAI pg. 37 -- 47
	Wednesday Jan 31	Local binding	- The \`let\` syntax - Scope - An evaluator for \`let\`	PLAI pg. 47 -- 57
	Monday Feb 5	First-class functions	- Syntax for functions - Adding functions to the \`Value\` type - Evaluating functions	PLAI pg. 58 -- 69
	Wednesday Feb 7	Growing SMoL: Macros	- Desugaring - An example: Strict If - define-syntax - Macro stepping in DrRacket	PLAI pg. 71 -- 84
	Monday Feb 12	Objects I	- The "standard model" of objects - State - Access control	PLAI pg. 85 -- 95

This course evolved as it went

Our original schedule

Types	Wednesday Feb 14	Objects II	- Extending objects: mixins, traits	PLAI pg. 97 -- 106
	Monday Feb 19	Introduction to types	- What are types? - A simple type checker - How to read and write typing judgments	PLAI pg. 109 -- 122
	Wednesday Feb 21	Typing functions	- The typing rule for functions - Assume-guarantee reasoning - Making a typechecker - Handling recursion	PLAI pg. 123 -- 132
	Monday Feb 26	The Simply Typed Lambda Calculus	- Syntax and a type checker - The Omega term and normalization	
	Wednesday Feb 28	Safety and soundness	- What is type safety, why do you want it - Enforcing type safety - Type safety for simply-typed lambda calculus	PLAI pg. 133 -- 144
	Monday Mar 4	**No Class -- Spring Break**		
	Wednesday Mar 6	**No Class -- Spring Break**		
	Monday Mar 11	Type inference	- Basic goals of type inference - Hindley-Milner - Complexity of type inference	PLAI pg. 145 -- 149
	Wednesday Mar 13	Algebraic datatypes and pairs	- Typechecking algebraic datatypes and pairs - Proofs and programs: Curry-Howard	PLAI pg. 150 -- 153
	Monday Mar 18	Subtyping	- Adding subtyping to typing judgments - Applications: information flow analysis	PLAI pg. 165 -- 170
	Wednesday Mar 20	Gradual typing	- TypeScript, typed Python, Typed Racket	PLAI pg. 170 -- 176
Paradigms	Monday Mar 25	Logic Programming I	- Programming with relations - Unification - A simple type checker	PLAI pg. 178 -- 184
	Wednesday Mar 27	Logic Programming II		PLAI pg. 193 -- 202
	Monday Apr 1	Laziness I	- Evaluation schemes: eager, lazy, call-by-need, call by name - Consequences of evaluation schemes - A lazy evaluator - Programming in lazy languages	
	Wednesday Apr 3	Laziness II	- Modeling state and mutation - A taste of Haskell	
	Monday Apr 8	Effects I	- Effects in Racket - Effect handlers	
	Wednesday Apr 10	Effects II		
	Monday Apr 15	**No Class -- Patriot's day**		
	Wednesday Apr 17	Slack day		

Why the changes?

- Some minor changes in pacing based on how fast things were going
- Big topic shifts:
  - Memory safety
  - Continuations

In response to:
- Expressed interest from students in learning Rust
- Government announcements on memory safety
- Ongoing research projects involving Rust becoming more interesting to me
- My own enjoyment of Rust

In response to:
- Research directions involving continuations
- Emerging programming language patterns (effect handlers, co-routines, etc.) involving continuations

Module 1: Growing a language

- Tiny language: calculator
  - Studied scope, syntax, semantics

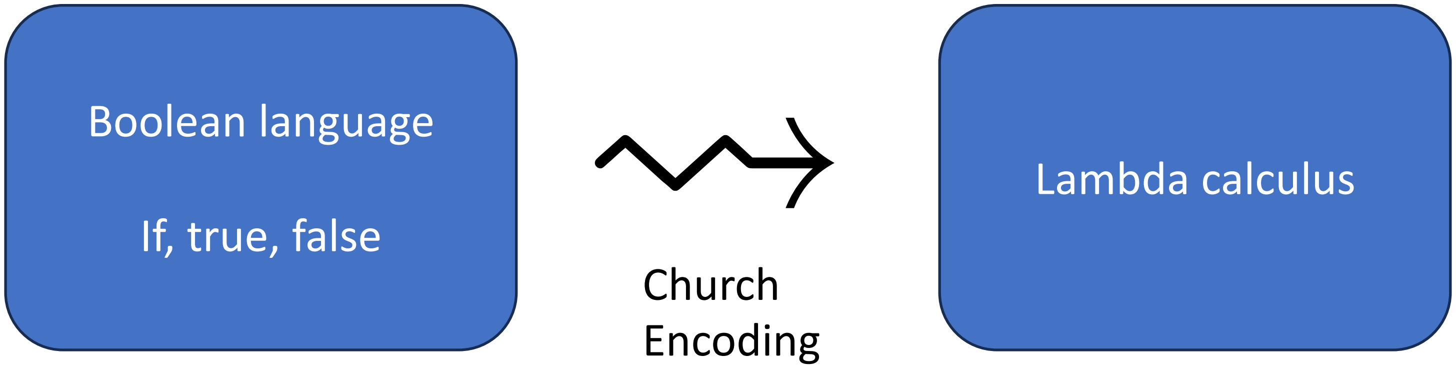
- Somehow even tinier language: λ-calculus

$$e ::= (\lambda x.e) \mid (e\ e) \mid x$$

- Implementation *with substitution* and *with environments*

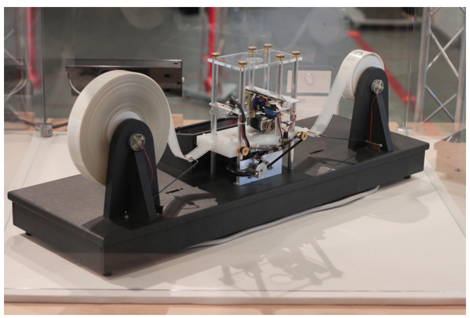
Module 1: Growing a language

- How big can we make the lambda calculus?



Module 1: Growing a language

- How big can we make the lambda calculus?



Any Turing machine

Church
Encoding

Lambda calculus

Module 1: Growing a language

- How do we make loopy programs?

$$\Omega = (\lambda x.(x\ x))\ (\lambda x.(x\ x))$$

- Want more? Check out the Y-combinator
  - A great blog post: https://matt.might.net/articles/python-church-y-combinator/

Bonus content: big-step semantics

- Remember type judgments? We can use those to describe how to run programs too

$$e ::= (e + e) \mid \text{num} \mid \texttt{let x = } e_1 \texttt{ in } e_2 \mid x$$

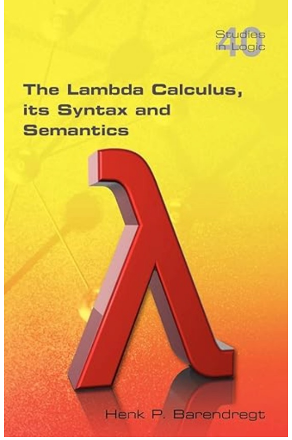
$$\frac{}{\texttt{num} \Downarrow \text{num}}$$

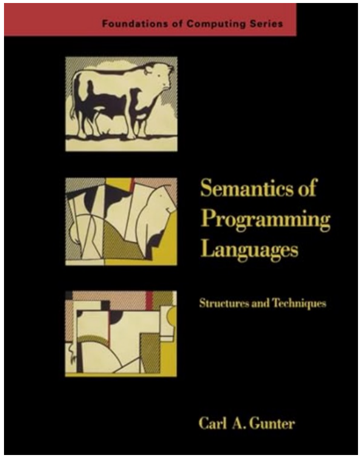
$$\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{(e_1 + e_2) \Downarrow v_1 + v_2}$$

$$\frac{e_1 \Downarrow v_1 \quad e_2[x \mapsto v] \Downarrow v_2}{\texttt{let x = } e_1 \texttt{ in } e_2 \Downarrow v_2}$$

Other aspects of semantics

- Small-step semantics: describe a program's behavior by the *sequence of steps* it takes to evaluate
- Denotational semantics: describe a program's behavior by associating it with a *mathematical object* (like a set)





Module 2: Types

- We studied how to design systems to prevent runtime errors

Given an interpreter…

$$e ::= (\lambda x.e) \mid (e\ e) \mid x$$

…design a type system that prevents runtime errors in that interpreter

$$\frac{}{\Gamma \vdash \texttt{num} : \texttt{Number}}\ \text{(T-Num)} \qquad \frac{x \in \Gamma \qquad \Gamma(x) = \tau}{\Gamma \vdash \texttt{x} : \tau}\ \text{T-Var}$$

$$\frac{\Gamma \cup \{\texttt{x} \mapsto \tau\} \vdash \texttt{e} : \tau'}{\Gamma \vdash \lambda\ \texttt{x}{:}\tau.\ \texttt{e} : \tau \rightarrow \tau'}\ \text{(T-Lambda)} \qquad \frac{\Gamma \vdash \texttt{e1} : \tau \rightarrow \tau' \qquad \Gamma \vdash \texttt{e2} : \tau}{\Gamma \vdash (\texttt{e1 e2}) : \tau'}\ \text{(T-App)}$$

Other aspects of types we didn't cover

- Existential and universal types
- Recursive types
- Dependent types
- **Connection between types and logic**
- Modules
- Calculus of constructions and formal verification

Propositions as types teaser

https://cacm.acm.org/research/propositions-as-types/

Phil Wadler

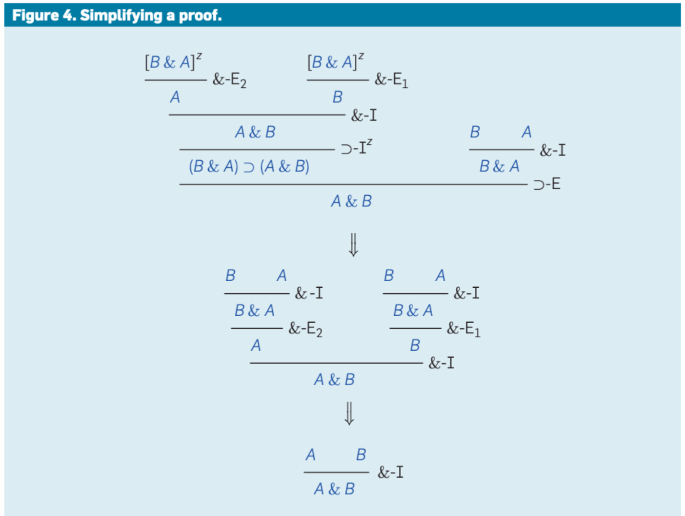


Figure 4. Simplifying a proof.

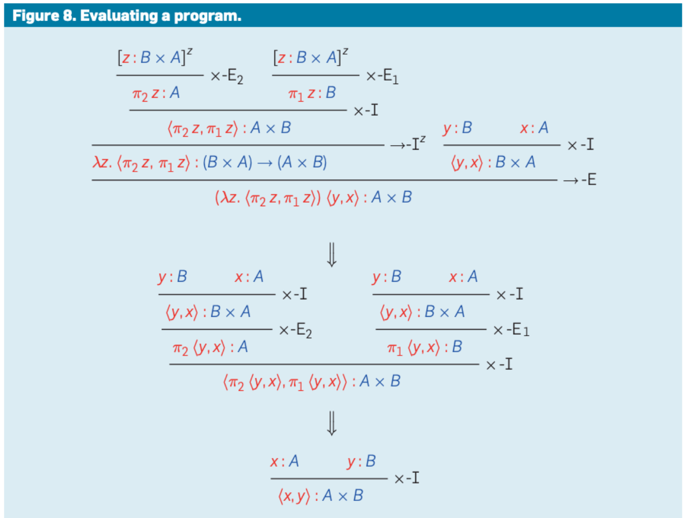


Figure 8. Evaluating a program.

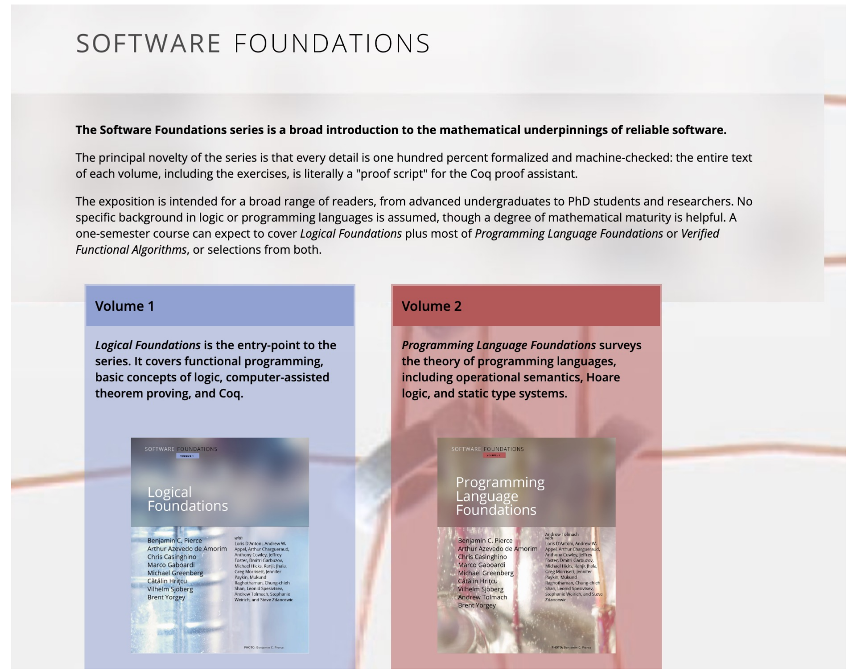
Formal verification

https://softwarefoundations.cis.upenn.edu/

Coq proof assistant

- There is a whole industry of *proving software correct* using programming language tools

- After this class, you are ready to explore this topic



Module 3: Memory safety

- Memory safety errors are pervasive and terrible

Heartbleed is a security bug in some outdated versions of the OpenSSL cryptography library, which is a widely used implementation of the Transport Layer Security (TLS) protocol. It was introduced into the software in 2012 and publicly disclosed in April 2014. Heartbleed could be exploited regardless of whether the vulnerable OpenSSL instance is running as a TLS server or client. It resulted from improper input validation (due to a missing bounds check) in the implementation of the TLS heartbeat extension.[5] Thus, the bug's name derived from *heartbeat*.[6] The vulnerability was classified as a buffer over-read,[7] a situation where more data can be read than should be allowed.[8]

Module 3: Memory safety

- A language-design approach: we can make memory-safety errors impossible by preventing low-level memory manipulation

OCaml

> Studied how to compile these languages into memory-unsafe languages

- Problem: *performance*!

Module 3: Memory safety

- A trend in modern language design: *memory safety + performance*

Rust

A language empowering everyone
to build reliable and efficient software.

Oxidizing OCaml: Locality

MAY 26, 2023 | 15 MIN READ

https://blog.janestreet.com/oxidizing-ocaml-locality/

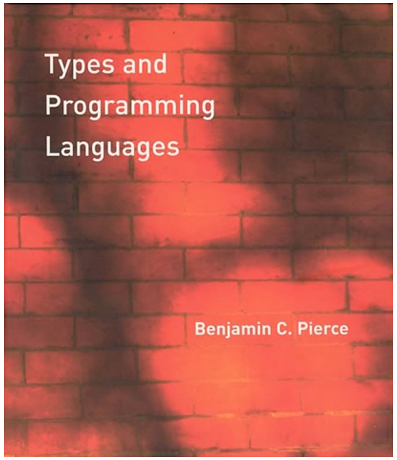
OCaml

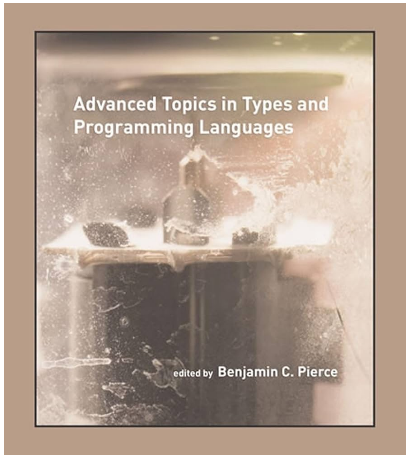
Module 4: Control & continuations

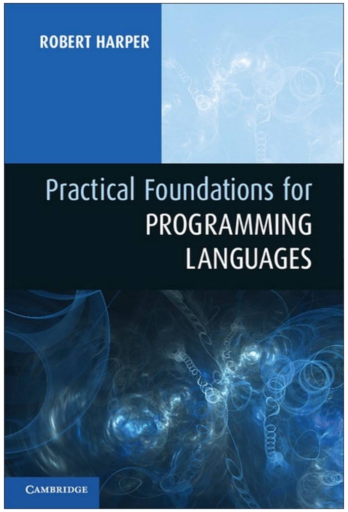
- **Theme**: how do we *implement interpreters* for languages with *interesting control-flow?*
  - Saw how *continuations* give us a way to implement control-flow constructs like exception handling
  - See how *continuation-passing style* lets us compile languages with interesting control-flow into simpler languages
- Forms the foundations for compiling functional programs
- The ideas come up in interesting places: call-backs in JavaScript, co-routines and concurrent programming, optimizing recursive programs

Resources

- Software foundations: https://softwarefoundations.cis.upenn.edu/
- Programming language foundations in Agda https://plfa.github.io/
- Types and Programming Languages by Ben Pierce
- Practical Foundations for Programming Languages by Bob Harper







Other courses at Northeastern

- Compilers (CS4410)
  - https://course.ccs.neu.edu/cs4410sp24
- Intensive programming languages (IPPL), CS7400
  - https://www.khoury.northeastern.edu/home/amal/course/7400-s15/
- Graduate seminars
  - CS7470/CS7480 (here's mine: https://neuppl.github.io/CS7470-Fall23/)

Research in PL at Northeastern

And beyond

Research overview at Northeastern

Amal Ahmed

- Research themes: broadly in programming language theory
  - Language interoperability
  - Type systems
  - Safe compilation
  - Rust, WASM

- Looking to recruit students working on web-assembly: theoretical and practical projects are available

Gradually Typed Languages Should be Vigilant!.
Olek Gierczak, Lucy Menon, Christos Dimoulas, and Amal Ahmed.
Proc. ACM Program. Lang. 8, OOPSLA1, Article 125, 29 pages, Apr 2024.

Semantic Encapsulation Using Linking Types.
Daniel Patterson, Andrew Wagner, and Amal Ahmed.
In *ACM SIGPLAN International Workshop on Type-Driven Development (TyDe '23)*, Seattle, Washington, September 2023.

Lilac: A Modal Separation Logic for Conditional Probability.
John M. Li, Amal Ahmed, and Steven Holtzen.
Proc. ACM Program. Lang. 7(PLDI):148-171 (2023).

ANF Preserves Dependent Types up to Extensional Equality.
Paulette Koronkevich, Ramon Rakow, Amal Ahmed, and William J. Bowman.
Journal of Functional Programming, 32, E22, 2022.

Semantic Soundness for Language Interoperability.
Daniel Patterson, Noble Mushtak, Andrew Wagner, Amal Ahmed.
In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'22)*, San Diego, California, June 2022.

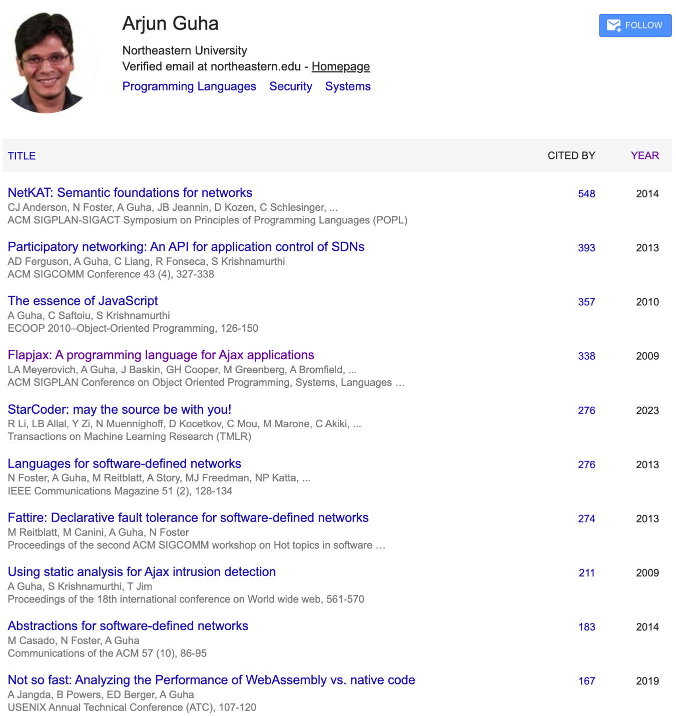
Gradual Type Theory.
Max S. New, Daniel R. Licata, Amal Ahmed.
Journal of Functional Programming, 31, E21, 2021.

Graduality and Parametricity: Together Again for the First Time.
Max S. New, Dustin Jamner, and Amal Ahmed.
In *ACM SIGPLAN Symposium on Principles of Programming Languages (POPL '20)*, New Orleans, Louisiana, January 2020.
Technical appendix, November 2019.

Research overview at Northeastern

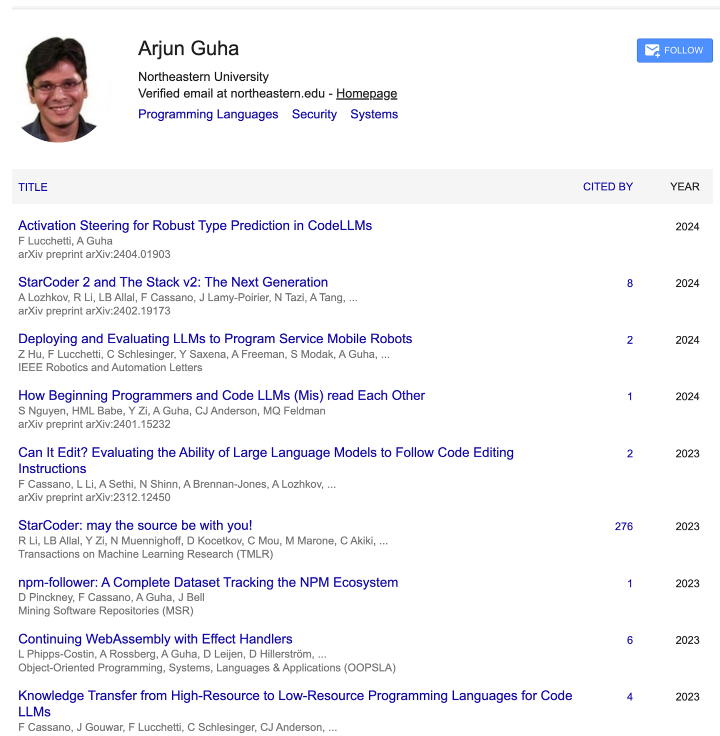
Arjun Guha

- Research themes: currently, language models for code



Arjun Guha
Northeastern University
Verified email at northeastern.edu - Homepage
Programming Languages Security Systems
FOLLOW

TITLE	CITED BY	YEAR
NetKAT: Semantic foundations for networks CJ Anderson, N Foster, A Guha, JB Jeannin, D Kozen, C Schlesinger, ... ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)	548	2014
Participatory networking: An API for application control of SDNs AD Ferguson, A Guha, C Liang, R Fonseca, S Krishnamurthi ACM SIGCOMM Conference 43 (4), 327-338	393	2013
The essence of JavaScript A Guha, C Saftoiu, S Krishnamurthi ECOOP 2010–Object-Oriented Programming, 126-150	357	2010
Flapjax: A programming language for Ajax applications LA Meyerovich, A Guha, J Baskin, GH Cooper, M Greenberg, A Bromfield, ... ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages ...	338	2009
StarCoder: may the source be with you! R Li, LB Allal, Y Zi, N Muennighoff, D Kocetkov, C Mou, M Marone, C Akiki, ... Transactions on Machine Learning Research (TMLR)	276	2023
Languages for software-defined networks N Foster, A Guha, M Reitblatt, A Story, MJ Freedman, NP Katta, ... IEEE Communications Magazine 51 (2), 128-134	276	2013
Fattire: Declarative fault tolerance for software-defined networks M Reitblatt, M Canini, A Guha, N Foster Proceedings of the second ACM SIGCOMM workshop on Hot topics in software ...	274	2013
Using static analysis for Ajax intrusion detection A Guha, S Krishnamurthi, T Jim Proceedings of the 18th international conference on World wide web, 561-570	211	2009
Abstractions for software-defined networks M Casado, N Foster, A Guha Communications of the ACM 57 (10), 86-95	183	2014
Not so fast: Analyzing the Performance of WebAssembly vs. native code A Jangda, B Powers, ED Berger, A Guha USENIX Annual Technical Conference (ATC), 107-120	167	2019



Arjun Guha
Northeastern University
Verified email at northeastern.edu - Homepage
Programming Languages Security Systems
FOLLOW

TITLE	CITED BY	YEAR
Activation Steering for Robust Type Prediction in CodeLLMs F Lucchetti, A Guha arXiv preprint arXiv:2404.01903		2024
StarCoder 2 and The Stack v2: The Next Generation A Lozhkov, R Li, LB Allal, F Cassano, J Lamy-Poirier, N Tazi, A Tang, ... arXiv preprint arXiv:2402.19173	8	2024
Deploying and Evaluating LLMs to Program Service Mobile Robots Z Hu, F Lucchetti, C Schlesinger, Y Saxena, A Freeman, S Modak, A Guha, ... IEEE Robotics and Automation Letters	2	2024
How Beginning Programmers and Code LLMs (Mis) read Each Other S Nguyen, HML Babe, Y Zi, A Guha, CJ Anderson, MQ Feldman arXiv preprint arXiv:2401.15232	1	2024
Can It Edit? Evaluating the Ability of Large Language Models to Follow Code Editing Instructions F Cassano, L Li, A Sethi, N Shinn, A Brennan-Jones, A Lozhkov, ... arXiv preprint arXiv:2312.12450	2	2023
StarCoder: may the source be with you! R Li, LB Allal, Y Zi, N Muennighoff, D Kocetkov, C Mou, M Marone, C Akiki, ... Transactions on Machine Learning Research (TMLR)	276	2023
npm-follower: A Complete Dataset Tracking the NPM Ecosystem D Pinckney, F Cassano, A Guha, J Bell Mining Software Repositories (MSR)	1	2023
Continuing WebAssembly with Effect Handlers L Phipps-Costin, A Rossberg, A Guha, D Leijen, D Hillerström, ... Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)	6	2023
Knowledge Transfer from High-Resource to Low-Resource Programming Languages for Code LLMs F Cassano, J Gouwar, F Lucchetti, C Schlesinger, CJ Anderson, ... arXiv preprint arXiv:2308.09895	4	2023

What is research?

- There are many goals, but at a high level, the goal is:
  - Deeply explore an area to understand its problems
  - Make progress towards solutions
- Main product of research is an academic paper. In general, it consists of:
  1. Problem statement and motivation
  2. Proposed problem solution
  3. Evidence for the quality and impact of the solution
  4. Related work
- Wonderful essay: "You and your research" https://www.cs.virginia.edu/~robins/YouAndYourResearch.html

What is research?

The structure of research at a university: the PhD. student

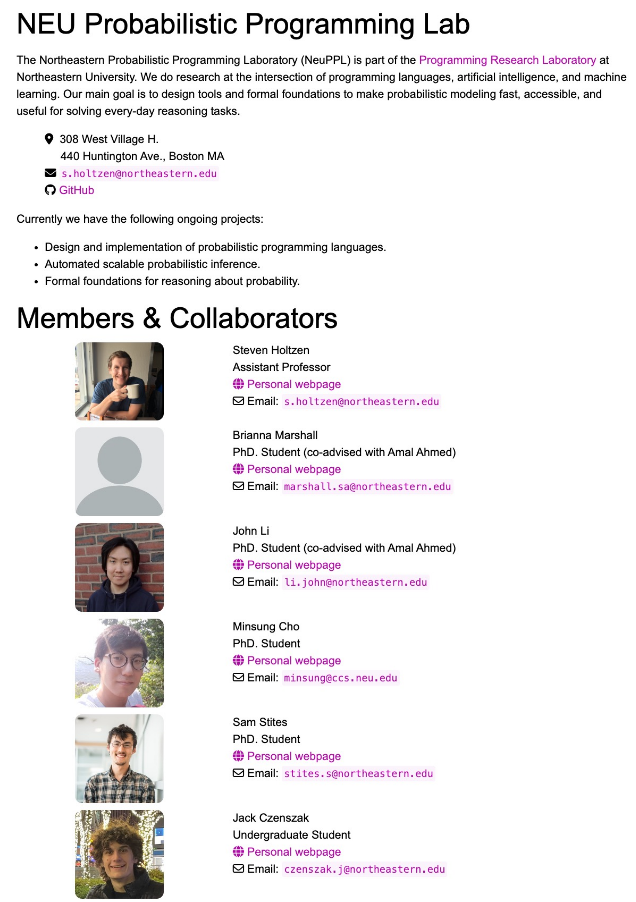
- PhD. students are fully-funded (meaning, they get a stipend) to do research
- Graduate in 5-6 years
- Highly competitive: hundreds of applicants for very small number of slots
- A prerequisite to becoming a professor at a university or pursuing other research-oriented jobs
- Main responsibilities:
    - Write papers
    - Help teach courses
    - Present work at conferences

Minsung Cho
PhD. Student
Personal webpage
Email: minsung@ccs.neu.edu

What is research?

The structure of research at a university: the research group

- Led by a tenure-track faculty-member called an **advisor**
- Majority of the group are PhD. students who are actively working on papers
  - Sometimes there are also post-docs: staff who have finished a PhD. but are not professors
- Advisor's job is helping select and guide research projects, raise funding
  - In addition to other responsibilities like teaching and service



What is research?

The structure of research at a university: the research area

- Above the research group is often a research area that organizes several groups together
- Consists of many faculty and PhD. students

https://prl.khoury.northeastern.edu/

What is research?

The structure of research at a university: the college

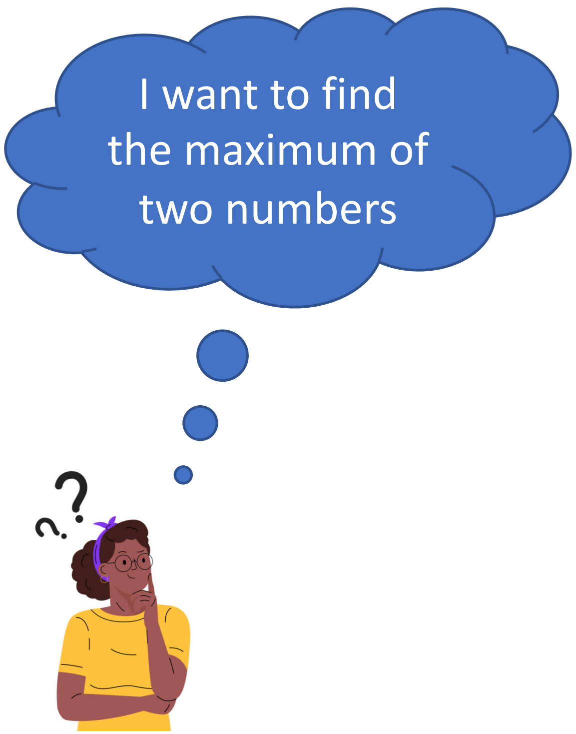
- Northeastern is a *high-research-output* (R1) university
  - 15,000 graduate students, 3000 faculty

- A large amount of university resources go into cultivating a research environment (in addition to a teaching environment)

PL Meets ML

A talk given at the “Programming Languages Mentorship Workshop” an PLDI 2024

A **programming language** is a language for unambiguously describing intent to the computer




```
def max(a, b):
    if a < b:
        return b
    else:
        return a
```

Programs can have ***bugs:*** when its input/output pair is not what the programmer intended

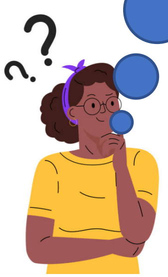
max(0, 10)

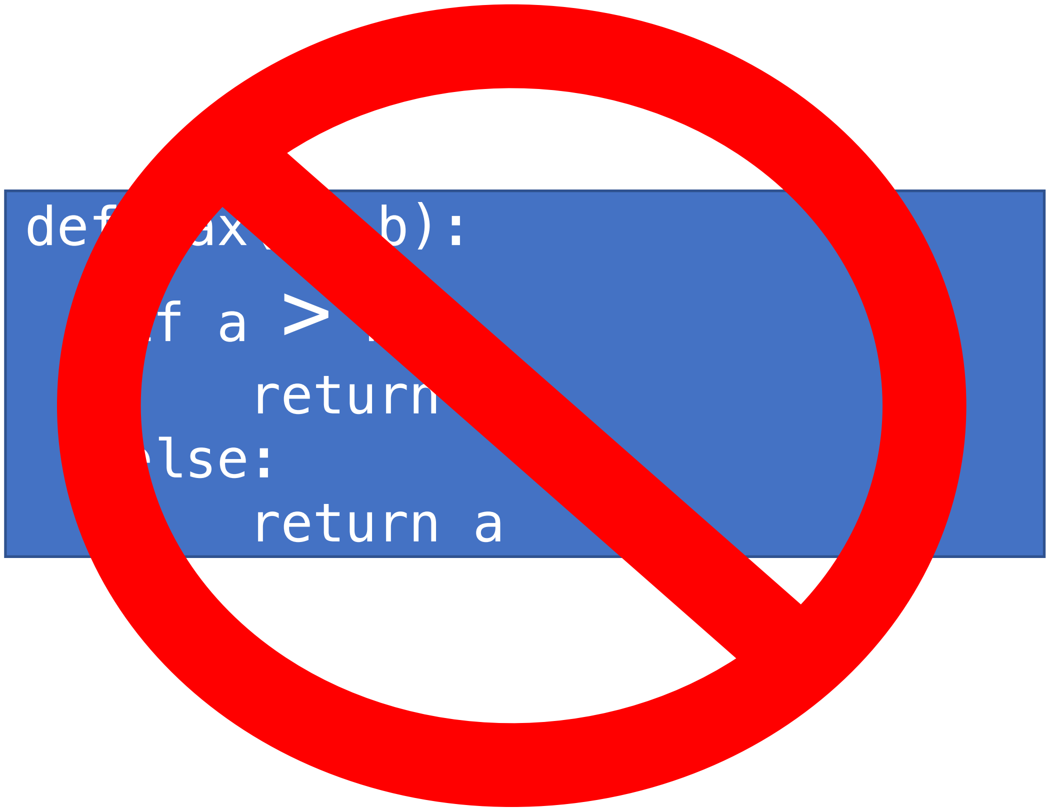
```
def max(a, b):
    if a > b:
        return b
    else:
        return a
```

0

We can (sometimes) even prove a program does the right thing!

Forall a, b.
max(a,b) >= a
&&
max(a,b) >= b

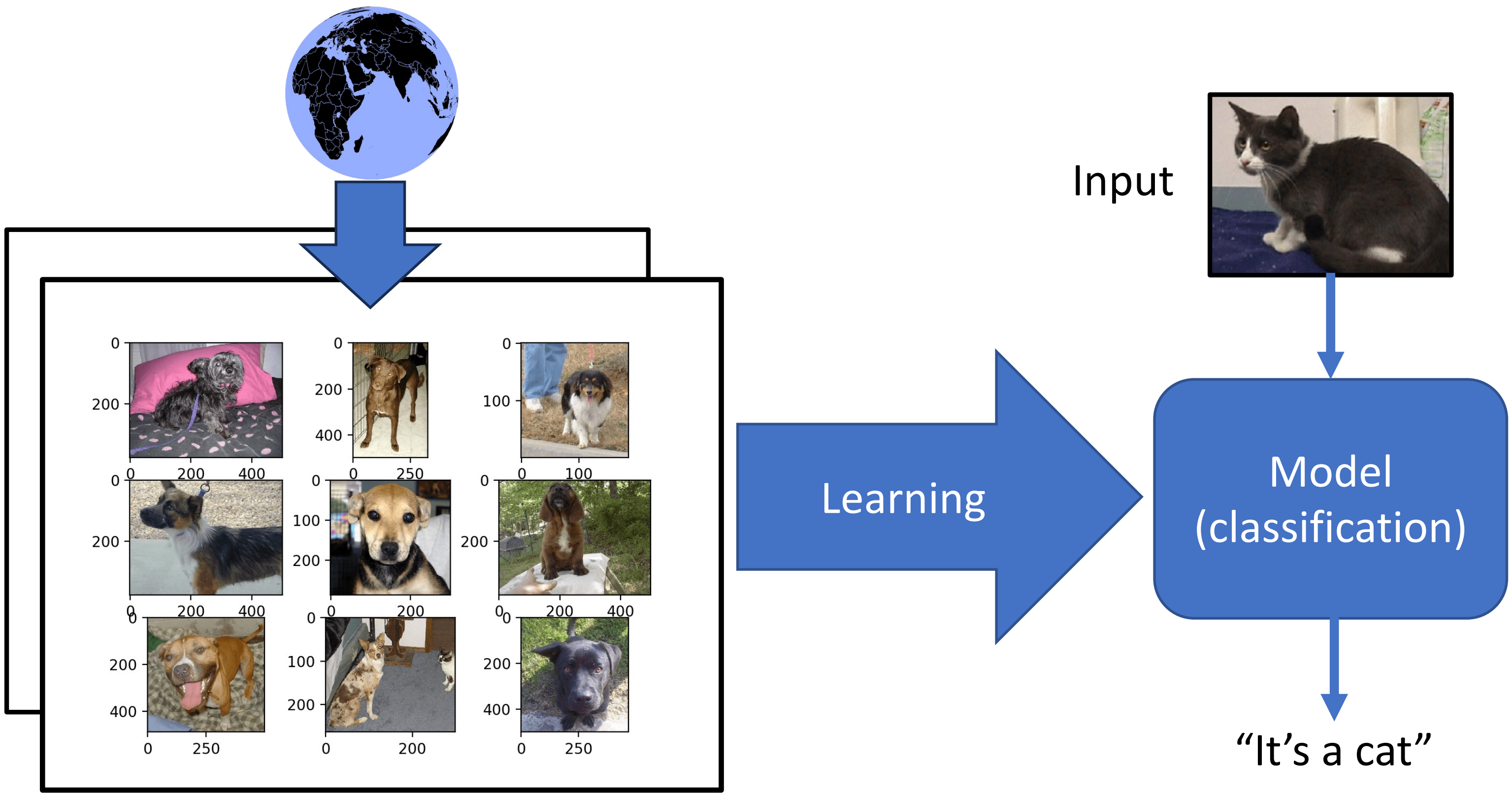




Machine learning

Machine learning is a means for using *data* to describe intent to the computer

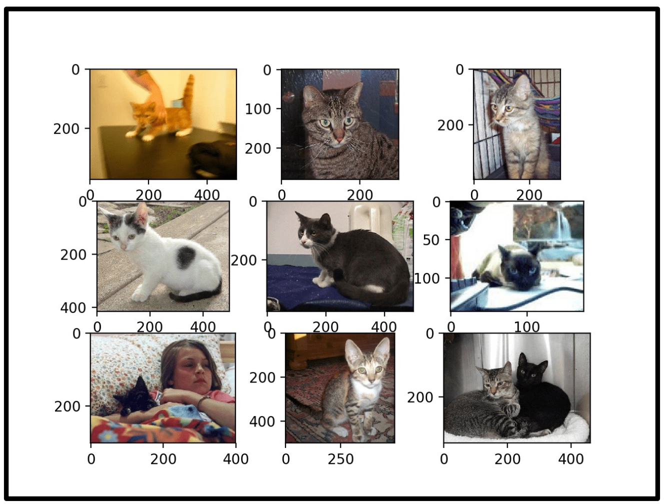
Two parts: Data + Model

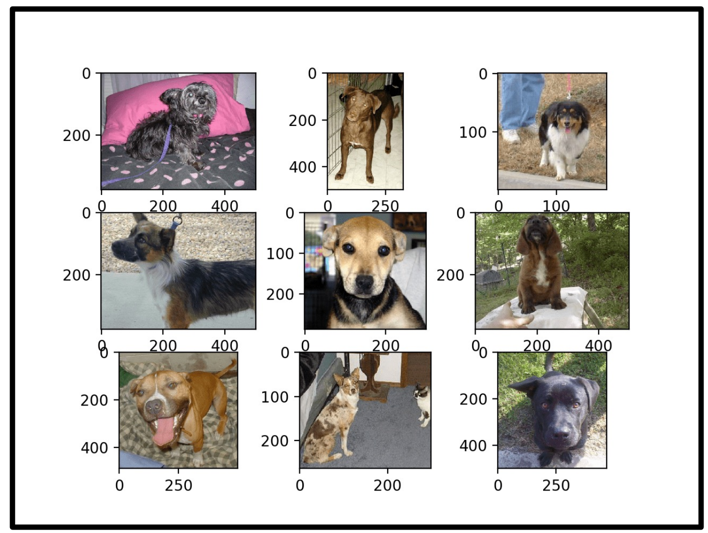


Good luck writing a program that can do this!

True goal of ML: Generalization

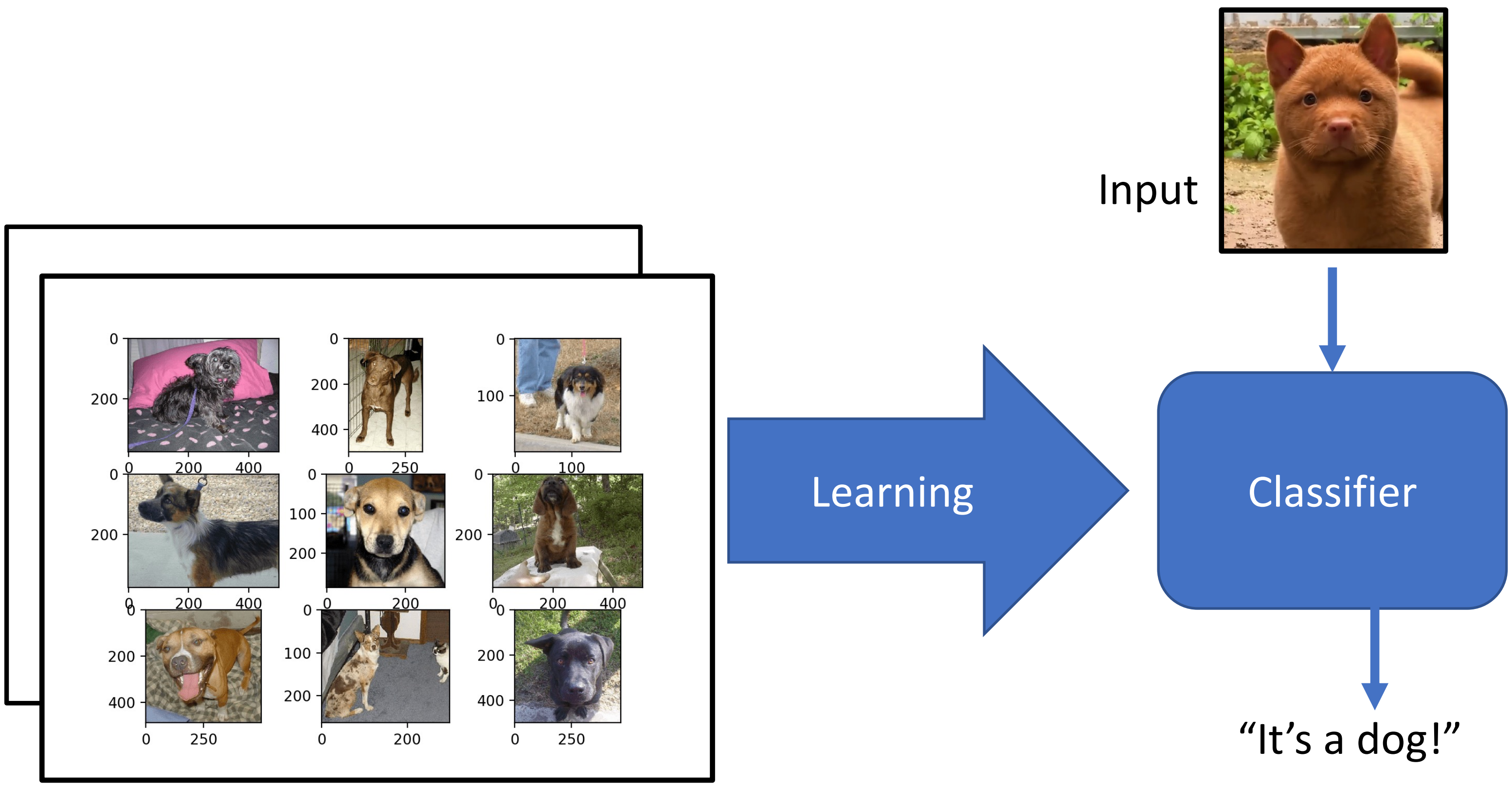
- Given some finite dataset describing the world…

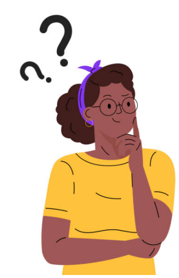




- … generalize to instances beyond that dataset and correctly predict things

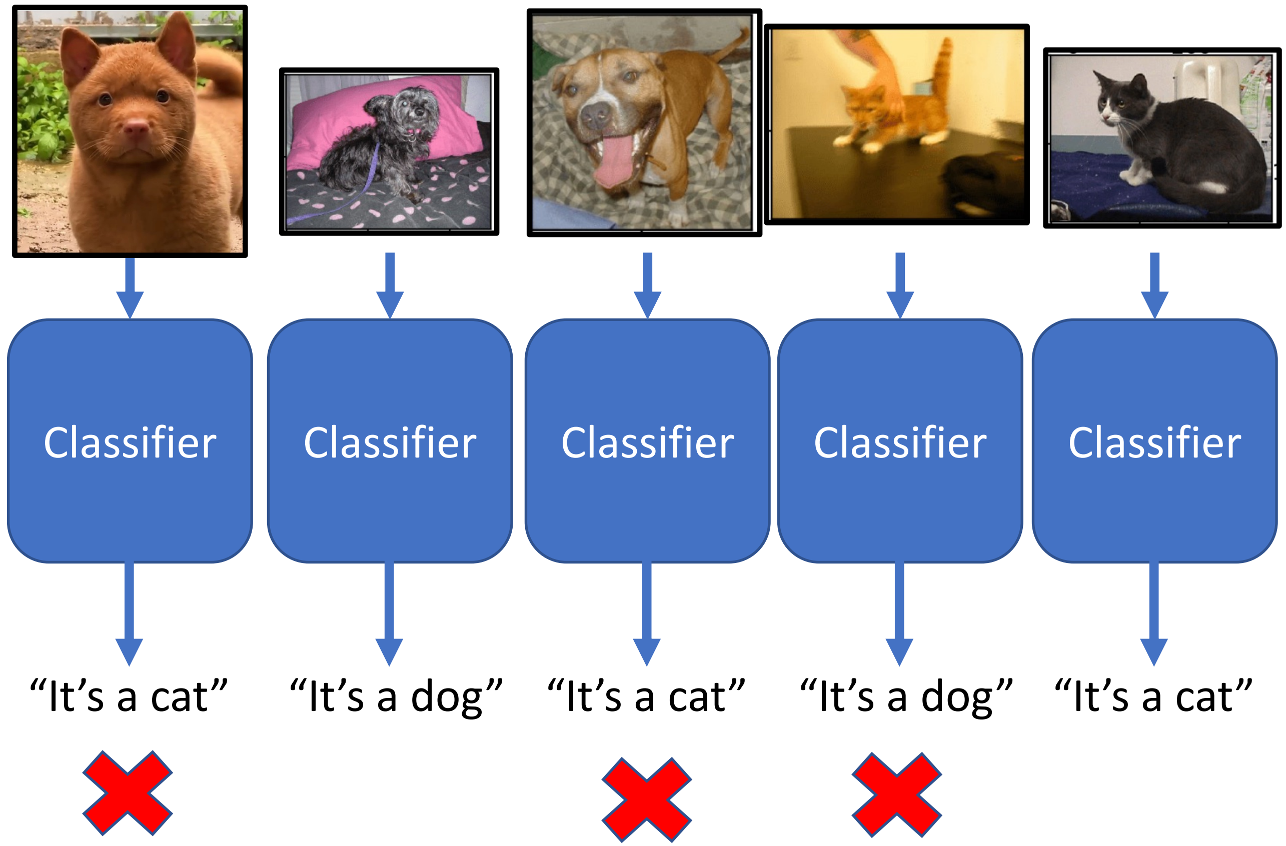
Machine learning must be wrong sometimes!





Measuring generalization: Accuracy

Test set
(Not used
for training)



Proportion of correctly predicted images on the test set is called ***accuracy***

The perils of accuracy

Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification*

Joy Buolamwini JOYAB@MIT.EDU
MIT Media Lab 75 Amherst St. Cambridge, MA 02139

Timnit Gebru TIMNIT.GEBRU@MICROSOFT.COM
Microsoft Research 641 Avenue of the Americas, New York, NY 10011

Classifier	Metric	DF	DM	LF	LM
MSFT	TPR(%)	76.2	**100**	**100**	**100**
	Error Rate(%)	**23.8**	0.0	0.0	0.0
	PPV(%)	**100**	84.2	**100**	**100**
	FPR(%)	0.0	**23.8**	0.0	0.0
Face++	TPR(%)	64.0	99.5	92.6	**100**
	Error Rate(%)	**36.0**	0.5	7.4	0.0
	PPV(%)	99.0	77.8	**100**	96.9
	FPR(%)	0.5	**36.0**	0.0	7.4
IBM	TPR(%)	66.9	94.3	**100**	98.4
	Error Rate(%)	**33.1**	5.7	0.0	1.6
	PPV(%)	90.4	78.0	96.4	**100**
	FPR(%)	5.7	**33.1**	1.6	0.0

3 major computer-vision-based gender recognition tools had a bug!

Light-skinned female error rate: 0%
Dark-skinned female (DF) error rate: 23.8%!

Buolamwini, Joy, and Timnit Gebru. "Gender shades: Intersectional accuracy disparities in commercial gender classification." Conference on fairness, accountability and transparency. PMLR, 2018.

Programming Languages

Behavior determined by program

Does exactly what the programmer says

Logical specification in terms of inputs and outputs

Machine Learning

Behavior determined by model + data

Generalizes beyond what programmer says

Correctness is a property of the world and the program

Complementary strengths and weaknesses

Big Challenges for ML Meets PL

Combining strengths and weaknesses of each

1. Synthesizing learning and programming

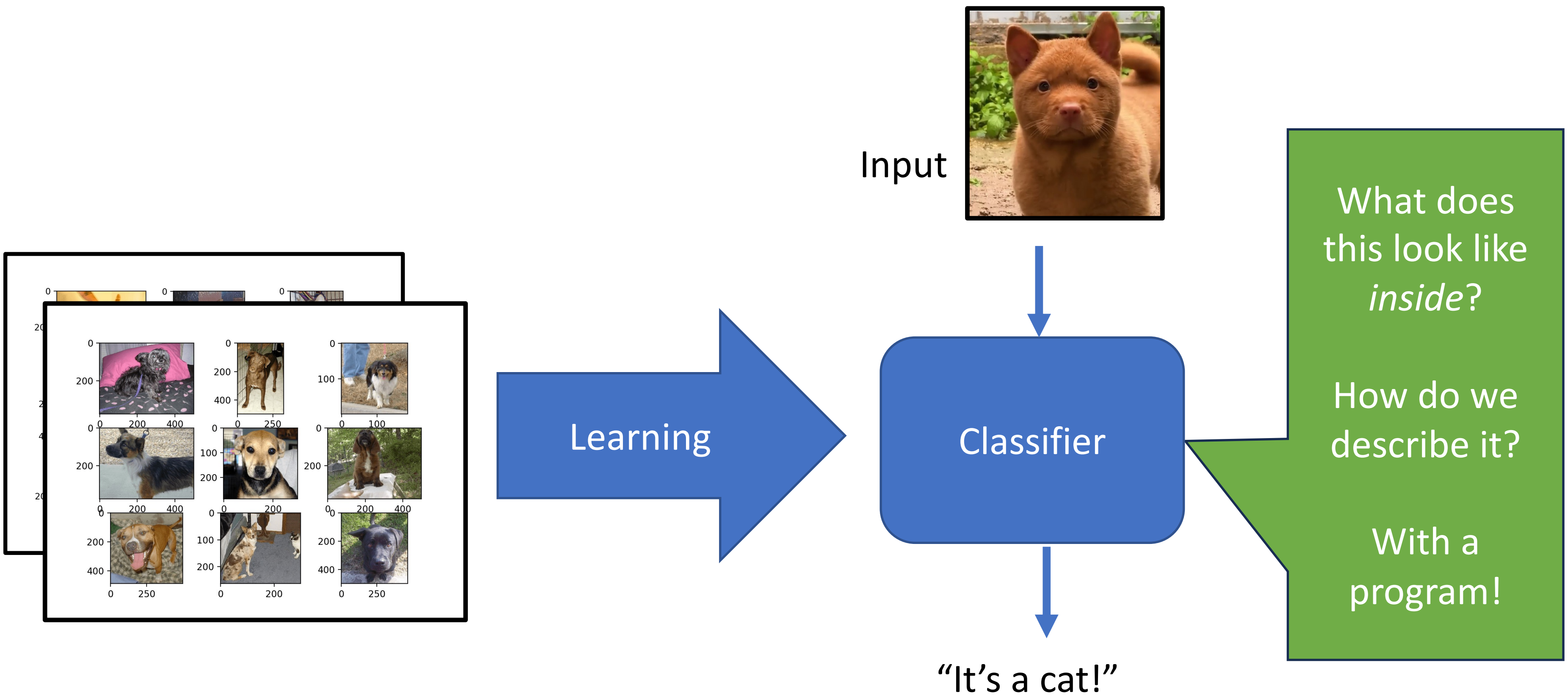
2. Verifying systems with learned components

3. Harnessing generative models

4. …?

Grand challenge #1:
Synthesizing learning and programming

Structured models



Model structure is critical

Accuracy & generalizability

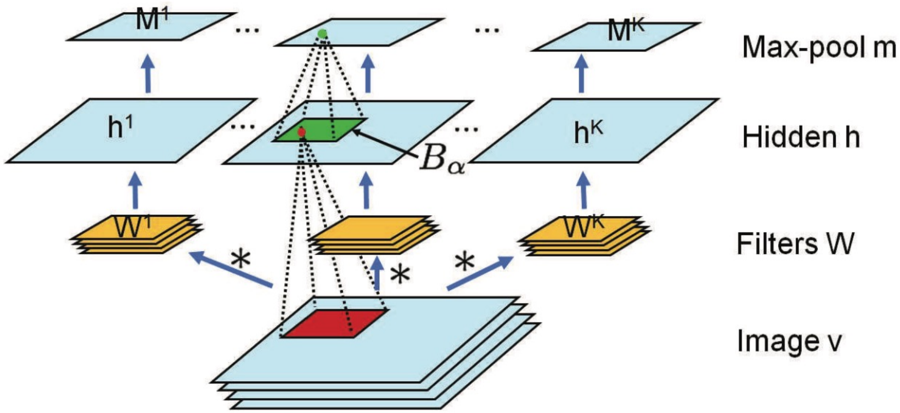


Fig 28.8: Convolutional Neural Network
Machine Learning: A Probabilistic Perspective

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

More reasons…

- Data efficiency
- Control
- Reliability

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84-90.

PL for Model Description

- **Big idea**: use a program to describe the model!

- Why use a PL to describe a model?

  - Accessible: any programmer can make a model

  - Expressive: full descriptive power of a PL

Why accessibility matters for making ML systems

The New York Times

Opinion

OPINION

Artificial Intelligence's White Guy Problem

By Kate Crawford
June 25, 2016

2016!!!

Kate Crawford
Sr. Principal Researcher
Microsoft Research

Like all technologies before it, artificial intelligence will reflect the values of its creators. So inclusivity matters — from who designs it to who sits on the company boards and which ethical perspectives are included. Otherwise, we risk constructing machine intelligence that mirrors a narrow and privileged vision of society, with its old, familiar biases and stereotypes.

Languages for building ML models

Differentiable Programming

JuliaDiff

PyTorch

TensorFlow

tinygrad

theano

Probabilistic Programming

Stan

Pyro

ProbLog

Gen

FairSquare

Psi

Dice

...

Approach #1: Differentiable Programming

Example system: Jax

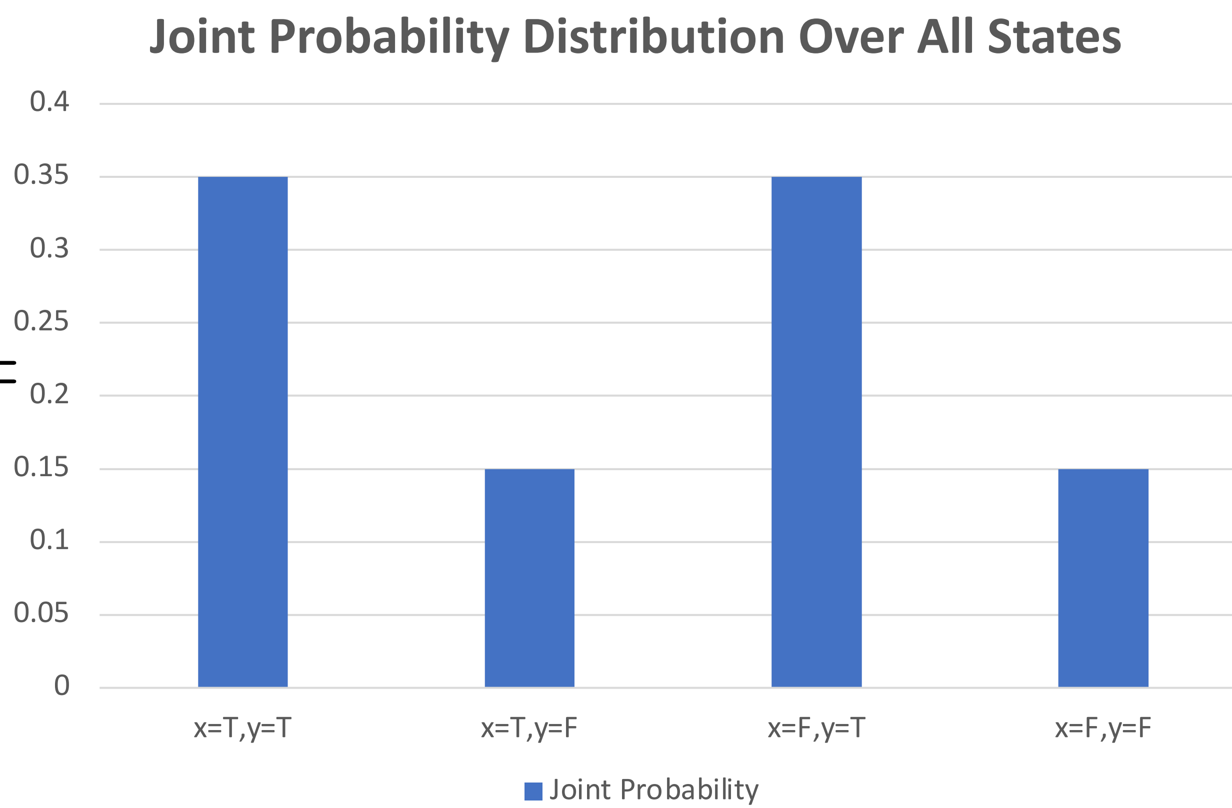
```
from jax import grad
import jax.numpy as jnp

def tanh(x):  # Define a function
  y = jnp.exp(-2.0 * x)
  return (1.0 - y) / (1.0 + y)

grad_tanh = grad(tanh)  # Obtain its gradient function
print(grad_tanh(1.0))   # Evaluate it at x = 1.0
# prints 0.4199743
```

Great blog post: https://thenumb.at/Autodiff/

Approach #2: Probabilistic Programming

Program defines a probabilistic model

$[\![$
```
x ~ flip(0.5);
y ~ flip(0.7);
```
$]\!] =$

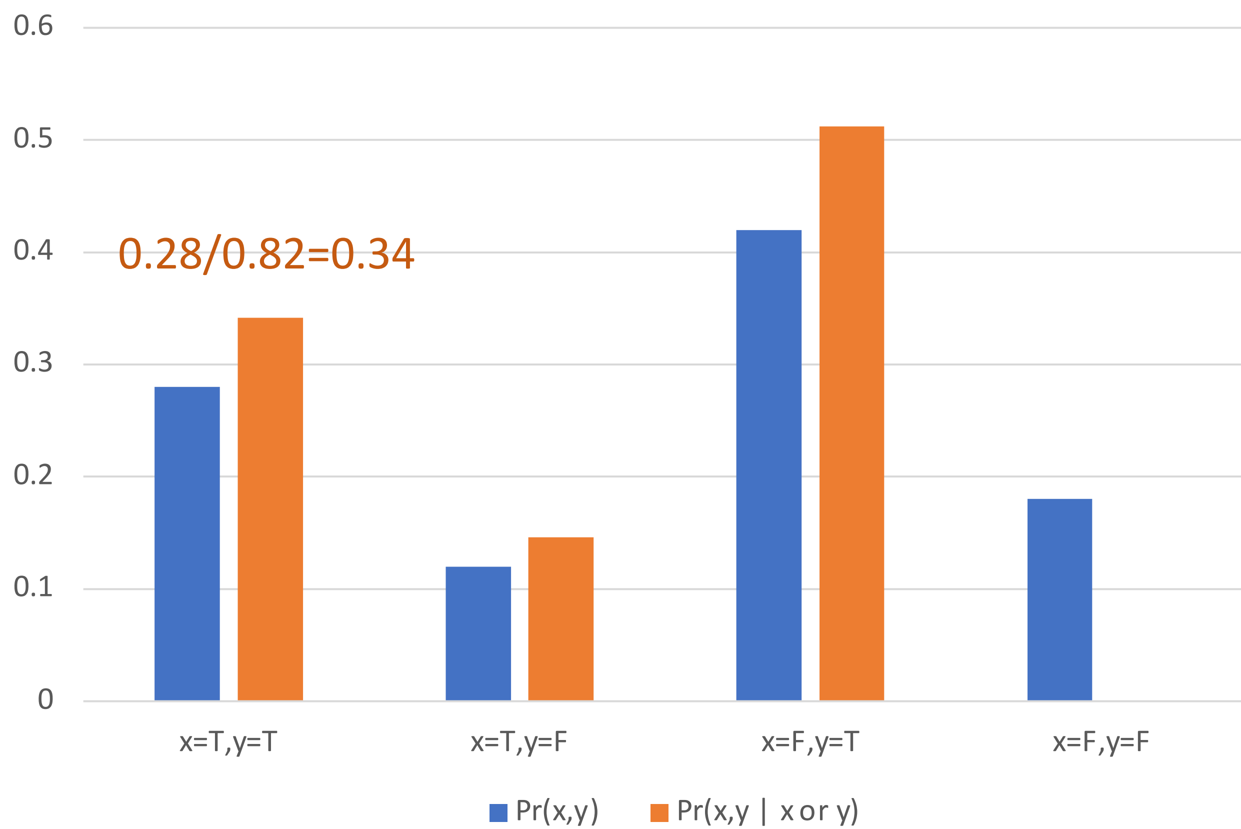


Generating this probability distribution is called *inference*

Bayesian reasoning

Learning with programs

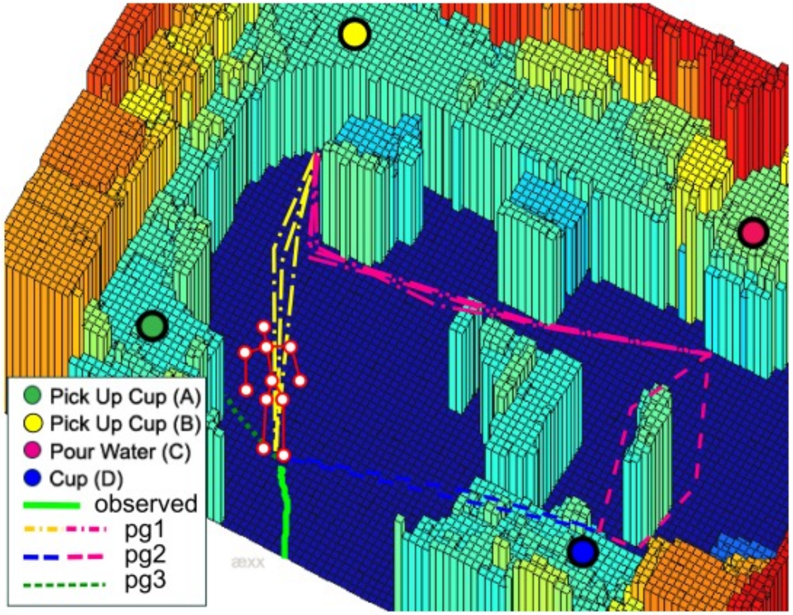
⟦
```
x ~ flip(0.4);
y ~ flip(0.7);
observe x or y;
return (x,y)
```
⟧ =



Application: Plan Inference

- Goal: Given a partially-observed trajectory of a human, infer the most likely underlying plan

- Represent the planning process as a program



Synthesizing learning and programming at PLDI'23

Scallop: A Language for Neurosymbolic Programming

ZIYANG LI*, University of Pennsylvania, USA
JIANI HUANG*, University of Pennsylvania, USA
MAYUR NAIK, University of Pennsylvania, USA

Monday @16:00

Prompting Is Programming: A Query Language for Large Language Models

Monday @17:40

LUCA BEURER-KELLNER, MARC FISCHER, and MARTIN VECHEV, ETH Zurich, Switzerland

Probabilistic Programming with Stochastic Probabilities

ALEXANDER K. LEW, MIT, USA
MATIN GHAVAMIZADEH, MIT, USA
MARTIN C. RINARD, MIT, USA
VIKASH K. MANSINGHKA, MIT, USA

Tuesday @14:40

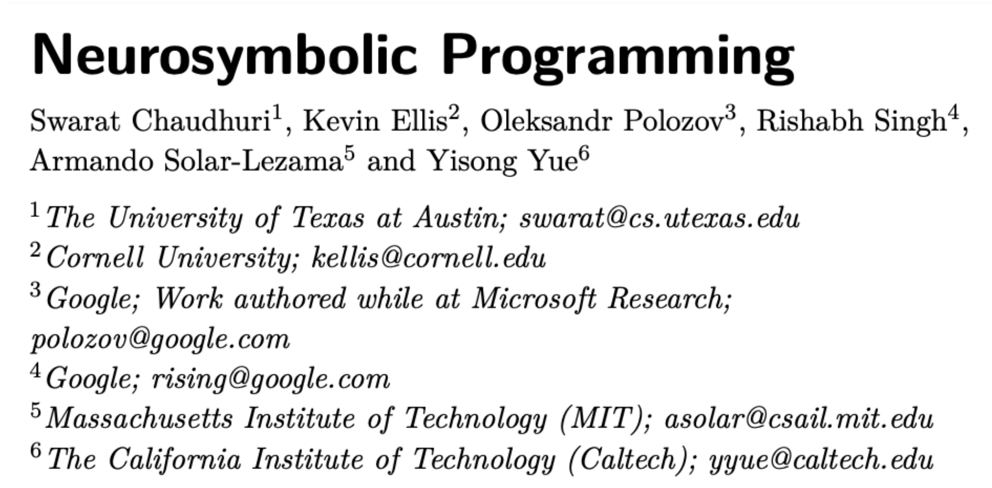
Synthesizing learning and programming at PLDI'23

Passport: Improving Automated Formal Verification Using Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

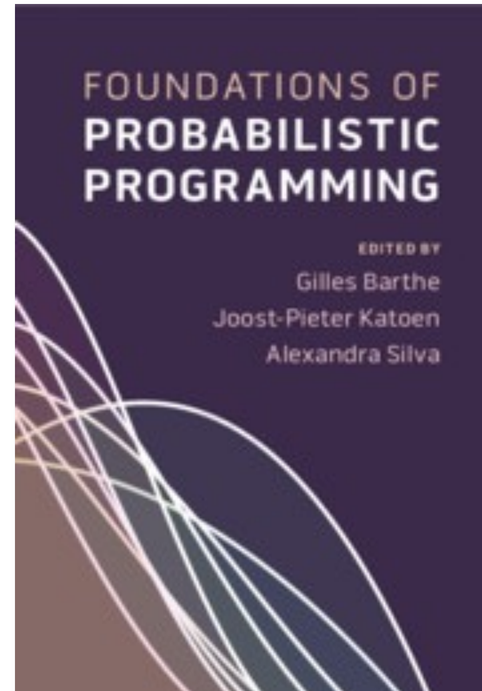
Wed @13:40

For more


Neurosymbolic Programming

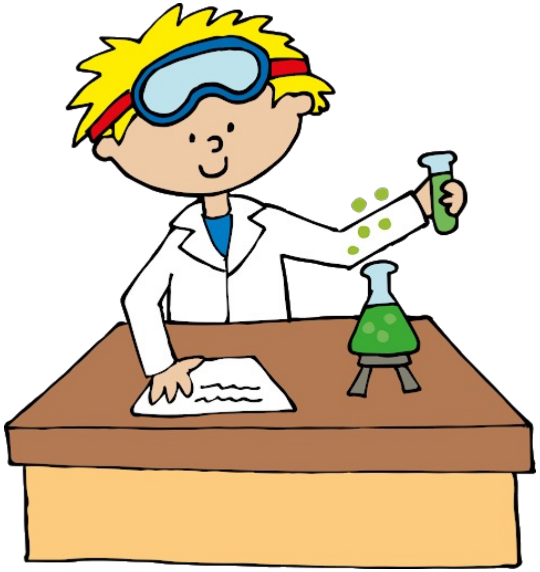
Swarat Chaudhuri[1], Kevin Ellis[2], Oleksandr Polozov[3], Rishabh Singh[4], Armando Solar-Lezama[5] and Yisong Yue[6]

[1] *The University of Texas at Austin; swarat@cs.utexas.edu*
[2] *Cornell University; kellis@cornell.edu*
[3] *Google; Work authored while at Microsoft Research; polozov@google.com*
[4] *Google; rising@google.com*
[5] *Massachusetts Institute of Technology (MIT); asolar@csail.mit.edu*
[6] *The California Institute of Technology (Caltech); yyue@caltech.edu*



Grand challenge #2:
Verified systems with learned & uncertain components

ML is being used to build important systems **now**



ehaving

EXCLUSIVE: SURVEILLANCE FOOTAGE OF TESLA CRASH ON SF'S BAY BRIDGE HOURS AFTER ELON MUSK ANNOUNCES "SELF-DRIVING" FEATURE

Musk has said Tesla's problematic autopilot features are "really the difference between Tesla being worth a lot of money or worth basically zero."

Ken Klippenstein

January 10 2023, 11:22 a.m.

The Intercept
2023

he New York Times

tal Accident

16

The New York Times

2 Killed in Driverless Tesla Car Crash, Officials Say

"No one was driving the vehicle" when the car crashed and burst into flames, killing two men, a constable said.

April 2021

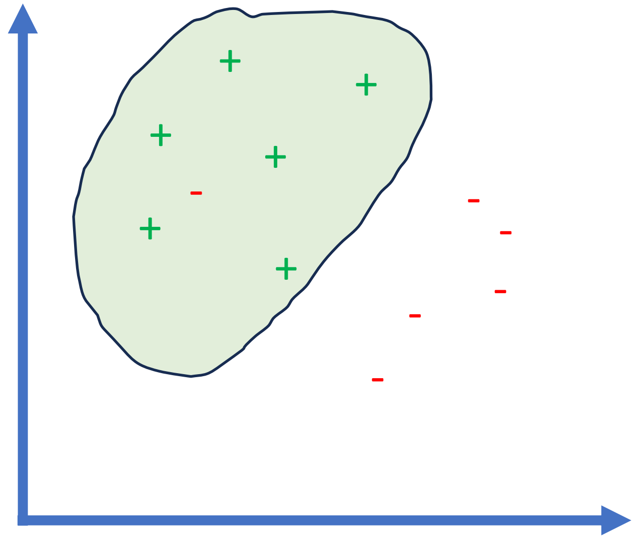
We want to *verify* ML systems

- **Safety property**: System never does something bad
- **Probabilistic safety**: Program does something bad with *low probability*
- Systems with learned components need both kinds of safety

Verification is hard because…

…Notion of correctness is unclear

May be high accuracy but unsafe!



Verification is hard because…

Scalability for analysis

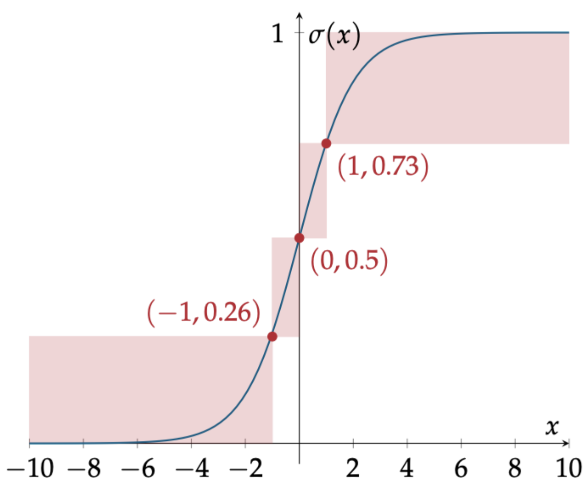


Figure 5.2 Sigmoid function with overapproximation

Introduction to Neural Network Verification

A book by Aws Albarghouthi

Verified systems with learned & uncertain components @ FCRC '23

Abstract Interpretation of Fixpoint Iterators with Applications to Neural Networks

Mon @16:20

MARK NIKLAS MÜLLER, MARC FISCHER, ROBIN STAAB, and MARTIN VECHEV,
ETH Zurich, Switzerland

Architecture-Preserving Provable Repair of Deep Neural Networks

ZHE TAO, University of California, Davis, U.S.A.
STEPHANIE NAWAS, University of California, Davis, U.S.A.
JACQUELINE MITCHELL, University of California, Davis, U.S.A.
ADITYA V. THAKUR, University of California, Davis, U.S.A.

Mon @17:00

Incremental Verification of Neural Networks

Mon @17:20

SHUBHAM UGARE, University of Illinois Urbana-Champaign, USA
DEBANGSHU BANERJEE, University of Illinois Urbana-Champaign, USA
SASA MISAILOVIC, University of Illinois Urbana-Champaign, USA
GAGANDEEP SINGH, University of Illinois Urbana-Champaign and VMware Research, USA

Verified systems with learned components @ FCRC '23

Lilac: A Modal Separation Logic for Conditional Probability

JOHN M. LI, Northeastern University, USA
AMAL AHMED, Northeastern University, USA
STEVEN HOLTZEN, Northeastern University, USA

Tues @13:40

One Pixel Adversarial Attacks via Sketched Programs

TOM YUVILER and DANA DRACHSLER-COHEN, Technion, Israel

Tues @10:00

Scalable Verification of GNN-Based Job Schedulers

HAOZE WU, Stanford University, USA
CLARK BARRETT, Stanford University, USA
MAHMOOD SHARIF, Tel Aviv University, Israel
NINA NARODYTSKA, VMware Research, USA
GAGANDEEP SINGH, University of Illinois at Urbana-Champaign, USA

Wed @14:00

Verified systems with learned components @ FCRC '23

Verified Density Compilation for a Probabilistic Programming Language

JOSEPH TASSAROTTI, NYU, USA
JEAN-BAPTISTE TRISTAN, AWS, USA

Tues @14:00

Formally Verified Samplers from Probabilistic Programs with Loops and Conditioning

ALEXANDER BAGNALL, Ohio University, USA
GORDON STEWART, BedRock Systems, Inc., USA
ANINDYA BANERJEE, IMDEA Software Institute, Spain

Tues @14:20

Grand challenge #3:
Harnessing generation

Themes in code generation

1. Models for generating code are rapidly becoming widely-used in practice

2. Trust but verify

Learning Loop Invariants for Program Verification

Xujie Si*
University of Pennsylvania
xsi@cis.upenn.edu

Hanjun Dai *
Georgia Tech
hanjundai@gatech.edu

Mukund Raghothaman
University of Pennsylvania
rmukund@cis.upenn.edu

Mayur Naik
University of Pennsylvania
mhnaik@cis.upenn.edu

Le Song
Georgia Tech and Ant Financial
lsong@cc.gatech.edu

3. Rise of open-source models

BigCode   starcoder   …

4. Things are moving fast

Harnessing generation @ FCRC '23

Prompting Is Programming: A Query Language for Large Language Models

Monday @17:40

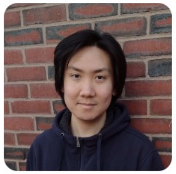
LUCA BEURER-KELLNER, MARC FISCHER, and MARTIN VECHEV, ETH Zurich, Switzerland

Conclusions

- ML Meets PL is a thriving intersection today, dozens of papers at just this PLDI on this topic
  - Many other intersections we did not have time to discuss
- Complementary strengths of ML and PL
- Some grand challenges:
  1. Synthesizing learning and programming
  2. Verified systems with learned and uncertain components
  3. Harnessing generative models
  4. Any more?

This was a team effort

Brianna Marshall
PhD. Student (co-advised with Amal Ahmed)
Personal webpage
Email: `marshall.sa@northeastern.edu`



John Li
PhD. Student (co-advised with Amal Ahmed)
Personal webpage
Email: `li.john@northeastern.edu`

Minsung Cho
PhD. Student
Personal webpage
Email: `minsung@ccs.neu.edu`

Sam Stites
PhD. Student
Personal webpage
Email: `stites.s@northeastern.edu`

Jack Czenszak
Undergraduate Student
Personal webpage
Email: `czenszak.j@northeastern.edu`

Conclusion: ask me anything!

- I hope you learned something:
  - Useful
  - Memorable
  - Enriching