

---

---

# Announcements

Monday Presentations (I will send a reminder email on Friday)

Graduate Students: Project Guidelines on website

## Help selecting a topic

To facilitate project discussion, I will have extra availability:

- Wednesday, Sep. 29, Innovation E456, 1-5pm
- Thursday, Sep. 30, Teams only, 1-5pm
- Friday, Oct. 1, Innovation E456 1-5pm
- Monday, Oct. 4, Innovation E456 1-5pm
- Tuesday, Oct. 5, Teams only, 1-5pm

Please feel free to stop by my my office or drop into Teams! This is **your time**.

2-3 page project proposals

Due Fri, Oct 8 in Blackboard

**CS 295B/CS 395B**  
**Systems for Knowledge**  
**Discovery**

Lecture 4: Formal  
Language Design as  
Method



The University of Vermont



# Topics

Methods overview/refresher

What are formal methods?

What is formal language design?

Formal language design as method

If we have time:

Notation: The Good, the Bad, the Ugly



---

---

# Methods refresher

Recall: methods are *how* you do something

Closely tied to research questions and hypotheses

Specifically: Methods are the community-recognized accepted evaluation techniques.



# Methods refresher: Qualitative methods

Research area: Human Factors

Project: Design for accessibility

Hypothesis: This design is usable

Method: Scenario Based Design

- Tell a “story”, record how a user responds





# Methods refresher: Quantitative methods

Research area: Reinforcement Learning

Project: Novel learning algorithm

Hypothesis: New algorithm is better

Method: Benchmarking

- Naïve sample of environments not representative
- Select representative collection
- Must be updated



# Methods refresher: Formal methods

Research area: Programming Languages

Project: Language-based security

Hypothesis: This approach is less conservative than past approach

Method: Formal proof

- Today: would also do an empirical evaluation





# Topics

~~Methods overview/refresher~~

What are formal methods?

What is formal language design?

Formal language design as method

If we have time:

Notation: The Good, the Bad, the Ugly





---

---

## What are formal methods?

[A] **Formal system** ... [is the] theoretical organization of terms and implicit relationships that is used as a tool for the analysis of the concept of deduction.

Models—structures that interpret the symbols of a formal system—are often used in conjunction with formal systems.

---

---

## What are formal methods?

[A] **Formal system** ... [is the] theoretical organization of terms and implicit relationships that is used as a tool for the analysis of the concept of deduction.

# Apply mathiness directly to forehead

Models—structures that interpret the symbols of a formal system—are often used in conjunction with formal systems.



Freya Holmér @FreyaHolmer · Sep 11

btw these large scary math symbols are just for-loops

<b>Summation</b> (capital sigma)	$\sum_{n=0}^4 3n$	<pre>sum = 0; for( n=0; n&lt;=4; n++ )   sum += 3*n;</pre>
<b>Product</b> (capital pi)	$\prod_{n=1}^4 2n$	<pre>prod = 1; for( n=1; n&lt;=4; n++ )   prod *= 2*n;</pre>

549 7.3K 35.2K



Franklin Lynam @FranklinLynam · Sep 11

I hate the symbols in math so much. It just feels like unnecessary gatekeeping for trivial concepts. How many mathematical proofs could be made widely accessible with just a little bit of psuedocode?

96 95 369



πsquaredbyi @pisquaredbyi

Replying to @FranklinLynam and @FreyaHolmer

The for loops are equally inaccessible. Mathematics just has different conventions, and many people might also be helped by learning for loops via these symbols

1:27 PM · Sep 11, 2021 · Twitter Web App

# Why formalize?

Classically: formal notation is a kind of language; purpose is communication

Can also think of formal notation as a kind of *compression*

**Purpose: to make *something* easier than it would have been without the notation**



## Troubling Trends in Machine Learning Scholarship

Zachary C. Lipton\* & Jacob Steinhardt\*  
 Carnegie Mellon University, Stanford University  
[zlipton@cmu.edu](mailto:zlipton@cmu.edu), [jsteinhardt@cs.stanford.edu](mailto:jsteinhardt@cs.stanford.edu)

July 27, 2018

### 1 Introduction

Collectively, machine learning (ML) researchers are engaged in the creation and dissemination of knowledge about data-driven algorithms. In a given paper, researchers might aspire to any subset of the following goals, among others: to theoretically characterize what is learnable, to obtain understanding through empirically rigorous experiments, or to build a working system that has high predictive accuracy. While determining which knowledge warrants inquiry may be subjective, once the topic is fixed, papers are most valuable to the community when they act in service of the reader, creating foundational knowledge and communicating as clearly as possible.

What sort of papers best serve their readers? We can enumerate desirable characteristics: these papers should (i) provide intuition to aid the reader's understanding, but clearly distinguish it from stronger conclusions supported by evidence; (ii) describe empirical investigations that consider and rule out alternative hypotheses [62]; (iii) make clear the relationship between theoretical analysis and intuitive or empirical claims [64]; and (iv) use language to empower the reader, choosing terminology to avoid misleading or unproven connotations, collisions with other definitions, or conflation with other related but distinct concepts [56].

Recent progress in machine learning comes despite frequent departures from these ideals. In this paper, we focus on the following four patterns that appear to us to be trending in ML scholarship:

1. Failure to distinguish between explanation and speculation.
2. Failure to identify the sources of empirical gains, e.g. emphasizing unnecessary modifications to neural architectures when gains actually stem from hyper-parameter tuning.
3. Mathiness: the use of mathematics that obfuscates or impresses rather than clarifies, e.g. by confusing technical and non-technical concepts.
4. Misuse of language, e.g. by choosing terms of art with colloquial connotations or by overloading established technical terms.

While the causes behind these patterns are uncertain, possibilities include the rapid expansion of the community, the consequent thinness of the reviewer pool, and the often-misaligned incentives between scholarship and short-term measures of success (e.g. bibliometrics, attention, and entrepreneurial opportunity). While each pattern offers a corresponding remedy (don't do it), we also discuss some speculative suggestions for how the community might combat these trends.

As the impact of machine learning widens, and the audience for research papers increasingly includes students, journalists, and policy-makers, these considerations apply to this wider audience

\*Equal Authorship

# How can this possibly be easier?

1. "Fluent speakers" can understand quickly.
2. Disambiguates things that are not disambiguated in other languages.
3. Makes some properties easier to prove

<i>Level</i>	::= $\perp \mid \top$	levels
<i>Exp</i>	::= $v \mid Exp_1 \text{ (op) } Exp_2$ $\text{if } Exp_1 \text{ then } Exp_t \text{ else } Exp_f$ $Exp_1 Exp_2$ $\langle Exp_\perp \mid Exp_\top \rangle(\ell)$ $\text{level } \ell \text{ in } Exp$ $\text{policy } \ell: Exp_p \text{ then } Level \text{ in } Exp$	expressions
<i>Stmt</i>	::= $\text{let } x: \tau = Exp$ $\text{print } \{Exp_c\} Exp$	

Figure 1: Jeeves syntax.

# What speakers see

Example from Yang et al., *A Language for Automatically Enforcing Privacy Policies*, POPL 2012

Top: tells me how write example programs

Non-standard content: levels

Bottom: Tells me what structure to reason over

<i>c</i>	::= $n \mid b \mid \lambda x: \tau. e \mid \text{record } x: \vec{v}$ $\text{error} \mid ()$	concrete primitives
$\sigma$	::= $x \mid \text{context } \tau$ $c_1 \text{ (op) } \sigma_2 \mid \sigma_1 \text{ (op) } c_2$ $\sigma_1 \text{ (op) } \sigma_2$ $\text{if } \sigma \text{ then } v_t \text{ else } v_f$	symbolic values
<i>v</i>	::= $c \mid \sigma$	values
<i>e</i>	::= $v \mid e_1 \text{ (op) } e_2$ $\text{if } e_1 \text{ then } e_t \text{ else } e_f \mid e_1 e_2$ $\text{let } x: \tau = e_1 \text{ in } e_2$ $\text{let rec } f: \tau = e_1 \text{ in } e_2$ $\text{defer } x: \tau \{e\} \text{ default } v_d$ $\text{assert } e$ $\text{concretize } e \text{ with } v_c$	expressions

Figure 2: The  $\lambda_J$  abstract syntax.

<i>Level</i>	::= $\perp \mid \top$	levels
<i>Exp</i>	::= $v \mid Exp_1 \text{ (op) } Exp_2$ $\text{if } Exp_1 \text{ then } Exp_t \text{ else } Exp_f$ $Exp_1 Exp_2$ $\langle Exp_\perp \mid Exp_\top \rangle(\ell)$ $\text{level } \ell \text{ in } Exp$ $\text{policy } \ell: Exp_p \text{ then } Level \text{ in } Exp$	expressions
<i>Stmt</i>	::= $\text{let } x: \tau = Exp$ $\text{print } \{Exp_c\} Exp$	

Figure 1: Jeeves syntax.

# Disambiguation

Example from Yang et al., *A Language for Automatically Enforcing Privacy Policies*, POPL 2012

Let vs. let-rec

- Just let – need to have a function call, dynamic check
- Let-rec – pushes this reasoning to the syntax level

<i>c</i>	::= $n \mid b \mid \lambda x: \tau. e \mid \text{record } x: \vec{v}$ $\text{error} \mid ()$	concrete primitives
$\sigma$	::= $x \mid \text{context } \tau$ $c_1 \text{ (op) } \sigma_2 \mid \sigma_1 \text{ (op) } c_2$ $\sigma_1 \text{ (op) } \sigma_2$ $\text{if } \sigma \text{ then } v_t \text{ else } v_f$	symbolic values
<i>v</i>	::= $c \mid \sigma$	values
<i>e</i>	::= $v \mid e_1 \text{ (op) } e_2$ $\text{if } e_1 \text{ then } e_t \text{ else } e_f \mid e_1 e_2$ $\text{let } x: \tau = e_1 \text{ in } e_2$ $\text{let rec } f: \tau = e_1 \text{ in } e_2$ $\text{defer } x: \tau \{e\} \text{ default } v_d$ $\text{assert } e$ $\text{concretize } e \text{ with } v_c$	expressions

Figure 2: The  $\lambda_J$  abstract syntax.



## 4. Properties

We describe more formally the guarantees that Jeeves provides. We prove progress and preservation properties for  $\lambda_J$ . We show that the only way the value for the high component of a sensitive value to affect the output of the computation is if the policies permit it.

### 4.1 Progress and Preservation

We first show the correctness of evaluation. We can prove progress and preservation properties for  $\lambda_J$ : evaluation of an expression  $e$  always results in a value  $v$  and preserves the type of  $e$ , including the internal nondeterminism tag  $\delta$ .

There are two interesting parts to the proof: showing that all function applications can be reduced and showing that all **defer** and **assert** expressions can be evaluated to produce appropriate constraint expressions. We can first show that the  $\lambda_J$  type system guarantees that all functions are concrete.

**Lemma 1** (Concrete Functions). *If  $v$  is a value of type  $\tau_1 \rightarrow \tau_2$ , then  $v = \lambda x : \tau_1 . e$ , where  $e$  has type  $\tau_2$ .*

**Theorem 4.1** (Progress). *Suppose  $e$  is a closed, well-typed expression. Then  $e$  is either a value  $v$  or there is some  $e'$  such that  $\vdash \langle \emptyset, \emptyset, e \rangle \rightarrow \langle \Sigma', \Delta', e' \rangle$ .*

## Proofs

Example from Yang et al., *A Language for Automatically Enforcing Privacy Policies*, POPL 2012

- Proving things we care about generally (progress and preservation)
- Proving a specific property (soundness of data release)
- Highlighting the non-standard parts

## FairSquare: Probabilistic Verification of Program Fairness

AWS ALBARGHOUTH, University of Wisconsin–Madison, USA

LORIS D'ANTONI, University of Wisconsin–Madison, USA

SAMUEL DREWS, University of Wisconsin–Madison, USA

ADITYA V. NORI, Microsoft Research, UK

With the range and sensitivity of algorithmic decisions expanding at a break-neck speed, it is imperative that we aggressively investigate *fairness and bias* in decision-making programs. First, we show that a number of recently proposed formal definitions of fairness can be encoded as probabilistic program properties. Second, with the goal of enabling rigorous reasoning about fairness, we design a novel technique for verifying probabilistic properties that admits a wide class of decision-making programs. Third, we present FairSquare, the first verification tool for automatically certifying that a program meets a given fairness property. We evaluate FairSquare on a range of decision-making programs. Our evaluation demonstrates FairSquare's ability to verify fairness for a range of different programs, which we show are out-of-reach for state-of-the-art program analysis techniques.

CCS Concepts: • **Mathematics of computing** → **Probabilistic inference problems**; • **Software and its engineering** → **Automated static analysis**;

Additional Key Words and Phrases: Algorithmic Fairness, Probabilistic Programming, Probabilistic Inference

### ACM Reference Format:

Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. 2017. FairSquare: Probabilistic Verification of Program Fairness. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 80 (October 2017), 30 pages. <https://doi.org/10.1145/3133904>

## 1 INTRODUCTION

A number of very interesting applications of program analysis have been explored in the probabilistic setting: reasoning about cyber-physical systems [Sankaranarayanan et al. 2013], proving differential privacy of complex algorithms [Barthe et al. 2014], reasoning about approximate programs and hardware [Carbin et al. 2013], synthesizing control programs [Chaudhuri et al. 2014], amongst many others. In this paper, we turn our attention to the problem of *verifying fairness of decision-making programs*.

**Program Bias** As software permeates our personal lives, corporate world, and bureaucracy, more and more of our critical decisions are being delegated to opaque algorithms. Software has thus become a powerful arbitrator of a range of significant decisions with far-reaching societal impact—hiring [Kobie 2016; Miller 2015], welfare allocation [Eubanks 2015], prison sentencing [Angwin

Authors' addresses: A. Albarghouthi, L. D'Antoni, S. Drews, Department of Computer Sciences, University of Wisconsin–Madison, 1210 West Dayton Street, Madison, WI, 53706, US; A. Nori, Microsoft Research Cambridge, 21 Station Road Cambridge CB1 2FB United Kingdom.

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, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2017 Copyright held by the owner/author(s). Publication rights licensed to Association for Computing Machinery. 2475-1421/2017/10-ART80 <https://doi.org/10.1145/3133904>

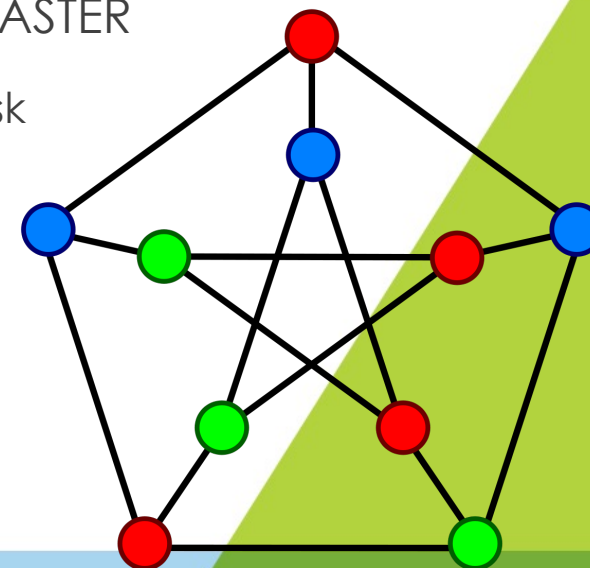
# Another example

(Do stuff on the board)

# A story about the power of models and formalisms

Problem domain: compilers

- A compiler transforms code in some way
- Optimizing compiler: make a new program that is EQUIVALENT and FASTER
  - How to do this? Memory hierarchy and locality: register -> cache -> disk
    - Central question: how to optimally allocate to registers?
    - Model as a graph, equivalent to graph coloring
      - Nodes: program variables; edges: "liveness"





---

---

## A story about the power of models and formalisms

Problem domain: compilers

- A compiler transforms code in one language
- Optimizing compiler: make a new program that is EQUIVALENT and FASTER

- How to do this? Memory hierarchy and locality: register -> cache -> disk

- Can we do this automatically? (e.g. register allocation?)

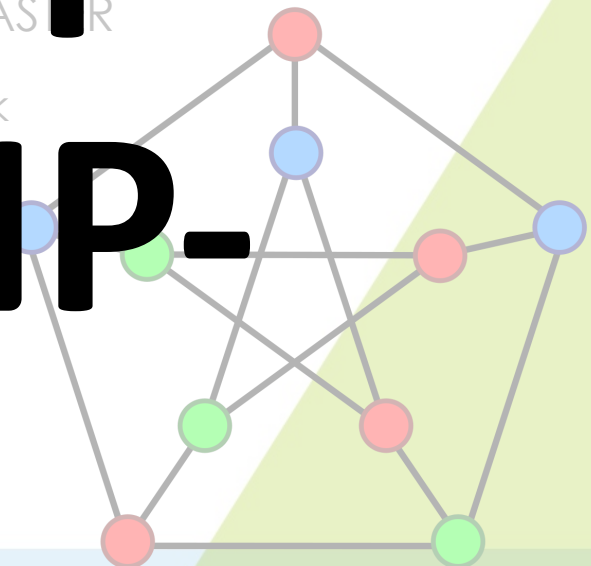
- Model as a graph, equivalent to graph coloring

- Nodes: program variables; edges: "liveness"

# Problem: graph

# coloring is NP-

# complete



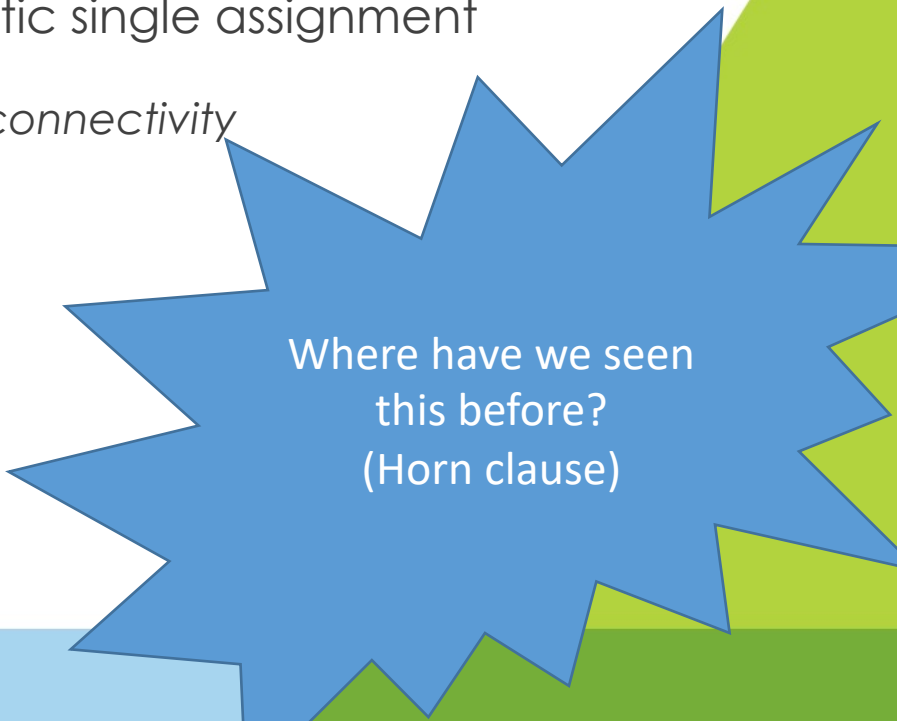
---

---

# A story about the power of models and formalisms

Formal language solution:

- Rewrite the program to an *intermediate representation (IR)*: static single assignment
  - This IR generates *many more variables*, but changes the *graph connectivity*
  - No longer the general graph-coloring problem
    - Now: special subset of graphs that are easier to reason about
    - Problem is now tractable (NO LONGER NP-COMPLETE)



Where have we seen  
this before?  
(Horn clause)

---

---

## Big idea

**Syntactic rewrites can change  
the complexity of a problem.**



# Topics

Methods overview/refresher

What are formal methods?

What is formal language design?

Formal language design as method

If we have time:

Notation: The Good, the Bad, the Ugly





---

---

# What is formal language design?

Each formal system has a **formal language** composed of **primitive symbols** acted on by certain **rules of formation**

*(statements concerning the symbols, functions, and sentences allowable in the system)*

and developed by [inference](#) from a [set](#) of axioms.

The system thus consists of **any number of formulas** built up through **finite combinations** of the primitive symbols—*combinations that are formed from the axioms in accordance with the stated rules.*

<i>Level</i>	::=	$\perp$   $\top$	levels
<i>Exp</i>	::=	$v$   $Exp_1$ (op) $Exp_2$   <b>if</b> $Exp_1$ <b>then</b> $Exp_t$ <b>else</b> $Exp_f$   $Exp_1$ $Exp_2$   $\langle Exp_\perp$   $Exp_\top \rangle(\ell)$   <b>level</b> $\ell$ <b>in</b> $Exp$   <b>policy</b> $\ell: Exp_p$ <b>then</b> $Level$ <b>in</b> $Exp$	expressions
<i>Stmt</i>	::=	<b>let</b> $x: \tau = Exp$   <b>print</b> $\{Exp_c\} Exp$	

Figure 1: Jeeves syntax.

<i>c</i>	::=	$n$   $b$   $\lambda x: \tau. e$   record $x: \tau$ $v$   <b>error</b>   $()$	concrete primitives
$\sigma$	::=	$x$   <b>context</b> $\tau$   $c_1$ (op) $\sigma_2$   $\sigma_1$ (op) $c_2$   $\sigma_1$ (op) $\sigma_2$   <b>if</b> $\sigma$ <b>then</b> $v_t$ <b>else</b> $v_f$	symbolic values
$v$	::=	$c$   $\sigma$	values
<i>e</i>	::=	$v$   $e_1$ (op) $e_2$   <b>if</b> $e_1$ <b>then</b> $e_t$ <b>else</b> $e_f$   $e_1$ $e_2$   <b>let</b> $x: \tau = e_1$ <b>in</b> $e_2$   <b>let rec</b> $f: \tau = e_1$ <b>in</b> $e_2$   <b>defer</b> $x: \tau \{e\}$ <b>default</b> $v_d$   <b>assert</b> $e$   <b>concretize</b> $e$ <b>with</b> $v_c$	expressions

Figure 2: The  $\lambda_J$  abstract syntax.

# Primitive Symbols

## Rules of formation

<i>Level</i>	$::= \perp \mid \top$	levels
<i>Exp</i>	$::= v \mid Exp_1 \text{ (op) } Exp_2$ $\mid \text{if } Exp_1 \text{ then } Exp_t \text{ else } Exp_f$ $Exp_1 Exp_2$ $\langle Exp_\perp \mid Exp_\top \rangle(\ell)$ $\text{level } \ell \text{ in } Exp$ $\text{policy } \ell: Exp_p \text{ then } Level \text{ in } Exp$	expressions
<i>Stmt</i>	$::= \text{let } x: \tau = Exp$ $\mid \text{print } \{Exp_c\} Exp$	

Figure 1: Jeeves syntax.

# Rule-based finite combinations

```
let x = fn1("asdf")
let y = fn2("fdsa")
```

```
let z =
  if x
  then
    level top in
      policy bot then top in y
  else
    y
print z
```

<i>c</i>	$::= n \mid b \mid \lambda x: \tau. e \mid \text{record } x: \vec{v}$ $\mid \text{error} \mid ()$	concrete primitives
$\sigma$	$::= x \mid \text{context } \tau$ $c_1 \text{ (op) } \sigma_2 \mid \sigma_1 \text{ (op) } c_2$ $\sigma_1 \text{ (op) } \sigma_2$ $\text{if } \sigma \text{ then } v_t \text{ else } v_f$	symbolic values
<i>v</i>	$::= c \mid \sigma$	values
<i>e</i>	$::= v \mid e_1 \text{ (op) } e_2$ $\text{if } e_1 \text{ then } e_t \text{ else } e_f \mid e_1 e_2$ $\text{let } x: \tau = e_1 \text{ in } e_2$ $\text{let rec } f: \tau = e_1 \text{ in } e_2$ $\text{defer } x: \tau \{e\} \text{ default } v_d$ $\text{assert } e$ $\text{concretize } e \text{ with } v_c$	expressions

Figure 2: The  $\lambda_J$  abstract syntax.

<i>Level</i>	::= $\perp \mid \top$	levels
<i>Exp</i>	::= $v \mid Exp_1 \text{ (op) } Exp_2$ $\text{if } Exp_1 \text{ then } Exp_t \text{ else } Exp_f$ $Exp_1 Exp_2$ $\langle Exp_\perp \mid Exp_\top \rangle(\ell)$ $\text{level } \ell \text{ in } Exp$ $\text{policy } \ell: Exp_p \text{ then } Level \text{ in } Exp$	expressions
<i>Stmt</i>	::= $\text{let } x: \tau = Exp$ $\text{print } \{Exp_c\} Exp$	

Figure 1: Jeeves syntax.

# Rule-based finite combinations

```
let x = fn1("asdf")
Let y = fn2("fdsa")
```

```
let z =
  if x
  then
    level y in
      policy bot then top in y
  else
    y
print z
```

<i>c</i>	::= $n \mid b \mid \lambda x: \tau. e \mid \text{record } x: \vec{v}$ $\text{error} \mid ()$	concrete primitives
$\sigma$	::= $x \mid \text{context } \tau$ $c_1 \text{ (op) } \sigma_2 \mid \sigma_1 \text{ (op) } c_2$ $\sigma_1 \text{ (op) } \sigma_2$ $\text{if } \sigma \text{ then } v_t \text{ else } v_f$	symbolic values
<i>v</i>	::= $c \mid \sigma$	values
<i>e</i>	::= $v \mid e_1 \text{ (op) } e_2$ $\text{if } e_1 \text{ then } e_t \text{ else } e_f \mid e_1 e_2$ $\text{let } x: \tau = e_1 \text{ in } e_2$ $\text{let rec } f: \tau = e_1 \text{ in } e_2$ $\text{defer } x: \tau \{e\} \text{ default } v_d$ $\text{assert } e$ $\text{concretize } e \text{ with } v_c$	expressions

Figure 2: The  $\lambda_J$  abstract syntax.



$$F = G \frac{m_1 m_2}{d^2}$$

$$i\hbar \frac{\partial}{\partial t} \psi = \hat{H} \psi$$

$$\phi(x) = \frac{1}{\sqrt{2\pi}}$$

$$E = mc^2$$

$$u = c^2 \frac{\partial^2 u}{\partial x^2}$$

$$\frac{df}{dt}$$

## What kinds of things do we prove?

All kinds of things

- Program equivalence (may not be decidable)
- Type inference is sound (progress and preservation)
- Domain-specific properties ("correct by construction")



# Topics

~~Methods overview/refresher~~

~~What are formal methods?~~

~~What is formal language design?~~

Formal language design as method

If we have time:

Notation: The Good, the Bad, the Ugly



---

---

# Formal language design as method

“Correct by construction”

**Goal:** *Design a language (primitives and rules for connecting them) such that  
by virtue of being expressed in that language,  
a program must have certain properties.*

---

---

# Formal language design as method

“Correct by construction”

**As method:** A community-approved way of evaluating hypotheses and research questions



---

---

# Formal language design as method

“Correct by construction”

**Research questions:** Often appear to be *can we* (b/c we focus on the design), but are often actually mechanistic. Why?

---

---

## Method of the method (recipe)

1. Surface syntax (easy to write, easy to extract, etc.)
2. Typically translate into a core language.
  1. May be smaller/simpler (reduces redundancies required by "users")
  2. May have more rules (disambiguation)
3. Core language has a semantics, rules for what syntax means.
  1. Semantics may add a bunch of new notation, depending on the task.
4. Prove things over the semantics (inductively).

# Formal language semantics example

Important things to remember:

1. These rules are mechanistic and tightly coupled with the syntax
2. Every syntactic construct (of the core lang.) gets used at least once.
3. You make this stuff up.

$$\begin{array}{c}
 \boxed{\tau_1 <: \tau_2} \\
 \frac{}{\tau <: \tau} \text{ S-REFLEXIVE} \quad \frac{}{\text{int}_c <: \text{int}} \text{ S-INT} \quad \frac{}{\text{bool}_c <: \text{bool}} \text{ S-BOOL} \quad \frac{}{\tau_1 \xrightarrow{nr} \tau_2 <: \tau_1 \rightarrow \tau_2} \text{ S-RECFUN} \quad \frac{\tau'_1 <: \tau_1 \quad \tau_2 <: \tau'_2}{\tau_1 \rightarrow \tau_2 <: \tau'_1 \rightarrow \tau'_2} \text{ S-FUN} \\
 \\
 \boxed{\text{rep } \tau} \\
 \frac{\text{rep } \tau' \quad \tau' <: \tau}{\text{rep } \tau} \text{ OK-SUBTYPE} \quad \frac{}{\text{rep } \beta} \text{ OK-BASETYPE} \quad \frac{\text{rep } \tau_2}{\text{rep } \beta_1 \rightarrow \tau_2} \text{ OK-BASEFUNCTION} \\
 \frac{\text{rep } (\tau_1 \xrightarrow{nr} \tau') \quad \text{rep } \tau_2}{\text{rep } (\tau_1 \xrightarrow{nr} \tau') \xrightarrow{nr} \tau_2} \text{ OK-HOFUNCTION} \quad \frac{\text{rep } \tau_1 \rightarrow \tau_2}{\text{rep } (\tau_1 \rightarrow \tau_2) \rightarrow \beta} \text{ OK-RECFUNCTIONBASE} \quad \frac{\text{rep } \tau_1 \rightarrow \tau_2 \quad \text{rep } \tau'_1 \rightarrow \tau'_2}{\text{rep } (\tau_1 \rightarrow \tau_2) \rightarrow (\tau'_1 \rightarrow \tau'_2)} \text{ OK-RECFUNCTION} \\
 \\
 \boxed{\Gamma; \gamma \vdash e : (\tau, \delta)} \\
 \frac{x \in \Gamma}{\Gamma; \gamma \vdash x : \Gamma(x)} \text{ T-VAR} \quad \frac{}{\Gamma; \gamma \vdash n : \text{int}_c} \text{ T-INT} \quad \frac{}{\Gamma; \gamma \vdash b : \text{bool}_c} \text{ T-BOOL} \quad \frac{}{\Gamma; \gamma \vdash () : \text{unit}} \text{ T-UNIT} \quad \frac{\text{rep } \tau}{\Gamma; \gamma \vdash \text{context } \tau : \tau} \text{ T-CONTEXT} \\
 \frac{\Gamma; \gamma \vdash e_1 : \tau_1 \quad \Gamma; \gamma \vdash e_2 : \tau_2 \quad \tau_1, \tau_2 <: \tau \quad \text{rep } \tau}{\Gamma; \gamma \vdash e_1 \text{ (op) } e_2 : \tau} \text{ T-OP} \quad \frac{\Gamma; \gamma \vdash e : \text{bool}_c \quad \Gamma; \gamma \vdash e_l : \tau_l \quad \Gamma; \gamma \vdash e_r : \tau_r \quad \tau_l, \tau_r <: \tau \quad \text{rep } \tau}{\Gamma; \gamma \vdash \text{if } e \text{ then } e_l \text{ else } e_r : \tau} \text{ T-CONDC} \\
 \frac{\Gamma; \gamma \vdash e : \text{bool} \quad \Gamma; \text{sym} \vdash e_l : \beta_l \quad \Gamma; \text{sym} \vdash e_r : \beta_r \quad \beta_l, \beta_r <: \beta_c \quad \text{rep } \beta_c}{\Gamma; \gamma \vdash \text{if } e \text{ then } e_l \text{ else } e_r : \beta_c} \text{ T-CONDSYM} \\
 \frac{\Gamma, x : \tau_d; \gamma \vdash e : \tau' \quad \text{rep } \tau \quad \text{rep } \tau'}{\Gamma; \gamma \vdash (\lambda x : \tau_d. e) : \tau_d \rightarrow \tau'} \text{ T-LAMBDA} \quad \frac{\Gamma; \gamma \vdash e_1 : \tau_1 \xrightarrow{nr} \tau_2 \quad \Gamma; \gamma \vdash e_2 : \tau'_1 \quad \tau'_1 <: \tau_1 \quad \text{rep } \tau_1 \quad \text{rep } \tau_2}{\Gamma; \gamma \vdash (e_1 e_2) : \tau_2} \text{ T-APP} \\
 \frac{\Gamma, f : \tau_1 \rightarrow \tau_2; \gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma, f : \tau_1 \xrightarrow{nr} \tau_2; \gamma \vdash e_2 : \tau_2 \quad \text{rep } \tau \quad \text{rep } \tau_2}{\Gamma; \gamma \vdash \text{let rec } f : \tau_1 \xrightarrow{nr} \tau_2 = e_1 \text{ in } e_2 : \tau_2} \text{ T-LETREC} \\
 \frac{\gamma = \text{concrete} \quad \Gamma; \gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma; \gamma \vdash e_2 : \tau'_1 \quad \tau'_1 <: \tau_1 \quad \text{rep } \tau_1 \quad \text{rep } \tau_2}{\Gamma; \gamma \vdash (e_1 e_2) : \tau_2} \text{ T-APPCURREC} \\
 \\
 \frac{\Gamma, x : \beta; \gamma \vdash e_c : \text{bool} \quad \Gamma; \gamma \vdash v : \beta}{\Gamma; \gamma \vdash (\text{defer } x : \beta \{ e_c \} \text{ default } v) : \beta} \text{ T-DEFER} \quad \frac{\Gamma; \gamma \vdash e_c : \text{bool}}{\Gamma; \gamma \vdash (\text{assert } e_c) : \text{unit}} \text{ T-ASSERT} \\
 \frac{\Gamma; \gamma \vdash e_1 : \beta \quad \Gamma; \gamma \vdash e_1 : \beta' \quad \Gamma; \gamma \vdash v : \beta'}{\Gamma; \gamma \vdash (\text{concretize } e_1 \text{ with } v) : \beta_c} \text{ T-CONCRETIZE}
 \end{array}$$

Figure 5: Static semantics for  $\lambda_J$  describing simple type-checking and enforcing restrictions on scope of nondeterminism and recursion. Recall that  $\beta$  refers to base (non-function) types.



# Topics

Methods overview/refresher

What are formal methods?

What is formal language design?

Formal language design as method

If we have time:

Notation: The Good, the Bad, the Ugly





---

---

# Notations you might see

Three styles of semantics:

## 1. Denotation

1. Mostly in linguistics, logic, mathematics, old-school PL, defined over abstract objects.
2. "What this program means"

## 2. Axiomatic

1. Most in distributed and large-scale systems, defined over axioms (things that must be true).
2. "What collection of things are true in this program."

## 3. Operational

1. Mostly in modern PL, novel languages, small systems, defined over a machine that takes steps.
2. "What this program does."