An Intro to Curry-Howard and Verification: Relating Logic and Machine Learning

Matthijs Vákár

New York, 5 April 2017

UNIVERSITY OF OXFORD

This talk

- Logic/verification and statistics/ML are complementary!
- Intro to my PhD research on relationship between logic and programming languages
- Specifically, about programming logics
- Helps foundational understanding of CS
- Framework for software verification
- Great opportunity to apply ML!
- This talk: an informal introduction to the area of research

Complementary Forms of Reasoning

- Reasoning is crucial for intelligence
- Reasoning: inferring new knowledge from existing knowledge
- Complex systems (lots of axioms/data) $\rightsquigarrow$ irrationality
- My dream: build computer systems that help us be rational

Deductive reasoning	**Inductive reasoning**
logic $\rightsquigarrow$ proof-assistants/verification	statistics $\rightsquigarrow$ machine learning
Guarantees conclusions	Derives probable (but uncertain) conclusions
General to specific	(Often) specific to general (through ind.biases)
Relies on axioms	Relies on data (cf. axioms) + ind. biases/priors
Fragile w.r.t. inconsistent axioms	Robust w.r.t. inconsistencies in data
Foundation of mathematics	Foundation of science

An Application: Catching Software Bugs Through Logic

- Software bugs cost $312 billion annually
- Developers spend half their time debugging
- Critical applications: medical equipment (Therac-25), auto pilots (Ariane 5), nuclear weapons technology (1980 US & 1983 USSR false alarms), cryptography (Heartbleed), financial markets (Knight Capital: $440 mln loss in 30min), foundational code (Pentium FDIV)
- Formal verification: using logic to catch bugs

"His debugging skills are exceptional."

Formal Verification?

Check that program meets specification, through

- (Semantic) model checking: machine-check that mathematical model of system satisfies property through exhaustive exploration
- **(Syntactic) deductive verification**: formulate property as proposition in some logic and find and machine-check proof

In particular, can hope to perform deductive verification *while type checking*: use types to express program properties

Formal Logic

- Formalism for giving proofs b of propositions B, from assumptions $A_1, \ldots, A_n$
- Have propositions like $\bot, \top, A \vee B, A \wedge B, A \Rightarrow B$ and $\neg A := A \Rightarrow \bot$
- Have canonical proofs with these propositions as conclusions and assumptions
- $+$ can trivially prove any assumption and can compose proofs
- (Can add proof dne_A of $\neg\neg A \Rightarrow A$ to go from intuitionistic to classical logic)

(Typed) Pure Functional Programming

- Important programming paradigm
- Language in which programs b are functions: back boxes which take inputs and produce output
- No effects! I.e. no mutable state, recursion, random nums etc.
- Inputs and output of program have "types" $A_1, \ldots, A_n$ and B
- Means to express properties of programs, organising them
- (Think of as sets)
- Types are practically useful for writing correct code
- Have canonical functions into and out of certain types: e.g. $0, 1 : \texttt{bool}$ and $\mathrm{case} : \texttt{bool} \Rightarrow A \Rightarrow A \Rightarrow A$
- $+$ functions can output any input and we can compose them

The Curry-Howard Correspondence

(Intuitionistic) logic = (typed purely functional) programming:

intuitionistic logic	**purely functional programming**
proposition A	type A
proof a of A	program a with output of type A
assumptions $A_1, \ldots, A_n$	inputs of types $A_1, \ldots, A_n$
proof normalisation $a \rightsquigarrow a'$	program execution $a \rightsquigarrow a'$
true proposition $\top$	type `void`
conjunction $A \wedge B$	product type $A \times B$
false proposition $\bot$	type `error`
disjunction $A \vee B$	sum type $A + B$
implication $A \Rightarrow B$	function type $A \Rightarrow B$.

Basic datatypes: $\texttt{bool} := \texttt{void} + \texttt{void} = \top \vee \top$

Some Famous Proofs As Programs

- (Contraposition) Proposition/type: $(A \Rightarrow B) \Rightarrow (\neg B \Rightarrow \neg A)$

  Proof/program: $\lambda_f \lambda_g \lambda_x g(f(x))$

- (De Morgan) Proposition/type: $(\neg A \vee \neg B) \Rightarrow \neg(A \wedge B)$

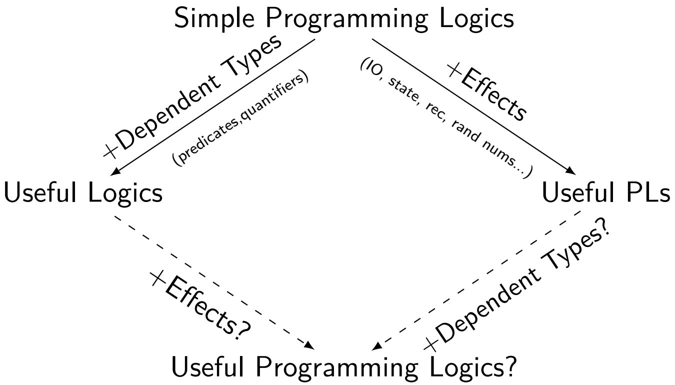
  Proof/program: $\lambda_x \text{case } x \text{ of } \langle i, x_i \rangle \mapsto \begin{cases} \lambda_y x_0(\mathrm{fst}(y)) \\ \lambda_y x_1(\mathrm{snd}(y)) \end{cases}$

- (Negation) Proposition/type: $\texttt{bool} \Rightarrow \texttt{bool}$ $(\texttt{bool} := \top \vee \top)$

  Proof/program: $\lambda_x \text{case } x \text{ of } \langle i, x_i \rangle \mapsto \begin{cases} \langle 1, x_0 \rangle \\ \langle 0, x_1 \rangle \end{cases}$

- Can execute any intuitionistic proof as program! (E.g. Picard-Lindelöf theorem for ODE's, if have quantifiers)

The Limits of Curry-Howard: Programming Logics?



Combination remains a puzzle: languages for verified practical programming $\rightsquigarrow$ goal of my PhD.

Thesis

Thesis solves 3 related open problems:

1. Shows how dependent types and effects can be elegantly and robustly combined
2. Gives game theoretic interpretation of dependent type theory
    - Types are 2-Player games; terms are strategies
    - Think Socratic dialogues: debates about a proposition
    - Put conditions (e.g. winning) on strategies to exclude effects
    - Unique uniform semantic framework for PL design space
3. Develops the theory of linear logic with dependent types

Combining Dependent Types and Effects

- Problem: effectful programs a that return a value of type A are inconsistent as proofs
- Idea: interpret as proofs ($\texttt{pf}\ a$) of modal formula $\Diamond A$
- Problem: effectful programs can be dynamic (e.g. could make n.d. choice); therefore, cannot simply apply dependent types (predicates) to them
- Idea: distinguish static proofs/values and dynamic programs
- Types can only depend on static objects
- Can express properties of (static) frozen programs ($\texttt{pf}\ a$) using dependent types
- Leads to well-behaved theory, developed in thesis

How is this useful?

- Better foundational understanding of disciplines of programming languages and formal logic
  - Made precise why there is tension between dependent types and effects
  - Showed how to resolve that tension
  - Developed rich and elegant mathematical theory (syntax, operational semantics, denotational semantics) for studying the combination: real programming logics
- Hopefully: verification of effectful programs
  - Our framework could be implemented as PL
  - Microsoft's F* comes close
  - Allows to write and check proofs about real world code

Augmenting Logic with ML

- Labour intensive to write proofs; but mostly routine
- Idea: use ML for proof search (cf. game semantics / AlphaGo)
- Note that equivalent problem: program synthesis (of purely functional programs)
- Logic for guarantees, ML for heuristics
- Supervised learning: promising results by Alemi et al. using deep learning
- Reinforcement learning?: proof-checking gives score

Conclusions

- Logic and ML are complementary by design
- Case study of my PhD gives concrete example of this
- Need smart ML people tackling hugely important problems in verification
- Logic and ML have similar motivations; different, complementary techniques
- As long as don't repeat past mistake of confusing induction and deduction...
- ...more collaboration would benefit all!