Little Engines of Proof: Lecture 1

N. Shankar, L. de Moura, H. Ruess, A. Tiwari
shankar@csl.sri.com
URL: http://www.csl.sri.com/~shankar/LEP.html

Computer Science Laboratory
SRI International
Menlo Park, CA

Why Automate Proof?

Computers can calculate numerical results as well as symbolic ones.

From very early on, philosophers have dreamt of machines that can reason. **Leibniz** spoke thus of the consequence of a reasoning machine:

> What must be achieved is in fact this: that every paralogism be recognized as an *error of calculation*, and every *sophism* when expressed in this new kind of notation, appear as a *solecism* or *barbarism*, to be corrected easily by the laws of this philosophical grammar.
>
> Once this is done, then when a controversy arises, disputation will no more be needed between two philosophers than between two computers. It will suffice that, pen in hand, they sit down to their abacus and (calling in a friend, if they so wish) say to each other: *let us calculate*.

Why Automate Proof?

In more recent times, **Vannevar Bush** (`http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm`) in a penetratingly prescient 1945 article entitled *As We May Think* wrote:

> Logic can become enormously difficult, and it would undoubtedly be well to produce more assurance in its use. . . . We may some day click off arguments on a machine with the same assurance that we now enter sales on a cash register.

Or hear **Alfred, Lord Tennyson** declaim:

> O purblind race of miserable men,
> How many among us at this very hour
> Do forge a lifelong trouble for ourselves,
> By taking true for false, and false for true!

Calculating Proofs

A big surprise of the modern world is *everybody needs proof.*

In some areas of mathematics, proofs might require more creativity than labor to when constructed by hand, but most applications of proof construction require automation.

Within computing, application areas include program analysis, logic and constraint programming, hardware verification, databases, and the semantic web.

In science, deduction makes it possible to calculate the consequences of mathematical models through deduction rather than simulation.

Historical Background

Logic itself dates back to 4th century BC with **Aristotle's** attempt to analyze sound reasoning.

Leibniz in the 17th century had great ambitions for a symbolic logic but the subject only blossomed out of the 19th century work of **Boole**, **de Morgan**, **Dedekind**, **Frege**, **Peano**, and **Pierce**.

The idea of thinking engine is a long-held romantic fancy, but serious efforts at *automated deduction* had their origin in the 1950s with the work of **Davis**, **Newell/Shaw/Simon** **Gilmore**, **Wang**, **Prawitz**, and **Kanger**.

The more modest endeavor of *automated proof checking* had its origin in the work of **McCarthy**, **de Bruijn**, and **Bledsoe** in the 1960s, and **Milner** in the 1970s.

Early History of Automated Reasoning

1954: Martin Davis programs a Presburger Arithmetic decision procedure.

> *Its great triumph was to prove that the sum of two even numbers is even.* Martin Davis

1957: **Newell**, **Shaw**, and **Simon's** logic theorist (LT): Introduced subgoaling, substitution, replacement, and forward and backward chaining, with *human*-oriented heuristics. Applied to theorems from **Russell** & **Whitehead's** *Principia Mathematica*.

Many early papers are collected in *Automated Reasoning: Vols. 1 & 2*, edited by Siekmann and Wrightson.
The Handbook of Automated Reasoning, edited by Robinson and Voronkov, is a good modern summary.

Wang versus Newell–Shaw–Simon

1958-60: **Hao Wang** showed that many LT proofs (and others from Russell/Whitehead) were in *easily* decidable fragments: propositional logic, Bernays–Schönfinkel. Hundreds of these theorems could be proved in minutes.

> *The most interesting lesson from these results is perhaps that even in a fairly rich domain, the theorems actually proved are mostly ones which call on a very small portion of the available resources of the domain.* —Hao Wang

Wang versus Newell–Shaw–Simon

The controversy referred to may be succinctly characterized as being between the two slogans: "Simulate people" and "Use mathematical logic". . . . Thus as early as 1961 Minsky remarked

> . . . it seems clear that a program to solve real mathematical problems will have to combine the mathematical sophistication of Wang with the heuristic sophistication of Newell, Shaw, and Simon.

—Martin Davis

Hao Wang's Programme: Inferential Analysis

In contrast with pure logic, the chief emphasis of inferential analysis is on the efficiency of algorithms, which is usually obtained by paying a great deal of attention to the detailed structure of problems and their solutions, to take advantage of possible systematic short cuts.

⋮

That proof procedures for elementary logic can be mechanized is familiar. In practice, however, were we slavishly to follow these procedures without further refinements, we should encounter a prohibitively expansive element. . . . In this way we are led to a closer study of reduction procedures and of decision procedures for special domains, as well as of proof procedures of more complex sorts. —Hao Wang

Big Engines versus Little Engines

Much of the focus in automated deduction has been on finding uniform procedures for large classes of theorems.

For example, the resolution method is a simple, sound and complete inference procedure for first-order logic.

Resolution-based methods have had significant successes solving open problems in diverse branches of mathematics.

However, big engines are not always predictable enough for serious applications — they do a poor job of exploiting domain knowledge.

The little engines ideology is based on composing small theory-specific engines.

Some Little Engines

- Propositional satisfiability solvers
- Binary Decision Diagrams
- Congruence Closure for Equality Propagation
- Real and Integer linear arithmetic solvers
- Decision procedures for lists, arrays, bit-vectors.
- Presburger arithmetic
- Monadic Second Order Logic

Applications of Automated Deduction

The aggregate of all applications of logic will not compare with the treasure of the pure theory itself. For when one has surveyed the whole subject, one will see that the theory of logic insofar as we attain to it, is the vision and the attainment of that Reasonableness for the sake of which the Heavens and the Earth have been created. C. S. Peirce

Already, in the 1970s, advances in deduction led to the introduction of the logic programming paradigm.

In the last fifteen years, as proof engines have become more powerful, there have been a growing number of applications of automated deduction in hardware and software verification, and in solving planning and search problems.

We want to focus on *embedded* applications of deduction.

Course Outline

- Programming prerequisites
- Background in Logic and Recursion Theory
- Propositional Logic
- Ground Decision Procedures for Equality and Arithmetic Inequality
- Combination Decision Procedures for the union of theories
- Quantifier Elimination
- Rewriting
- Proof search in quantificational logic
- Applications

What is Cruel and Unusual?

- Focus on little engines that are theory-specific, and their various combinations.
- Develop theoretical foundations emphasizing simplicity, modularity, and ease of implementation.
- Survey innovative applications of embedded deduction.
- Most of the material is based on recent research, and many open research problems/areas remain.

Sample Conjectures

1. $\neg p \wedge (q \vee p) \Rightarrow q$
2. $f(x) = f(f(f(x))) \Rightarrow f(f(f(f(f(x))))) = f(x)$
3. $x + y = 19 \wedge x - y = 7 \Rightarrow (x = 13 \wedge y = 6)$
4. $x + 7 = 2 * y + 3 * z \Rightarrow f(2 * x - 6 * z) = f(4 * y - 14)$
5. $2 * x \leq 3 * y + 1 \wedge 2 * z \leq 3 * x - 1 \Rightarrow 4 * z \leq 9 * y + 5$
6. $car(x) = cdr(x) - 4 \wedge x = cons(y, 7) \Rightarrow y = 3$
7. $(\exists x, y : x \neq y) \Rightarrow (\forall x : \exists y : x \neq y)$
8. $\forall i, j : (\exists i' : 2 * i' = i) \wedge (\exists j' : 2 * j' = j) \Rightarrow (\exists k : 2 * k - i - j = 0)$ for integers i, j, k, i', j'.

What is Logic?

- Logic is the art and science of effective reasoning.
- How can we draw general and reliable conclusions from a collection of facts?
- Formal logic: Precise, syntactic characterizations of well-formed expressions and valid deductions.
- Formal logic makes it possible to *calculate* consequences at the symbolic level.
- Computers can be used to automate such symbolic calculations.

What is Logic?

Logic studies the relationship between language, meaning, and (proof) method.

A logic consists of a language in which (well-formed) sentences are expressed.

A semantics that distinguishes the valid sentences from the refutable ones.

A proof system for constructing arguments justifying valid sentences.

Examples of logics include propositional logic, equational logic, first-order logic, higher-order logic, and modal logics.

What is a Logical Language?

A language consists of logical symbols whose interpretations are fixed, and non-logical ones whose interpretations vary.

These symbols are combined together to form well-formed formulas.

Thus, in propositional logic PL, the connectives $\wedge$, $\neg$, and $\vee$ have a fixed interpretation, whereas the constants p, q, r may be interpreted at will.

Propositional Logic

Formulas: $\phi := P \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \Rightarrow \phi_2$.

P is a class of propositional variables: $p_0, p_1, \ldots$.

Examples: p_0, $\neg p_0$, $p_0 \vee \neg p_1$, $p_0 \vee p_1 \Rightarrow (\neg p_3 \wedge p_4)$.

Exercise 1 *Using a programming language, define a representation for propositional formulas and a checker for well-formed propositional formulas.*

Interpretation

An interpretation $\mathcal{M}$ assigns truth values $\{\top, \bot\}$ to propositional variables.

Let A and B range over PL formulas.

$\mathcal{M}[\![\phi]\!]$ is the meaning of ϕ in $\mathcal{M}$ and is computed using *truth tables*:

ϕ	A	B	$\neg A$	$A \vee B$	$A \wedge \neg A$	$A \Rightarrow B$	$A \Rightarrow (B \vee A)$
$\mathcal{M}_1(\phi)$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\mathcal{M}_2(\phi)$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\mathcal{M}_3(\phi)$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$
$\mathcal{M}_4(\phi)$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\top$

Satisfiability and Validity

A formula A is *satisfiable* if it has a *model*, i.e., an interpretation $\mathcal{M}$ under which the formula is logically true.

We write $\mathcal{M} \models \mathcal{A}$ when this is the case. E.g., $\mathcal{M}_1 \models \neg A$.

For example, $A \wedge \neg A$ is *unsatisfiable*, but all the other formulas are satisfiable.

A formula A is *valid* if it is logically true under any interpretation, written as $\models A$.

For example, $A \Rightarrow (B \vee A)$ is valid, but the other formulas are not.

A propositional formula is valid if and only if its negation is unsatisfiable. Otherwise, the formula is invalid, i.e., its negation is satisfiable.

Equivalence

Two formulas A and B are equivalent, $A \iff B$, if their truth values agree in each interpretation.

Exercise 2 *Prove that the following are equivalent (TFAE):*

1. $\neg\neg A \iff A$
2. $A \Rightarrow B \iff \neg A \vee B$
3. $\neg(A \wedge B) \iff \neg A \vee \neg B$
4. $\neg(A \vee B) \iff \neg A \wedge \neg B$
5. $\neg A \Rightarrow B \iff \neg B \Rightarrow A$

Normal Forms

A formula where negation is applied only to propositional atoms is said to be in negation normal form (NNF).

A literal is either a propositional atom or its negation.

A formula that is a multiary conjunction of multiary disjunctions of literals is in conjunctive normal form (CNF).

A formula that is a multiary disjunction of multiary conjunctions of literals is in disjunctive normal form (DNF).

Exercise 3 *Show that every propositional formula is equivalent to one in NNF, CNF, and DNF.*

Exercise 4 *Show that every n-ary Boolean function can be expressed using just $\neg$ and $\vee$.*

Proofs and Theorems

For propositional formulas, it is possible to do this by exhaustive evaluation, but this is impossible when the domains involved are infinite.

Proofs provide a finitary means for demonstrating validity.

Proofs are constructed starting from *axiom* formulas using *inference rules*.

A proof system is an *inductive definition* of a provability judgement that asserts that

1. Axioms are provable.
2. The conclusion of the application of an inference rule to provable premises, is provable.
3. Only formulas that are derived according to rules 1 and 2 are provable.

A Propositional Proof System

A *sequent* has the form $\Gamma \vdash \Delta$.

Γ is the *set* of *antecedent* formulas.

Δ is the *set* of *consequent* formulas.

A sequent $\Gamma \vdash \Delta$ captures the judgement: $\bigwedge \Gamma \Rightarrow \bigvee \Delta$ is provable.

A formula A is provable if the judgement $\vdash A$ can be derived in the proof system.

A Propositional Proof System: LK_0

	Left	Right
Ax	$\dfrac{}{\Gamma, A \vdash A, \Delta}$	
$\neg$	$\dfrac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}$	$\dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}$
$\vee$	$\dfrac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}$	$\dfrac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}$
$\wedge$	$\dfrac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}$	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}$
$\Rightarrow$	$\dfrac{\Gamma, B \vdash \Delta \quad \Gamma \vdash A, \Delta}{\Gamma, A \Rightarrow B \vdash \Delta}$	$\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \Rightarrow B, \Delta}$
Cut	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}$	

Soundness and Completeness

A proof system is *sound* if all provable formulas are valid, i.e., $\vdash A$ implies $\models A$.

Exercise 5 *Demonstrate the soundness of LK_0.*

A proof system is *complete* if all valid formulas are provable, i.e., $\models A$ implies $\vdash A$. In other words, any unprovable formula must be satisfiable.

Exercise 6 *Demonstrate the completeness of LK_0.*

A set of formulas Γ is *consistent* iff there is no formula A in Γ such that $\Gamma \vdash \neg A$.

A logic is *compact* if any set of sentences Γ is satisfiable if all finite subsets of it are.

Exercise 7 *Demonstrate the compactness of PL.*