Little Engines of Proof

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

Programming Options

Scheme is a compact, easy-to-learn variant of Lisp. The `bigloo` compiler is available at `http://www-sop.inria.fr/mimosa/fp/Bigloo/`.

ML is a typed variant with type-inference, polymorphic typing, references, and modules. `oCaml` is a popular implementation that is available at `http://www.ocaml.org/`.

Many of the programs in the course can be easily implemented with a rewriting engine such as Maude (`http://maude.cs.uiuc.edu`).

PVS (`http://pvs.csl.sri.com`) is a theorem-proving/proof-checking environment based on higher-order logic. PVS could be used to generate executable code from verified descriptions.

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	$\dfrac{\Gamma \vdash A, \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta}$	

Completeness

A sequent $\Gamma \vdash \Delta$ is just $\bigvee \overline{\Gamma} \vee \bigvee \Delta$, where $\overline{\Gamma}$ is the result of negating the formulas in Γ.

If you drop the Cut rule, the LK_0 rules preserve equivalence and transform the formula A to CNF form as a conjunction of *clauses*.

The CNF form is unprovable when it contains a non-axiom clause C.

Clause C yields a satisfying truth assignment $\mathcal{M}$ for its negation $\neg C$ which also satisfies $\neg A$.

Completeness, Alternately

A set of formulas is *complete* if for each formula A, it contains A or $\neg A$.

Any consistent set of formulas Γ can be made complete as $\hat{\Gamma}$.

Let A_i be the i'th formula in some enumeration of PL formulas. Define

$$\begin{aligned} \Gamma_0 &= \Gamma \\ \Gamma_{i+1} &= \Gamma_i \cup \{A_i\}, \text{ if } Con(\Gamma_i \cup \{A_i\}) \\ &= \Gamma_i \cup \{\neg A_i\}, \text{ otherwise.} \\ \hat{\Gamma} &= \Gamma_\omega = \bigcup_i \Gamma_i \end{aligned}$$

Ex: Check that $\hat{\Gamma}$ yields an interpretation $\mathcal{M}_{\hat{\Gamma}}$ satisfying Γ.

Equational Logic

Equational logic deals with terms τ such that

$$\begin{aligned} \tau &:= f(\tau_1, \ldots, \tau_n), \text{ for } n \geq 0 \\ \phi &:= P \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \supset \phi_2 \mid \tau_1 = \tau_2 \end{aligned}$$

The meaning $\mathcal{M}[\![a]\!]$ is an element of a *domain* D, and $\mathcal{M}(f)$ is a map from D^n to D, where n is the arity of f.

$$\begin{aligned} \mathcal{M}[\![a = b]\!] &= \mathcal{M}[\![a]\!] = \mathcal{M}[\![b]\!] \\ \mathcal{M}[\![f(a_1, \ldots, a_n)]\!] &= (\mathcal{M}[\![f]\!])(\mathcal{M}[\![a_1]\!], \ldots, \mathcal{M}[\![a_n]\!]) \end{aligned}$$

Proof Rules for Equational Logic (EL_0)

The following rules are added to LK_0 to obtain EL_0.

Reflexivity	$\Gamma \vdash a = a, \Delta$
Symmetry	$\dfrac{\Gamma \vdash a = b, \Delta}{\Gamma \vdash b = a, \Delta}$
Transitivity	$\dfrac{\Gamma \vdash a = b, \Delta \qquad \Gamma \vdash b = c, \Delta}{\Gamma \vdash a = c, \Delta}$
Congruence	$\dfrac{\Gamma \vdash a_1 = b_1, \Delta \ldots \Gamma \vdash a_n = b_n, \Delta}{\Gamma \vdash f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n), \Delta}$

Soundness and Completeness

Ex: Demonstrate the soundness of EL_0: $\vdash A$ implies $\models A$.

A consistent set of formulas Γ can be extended to a complete and consistent set $\hat{\Gamma}$.

From $\hat{\Gamma}$ it is possible to construct a *term model* consisting of the equivalence classes $[t]$ of terms t in $\hat{\Gamma}$.

The interpretation $\mathcal{M}_{\hat{\Gamma}}$ is given by $\mathcal{M}_{\hat{\Gamma}}(f)([t_1], \ldots, [t_n]) = [f(t_1, \ldots, t_n)]$.

Ex: Prove that $\mathcal{M}_{\hat{\Gamma}}$ is a model of $\hat{\Gamma}$.

First-order Logic
Variables and Quantifiers

The terms and formulas of FOL are given by

$$
\begin{array}{lcl}
\tau & := & X \\
 & | & f(\tau_1, \ldots, \tau_n), \text{ for } n \geq 0 \\
\phi & := & \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \supset \phi_2 \mid \tau_1 = \tau_2 \\
 & & \mid \forall x : \phi \mid \exists x : \phi \mid q(\tau_1, \ldots, \tau_n), \text{ for } n \geq 0
\end{array}
$$

Terms contain variables, and formulas contain atomic and quantified formulas.

Semantics for Variables and Quantifiers

$\mathcal{M}[\![q]\!]$ is a map from D^n to $\{\top, \bot\}$, where n is the arity of predicate q.

$$
\begin{array}{rcl}
\mathcal{M}[\![x]\!]\rho & = & \rho(x) \\
\mathcal{M}[\![q(a_1, \ldots, a_n)]\!]\rho & = & \mathcal{M}[\![q]\!](\mathcal{M}[\![a_1]\!]\rho, \ldots, \mathcal{M}[\![a_n]\!]\rho) \\
\mathcal{M}[\![\forall x : A]\!]\rho & = & \begin{cases} \top, & \text{if } \mathcal{M}[\![A]\!]\rho[x := d] \text{ for all } d \in D \\ \bot, & \text{otherwise} \end{cases} \\
\mathcal{M}[\![\exists x : A]\!]\rho & = & \begin{cases} \top, & \text{if } \mathcal{M}[\![A]\!]\rho[x := d] \text{ for some } d \in D \\ \bot, & \text{otherwise} \end{cases}
\end{array}
$$

Proof Rules for Quantifiers (LK)

The following rules are added to EL_0 to obtain LK.

	Left	Right
$\forall$	$\dfrac{\Gamma, \forall x : A, A[t/x] \vdash \Delta}{\Gamma, \forall x : A \vdash \Delta}$	$\dfrac{\Gamma \vdash A[c/x], \forall x : A, \Delta}{\Gamma \vdash \forall x : A, \Delta}$
$\exists$	$\dfrac{\Gamma, \exists x : A, A[c/x] \vdash \Delta}{\Gamma, \exists x : A \vdash \Delta}$	$\dfrac{\Gamma \vdash A[t/x], \exists x : A, \Delta}{\Gamma \vdash \exists x : A, \Delta}$

Constant c must be chosen to be new so that it does not appear in the conclusion sequent.

Exercises

1. Define operations for collecting the free variables in a given formula, and substituting a term for a free variable in a formula.
2. Prove $\exists x : (p(x) \Rightarrow \forall y : p(y))$.
3. Give at least two satisfying interpretations for the statement $(\exists x : p(x)) \supset (\forall x : p(x))$.
4. A sentence is a formula with no free variables. Find a sentence A such that both A and $\neg A$ are satisfiable.
5. Write a formula asserting the unique existence of an x such that $p(x)$.
6. Show that any quantified formula is equivalent to one in *prenex normal form*, i.e., where the only quantifiers appear at the head of the formula.

Soundness and Completeness

Unlike LK_0 and EL_0, the LK quantifier rules require copying.

Proof branches can be extended without bound.

Ex: Show that LK is sound: $\vdash A$ implies $\models A$.

The Henkin closure $H(\Gamma)$ is the smallest extension of a set of sentences Γ that is Henkin-closed, i.e., contains $B \Rightarrow A(c_B)$ for every $B \in H(\Gamma)$ of the form $\exists x : A$.

Any consistent set of formulas Γ has a *consistent* Henkin closure $H(\Gamma)$.

As before, any consistent, Henkin closed set of formulas Γ has a complete, Henkin-closed extension $\hat{\Gamma}$.

Ex: Show that the resulting interpretation $\mathcal{M}_{\widehat{H(\Gamma)}}$ is a model for $H(\Gamma)$.

Herbrand's Theorem

Every FOL sentence has a prenex equivalent.

In a cut-free sequent proof of a prenex formula, the quantifier rules can be made to appear below all the other rules.

Such proofs must have a quantifier-free mid-sequent above which the proof is entirely equational/propositional.

Thus for any sentence A there is a quantifier-free sentence A_H such that $\vdash A$ in LK iff $\vdash A_H$ in EL_0.

Computability

In the 1930s, **Herbrand**, **Gödel**, **Church**, **Turing**, **Post**, and **Kleene** clarified the nature of computability.

Turing proposed the notion of a machine with a finite state controller with its head at a specific location on an infinite tape, a sequence of discrete cells.

In each machine transition, the controller reads a symbol off the tape, and writes a new symbol in its place, and moves to a new controller state and a new tape location that is adjacent to the previous one, until it reaches the halting state.

Turing Machines

For example, a constant function just writes out a constant past the given input.

The identity function copies the contents of the input past the end of the input.

Turing also constructed a universal Turing machine U such that for every other Turing machine M and input x, it was possible to code the machine as an input m so that $U(m, x) = M(x)$.

Computability

In 1936, **Church** had shown that the validity problem for predicate calculus was not solvable using the lambda calculus as a computation model.

This was the first instance of an computationally unsolvable problem.

Ex: Show that there is no machine H such that for any given Turing machine M and input x, M terminates on x iff $H(m, x)$ computes to 0.

Turing also showed the Turing-unsolvability of the validity problem for predicate calculus.

Different computational models like Turing machines, lambda calculus, and recursive definitions are equally expressive.

Recursive Functions

A function f is recursive if it can be defined by a collection of definitions of the form

1. Constant functions: $f(x_1, \ldots, x_n) = k$, for numeral k.
2. Projections: $f(x_1, \ldots, x_n) = x_i$.
3. Compositions $f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n))$, for h, g recursive.
4. Primitive recursion: $f(0, \ldots, x_n) = g(x_1, \ldots, x_n)$, and $f(x_1 + 1, \ldots, x_n) = h(f(x_1, \ldots, x_n), x_1, x_2, \ldots, x_n)$, for g, h, recursive.
5. Minimization: $f(x_1, \ldots, x_n) = \mu x : g(x, x_1, \ldots, x_n) = 0$, when $\vdash \forall x_1, \ldots, x_n : \exists x : g(x, x_1, \ldots, x_n) = 0$.

Recursive and Enumerable Sets

A set S is computable or recursive if it has a computable characteristic function f_S such that $f_S(x) = 0 \iff x \in S$.

The set of well-formed formulas in most logics are recursive.

A set is S (recursively) enumerable if there is a recursive predicate P_S such that $x \in S \iff \exists i : P_S(x, i)$.

The set of theorems in most logics are recursively enumerable (r.e.).

Every recursive set is also r.e.

The complement of a recursive set is recursive.

The complement of an r.e. set, i.e., a co-r.e. set may not be r.e., but if it is, the set is recursive.

There are r.e. sets that are not recursive.

Decidability

A logic L is *decidable* if the set of theorems is recursive.

Propositional logic is decidable.

Most modal propositional logics are decidable.

First-order logic is undecidable, but some fragments are decidable.

Some first-order theories are decidable: Presburger arithmetic $\langle 0, 1, + \rangle$.

Most first-order theories are not decidable: Peano arithmetic $\langle 0, 1, +, * \rangle$.

Small Decidability Problems

Word problems (WP): $\vdash A$ for atomic A. Many word problems are decidable.

Uniform word problems (UWP): $A_1, \ldots, A_n \vdash A$ for atomic $A, A_1, \ldots, A_n$.

Some theories do have undecidable UWPs: semigroups, groups.

There are decidable UWPs: transitive closure, congruence closure, partial orders, commutative semigroups.

Complexity Classes

There is a hierarchy of complexity classes among the problems that are computable.

The notion of bounded computability is given in terms of a Turing machine with an input tape of length n, an output tape, and a work tape of length $S(n)$ and a bound $T(n)$ on the number of steps.

A polynomially computable operation requires $T(n)$ to be a polynomial.

Decidability problems are often the canonical hard problems in each complexity class.

E.g., Unification is P-complete, propositional satisfiability is NP-complete, QBF validity is PSPACE-complete, the word problem for commutative semigroups is EXPSPACE-complete.

Refutation Decision Procedures

A decision procedure determines if a collection of formulas is satisfiable.

A decision procedure is given by a collection of reduction rules on a *logical state* ψ.

State ψ is either $\perp$ or of the form $\kappa_1|\ldots|\kappa_n$, where each κ_i is a *configuration*.

The logical content of κ is either $\perp$ or is given by a finite set of formulas of the form $A_1, \ldots, A_m$.

A state ψ of the form $\kappa_1, \ldots, \kappa_n$ is satisfiable if some configuration κ_i is satisfiable.

A configuration κ of the form $A_1, \ldots, A_m$ is satisfiable if there is an interpretation M and an assignment ρ such that $M, \rho \models A_i$ for $1 \leq i \leq m$.

Inference Systems for Decision Procedures

A refutation procedure proves A by refuting $\neg A$ through the application of reduction rules.

An application of an reduction rule transforms a state ψ to a state ψ' (written $\psi \vdash \psi'$).

The states ψ and ψ' must be *equivalent*: Any $\mathcal{M}, \rho$ such that $\mathcal{M}, \rho \models \psi$ there is a ρ' extending ρ such that $\mathcal{M}, \rho' \models \psi'$, and conversely whenever $\mathcal{M}, \rho' \models \psi'$, there is a ρ extending ρ' such that $\mathcal{M}, \rho' \models \psi$.

If relation $\vdash$ between states is well-founded and any non-$\perp$ irreducible state is satisfiable, we say that the inference system is a decision procedure.

Ex: Prove that a decision procedure as given above is sound and complete.

An Example

An inference rule $\dfrac{\kappa}{\kappa_1 | \ldots | \kappa_n}$ is shorthand for $\dfrac{\psi[\kappa]}{\psi[\kappa_1 | \ldots | \kappa_n]}$.

The inference rules for a sequent search procedure are

$\dfrac{A \wedge B, \Gamma}{A, B, \Gamma} \wedge +$	$\dfrac{\neg(A \wedge B), \Gamma}{\neg A, \Gamma \| \neg B, \Gamma} \wedge -$
$\dfrac{\neg(A \vee B), \Gamma}{\neg A, \neg B, \Gamma} \vee -$	$\dfrac{(A \vee B), \Gamma}{A, \Gamma \| B, \Gamma} \wedge +$
$\dfrac{\neg(A \Rightarrow B), \Gamma}{A, \neg B, \Gamma} \Rightarrow +$	$\dfrac{(A \Rightarrow B), \Gamma}{\neg A, \Gamma \| B, \Gamma} \Rightarrow -$
$\dfrac{\neg\neg A, \Gamma}{A, \Gamma} \neg$	$\dfrac{A, \neg A, \Gamma}{\bot} \bot$

Ex: Prove soundness and completeness of the above inference system.