Using linear logic and proof theory to unify computational logic

Dale Miller

Inria Saclay & LIX, École Polytechnique
Palaiseau, France

3 September 2017

Purpose: Outline a textbook that provides a single, formal foundations for a wide range of topics in computational logic.

Scope of computational logic

Includes the following topics using classical and intuitionistic logics

- Theorem proving in logic: resolution, tableaux, SAT, etc
- Logic programming (proof search)
- Term representation
- Type systems (Curry-Howard)
- Functional programming (proof normalization)

The scope should also include topics from arithmetic.

- Model checking
- Theorem proving in arithmetic (induction & coinduction)

Topics explicitly related to linear logic (e.g., linear logic programming, linear types, GoI) may not be part of such a text.

Higher-order quantification should play a (limited) role.

A textbook for computational logic

For the sake of discussion, I propose to structure a textbook that provides

- a substantial collections of definitions and theorems that
- can be applied systematically to describe a diversity of topics in computational logic.

The main threads in this proposed structure.

1. Sequent calculus, rule permutations, and cut-elimination.
2. Lessons learned from linear logic: additive/multiplicative distinguish, control of structural rules, and polarity.
3. Focused proof system and the resulting notion of synthetic inference rules.

What about category theory/type theory/model theory?

These theories are valuable in order to

- place computational logic into the broader topics of logic, philosophy, semantics, and mathematics;
- provide deeper results about computational logic (e.g., completeness and incompleteness theorems); and
- provide alternative descriptions of important concepts: theorems are also formulas valid in all models.

What about category theory/type theory/model theory?

These theories are valuable in order to

- place computational logic into the broader topics of logic, philosophy, semantics, and mathematics;
- provide deeper results about computational logic (e.g., completeness and incompleteness theorems); and
- provide alternative descriptions of important concepts: theorems are also formulas valid in all models.

But proof theory can provide

- a common definition of terms (with and without bindings), formulas, proof structures, substitutions, etc;
- a small but powerful meta-theory (rule permutations, cut-elimination);
- a flexible approach to synthetic inference rules, and
- two dynamics: proof search and proof normalization.

Terms and formulas

Identify terms and formulas as simply typed λ-terms. Formulas are simple terms of a special type (o in Church 1940).

Types are best consider as just "syntactic categories": term, formula, abstraction of a term over a formula, etc.

Many operations on terms and formulas follow from the literature on λ-calculus.

Sequent calculus: the good points

Gentzen introduction of sequents provided the first "unity of logic" result: cut-elimination for both classical and intuitionistic logics.

Cut-elimination implies consistency, new decision algorithms for intuitionistic propositional logic, etc.

The sequent calculus is not just some proof system.

The sequent calculus is usually considered just some choice among many equal choices including natural deduction, tableaux, expansion trees, dependently typed λ-terms, etc.

Sequent calculus is the *assembly language for proofs*.

Most other proof systems can be compiled into sequent calculus.

Sequent calculus: the bad points

From a computer science point-of-view, a direct use of sequent calculus is *crazy*.

- It uses tiny inference rules.
- It is amorphous, chaotic. If a sequent has a proof, it probably has many proofs, all different (but not really).
- There is no treatment of parallelism in proof.

Focused proof systems and synthetic inference rules

Focused proof systems can build large scale synthetic inference rules (bipoles) from tiny (introduction) rules.

Core technical devices

- Use a multifocused versions of LJF and LKF [Liang & Miller 2007]. Can capture parallelism in proofs.
- In LJF, atoms and conjunctions can be polarized arbitrarily; in LKF, also disjunctions can be polarized arbitrarily.
- Different polarizations mean different proofs but not different theorems.
- Contraction is applied only to positive formulas.
- Negative non-atomic formulas are treated as if they were linear (never contracted, never weakened).

Extending MALL

MALL is a good starting point since multiplicative/additive distinction and polarity are clear.

Extend with first-order quantifiers and predicates (i.e., non-logical formula symbols). CS exploits these extensively.

Also allow weakening and contraction. Two popular choices.

- exponentials ?, ! (Linear logic)
- polarity and zones: LC & LU [Girard], LLP [Laurent], LKU [Liang-Miller]

Later a separate extension using fixed points.

General proof theory results

The following can be addressed naturally using focusing and polarization.

- Geometric formulas (bipolar formulas).
- Double negation translations, Glivenko's theorem, etc.
- Completeness of Herbrand's theorem, interpolation
- Correctness of Skolemization
  - Two different schemes for Skolemization.
  - One corresponds to the use of eigenvariables. One requires quantifier movement and the use of cuts.
  - Conventionally, Skolemization is justified semantically...

Theorem proving in logic

There are now many presentations of proofs in

resolution, tableaux, SAT, SMTP, etc

as sequent calculus proofs (in part, for certification purposes).

Focusing provides information about contraction and that enables the description of decision and semi-decision procedures.

Logic programming as proof search

Uniform proofs (1986) (alternations of goal-reduction and backchaing phases) is replaced by focused proofs.

Top-down (Prolog) and bottom-up (Datalog) distinctions are polarity issues.

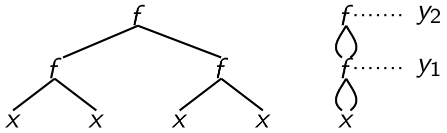
Proof theory allows abstractions within logic programming (modules, abstract datatypes, higher-order programming, bindings in terms).

Cut-elimination allows some reasoning about logic programs.

Term representation and typing

Canonical forms: changing polarity switches between *head normal form* and *administrative normal form*.

Given variables $x\colon i$ and $f : i \to i \to i$.



When i is polarized negatively, we get the left structure, namely, $f\ (f\ (x,\ x),\ f\ (x,\ x))$.

When i is polarized positively, we get the right structure, namely, $\textbf{name}\ y_1 = (f\ x\ x)\ \textbf{in}\ \textbf{name}\ y_2 = (f\ y_1\ y_1)\ \textbf{in}\ y_2$,
i.e., sharing is a cut-free, bottom-up structure.

Dependently typed λ-terms are proofs in intuitionistic f.o. logic.

λ-calculus and functional programming

Untyped lambda-terms. Typing judgments.

Beta-reduction and cut-elimination. Explicit substitutions.

Evaluation: Call-by-name, call-by-value is a polarization choice. Call-by-push-value.

Curry-Howard correspondence

Conventionally, typed λ-calculi were based on negatively biased connectives (implication, universal quantifiers).

There is a need for positively polarized connectives also: existential types, pattern matching, let-expressions, sharing, etc.

Some proof theory behind arithmetic

μMALL is MALL extended with least/greatest fixed points (plus first-order structures). Induction and co-induction rules.

Focused proof system: μMALLF. μ is positive, ν is negative.

Purely positive fixed point: $\vdash B \equiv\ !B$. E.g., Horn clauses.

```
nat z.
nat (s N) :- nat N.

app nil K K.
app (cons X L) K (cons X M) :- app L K M.
```

Some proof theory behind arithmetic

μMALL is MALL extended with least/greatest fixed points (plus first-order structures). Induction and co-induction rules.

Focused proof system: μMALLF. μ is positive, ν is negative.

Purely positive fixed point: $\vdash B \equiv\ !B$. E.g., Horn clauses.

```
nat z.
nat (s N) :- nat N.

app nil K K.
app (cons X L) K (cons X M) :- app L K M.
```

$$nat = \mu\lambda N\lambda n\,(n = \mathtt{z} \vee \exists n'(n = \mathtt{s}\ n' \wedge^{+} N\ n'))$$

$$app = \mu\lambda A\lambda l\lambda k\lambda m\,((l = \mathtt{nil} \wedge^{+} k = m) \vee^{+}$$
$$\exists x\exists l'\exists m'(l = \mathtt{cons}\ x\ l' \wedge^{+} m = \mathtt{cons}\ x\ m' \wedge^{+} A\ l'\ k\ m'))$$

What about μLJF and μLKF?

Arithmetic hierarchy and alternation of polarity.

Model checking

Model-theoretic semantics is easily captured using additive inference rules. But, purely additive rules are too weak.

Additive synthetic inference rules: allow multiplicative behavior only within phases.

Let $P \xrightarrow{A} Q$ be a labeled transition system between processes and actions that has a purely positive definition.

The simulation for this labeled transition systems is defined as

$$\nu\big(\lambda S \lambda p \lambda q.\, \forall a \forall p'.\, p \xrightarrow{a} p' \supset \exists q'.\, q \xrightarrow{a} q' \wedge^{+} S\, p'\, q'\big)$$

To extend expressiveness of model checking, additional multiplicative structures are needed:

- tables (lemmas)
- constraints (goals that are delayed).

Theorem proving in arithmetic: induction, coinduction

Modern provers need to separate computation and deduction. Such a distinction arises in focusing: the negative phase is functional.

Singleton sets provide an ambiguity of polarity: if $P(\cdot)$ denotes a singleton set, then

$$\exists x(P(x) \wedge^{+} Q(x)) \;\equiv\; \forall x(P(x) \supset Q(x))$$

Change "generate-and-test" by calculate. This observation can be used to separate within proofs functional computations from more general proof search [CSL 2017].

Certify the result of theorem proving by outputing key aspects of discovered proofs. Attempt to capture the provers actual rules using synthetic rules.

Thank you

Questions?