Applications of structural proof theory to computer science

Dale Miller

Inria Saclay & LIX, École Polytechnique
Palaiseau, France

Séminaire "Réflexions sur les processus de calcul,
d'information et de programmation"
Université de Lille 3

15 June 2016

Scope of today's talk

By "structural proof theory" I mean Gentzen's sequent calculus [1935] with the refinements that we have learned from Girard's linear logic [1987].

By computational logic, I mean a broad collection of computational topics which make significant use of logic.

- logic programming
- functional programming
- type theories
- model checking
- theorem proving

Theme of this talk: A rich interactions between Computational Logic and Proof Theory has shaped their evolution.

Outline

Part 1: Focused proofs system: sequent calculus in CS

Part 2: Certain computer science reasoning tasks might require us to leave mathematics behind.

An early application of sequent calculus in CS

What is a logic programming language? Until the mid-1980's, there were examples of logic programming languages but no framework.

Definition: An abstract logic programming language is a proof system where uniform proofs are complete. [LICS 1987/APAL 1991]

A uniform proof are built from two phases.

- Goal-reduction phase: sequents with non-atomic right-hand-sides are the consequent of a right-introduction rules.
- Backchaining phase: left introduction rules are grouped together to establish the atomic right-hand-side.

Sequent calculus plays a central role in this definition.

Linear Logic

J.-Y. Girard, Theoretical Computer Science, 1987, pp. 1–102.

A new logic. The "logic behind logics".

Provided new approaches to proof: proof nets, geometry of interaction.

However: the simplest and most natural presentation of linear logic was given by the sequent calculus.

As computer scientists attempted to understand and exploit linear logic, they needed to learn the sequent calculus.

Sequent calculi for linear logic

Multiplicative, additive, exponential connectives

$$\frac{\Gamma \vdash B_1 \qquad \Gamma' \vdash B_2}{\Gamma, \Gamma' \vdash B_1 \otimes B_2} \qquad \frac{\Gamma \vdash B_1 \qquad \Gamma \vdash B_2}{\Gamma \vdash B_1 \& B_2} \qquad \frac{!\Gamma \vdash B}{!\Gamma \vdash !B}$$

$$!(B_1 \& B_2) \equiv (!B_1 \otimes !B_2)$$

Multiple conclusion sequent calculus

$$\frac{\Gamma \vdash B_1, \Delta \qquad \Gamma' \vdash B_2, \Delta'}{\Gamma, \Gamma' \vdash B_1 \otimes B_2, \Delta, \Delta'} \qquad \frac{\Gamma \vdash B_1, \Delta \qquad \Gamma \vdash B_2, \Delta}{\Gamma \vdash B_1 \& B_2, \Delta} \qquad \frac{!\Gamma \vdash B, ?\Delta}{!\Gamma \vdash !B, ?\Delta}$$

The multiple conclusion sequent calculus of Gentzen was not an option in presenting full linear logic.

Focused proof systems

The two phase structure of uniform proofs can be extended to all of linear logic with some remarkable symmetries and applications.

Andreoli [PhD Paris VI, 1990 / JLC 1992] designed what he called a "focused proof system" for linear logic.

Many focusing proof systems (besides uniform proofs) have been seen before, but his was the most comprehensive and general.

Liang & Miller [CSL 2007, TCS 2009] designed focusing systems for classical and intuitionistic logic: LKF and LJF. These account for all previous focused proof systems.

The *LKneg* proof system

Use invertible rules where possible. In propositional classical logic, both conjunction and disjunction can be given invertible rules.

$$\frac{\vdash \cdot; B}{\vdash B}\ \mathit{start} \qquad \frac{\vdash \Delta, L; \Gamma}{\vdash \Delta; L, \Gamma}\ \mathit{store} \qquad \frac{}{\vdash \Delta, A, \neg A; \cdot}\ \mathit{init}$$

$$\frac{\vdash \Delta; \Gamma}{\vdash \Delta; \mathit{false}, \Gamma} \qquad \frac{\vdash \Delta; B, C, \Gamma}{\vdash \Delta; B \vee C, \Gamma} \qquad \frac{}{\vdash \Delta; \top, \Gamma} \qquad \frac{\vdash \Delta; B, \Gamma \quad \vdash \Delta; C, \Gamma}{\vdash \Delta; B \wedge C, \Gamma}$$

Here, A is an atom, L a literal, Δ a multiset of literals, and Γ a list of formulas. Sequents have two *zones*.

This proof system provides a decision procedure (resembling conjunctive normal forms).

A small (constant sized) certificate is possible.

The *LKneg* proof system

Use invertible rules where possible. In propositional classical logic, both conjunction and disjunction can be given invertible rules.

$$\frac{\vdash \cdot; B}{\vdash B}\ \mathit{start} \qquad \frac{\vdash \Delta, L; \Gamma}{\vdash \Delta; L, \Gamma}\ \mathit{store} \qquad \frac{}{\vdash \Delta, A, \neg A; \cdot}\ \mathit{init}$$

$$\frac{\vdash \Delta; \Gamma}{\vdash \Delta; \mathit{false}, \Gamma} \qquad \frac{\vdash \Delta; B, C, \Gamma}{\vdash \Delta; B \vee C, \Gamma} \qquad \frac{}{\vdash \Delta; \top, \Gamma} \qquad \frac{\vdash \Delta; B, \Gamma \quad \vdash \Delta; C, \Gamma}{\vdash \Delta; B \wedge C, \Gamma}$$

Here, A is an atom, L a literal, Δ a multiset of literals, and Γ a list of formulas. Sequents have two *zones*.

This proof system provides a decision procedure (resembling conjunctive normal forms).

A small (constant sized) certificate is possible.

Consider proving $(p \vee C) \vee \neg p$ for large C.

The *LKpos* proof system

Non-invertible rules are used here.

$$\frac{\vdash B; \cdot; B}{\vdash B}\ \textit{start} \qquad \frac{\vdash B; \mathcal{N}, \neg A; B}{\vdash B; \mathcal{N}; \neg A}\ \textit{restart} \qquad \frac{}{\vdash B; \mathcal{N}, \neg A; A}\ \textit{init}$$

$$\frac{\vdash B; \mathcal{N}; B_i}{\vdash B; \mathcal{N}; B_1 \vee B_2} \qquad \frac{}{\vdash B; \mathcal{N}; \top} \qquad \frac{\vdash B; \mathcal{N}; B_1 \quad \vdash B; \mathcal{N}; B_2}{\vdash B; \mathcal{N}; B_1 \wedge B_2}$$

Here, A is an atom and $\mathcal{N}$ is a multiset of negated atoms.
Sequents have three *zones*.

The $\vee$ rule *consumes* some external information or some non-determinism.

An *oracle string*, a series of bits used to indicate whether to go left or right, can be a proof certificate.

A proof in *LKpos*

Let C have several alternations of conjunction and disjunction.

Let $B = (p \vee C) \vee \neg p$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash B; \neg p; p}\,\mathit{init}}{\vdash B; \neg p; p \vee C}\,*}{\vdash B; \neg p; (p \vee C) \vee \neg p}\,*}{\vdash B; \cdot\ ; \neg p}\,\mathit{restart}}{\vdash B; \cdot\ ; (p \vee C) \vee \neg p}\,*}{\vdash B}\,\mathit{start}$$

The subformula C is avoided. Clever choices $*$ are injected at these points: right, left, left. We have a small certificate and small checking time. In general, these certificates may grow large.

Combining the *LKneg* and *LKpos* proof systems

Introduce two versions of conjunction, disjunction, and their units.

$$t^-, t^+, f^-, f^+, \vee^-, \vee^+, \wedge^-, \wedge^+$$

The inference rules for negative connectives are invertible.

These polarized connectives also exist in linear logic.

Introduce the two kinds of sequent, namely,
$\vdash \Theta \Uparrow \Gamma$: for invertible (negative) rules (Γ a list of formulas)
$\vdash \Theta \Downarrow B$: for non-invertible (positive) rules (B a formula)

LKF : a focused proof systems for classical logic

$$\frac{}{\vdash \Theta \Uparrow \Gamma, t^-} \qquad \frac{\vdash \Theta \Uparrow \Gamma, B \quad \vdash \Theta \Uparrow \Gamma, B'}{\vdash \Theta \Uparrow \Gamma, B \wedge^- B'} \qquad \frac{\vdash \Theta \Uparrow \Gamma}{\vdash \Theta \Uparrow \Gamma, f^-} \qquad \frac{\vdash \Theta \Uparrow \Gamma, B, B'}{\vdash \Theta \Uparrow \Gamma, B \vee^- B'}$$

$$\frac{}{\vdash \Theta \Downarrow t^+} \qquad \frac{\vdash \Theta \Downarrow B \qquad \vdash \Theta \Downarrow B'}{\vdash \Theta \Downarrow B \wedge^+ B'} \qquad \frac{\vdash \Theta \Downarrow B_i}{\vdash \Theta \Downarrow B_1 \vee^+ B_2}$$

Init

$$\frac{}{\vdash \neg A, \Theta \Downarrow A}$$

Store

$$\frac{\vdash \Theta, C \Uparrow \Gamma}{\vdash \Theta \Uparrow \Gamma, C}$$

Release

$$\frac{\vdash \Theta \Uparrow N}{\vdash \Theta \Downarrow N}$$

Decide

$$\frac{\vdash P, \Theta \Downarrow P}{\vdash P, \Theta \Uparrow \cdot}$$

P is a positive formula; N is a negative formula;
A is an atom; C positive formula or negative literal

Results about LKF

Let B be a propositional logic formula and let $\hat{B}$ result from B by placing $+$ or $-$ on t, f, $\wedge$, and $\vee$ (there are exponentially many such placements).

Theorem. [Liang & M, TCS 2009]

- If B is a tautology then every polarization $\hat{B}$ has an LKF proof.
- If some polarization $\hat{B}$ has an LKF proof, then B is a tautology.

The different polarizations do not change *provability* but can radically change the *proofs*.

Also:

- Negative (non-atomic) formulas are treated linearly (never weakened nor contracted).
- Only positive formulas are contracted (in the Decide rule).

Example: deciding on a simple clause

Assume that Θ contains the formula $a \wedge^+ b \wedge^+ \neg c$ and that we have a derivation that Decides on this formula.

$$\dfrac{\dfrac{}{\vdash \Theta \Downarrow a}\,\mathit{Init} \quad \dfrac{}{\vdash \Theta \Downarrow b}\,\mathit{Init} \quad \dfrac{\dfrac{\vdash \Theta, \neg c \Uparrow \cdot}{\vdash \Theta \Uparrow \neg c}\,\mathit{Store}}{\vdash \Theta \Downarrow \neg c}\,\mathit{Release}}{\dfrac{\vdash \Theta \Downarrow a \wedge^+ b \wedge^+ \neg c}{\vdash \Theta \Uparrow \cdot}\,\mathit{Decide}}\,\wedge^+$$

This derivation is possible iff Θ is of the form $\neg a, \neg b, \Theta'$. Thus, the "macro-rule" is

$$\dfrac{\vdash \neg a, \neg b, \neg c, \Theta' \Uparrow \cdot}{\vdash \neg a, \neg b, \Theta' \Uparrow \cdot}$$

Applications of LKF: to proof theory

Herbrand's theorem is an immediate consequence of the completeness of LKF.

> *Polarized all connectives negative. The only remaining positive connective is the existential which is the only one used more than once.*

A central theme of modern proof theory has been the treatment of parallelism in proof structures.

The introduction of "maximal multifocused proof systems" provides the sequent calculus with a means of describing parallelism in both *proof nets* and *expansion trees* (a generalization of Herbrand disjunctions).

Applications of LKF: Computational Logic

In logic programming,

- polarizing atoms negatively yields "top-down search" and
- polarizing atoms positively yields "bottom-up search".

In functional programming,

- polarize atoms negative yield "head-normal form" and
- polarize atoms positive yield "administrative normal form".

What about first-order terms and quantification?

Computer Science needs terms, natural numbers, lists, trees, etc.

We need first-order quantifiers ($\forall$, $\exists$), term equality, and fixed point definitions.

These were first addressed independently by Schroeder-Heister [1991/2] and Girard [1992].

Their extensions required eigenvariables to be substituted within cut-free proofs (this differs from Gentzen's treatment).

Theorems are those true in all fixed points. This is too weak.

In Computer Science, we have properties defined by induction (least fixed points) and coinduction (greatest fixed points).

μMALL and μLJ

μMALL = MALL plus equality, quantification, and least and greatest fixed points. Similarly, adding these to LJ yields μLJ.

The proof theory of least and greatest fixed points developed by three of my previous PhD students.

- McDowell: natural numbers induction.
- Tui: more general forms of induction and coinduction
- Baelde: μMALL as an alternative way of extending MALL.

The rules for induction (coinduction) both use higher-order variables for invariant (coinvariant).

With higher-order variables

- automated discovery of invariants is not expected and
- cut-free proofs do not have the subformula property.

Applications of μMALL and μLJ

Model checking seems like a question of semantics (checking truth in a model). This is a job for "additive connectives".

Model checking procedures are algorithms: e.g., check for the existence of a path, etc. This is a job for "multiplicative connectives".

Focusing in μMALL allows us to blend these: the synthetic connectives must be additive while internally they are multiplicative.

The Parsifal team is developing a model checker (Bedwyr) and an inductive theorem prover (Abella) based on insights from μLJF.

Part 2: We might need to leave behind mathematics

In some areas of computational logic, we need to reason formally at a meta-level.

Example: Let M range over programs in some (functional) programming language.

- If M has type T and has value V, then V has type T.
- If M has value V and U then $V = U$.

Consider also formalizing the meta-theory of the λ-calculus or the π-calculus.

Metatheory is unlike other domains

Formalizing metatheory requires dealing with *linguistic* items (e.g., types, terms, formulas, proofs, programs, etc) which are not typical data structures (e.g., integers, trees, lists, etc).

The authors of the POPLmark challenge tried metatheory problems on existing systems and urged the developers of proof assistants to make improvements:

> *Our conclusion [...] is that the relevant technology has developed almost to the point where it can be widely used by language researchers. We seek to push it over the threshold, making the use of proof tools common practice in programming language research* *[TPHOLS 2005]*

The treatment of bindings in syntax is a major stumbling block.

Major first step: Drop mathematics as an intermediate

A traditional approach to formalizing metatheory.

1. Implement mathematics
   - Pick a rich logic: intuitionistic higher-order logic, classical first-order logic, set theory, etc.
   - Provide abstractions such as sets, functions, etc.
2. Model computation via mathematical structures:
   - via denotational semantics and/or
   - via inductively defined data types and proof systems.

What could be wrong with this approach? Isn't mathematics the universal language?

Major first step: Drop mathematics as an intermediate

A traditional approach to formalizing metatheory.

1. Implement mathematics
   - Pick a rich logic: intuitionistic higher-order logic, classical first-order logic, set theory, etc.
   - Provide abstractions such as sets, functions, etc.
2. Model computation via mathematical structures:
   - via denotational semantics and/or
   - via inductively defined data types and proof systems.

What could be wrong with this approach? Isn't mathematics the universal language?

Various "intensional aspects" of computational specifications — bindings, names, resource accounting, etc — are challenges to this approach to reasoning about computation.

Examples of intensional aspects of expressions

Consider *algorithms*: two sort programs describe the same function but should not be replaced in all contexts.

A more explicit example: Is the following a theorem?

$$\forall w_i \ \neg(\lambda x.x = \lambda x.w) \qquad (*)$$

Examples of intensional aspects of expressions

Consider *algorithms*: two sort programs describe the same function but should not be replaced in all contexts.

A more explicit example: Is the following a theorem?

$$\forall w_i \; \neg(\lambda x.x = \lambda x.w) \qquad (*)$$

If λ-abstractions denote functions, $(*)$ is equivalent to

$$\forall w_i \; \neg \; \forall x(x = w).$$

This is not a theorem (consider the singleton model).

Examples of intensional aspects of expressions

Consider *algorithms*: two sort programs describe the same function but should not be replaced in all contexts.

A more explicit example: Is the following a theorem?

$$\forall w_i \; \neg(\lambda x.x = \lambda x.w) \qquad (*)$$

If λ-abstractions denote functions, $(*)$ is equivalent to

$$\forall w_i \; \neg \; \forall x(x = w).$$

This is not a theorem (consider the singleton model).

If λ-abstractions denote syntactic expressions, then $(*)$ should be a theorem since no (capture avoiding) substitution of an expression of type i for the w in $\lambda x.w$ can yield $\lambda x.x$.

Two Type Theories of Church [JSL 1940]

Tension between a logic for metatheory and for mathematics.

Axioms 1-6: Elementary Type Theory (ETT). Foundations for a higher-order predicate calculus.

Two Type Theories of Church [JSL 1940]

Tension between a logic for metatheory and for mathematics.

Axioms 1-6: Elementary Type Theory (ETT). Foundations for a higher-order predicate calculus.

Axioms 7-11: Simple Theory of Types (STT)

- non-empty domains
- Peano's axioms,
- axioms of description and choice, and
- extensionality for functions.

Adding these gives us a foundations for much of mathematics.

Two Type Theories of Church [JSL 1940]

Tension between a logic for metatheory and for mathematics.

Axioms 1-6: Elementary Type Theory (ETT). Foundations for a higher-order predicate calculus.

Axioms 7-11: Simple Theory of Types (STT)

- non-empty domains
- Peano's axioms,
- axioms of description and choice, and
- extensionality for functions.

Adding these gives us a foundations for much of mathematics.

With extensionality, description, and choice, STT goes too far for our interests in metatheory.

We keep to ETT and eventually extend it for our metatheory needs.

Simple types as syntactic categories

The type o (omicron) is the type of formulas.

Other primitive types provide for multisorted terms.

The arrow type denotes the syntactic category of one syntactic category over another.

For example, the universal quantifier $\forall_\tau$ is not applied to a term of type τ and a formula (of type o) but rather to an abstraction of type $\tau \to o$.

Both $\forall_\tau$ and $\exists_\tau$ belong to the syntactic category $(\tau \to o) \to o$.

Typing in this sense is essentially the same as Martin-Löf's notion of *arity types*.

How abstract is your syntax?

Gödel and Church did their formal metatheory on string representation of formulas! Today, we parse strings into *abstract syntax* (a.k.a *parse trees*). But how abstract is that syntax?

Principle 1: *The names of bound variables should be treated as the same kind of fiction as white space.*

Principle 2: *There is "one binder to ring them all."*[1]

Principle 3: *There is no such thing as a free variable. (Alan Perlis's epigram 47.)*

Principle 4: *Bindings have* mobility *and the equality theory of expressions must support such mobility.*

[1] A scrambling of J. R. R. Tolkien's "One Ring to rule them all, ... and in the darkness bind them."

Dynamics of binders during proof search

During proof search, binders can be *instantiated* (using β implicitly)

$$\frac{\Sigma : \Delta, \mathit{typeof}\ c\ (\mathit{int} \to \mathit{int}) \vdash C}{\Sigma : \Delta, \forall \alpha(\mathit{typeof}\ c\ (\alpha \to \alpha)) \vdash C}\ \forall L$$

They also have *mobility* (they can move):

$$\frac{\Sigma, x : \Delta, \mathit{typeof}\ x\ \alpha \vdash \mathit{typeof}\ \lceil B \rceil\ \beta}{\dfrac{\Sigma : \Delta \vdash \forall x(\mathit{typeof}\ x\ \alpha \supset \mathit{typeof}\ \lceil B \rceil\ \beta)}{\Sigma : \Delta \vdash \mathit{typeof}\ \lceil \lambda x.B \rceil\ (\alpha \to \beta)}}\ \forall R$$

In this case, the binder named x moves from *term-level* (λx) to *formula-level* ($\forall x$) to *proof-level* (as an eigenvariable in Σ, x).

Proof theory provides an elegant framework for explaining this style of reasoning. Doing this with mathematical concepts is possible but much less natural and immediate.

Thank you