

A Multiple-Conclusion Specification Logic

Alwen Tiu

Australian National University

ANU Logic Summer School
Lecture 3, December 14, 2007

Linear logic programming?

- The abstract logic programming languages we've seen so far are single-conclusioned. Goal-directed proof search is straightforwardly defined.
- What are the natural extension of the notion of goal-directedness or uniform proofs to multiple-conclusion setting?
- Why do we need multiple conclusion? Think of concurrency: a sequent

$$\Gamma \longrightarrow A_1, \dots, A_n$$

represents 'processes' A_1, \dots, A_n running in parallel, which can *interact* with each other.

- This means we must generalise program clauses as well, to describe possible interactions, e.g.,

$$G \multimap A_1 \wp A_2.$$

Processes A_1 and A_2 interacts to produce another goal G .

Uniform provability for multiple-conclusion sequents

Definition

A cut-free sequent proof Π is *uniform* if for every subproof Π' of Π and for every non-atomic formula occurrence B in the right-hand side of the end sequent of Π' , there is a proof Π'' that is equal to Π' up to a permutation of inference rules and is such that the last inference rule in Π'' introduces the top-level logical connective of B .

Notice the “up to a permutation of inference rules” part.

Definition

A logic with a sequent calculus proof system is an *abstract logic programming language* if restricting to uniform proofs does not lose completeness.

Design of the logic programming language

- We still have the same problem as in intuitionistic case with the connectives $\mathbf{1}$, \otimes , \oplus and $!$.
- We extend the language of \mathcal{L} with \wp , $?$ and \perp . So the set of connectives we consider is: \top , $\&$, \perp , \wp , \multimap , $?$ and \forall .
- This language is called Forum. It is a superset of all other previous languages we've seen.

Linear logic = Forum

Forum is actually equivalent to linear logic. The missing connectives can be defined as follows:

$$B^\perp \equiv B \multimap \perp \quad \mathbf{0} \equiv \top \multimap \perp \quad \mathbf{1} \equiv \perp \multimap \perp$$

$$B \oplus C \equiv (B^\perp \& C^\perp)^\perp \quad B \otimes C \equiv (B^\perp \wp C^\perp)^\perp$$

$$!B \equiv (B \Rightarrow \perp) \multimap \perp \quad \exists x.B \equiv (\forall x.B^\perp)^\perp$$

The proof system \mathcal{F}

$$\frac{}{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, \top, \Gamma; \Upsilon} \top_R$$

$$\frac{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, B, \Gamma; \Upsilon \quad \Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, C, \Gamma; \Upsilon}{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, B \wedge C, \Gamma; \Upsilon} \&_R$$

$$\frac{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, \Gamma; \Upsilon}{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, \perp, \Gamma; \Upsilon} \perp_R \quad \frac{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, B, C, \Gamma; \Upsilon}{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, B \wp C, \Gamma; \Upsilon} \wp_R$$

$$\frac{\Sigma : \Psi; B, \Delta \longrightarrow \mathcal{A}, C, \Gamma; \Upsilon}{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, B \multimap C, \Gamma; \Upsilon} \multimap_R \quad \frac{\Sigma : B, \Psi; \Delta \longrightarrow \mathcal{A}, C, \Gamma; \Upsilon}{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, B \Rightarrow C, \Gamma; \Upsilon} \Rightarrow_R$$

$$\frac{y : \tau, \Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, B[y/x], \Gamma; \Upsilon}{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, \forall_{\tau} x. B, \Gamma; \Upsilon} \forall_R \quad \frac{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, \Gamma; B, \Upsilon}{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, ?B, \Gamma; \Upsilon} ?_R$$

The proof system \mathcal{F}

$$\frac{\Sigma : B, \Psi; \Delta \xrightarrow{B} \mathcal{A}; \Upsilon}{\Sigma : B, \Psi; \Delta \longrightarrow \mathcal{A}; \Upsilon} \textit{decide!} \qquad \frac{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}, B; B, \Upsilon}{\Sigma : \Psi; \Delta \longrightarrow \mathcal{A}; B, \Upsilon} \textit{decide?}$$

$$\frac{\Sigma : \Psi; \Delta \xrightarrow{B} \mathcal{A}; \Upsilon}{\Sigma : \Psi; B, \Delta \longrightarrow \mathcal{A}; \Upsilon} \textit{decide}$$

$$\frac{}{\Sigma : \Psi; . \xrightarrow{A} \mathcal{A}; \Upsilon} \textit{initial}$$

$$\frac{}{\Sigma : \Psi; . \xrightarrow{A} .; \mathcal{A}, \Upsilon} \textit{initial?}$$

The proof system \mathcal{F}

$$\begin{array}{c}
 \frac{}{\Sigma : \Psi; . \xrightarrow{\perp} .; \Upsilon} \perp_L \quad \frac{\Sigma : \Psi; \Delta \xrightarrow{B_i} \mathcal{A}; \Upsilon}{\Sigma : \Psi; \Delta \xrightarrow{B_1 \& B_2} \mathcal{A}; \Upsilon} \&_L \quad \frac{\Sigma : \Psi; B \longrightarrow .; \Upsilon}{\Sigma : \Psi; . \xrightarrow{?B} .; \Upsilon} ?_L \\
 \\
 \frac{\Sigma : \Psi; \Delta_1 \xrightarrow{B} \mathcal{A}_1; \Upsilon \quad \Sigma : \Psi; \Delta_2 \xrightarrow{C} \mathcal{A}_2; \Upsilon}{\Sigma : \Psi; \Delta_1, \Delta_2 \xrightarrow{B \wp C} \mathcal{A}_1 + \mathcal{A}_2; \Upsilon} \&_L \quad \frac{\Sigma : \Psi; \Delta \xrightarrow{B[t/x]} \mathcal{A}; \Upsilon}{\Sigma : \Psi; \Delta \xrightarrow{\forall_T x. B} \mathcal{A}; \Upsilon} \forall_L \\
 \\
 \frac{\Sigma : \Psi; \Delta_1 \longrightarrow \mathcal{A}_1, B; \Upsilon \quad \Sigma : \Psi; \Delta_2 \xrightarrow{C} \mathcal{A}_2; \Upsilon}{\Sigma : \Psi; \Delta_1, \Delta_2 \xrightarrow{B \multimap C} \mathcal{A}_1 + \mathcal{A}_2; \Upsilon} \multimap_L \\
 \\
 \frac{\Sigma : \Psi; . \longrightarrow B; \Upsilon \quad \Sigma : \Psi; \Delta \xrightarrow{C} \mathcal{A}; \Upsilon}{\Sigma : \Psi; \Delta \xrightarrow{B \Rightarrow C} \mathcal{A}; \Upsilon} \Rightarrow_L
 \end{array}$$

Soundness and completeness

Theorem

(Soundness) *If the sequent $\Sigma : \Psi; \Delta \longrightarrow \Gamma; \Upsilon$ has an \mathcal{F} proof then*

$$!\Psi, \Delta \vdash_{LL} \Gamma, ?\Upsilon.$$

If the sequent $\Sigma : \Psi; \Delta \xrightarrow{B} \mathcal{A}; \Upsilon$ has an \mathcal{F} proof then

$$!\Psi, \Delta, B \vdash_{LL} \Gamma, ?\Upsilon.$$

Theorem

(Completeness) *If $!\Psi, \Delta \vdash_{LL} \Gamma, ?\Upsilon$ then the sequent*

$$\Sigma : \Psi; \Delta \longrightarrow \Gamma; \Upsilon$$

has a proof in \mathcal{F} .

Clause with empty head

In Forum, we are allowed to write a program clause with empty head (i.e., empty \mathcal{F} -disjunction), e.g.,

$$\forall x(G \multimap \perp).$$

This clause, when used in backchaining, will match any goal(s), and easily leads to non-termination.

Clauses with empty head can be used to encode cut-rule in specifying object-logic sequent calculus. It mimics the behavior cut-rule in proof search.

Example: multiset rewriting

Multiset rewriting has been shown to capture some aspects of concurrent computations, especially in the specification of communication protocols. Forum can be used naturally to specify multiset rewriting. For example,

$$\{a, b\} \rightsquigarrow \{c, d, e\}$$

can be encoded as the clause

$$F = a \wp b \multimap c \wp d \wp e.$$

Then we can prove in \mathcal{F}

$$\Sigma : F; c \wp d \wp e \longrightarrow a \wp b; .$$

Example: encoding object-level sequent

We can use Forum to “implement” other sequent systems. Consider for example the sequent calculus for minimal logic (a subset of intuitionistic logic). Sequents in minimal logic take the form

$$B_1, \dots, B_n \longrightarrow C.$$

This is encoded as the (right-hand side) formula in \mathcal{F} sequent

$$?left(B_1), \dots, ?left(B_n), right(C).$$

Note that

- Object-level formula in minimal logic becomes terms in Forum.
- We use a meta-level (Forum) predicate *left* and *right* to encode which side a formula belongs to in the object sequent.
- The encoding reflects the fact that we can contract object-level formula on the ‘left’ but not on the ‘right’ side of the object sequents.

Example: specification of minimal logic

- (\supset_R) $right(A \supset B) \multimap (?left(A) \wp right(B)).$
- (\supset_L) $?left(A \supset B) \multimap right(A) \multimap ?left(B).$
- (\wedge_R) $right(A \wedge B) \multimap right(A) \multimap right(B).$
- (\wedge_{L_1}) $?left(A \wedge B) \multimap ?left(A).$
- (\wedge_{L_2}) $?left(A \wedge B) \multimap ?left(B).$
- (Id) $right(B) \wp ?left(B).$
- (Cut) $\perp \multimap ?left(B) \multimap right B.$

Note that clause (\supset_L) is equivalent to the “uncurried” form

$$?left(A \supset B) \multimap (right(A) \otimes ?left(B)).$$

Example: encoding CCS

The *calculus of communicating systems* (CCS) is a formal model of concurrent computation developed by Milner. The language of processes:

$$P := nil \mid (P \mid P) \mid (P + P) \mid !P \mid a.P \mid \bar{a}.P$$

Two processes can *synchronise* and evolve into another process:

$$a.P \mid \bar{a}.Q \rightarrow P \mid Q.$$

Encode processes as formulas in Forum:

$$\begin{aligned} nil^\bullet &= nil & (P \mid Q)^\bullet &= P^\bullet \wp Q^\bullet \\ (P + Q)^\bullet &= plus P^\bullet Q^\bullet & (!P)^\bullet &= bang P^\bullet \\ (a.P)^\bullet &= send a P^\bullet & (\bar{a}.P)^\bullet &= get a P^\bullet. \end{aligned}$$

Example: specification of CCS

$$\forall x \forall S \forall R [R \wp S \multimap \text{get } x \ R \wp \text{send } x \ S]$$

$$\forall P \forall Q [P \multimap \text{plus } P \ Q] \quad \forall P \forall Q [Q \multimap \text{plus } P \ Q]$$

$$\forall P [\text{bang } P \wp \text{bang } P \multimap \text{bang } P] \quad \forall P [P \multimap \text{bang } P]$$

Let us denote this specification with Γ and let Σ contains the appropriate constant declarations. Then:

Theorem

If $P \rightsquigarrow^* Q$ then $\Sigma : \Gamma; Q^\bullet \longrightarrow P^\bullet; \cdot$ is provable in \mathcal{F} .

Summary

- We have seen some basic proof theory for intuitionistic and linear logic.
- We have looked at the computational interpretation of proof search in both logic.
- This has been applied to the design of logic programming languages. The essence of abstract logic programming is *goal-directed* proof search.
- By moving to richer logics, we get increasing expressivity: modules and abstract data types, resource models, and concurrency.
- By staying within logic, we can make use of the logical structures of the programs to reason about their behaviors.

Implementations

- Hereditary Harrop fragment has been implemented in the language called λ Prolog. It was developed by Nadathur and Miller. The current (maintained) implementation is called Teyjus.
- Intuitionistic linear logic \mathcal{L} was implemented by Hodas and Miller in a language called Lolli. This doesn't seem to be maintained.
- There is no serious implementations of Forum, due to some technical difficulty. There is a prototype implemented by Miller.
- Links for these can be found on Dale Miller's website.
- The HH fragment has been also implemented to be used as a *logical framework*. This line of work is more oriented towards theorem proving. Example of this include: Elf/Twelf, Isabelle.

Research topics

- Finding a logical notion of *control*: aborting proof search, pruning search, etc.
- Failed proof search. Proof theory deals only with proofs, yet implementation of proof search often deals with partial/incomplete proofs.
- Game semantics for proof search: viewing proof search process as two players game.
- Extending the notion of focussed proofs.
- Reasoning about logic specifications: need logic with fixed points, induction/co-induction, etc.