PART II

An Efficient Refiner for First-order Intuitionistic Logic

— J. P. River —
CONNECTION TO NUPRL-5

Concept: Add JProver as a new Refiner to the NUPRL-5 architecture.
calculus L^\perp, i.e., the first-order fragment of L^\perp

L^\perp sequent proof in Gentzen's single conclusion sequent


search-free proof reconstruction procedure into L^\perp or L^\perp

Reconstruction component: reconstruction procedure based on the extension procedure

Proof search strategy based on the extension procedure

Connection prover: MetapRL term, interface between NupRL and OCaml

J-formula / J-sequent: J-formula / J-sequent

Architecture of the J-prover Refiner
Requires irredundancy check during proof search

\[ \varphi \land (x \in A) \equiv (\varphi \land x \in A) \]

Example:

Non-permutability of sequent rules: Efficient computation of a global substitution \( \varphi \)

Connection calculus: Proof search guided by connectives' successive decomposition

CONNECTION-BASED THEOREM PROVING
Proof Reconstruction: Requirements

Requirements:

\[ \mathcal{O} \cap \mathcal{C} \ni \mathcal{F} \]

Basic Idea: Traverse reduction ordering \( \mathcal{O} \) to construct a sequent proof in \( \mathcal{L}^m \mathcal{C} \)
Incompatibility between $\mathbf{LTL}$ and pure first-order theorem proving

3. Providing non-empty types: Kripke semantics assume non-empty domains

$\sim$

2. First-order fragment of $\mathbf{LTL}$: NuPRL allows only closed sequents

$\sim$

1. Interface: MetapRL terms; first hybrid system using NuPRL's and MetapRL's reinter

Integration into NuPRL: Variables / functions range over the same type

Output: Sequent proof in the first order fragment of $\mathbf{LTL}$

Input: NuPRL sequent, including function symbols and free-variables (constants)

Reifier: Reifier embedded into MetapRL to avoid code duplication

Integration of JProver into NuPRL
Efficient computation of $\forall$-proofs vs. $\forall$-nodes

- $\forall$-proof for $\forall$-proof
  - Encoding exactly the inference steps for every possible $\forall$-proof

Define two operations on $\forall$-proofs, resulting in a general relation:

Solution: Explicit representation of the inference steps for $\forall$-proofs

\[
\begin{align*}
\frac{\top \lor \neg \top}{B \land (B \land \forall) \land (B \land \forall) \lor ((B \land \forall) \lor B) \land} & \\
\frac{A \land (B \land \forall) \land (B \land \forall) \land (\neg \exists \quad \text{let subgoal})}{}
\end{align*}
\]

Purity concept: Insufficient (deletion of "non-connected" subformulae)

Complete redundancy deletion in $\forall$ to avoid search / deadlocks

REQUIREMENTS I: INTEGRATION OF PROOF KNOWLEDGE
Requires complete redundancy delation in $\preceq$

\[ \vdash \quad \frac{B \land A \vdash I}{I} \quad \frac{\vdash C, B \vdash A}{I, C, B \vdash A} \quad \frac{I, \vdash \Box C}{I, \vdash B \lor A} \]

Selection strategy: $\vdash \Box \lor A$ or $\vdash \Box \lor B$

\[ \vdash \Box \lor A \quad \vdash \Box \lor B \]

Dynamic completion of reduction ordering $\preceq$ during proof reconstruction necessary

SWITCHING BETWEEN $\LL^c$ AND $\LL$
proof permutations · by need · additional eigenvariable renaming required

S

– Apply permutation-based transformations on LL macro proof, obtaining an LL proof for S
– If a deadlock occurs at a sequent S, reconstruct an LL macro proof for S
– Apply selection strategy for reconstructing an LL proof as far as possible

Solution: Combine proof reconstruction with proof permutation approach

\[
1 \lor x \eta \iff \frac{xB \cdot xE \iff ((z \forall \neg \cdot zE) \iff (\neg \forall \cdot \neg \eta E)) \lor (xB \land x \forall \cdot xA) \land xB \cdot xE \lor (z \forall \neg \cdot zE) \iff (\neg \forall \cdot \neg \eta E), xB \land x \forall \cdot xA}{xB \cdot xE \lor (z \forall \neg \cdot zE) \iff (\neg \forall \cdot \neg \eta E), xB \land x \forall \cdot xA}
\]

Reason: Global substitution from the matrix proof

Problem: Selection strategy for LL is incomplete for the first-order case

Requirements II: Permutation-Based Transformations
Make sure that $\forall$-prover is declared in the original NuPRL sequent.

- Map free variables of the whole sequent proof to a unique variable $\forall$-prover
- Split $\forall$ by replacing non-declared eigenvariables with original variables, e.g. $x$

Solution: "localize" Global substitution $\forall$ after splitting.

\[
\begin{align*}
\forall \langle x \forall \cdot x A \rangle & \quad \forall \langle x \forall \cdot x A \rangle \\
\forall \langle x \forall \cdot x A \rangle & \quad \forall \langle x \forall \cdot x A \rangle
\end{align*}
\]

Reason: NuPRL allows only closed sequents (every used parameter has to be declared).

Problem: Global substitution $\forall$ cannot be used in different branches.
Incomparability between \( \mathcal{M} \) and pure first-order theorem proving

Solution: Introduce unique object \( \forall \mathcal{O} \)-prover via the cut rule (by “need”)

\[
\begin{align*}
\forall A & \quad T \\
\mathcal{O} A & \quad \forall x \mathcal{O} A \\
\forall x \mathcal{O} A & \quad \forall \mathcal{O} A \\
\forall \mathcal{O} A & \quad T
\end{align*}
\]

Problem: Provide at least one element for the unique type used by \( \mathcal{O} \)-prover

Integration II: Providing non-empty types
SUMMARY: JProver

- Introduce unique object vs. JProver to provide non-empty types for JProver
- Split global substitutions \( B \) to handle closed sequents in NuPRL
- Complete selection strategy for \( \alpha \) proofs via permutational-based transformations
- Efficient computation of \( \alpha \)-proofs wrt. used proof calculus
- Integrate proof knowledge \( \alpha \) - \( \beta \) (\( \beta \)-proofs) to avoid search / deadlocks
- Traverse reduction ordering decision respecting constraints from \( \langle \alpha, \beta \rangle \)
- Combination of different transformation approaches
- Admissibility conditions require iterative check during proof search
- First-order logic: combined prefix yields satisfiability, \( \forall \) quantifier unification
- Propositional fragment: efficient prefix unification, yields substitution
- Reels on combined unification modules to encode rule non-permutability
- Connection-search: connection-driven path checking algorithm
Special thanks to Lor and Alexey

To do: Integrate JProver into MetaPRL proof sessions

Alternative: call JProver with transformation tactics – where did they go?

Suggestion: switch off dependency check for JProver

– Tree construction lasts much longer than a „plain“ tactic call (small examples)

– Superfluous dependency check between the tree nodes

Both solutions suffer from an inefficient tree construction

Proft trees as „real proofs“ vs. „display proofs“ (no further refinement steps)

Representation of first-order proof trees: