Bridging holes on Dedukti proofs, an overview

UPSCaLe Scientific Day

Guillaume Burel$^{1,2}$

Tuesday October 9th, 2018

$^1$Samovar, ENSIIE, Université Paris Saclay

$^2$Inria and LSV, CNRS and ENS Paris Saclay
Introduction

Dedukti

Logical framework based on the $\lambda\Pi$-calculus modulo rewriting

Can express many logics

- Import from Matita, OpenTheory, FoCaLize, ...
- Export to Coq, Matita, PVS, Lean, OpenTheory (HOL Light, HOL4, ...)
- Theorem provers with a Dedukti output: iProverModulo, Zenon modulo, ArchSat
Holes in (yet) incomplete proofs = metavariables

refine tactic of Dedukti v3.0

refine nat_ind (λ n, P n m) ?CZ[n,m] ?CS[n,m] n
Proof traces

Many tools do not provide proofs in the Curry-Howard sense

- SAT/SMT solvers
  - DRAT format
- FO automated theorem provers
  - TSTP format
- Instrumented provers
Proof interoperability

Delegation of proofs

Example: SMT solvers
Dealing with proof holes

Traces as incomplete proofs
  ▶ Needs to bridge missing informations

In ProofCert:
  ▶ clerks vs. experts

Here: using OCaml or Dedukti programs
A journey through incomplete proofs

Outline

■ Introduction

■ A journey through incomplete proofs
  • Proof terms
  • Proof trace reconstruction
  • Dealing with unprovability

■ Conclusion
OpenTheory

Common format for provers of the HOL family
- HOL Light, HOL4 and ProofPower

HOLiDe: translation from OpenTheory to Dedukti [Assaf 2012]
Proof format designed for proof exchange
- No proof holes!

(In the converse direction, needs to reconstruct $\beta$-reduction steps.)
Calculus of Construction

Coq/Matita $\rightarrow$ Dedukti

$\Pi x : A, B$ translated as $\pi_{s_1,s_2}|A|(\lambda x, |B|)$

Problem: needs to know sorts $s_1, s_2$

- Use Coq kernel to infer them
Res. \[ \frac{P \lor C \quad \neg Q \lor D}{\sigma(C \lor D)} \quad \sigma = \text{mgu}(P, Q) \]

Fact. \[ \frac{P \lor Q \lor D}{\sigma(P \lor D)} \quad \sigma = \text{mgu}(P, Q) \]

Proof trace from e.g. Prover9:
- which rule?
- which premises?
- which literals?
- which derived clause?

[Cauderlier 18] Dedukti tactic using metadedukti
- a program written in Dedukti
- produce Dedukti proof terms for each inference step

```plaintext
def C3 := resolution.resolve 0 2 C1 C2.

thm c3 : resolution.qcproof C3 := resolution.resolve_correct 0 2 C1 C2 c1 c2.
```
TSTP

Proof format of the CADE community

List of formulas
  - each annotated by an inference tree whose leafs are other formulas

\[
\text{cnf}(c_{0\_60}, \text{plain},
  ( \text{join}(X1, \text{join}(X2, X3)) = \text{join}(X2, \text{join}(X1, X3)) ),
  \text{inference}(rw, [\text{status(thm)}],
    [\text{inference}(spm, [\text{status(thm)}], [c_{0\_30}, c_{0\_18}]),
    c_{0\_30}))).
\]
TSTP

Proof format of the CADE community

List of formulas
  ▶ each annotated by an inference tree whose leafs are other formulas

cnf(c_0_60,plain,
    ( join(X1,join(X2,X3)) = join(X2,join(X1,X3)) ),
    inference(rw,[status(thm)],
        [inference(spm,[status(thm)],[c_0_30,c_0_18]),
            c_0_30]))).

Independent of the proof calculus
Proof calculus of E

- \( \text{valid}(S) \subseteq C \)
- \( \text{valid}(S) \cap \text{valid}(T) = \emptyset \) then \( \text{valid}(S \cup T) \)

We say that a literal \( L \) is selected (with respect to a given selection function) in a clause \( C \) if \( \text{valid}(C \cup \{L\}) \). We will use two kinds of restrictions on selecting new clauses. One is implied by selecting contraries and the other by selection functions. We combine these in the notion of eligible literals.

Definition 3.1.2 (Eligible literals) Let \( \text{valid}(V) \subseteq C \) be a clause, let \( \sigma \) be a substitution and let \( \theta \) be a selection function.

- \( \theta(V) \) is eligible for paramodulation if either
  - \( \theta(V) \) and \( \theta(\text{valid}(V)) \cup \text{valid}(C) \) is minimal in \( \theta(V) \).
  - \( \theta(V) \) and \( \theta(\text{valid}(V)) \cup \text{valid}(C) \) is minimal in \( \theta(V) \) and \( \theta(V) \).
- \( \theta(V) \) is eligible for paramodulation if \( \theta(V) \) in \( C \) is positive and \( \theta(V) \) is positive.

The clause is represented in the form of inference rules. For convenience, we write rules with a single line separating preconditions and results, the result is added to the set of all clauses. For constructing inference rules, written with a double line, the new clause is substituted for the clause in the premisses. In the following, \( \sigma, \tau, v \) and \( \theta \) are terms, \( = \) is a substitution, and \( = \) is a selection function. Variables \( x, y, z \) and \( a \) are terms.

Definition 3.1.3 (The inference system SP) Let \( \theta \) be a local simplification ordering (extended to subterms \( \gamma \) and \( \hbar \) on literals and clauses), let \( \sigma \) be a selection function, and let \( \theta \) be a set of fresh parameterized constants. The inference system SP consists of the following inference rules:

- \( \text{Equivalence substitution:} \)
  \[ \begin{align*}
  \text{G} & : \text{valid}(V) \\
  \sigma & = \text{valid}(V) \\
  \theta & = \text{valid}(V) \\
  \end{align*} \]
  \( \text{R} \) is eligible for paramodulation.

- \( \text{Supposition into positive literals:} \)
  \[ \begin{align*}
  \text{G} & : \text{valid}(V) \\
  \sigma & = \text{valid}(V) \\
  \theta & = \text{valid}(V) \\
  \end{align*} \]
  \( \text{R} \) is eligible for paramodulation.

- \( \text{Supposition into negative literals:} \)
  \[ \begin{align*}
  \text{G} & : \text{valid}(V) \\
  \sigma & = \text{valid}(V) \\
  \theta & = \text{valid}(V) \\
  \end{align*} \]
  \( \text{R} \) is eligible for paramodulation.

- \( \text{Resolving of positive literals:} \)
  \[ \begin{align*}
  \text{G} & : \text{valid}(V) \\
  \sigma & = \text{valid}(V) \\
  \theta & = \text{valid}(V) \\
  \end{align*} \]
  \( \text{R} \) is eligible for paramodulation.

- \( \text{Resolving of negative literals:} \)
  \[ \begin{align*}
  \text{G} & : \text{valid}(V) \\
  \sigma & = \text{valid}(V) \\
  \theta & = \text{valid}(V) \\
  \end{align*} \]
  \( \text{R} \) is eligible for paramodulation.
Proof reconstruction

Use information from the proof trace to guide proof building

Inspired by Sledgehammer and PRocH

Two approaches:
- premises selector
- trace steps reconstruction
A journey through incomplete proofs

Premises selector

Problems can contain many axioms

- (especially if they come from ITP in a huge development)

Proofs found by ATP only use a few of them

Use the trace to know which axioms are actually needed
Reconstruct the proof from scratch using only these axioms

- In a Dedukti-producing ATP
A journey through incomplete proofs

Premises selection, experimental results

[Pham 2016]:
Fork of Zenon modulo, reads a TSTP file and keep only needed axioms

On 12467 FO problems of the TPTP library:

<table>
<thead>
<tr>
<th></th>
<th>Zenon modulo (alone)</th>
<th>E prover + Zenon modulo</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Problems solved</td>
<td>2274</td>
<td>8901</td>
</tr>
<tr>
<td>%</td>
<td>18.2</td>
<td>71.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3249</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26.0</td>
</tr>
</tbody>
</table>
Proof step reconstruction

Axiom selection not enough, need to rebuild each proof steps

Part of Yacine El Haddad PhD thesis (ongoing work)

- agnostic wrt the proof calculus
- agnostic wrt the proof-producing reconstructor
A journey through incomplete proofs

Architecture

Guillaume Burel:
Bridging holes on Dedukti proofs, an overview
Proofs Modulo Theory

SMT solver VeriT

Proof traces:

- logical steps
- theory “axioms”
  - formulas valid in the theory
  - generated by the theory reasoner (learned lemma)

Verine: translation to Dedukti [Gilbert 15]

- Logical steps can be easily translated
- Needs theory specific Dedukti-proof producing solver
  - ArchSat? Coq(Omega) + translation?
SAT solving

De facto standard for SAT solvers: DRAT

List of clauses

- each new clause preserves satisfiability of preceding ones
  - using a criterion called Reverse Asymmetric Tautology
- Deletion: indicates which clauses are no longer needed

New clauses may not be logical consequences of preceding ones!

- think of Skolemization in FOL
Proof transformation

Satisfiability preservation:
Γ has a model ⇒ Γ, C has a model

Provability preservation:
Γ, C ⊢ ⊥ has a proof ⇒ Γ ⊢ ⊥ has a proof

1. Start from Γ, ⊥ ⊢ ⊥
2. Transform proof until Axioms ⊢ ⊥

RAT criterion leads to an algorithm to transform proofs
▶ using auxiliary clauses
Limits of proof transformation

Start from the end of the trace
  - Cannot benefit from deletion information

Can be adapted to follow the trace in the right order, but produces too many unneeded auxiliary clauses
Extended Resolution

Fortunately, [Kiesl et al. 2018]:

- Extended resolution simulates DRAT

Extended resolution [Tseitin 1968]:

- resolution + definitions of new propositional variables
- Easily expressible in Dedukti

The translation from DRAT to what we need of Extended resolution can be performed in quadratic time (better in practice)
Outline

- Introduction
- A journey through incomplete proofs
- Conclusion
Conclusion

Different widths of proof holes to bridge

- detailed and well-defined proof steps directly translatable in Dedukti
- general inference steps that need to be reconstructed by a Dedukti producing tool
- non-provable steps that involve proof transformation
Conclusion

Further work

What about systems that barely produce a trace?
  ▶ e.g. PVS?

Give a global name (an URL?) to proof holes to ease interoperability?
  ▶ see logipedia