Bridging holes on Dedukti proofs, an overview

UPSCaLe Scientific Day

Guillaume Burel[1,2]

Tuesday October 9th, 2018

[1]Samovar, ENSIIE, Université Paris Saclay

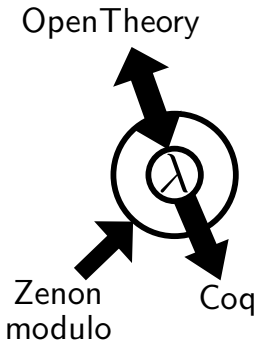
[2]Inria and LSV, CNRS and ENS Paris Saclay, Université Paris Saclay

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



Interactive Theorem Proving

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

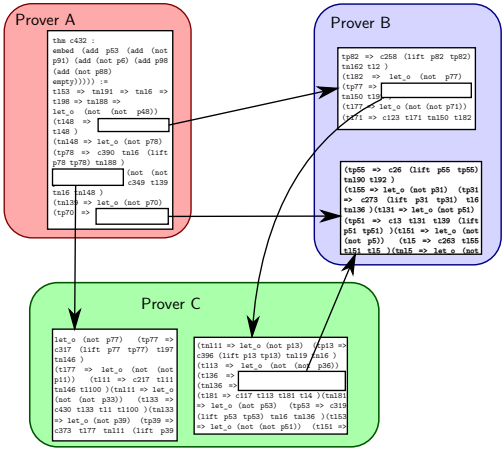
Many tools do not provide proofs in the Curry-Howard sense

- 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

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 β-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

Resolution proofs

Res. $\dfrac{P \vee C \qquad \neg Q \vee D}{\sigma(C \vee D)}\ \sigma = mgu(P, Q)$ Fact. $\dfrac{P \vee Q \vee D}{\sigma(P \vee D)}\ \sigma = mgu(P, Q)$

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

```
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

```
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])).
```

TSTP

Proof format of the CADE community

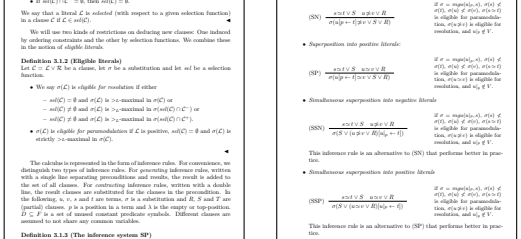
List of formulas

- 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



- $sel(\mathcal{C}) \subseteq \mathcal{C}$.
- If $sel(\mathcal{C}) \cap \mathcal{C}^- = \emptyset$, then $sel(\mathcal{C}) = \emptyset$.

We say that a literal L is *selected* (with respect to a given selection function) in a clause C if $L \in sel(C)$. ◂

We will use two kinds of restrictions on deducing new clauses: One induced by ordering constraints and the other by selection functions. We combine these in the notion of *eligible literals*.

Definition 3.1.2 (Eligible literals)
Let $\mathcal{C} \subset \mathbb{K}$ be a clause, let σ be a substitution and let sel be a selection function.

- We say $\sigma(L)$ is *eligible for resolution* if either
  - $sel(C) = \emptyset$ and $\sigma(L)$ is $\succ_{\mathcal{C}}$-maximal in $\sigma(C)$ or
  - $sel(C) \neq \emptyset$ and $\sigma(L)$ is $\succeq_{\mathcal{C}}$-maximal in $\sigma(sel(C) \cap \mathcal{C}^-)$ or
  - $sel(C) \neq \emptyset$ and $\sigma(L)$ is $\succeq_{\mathcal{C}}$-maximal in $\sigma(sel(C) \cap \mathcal{C}^+)$.
- $\sigma(L)$ is *eligible for paramodulation* if L is positive, $sel(\mathcal{C}) = \emptyset$ and $\sigma(L)$ is strictly $\succ_{\mathcal{C}}$-maximal in $\sigma(\mathcal{C})$.

◂

The calculus is represented in the form of inference rules. For convenience, we distinguish two types of inference rules. For *generating* inference rules, written with a single line separating preconditions and results, the result is added to the set of all clauses. For *contracting* inference rules, written with a double line, the result clauses are substituted for the clauses in the precondition. In the following, u, v, s and t are terms, σ is a substitution and R, S and T are (partial) clauses. p is a position in a term and λ is the empty or top-position. $D \subseteq F$ is a set of unused constant predicate symbols. Different clauses are assumed to not share any common variables.

Definition 3.1.3 (The inference system SP)
Let $\succ$ be a total simplification ordering (extended to orderings $\succ_L$ and $\succ_C$ on literals and clauses), let sel be a selection function, and let D be a set of fresh propositional constants. The inference system **SP** consists of the following inference rules:

- *Equality Resolution*:

$$\text{(ER)}\ \frac{u \not\approx v \vee R}{\sigma(R)} \qquad \text{if } \sigma = mgu(u, v) \text{ and } \sigma(u \not\approx v) \text{ is eligible for resolution.}$$

- *Superposition into negative literals*:

$$\text{(SN)}\ \frac{s \simeq t \vee S \quad u \not\simeq v \vee R}{\sigma(u[p \leftarrow t] \not\simeq v \vee S \vee R)}$$

if $\sigma = mgu(u|_p, s)$, $\sigma(s) \not\preceq \sigma(t)$, $\sigma(u) \not\preceq \sigma(v)$, $\sigma(s \simeq t)$ is eligible for paramodulation, $\sigma(u \not\simeq v)$ is eligible for resolution, and $u|_p \notin V$.

- *Superposition into positive literals*:

$$\text{(SP)}\ \frac{s \simeq t \vee S \quad u \simeq v \vee R}{\sigma(u[p \leftarrow t] \simeq v \vee S \vee R)}$$

if $\sigma = mgu(u|_p, s)$, $\sigma(s) \not\preceq \sigma(t)$, $\sigma(u) \not\preceq \sigma(v)$, $\sigma(s \simeq t)$ is eligible for paramodulation, $\sigma(u \simeq v)$ is eligible for resolution, and $u|_p \notin V$.

- *Simultaneous superposition into negative literals*

$$\text{(SSN)}\ \frac{s \simeq t \vee S \quad u \not\simeq v \vee R}{\sigma(S \vee (u \not\simeq v \vee R)[u|_p \leftarrow t])}$$

if $\sigma = mgu(u|_p, s)$, $\sigma(s) \not\preceq \sigma(t)$, $\sigma(u) \not\preceq \sigma(v)$, $\sigma(s \simeq t)$ is eligible for paramodulation, $\sigma(u \not\simeq v)$ is eligible for resolution, and $u|_p \notin V$.

This inference rule is an alternative to (SN) that performs better in practice.

- *Simultaneous superposition into positive literals*

$$\text{(SSP)}\ \frac{s \simeq t \vee S \quad u \simeq v \vee R}{\sigma(S \vee (u \simeq v \vee R)[u|_p \leftarrow t])}$$

if $\sigma = mgu(u|_p, s)$, $\sigma(s) \not\preceq \sigma(t)$, $\sigma(u) \not\preceq \sigma(v)$, $\sigma(s \simeq t)$ is eligible for paramodulation, $\sigma(u \simeq v)$ is eligible for resolution, and $u|_p \notin V$.

This inference rule is an alternative to (SP) that performs better in practice.

- *Equality factoring*:

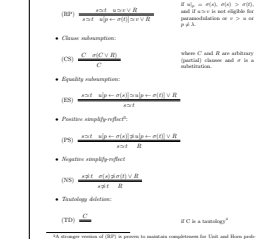
$$\text{(EF)}\ \frac{s \simeq t \vee u \simeq v \vee R}{\sigma(t \not\simeq v \vee u \simeq v \vee R)}$$

if $\sigma = mgu(s, u)$, $\sigma(t) \not\succeq \sigma(s)$ and $\sigma(s \simeq t)$ eligible for paramodulation.

- *Rewriting of negative literals*:

$$\text{(RN)}\ \frac{s \simeq t \quad u \not\simeq v \vee R}{s \simeq t \quad u[p \leftarrow \sigma(t)] \not\simeq v \vee R}$$

if $u|_p = \sigma(s)$ and $\sigma(s) > \sigma(t)$.



- *Rewriting of positive literals*[2]:

$$\text{(RP)}\ \frac{s \simeq t \quad u[p \leftarrow \sigma(t)] \simeq v \vee R}{s \simeq t \quad u[p \leftarrow \sigma(t)] \simeq v \vee R}$$

if $u|_p = \sigma(s)$, $\sigma(s) > \sigma(t)$, and if $u \simeq v$ is not eligible for paramodulation or $v > u$ or $p \neq \lambda$.

- *Clause subsumption*:

$$\text{(CS)}\ \frac{C \quad \sigma(C \vee R)}{C}$$

where C and R are arbitrary (partial) clauses and σ is a substitution.

- *Equality subsumption*:

$$\text{(ES)}\ \frac{s \simeq t \quad u[p \leftarrow \sigma(t)] \simeq u[p \leftarrow \sigma(t)] \vee R}{s \simeq t}$$

- *Positive simplify-reflect*[3]:

$$\text{(PS)}\ \frac{s \simeq t \quad u[p \leftarrow \sigma(s)] \not\simeq u[p \leftarrow \sigma(t)] \vee R}{s \simeq t \quad R}$$

- *Negative simplify-reflect*

$$\text{(NS)}\ \frac{s \not\simeq t \quad \sigma(s) \not\simeq \sigma(t) \vee R}{s \not\simeq t \quad R}$$

- *Tautology deletion*:

$$\text{(TD)}\ \frac{C}{}$$

if C is a tautology[4]

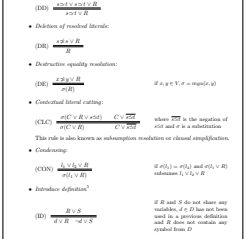
[2]A stronger version of (RP) is proven to maintain completeness for Unit and Horn problems and is generally believed to maintain completeness for the general case as well [Bac94]. However, the proof of completeness for the general case seems to be rather involved, as it requires a very different clause ordering than the one introduced [BG94], and we are not aware of any existing proof in the literature. The variant rule allows rewriting of maximal terms of maximal literals under certain circumstances:

$$\text{(RP}^+\text{)}\ \frac{s \simeq t \quad u[v \simeq v \vee R}{s \simeq t \quad u[p \leftarrow \sigma(t)] \simeq v \vee R}$$

if $u|_p = \sigma(s)$, $\sigma(s) > \sigma(t)$ and if $u \simeq v$ is not eligible for paramodulation or $u \not\simeq v$ or $p \neq \lambda$ or v is not a variable remaining.

This stronger rule is implemented successfully by both E and SPASS [Wei09].

[3]In practice, this rule is only applied if $\sigma(s)$ and $\sigma(t)$ are $\succeq$-incomparable – in all other cases this rule is subsumed by (RN) and the deletion of resolved literals (DR).



- *Deletion of duplicate literals*:

$$\text{(DD)}\ \frac{s \simeq t \vee s \simeq t \vee R}{s \simeq t \vee R}$$

- *Deletion of resolved literals*:

$$\text{(DR)}\ \frac{s \not\simeq s \vee R}{R}$$

- *Destructive equality resolution*:

$$\text{(DE)}\ \frac{x \not\simeq y \vee R}{\sigma(R)} \qquad \text{if } x, y \in V, \sigma = mgu(x, y)$$

- *Contextual literal cutting*:

$$\text{(CLC)}\ \frac{\sigma(C \vee R \vee s \simeq t) \qquad C \vee \overline{s \simeq t}}{\sigma(C \vee R)} \qquad \text{where } \overline{s \simeq t} \text{ is the negation of } s \simeq t \text{ and } \sigma \text{ is a substitution}$$

This rule is also known as *subsumption resolution* or *clausal simplification*.

- *Condensing*:

$$\text{(CON)}\ \frac{l_i \vee l_k \vee R}{\sigma(l_i \vee R)} \qquad \text{if } \sigma(l_i) = \sigma(l_k) \text{ and } \sigma(l_i \vee R) \text{ subsumes } l_i \vee l_k \vee R$$

- *Introduce definition*[5]

$$\text{(ID)}\ \frac{R \vee S}{d \vee R \quad \neg d \vee S}$$

if R and S do not share any variables, $d \in D$ has not been used in a previous definition and R does not contain any symbol from D

- *Apply definition*

$$\text{(AD)}\ \frac{\sigma(d \vee R) \quad R \vee S}{\sigma(d \vee R) \quad \neg d \vee S}$$

if σ is a variable renaming, R and S do not share any variables, $d \in D$ and R does not contain any symbol from D

[5]This rule can only be implemented approximately, as the problem of recognizing tautologies is only semi-decidable in equational logic. Current versions of E try to detect tautologies by checking if $s \simeq t$ is subsumed by completed negative literals imply at least one of the positive literals, as suggested in [SN04].

[5]This rule is always exhaustively applied to any clause, leaving a split-off clauses and one final link clause of all negative propositions.

Proof reconstruction

Use information from the proof trace to guide proof building

Inspired by Sledgehammer and PRocH

Two approaches:

- premises selector
- trace steps reconstruction

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

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:

	Zenon modulo (alone)	E prover	Premises selection + Zenon modulo
#Problems solved	2274	8901	3249
%	18.2	71.3	26.0

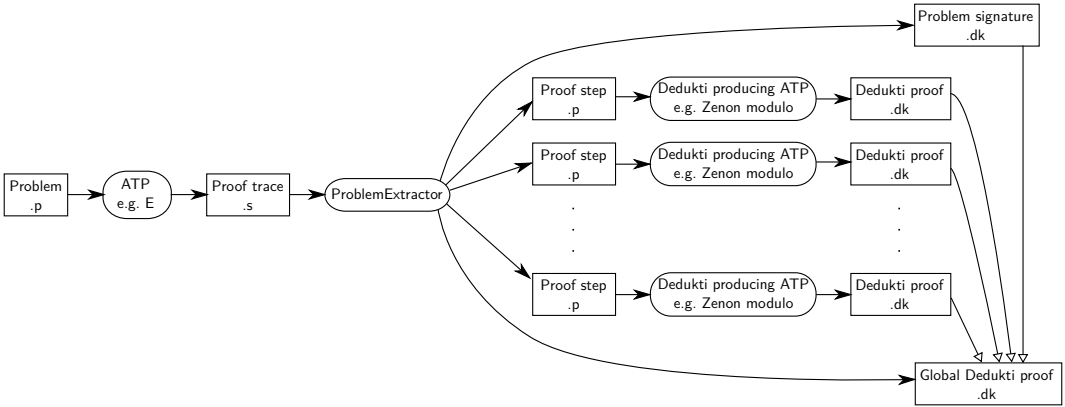
Proof step reconstruction

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

Architecture



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 $\Rightarrow$ Γ, C has a model

Provability preservation:

$\Gamma, C \vdash \bot$ has a proof $\Rightarrow$ $\Gamma \vdash \bot$ has a proof

1. Start from $\overline{\Gamma, \bot \vdash \bot}$
2. Transform proof until $Axioms \vdash \bot$

RAT criterion leads to a 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

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