Translating proofs from automated theorem provers to Logipedia

First Logipedia meeting

Guillaume Burel\textsuperscript{1,2}

Wednesday January 23rd, 2019

\textsuperscript{1}Samovar, ENSIIE, Université Paris Saclay

\textsuperscript{2}Inria and LSV, CNRS and ENS Paris Saclay, Université Paris Saclay
Why could Logipedia be interested in Automated Theorem Provers?

- Import proofs from databases of problems
  - TPTP yes, at least problems produced by humans
  - Proof obligations from program verification probably not
- Helping proof assistants
  - automated Logipedia tactic
- Transfer
  - cannot automated everything
  - but can definitively help (see [Cauderlier 17])
Families of automated theorem provers

- **SAT solvers**
  - propositional logic

- **SMT solvers**
  - combining a SAT solver with decision procedure for particular theories

- **FO theorem provers**
  - many based on resolution/superposition

- **HO provers**
  - TH1 of TPTP $\simeq$ classical STT\forall

Most of them are for classical logic
Output of ATPs

- proof term
  - the ATP produces directly a Dedukti file
  - Zenon modulo, iProverModulo, ArchSat

- proof script
  - tree (DAG) of formulas;
  - each formula is a logical consequence of its parents
  - TSTP format (partially), DRUP format

- proof trace
  - evolving set of formulas
  - satisfiability is preserved
  - TSTP format (Skolemization, splitting), DRAT format
Resolution proofs

\[
\begin{align*}
\text{Res.} & \quad \frac{P \lor C \quad \neg Q \lor D}{\sigma(C \lor D)} \quad \sigma = \text{mgu}(P,Q) \\
\text{Fact.} & \quad \frac{P \lor Q \lor D}{\sigma(P \lor D)} \quad \sigma = \text{mgu}(P,Q)
\end{align*}
\]

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

```haskell
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

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

- **Definition of clause literals**: Let \( C \in \mathcal{E} \) be a clause, let \( \mathcal{E} \) be a substitution, and let \( l \subseteq \mathcal{E} \) be a selection function.
  - \( \mathcal{E} \) is a total simplification ordering.
  - The calculus is represented in the form of inference rules. For convenience, we choose \( \mathcal{E} \) to be a substitution.

- **Definition of clause literals**: Let \( C \in \mathcal{E} \) be a clause, let \( \mathcal{E} \) be a substitution, and let \( l \subseteq \mathcal{E} \) be a selection function.
  - \( \mathcal{E} \) is a total simplification ordering.
  - The calculus is represented in the form of inference rules. For convenience, we choose \( \mathcal{E} \) to be a substitution.

- **Definition of clause literals**: Let \( C \in \mathcal{E} \) be a clause, let \( \mathcal{E} \) be a substitution, and let \( l \subseteq \mathcal{E} \) be a selection function.
  - \( \mathcal{E} \) is a total simplification ordering.
  - The calculus is represented in the form of inference rules. For convenience, we choose \( \mathcal{E} \) to be a substitution.

- **Definition of clause literals**: Let \( C \in \mathcal{E} \) be a clause, let \( \mathcal{E} \) be a substitution, and let \( l \subseteq \mathcal{E} \) be a selection function.
  - \( \mathcal{E} \) is a total simplification ordering.
  - The calculus is represented in the form of inference rules. For convenience, we choose \( \mathcal{E} \) to be a substitution.
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:

<table>
<thead>
<tr>
<th></th>
<th>Zenon modulo (alone)</th>
<th>E prover</th>
<th>Premises selection + Zenon modulo</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Problems solved</td>
<td>2274</td>
<td>8901</td>
<td>3249</td>
</tr>
<tr>
<td>%</td>
<td>18.2</td>
<td>71.3</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
Architecture
Remark

The structure of the original trace is kept in the global Dedukti proof:

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])).

let c_0_18 : ... = ...
let c_0_30 : ... = ...

...

let c_0_60 : P (eq(join(X1,join(X2,X3)), join(X2,join(X1,X3)))) =
    c_0_40.goal c_0_30 c_0_18 c_0_30 in x...
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:
\( \Gamma \) has a model \( \Rightarrow \) \( \Gamma, C \) has a model

Provability preservation:
\( \Gamma, C \vdash \bot \) has a proof \( \Rightarrow \) \( \Gamma \vdash \bot \) has a proof

1. Start from \( \Gamma, \bot \vdash \bot \)
2. Transform proof until \( Axioms \vdash \bot \)

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)
Questions

1. Constructivization

2. Which automatically found proofs do we want in Logipedia?

3. Is it possible to present them so that export out of Logipedia look nice?

4. How much can ATPs help in concept alignment?
   - see also nitpick