How to handle systems using automated theorem provers?

1st Dedukti School

Guillaume Burel

Saturday June 25th, 2022

Samovar, ENSIIE

Proof assistants and ATP

Limitation of proof assistants

- lack of automation
- need for specially trained experts
- bottleneck for widespread use

Limitation of automated theorem provers

- lack of confidence
- highly optimized tools
- code too complex to be certified

Cooperation

Proof assistants:

- use ATPs to discharge some obligations
  - *e.g.* Sledgehammer, SMTCoq, ...

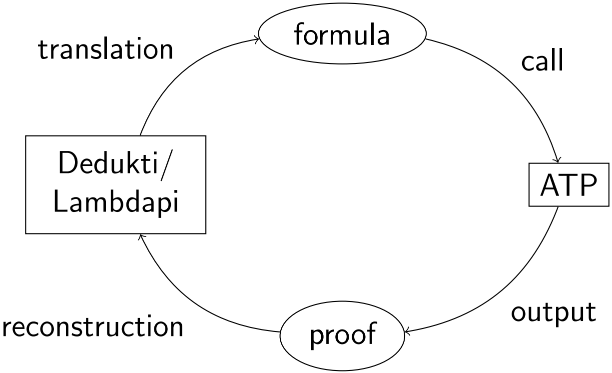
ATPs:

- Export proofs that can be independently checked
- Ideally, checkable by a well known tool

Dedukti

Dedukti as a pivot for proof interoperability
Export from/to ATPs should pass by Dedukti

Ideal goal


formula
translation
call
Dedukti/
Lambdapi
ATP
output
reconstruction
proof

From Lambdapi to ATPs

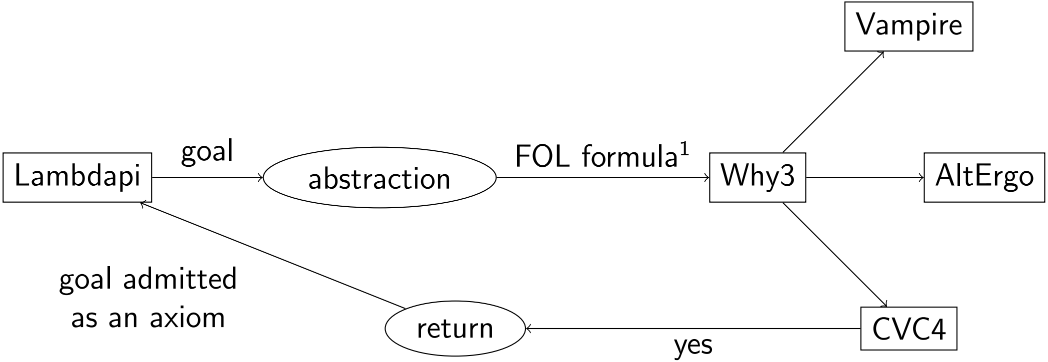
Why3:

- platform for deductive program verification
- able to delegate proofs to many provers
- `https://why3.lri.fr/`

Calling provers within Lambdapi:

- Tactic `why3`

Why3 tactic



[1] Actually, propositional logic for now

Outline

- Introduction
- Intrumenting provers for Dedukti proof production
  - iProverModulo
  - Zenon Modulo
- Reconstructing proofs
- Conclusion

Trusting automated theorem provers

Automated theorem provers:

- quite big piece of software
- complex proof calculi
- finely tuned, optimization hacks

Trust?

- Originally, only answer "yes" / "no" (more often, "maybe")
- More and more, produce at least proof traces (*i.e.* big steps)

Trusting ATPs

To increase confidence:

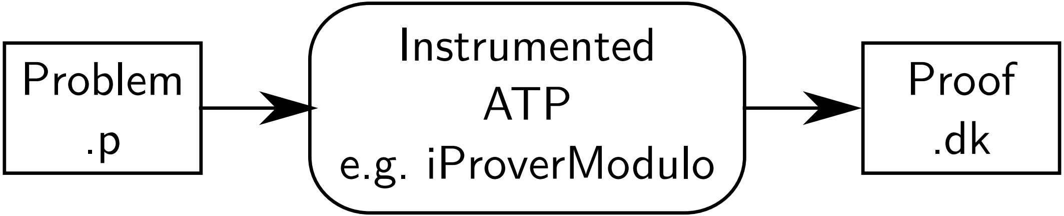
- either build a certified proof checker for proof traces
    - e.g. Coq certified proof checker for DRAT proof traces of SAT solvers
- or directly produce a proof checkable by your favorite assistant

Trusting ATPs

To increase confidence:

- either build a certified proof checker for proof traces
  - e.g. Coq certified proof checker for DRAT proof traces of SAT solvers
- or directly produce a proof checkable by your favorite assistant

Instrumenting a prover to produce a proof



Pros:

- Access to all needed informations

Cons:

- Needs to embed the calculus of the prover into Dedukti
- Needs to know precisely the code of the prover

- more or less easy depending on the complexity of the code/the proof calculus
- easier if a proof output was designed from the start (e.g. in Zenon)

Can only be done for a few provers

Provers outputing Dedukti proofs

iProverModulo: extension of iProver to handle Deduction Modulo Theory
https://github.com/gburel/iProverModulo.git

Zenon Modulo: extension of Zenon to handle Deduction Modulo Theory and arithmetic
https://github.com/Deducteam/zenon_modulo.git

ArchSAT: SMT solver
https://github.com/Gbury/archsat

Translating proofs

First, need to carefully choose in which theory we are working

- e.g. D[FOL]

Then, two approaches:

- Directly translating proofs into Dedukti
  - iProverModulo
- Embedding the proof calculus into Dedukti
  - Zenon Modulo

iProverModulo

[Burel 2011]

Patch to iProver [Korovin 2008]

iProver: Combination of two proof procedures:

- Inst-Gen (not relevant for us)
- Ordered resolution

iProverModulo: Add support of Deduction Modulo Theory

Resolution Calculus

Clause: set of literals (atoms or negation of atoms)
Derive new clauses using

$$\text{Resolution}\ \frac{P;C \qquad \neg Q;D}{\sigma(C;D)}\ \sigma = mgu(P,Q)$$

until the empty clause is produced

Representation of clauses

$\{L_1; \cdots ; L_m\}$ corresponds to $\forall X_1. \ \ldots \forall X_n.\ L_1 \vee \cdots \vee L_m$
$(X_1, \ldots, X_n$ free variables of $L_1, \ldots, L_m)$

$\{L_1; \cdots ; L_m\}$ translated as

$$\Pi X_1 : \texttt{El}\ \iota.\ \ldots \Pi X_n : \texttt{El}\ \iota.\ \Pi\ \flat : \texttt{Prop}.\ ||L_1||_\flat \to \cdots \to ||L_m||_\flat \to \texttt{Prf}\ \flat$$

with $||P||_\flat = \texttt{Prf}\ ||P|| \to \texttt{Prf}\ \flat$ and $||\neg P||_\flat = (\texttt{Prf}||P|| \to \texttt{Prf}\ \flat) \to \texttt{Prf}\ \flat$

$\texttt{Prf}\ ||\forall X_1.\ \ldots \forall X_n.\ L_1 \vee \cdots \vee L_m||$ implies
$\Pi X_1 : \texttt{El}\ \iota.\ \ldots \Pi X_n : \texttt{El}\ \iota.\ \Pi\ \flat : \texttt{Prop}.\ ||L_1||_\flat \to \cdots \to ||L_m||_\flat \to \texttt{Prf}\ \flat$

Translation of resolution

$$\text{Resolution}\ \frac{P; Q \qquad R; \neg P}{Q; R}$$

$$\begin{aligned}
c_1 &: \Pi\, \flat : \texttt{Prop}.\ (P \to \texttt{Prf}\ \flat) \to (Q \to \texttt{Prf}\ \flat) \to \texttt{Prf}\ \flat \\
c_2 &: \Pi\, \flat : \texttt{Prop}.\ (R \to \texttt{Prf}\ \flat) \to ((P \to \texttt{Prf}\ \flat) \to \texttt{Prf}\ \flat) \to \texttt{Prf}\ \flat \\
d &: \Pi\, \flat : \texttt{Prop}.\ (Q \to \texttt{Prf}\ \flat) \to (R \to \texttt{Prf}\ \flat) \to \texttt{Prf}\ \flat \\
&:= \lambda \flat.\ \lambda q.\ \lambda r. \\
&\qquad c_1\ \flat\ (\lambda tp : P.\ c_2\ \flat\ r\ (\lambda tnp : (P \to \texttt{Prf}\ \flat).\ tnp\ tp))\ q
\end{aligned}$$

Limits

Can handle various simplification rules, rewriting

Can be extended to superposition (E, Vampire, . . .)

But:

- works only if the proof is found using only resolution (i.e. not Inst-Gen)
- no translation of the transformation into clauses

Zenon Modulo

[Delahaye, Doligez, Gilbert, Halmagrand, and Hermant 2013]

- extension of Zenon to Deduction Modulo Theory
- tableau-based
- polymorphic first-order logic with equality

Tableau proofs

Proofs by contradiction
$\simeq$ bottom-up sequent-calculus with metavariables

$$\frac{P, \neg P}{\odot}\ \odot \qquad \frac{\neg(A \Rightarrow B)}{\neg A, B}\ \alpha_{\neg\Rightarrow} \qquad \frac{\neg(A \wedge B)}{\neg A \quad \mid \quad \neg B}\ \beta_{\neg\wedge}$$

Example, proof by refutation of $P \Rightarrow (P \wedge P)$:

$$\cfrac{\neg(P \Rightarrow (P \wedge P))}{\cfrac{P}{\cfrac{\neg(P \wedge P)}{\cfrac{\neg P}{\odot}\ \odot \qquad \cfrac{\neg P}{\odot}\ \odot}\ \beta_{\neg\wedge}}}\ \alpha_{\neg\Rightarrow}$$

Deep embedding of proof calculus

$$\frac{P, \neg P}{\odot}\ \odot :$$

```
symbol Rax p : Prf p → Prf (¬p) → Prf ⊥;
```

$$\frac{\neg(A \Rightarrow B)}{\neg A, B}\ \alpha_{\neg\Rightarrow} :$$

```
symbol R¬⇒ a b : (Prf a → Prf (¬b) → Prf ⊥) → Prf (¬(a ⇒ b)) → Prf ⊥;
```

$$\frac{\neg(A \wedge B)}{\neg A \quad | \quad \neg B}\ \beta_{\neg\wedge} :$$

```
symbol R¬∧ a b : (Prf (¬ a) → Prf ⊥) → (Prf (¬ b) → Prf ⊥) →
                    Prf (¬ (a ∧ b)) → Prf ⊥;
```

Deep translation of the example

(after η-reduction to make it more readable)

```
opaque symbol goal : Prf ᶜ (p ⇒ (p ∧ p)) ≔
  R¬⇒ p (p ∧ p)
    (λ π, R¬∧ p p (Rax p π) (Rax p π));
```

Making the embedding more shallow

Reducing it to Natural Deduction

$$\wedge\text{-e}_l\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad \wedge\text{-e}_r\ \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \qquad \wedge\text{-i}\ \frac{A \qquad B}{A \wedge B}$$

$$\Rightarrow\text{-e}\ \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \qquad \Rightarrow\text{-i}\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$$

Natural Deduction in LambdaPi:

```
symbol ∧I p q : Prf p → Prf q → Prf (p ∧ q);
symbol ∧El p q : Prf (p ∧ q) → Prf p;
symbol ∧Er p q : Prf (p ∧ q) → Prf q;

symbol ⇒I p q : (Prf p → Prf q) → Prf (p ⇒ q);
symbol ⇒E p q : Prf (p ⇒ q) → Prf p → Prf q;
```

Defining Tableau rules in term of ND:

```
rule Rax ↪ λ p h π, ¬E p π h;
rule R¬∧ ↪ λ p q h1 h2 h3,
  h1 (¬I p (λ h5, h2 (¬I q (λ h6,
    ¬E (p ∧ q) h3 (∧I p q h5 h6))))));
rule R¬⇒ ↪ λ p q h1 h2,
  ¬E (p ⇒ q) h2 (⇒I p q (λ h3, ⊥E (h1 h3
    (¬I q (λ h4, ¬E (p ⇒ q) h2 (⇒I p q (λ _, h4))))) q));
```

Proof that Tableaux rules are derivable in ND

Example in natural deduction

```
assert ⊢ goal : Prf^c (p ⇒ (p ∧ p));
assert ⊢ goal ≡ λ h2, ¬E (p ⇒ (p ∧ p)) h2 (⇒I p (p ∧ p)
  (λ h3, ⊥E (¬E (p ⇒ (p ∧ p)) h2
    (⇒I p (p ∧ p) (λ _, ∧I p p h3 h3))) (p ∧ p)));
```

Making it even more shallow

Reduce Natural Deduction thanks to the shallow encoding of FOL

```
rule ⇒I ↪ λ p q π, π;
rule ⇒E ↪ λ p q π, π;

rule ∧I ↪ λ p q πp πq r πp⇒q⇒r, πp⇒q⇒r πp πq;
rule ∧El ↪ λ p q πp∧q, πp∧q p (λ x _, x);
rule ∧Er ↪ λ p q πp∧q, πp∧q q (λ _ x, x);
```

Shallow proof from the example

```
assert ⊢ goal : Prfᶜ (p ⇒ (p ∧ p));
assert ⊢ goal ≡
  λ h2, h2 (λ h3, h2 (λ _ _ π, π h3 h3) (p ∧ p));
```

Outline

- Introduction
- Intrumenting provers for Dedukti proof production
- Reconstructing proofs
- Conclusion

Limits of instrumentation

Provers can be hard to instrument to produce exact Dedukti proofs

- large piece of software
- developers not expert in $\lambda\Pi$-calculus modulo theory
- non stable and quite big 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 $\mathcal{L}$ is *selected* (with respect to a given selection function) in a clause $\mathcal{C}$ if $\mathcal{L} \in sel(\mathcal{C})$. ◀

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 $\mathcal{C} = \mathcal{L} \vee \mathcal{R}$ be a clause, let σ be a substitution and let sel be a selection function.

- We say $\sigma(\mathcal{L})$ is *eligible for resolution* if either
  - $sel(\mathcal{C}) = \emptyset$ and $\sigma(\mathcal{L})$ is $>_L$-maximal in $\sigma(\mathcal{C})$ or
  - $sel(\mathcal{C}) \neq \emptyset$ and $\sigma(\mathcal{L})$ is $>_L$-maximal in $\sigma(sel(\mathcal{C}) \cap \mathcal{C}^-)$ or
  - $sel(\mathcal{C}) \neq \emptyset$ and $\sigma(\mathcal{L})$ is $>_L$-maximal in $\sigma(sel(\mathcal{C}) \cap \mathcal{C}^+)$.
- $\sigma(\mathcal{L})$ is *eligible for paramodulation* if $\mathcal{L}$ is positive, $sel(\mathcal{C}) = \emptyset$ and $\sigma(\mathcal{L})$ is strictly $>_L$-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 $>$ be a total simplification ordering (extended to orderings $>_L$ and $>_C$ 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*:

(ER) $\dfrac{u \not\simeq v \vee R}{\sigma(R)}$ if $\sigma = mgu(u, v)$ and $\sigma(u \not\simeq v)$ is eligible for resolution.

8

- *Superposition into negative literals*:

(SN) $\dfrac{s \simeq t \vee S \quad u \not\simeq v \vee R}{\sigma(u[p \leftarrow t] \not\simeq v \vee S \vee R)}$ if $\sigma = mgu(u|_p, s)$, $\sigma(s) \not< \sigma(t)$, $\sigma(u) \not< \sigma(v)$, $\sigma(s \simeq t)$ is eligible for paramodulation, $\sigma(u \not\simeq v)$ is eligible for resolution, and $u|_p \notin V$.

- *Superposition into positive literals*:

(SP) $\dfrac{s \simeq t \vee S \quad u \simeq v \vee R}{\sigma(u[p \leftarrow t] \simeq v \vee S \vee R)}$ if $\sigma = mgu(u|_p, s)$, $\sigma(s) \not< \sigma(t)$, $\sigma(u) \not< \sigma(v)$, $\sigma(s \simeq t)$ is eligible for paramodulation, $\sigma(u \simeq v)$ is eligible for resolution, and $u|_p \notin V$.

- *Simultaneous superposition into negative literals*

(SSN) $\dfrac{s \simeq t \vee S \quad u \not\simeq v \vee R}{\sigma(S \vee (u \not\simeq v \vee R)[u|_p \leftarrow t])}$ if $\sigma = mgu(u|_p, s)$, $\sigma(s) \not< \sigma(t)$, $\sigma(u) \not< \sigma(v)$, $\sigma(s \simeq t)$ 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*

(SSP) $\dfrac{s \simeq t \vee S \quad u \simeq v \vee R}{\sigma(S \vee (u \simeq v \vee R)[u|_p \leftarrow t])}$ if $\sigma = mgu(u|_p, s)$, $\sigma(s) \not< \sigma(t)$, $\sigma(u) \not< \sigma(v)$, $\sigma(s \simeq t)$ 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 (SP) that performs better in practice.

- *Equality factoring*:

(EF) $\dfrac{s \simeq t \vee u \simeq v \vee R}{\sigma(t \not\simeq v \vee u \simeq v \vee R)}$ if $\sigma = mgu(s, u)$, $\sigma(t) \not> \sigma(s)$ and $\sigma(s \simeq t)$ eligible for paramodulation.

- *Rewriting of negative literals*:

(RN) $\dfrac{s \simeq t \quad u \not\simeq v \vee R}{s \simeq t \quad u[p \leftarrow \sigma(t)] \not\simeq v \vee R}$ if $u|_p = \sigma(s)$ and $\sigma(s) > \sigma(t)$.

9

- *Rewriting of positive literals*[2]:

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

(CS) $\dfrac{C \quad \sigma(C \vee R)}{C}$ where C and R are arbitrary (partial) clauses and σ is a substitution.

- *Equality subsumption*:

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

- *Positive simplify-reflect*[3]:

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

- *Negative simplify-reflect*

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

- *Tautology deletion*:

(TD) $\dfrac{C}{}$ 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 [Bac98]. 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:

(RP') $\dfrac{s \simeq t \quad u \simeq v \vee R}{s \simeq t \quad u[p \leftarrow \sigma(t)] \simeq v \vee R}$ if $u|_p = \sigma(s)$, $\sigma(s) > \sigma(t)$ and if $u \simeq v$ is not eligible for paramdulation or $u \not> v$ or $p \neq \lambda$ or σ is not a variable renaming.

This stronger rule is implemented successfully by both E and SPASS [Wei99].

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

10

- *Deletion of duplicate literals*:

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

- *Deletion of resolved literals*:

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

- *Destructive equality resolution*:

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

- *Contextual literal cutting*:

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

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

- *Condensing*:

(CON) $\dfrac{l_1 \vee l_2 \vee R}{\sigma(l_1 \vee R)}$ if $\sigma(l_1) = \sigma(l_2)$ and $\sigma(l_1 \vee R)$ subsumes $l_1 \vee l_2 \vee R$

- *Introduce definition*[5]

(ID) $\dfrac{R \vee S}{d \vee R \quad \neg d \vee S}$ 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*

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

[4]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 the ground-completed negative literals imply at least one of the positive literals, as suggested in [NN93].

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

11

Proof trace

But often, provers produce at least a proof trace:

- list of formulas that were derived to obtain the proof
- sometimes with more informations
  - premises
  - name of the inference rules
  - theory
  - ...

Example of trace: TSTP format

Output format of E, Vampire, Zipperposition, ...

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])).
```

Example of trace: TSTP format

Output format of E, Vampire, Zipperposition, ...

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 reconstruction

Use the content of the proof trace to reconstruct a Dedukti proof

Idea:

- Reprove each step using a Dedukti producing tool
- Combine the proofs of the steps to get a proof of the original formula

Try to be agnostic:

- w.r.t. the prover that produces the trace
- w.r.t. the prover that reprove the steps

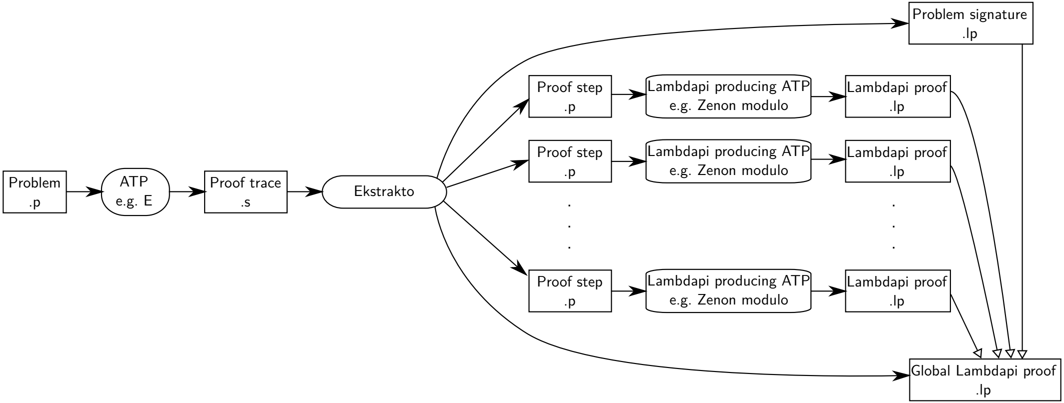
Ekstrakto

[El Haddad 2021]

- Input: TSTP proof trace
- Output: Reconstructed Lambdapi proof

`https://github.com/Deducteam/ekstrakto`

Ekstrakto architecture



Experimental evaluation

Benchmark:

- CNF problems of TPTP v7.4.0 (8118 files)

Trace producers:

- E and Vampire

Step provers:

- Zenon modulo and ArchSat

Results

Percentage of Lambdapi proofs on the extracted TPTP files

Prover	% **E**	% VAMPIRE
ZenonModulo	87%	60%
ArchSAT	92%	81%
ZenonModulo ∪ *ArchSAT*	95%	85%

Percentage of complete Lambdapi proofs

Prover	% **E** TSTP	% VAMPIRE TSTP
ZenonModulo	45%	54%
ArchSAT	56%	74%
ZenonModulo ∪ *ArchSAT*	69%	83%

Non provable steps

Problem:

- some steps are not provable
  their conclusion is not a logical consequence of their premises
- OK because they preserve provability
- but Ekstrakto cannot work for them

Non provable steps

Problem:

- some steps are not provable
  their conclusion is not a logical consequence of their premises
- OK because they preserve provability
- but Ekstrakto cannot work for them

Main instance: Skolemization

$$\Gamma, \forall\vec{x}, \exists y, A[\vec{x}, y] \vdash B \text{ iff } \Gamma, \forall\vec{x}, A[\vec{x}, f(\vec{x})] \vdash B \text{ for a fresh } f$$

Present in the CNF transformation used by almost all ATPs

Skonverto

[El Haddad 2021]

Inputs:

- an axiom and its Skolemized version
- a Lambdapi proof using the latter

Output:

- a Lambdapi proof using the non-Skolemized axiom

Content

Implementation of a constructive proof of Skolem theorem by [Dowek and Werner 2005]

- in the context of first-order natural deduction

Problem:

- the proof assumes that proofs are in normal form
- also w.r.t. so-called commuting cuts

Commuting cuts

$$\dfrac{\dfrac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \wedge D \qquad \Gamma, B \vdash C \wedge D}{\Gamma \vdash C \wedge D}\vee_E}{\Gamma \vdash C}\wedge_{El}$$

$$\rightsquigarrow$$

$$\dfrac{\Gamma \vdash A \vee B \qquad \dfrac{\Gamma, A \vdash C \wedge D}{\Gamma, A \vdash C}\wedge_{El} \qquad \dfrac{\Gamma, B \vdash C \wedge D}{\Gamma, B \vdash C}\wedge_{El}}{\Gamma \vdash C}\vee_E$$

Reducing commuting cuts

If we work on shallow proofs, these cuts are no longer visible

- cannot reduce them

(On the other hand, regular cuts are embedded into β-redexes, so they are reduced.)

- Needs to stay at the ND encoding level

Add rules to reduce the commuting cuts

```
rule ∧El $c $d (∨E $a $b $paorb ($c ∧ $d) $pac $pbc) ↪
    ∨E $a $b $paorb $c (λ pa, ∧El $c $d ($pac pa))
        (λ pb, ∧El $c $d ($pbc pb));
```

```
symbol axiom : Prf (∀ (λ X, ∃ (λ Y, (p X (s Y)))));

symbol goal
  (ax_tran : Prf (∀ (λ X1 : El ι, ∀ (λ X2 : El ι, ∀ (λ X3 : El
     (p X1 X2) ⇒ ((p X2 X3) ⇒ (p X1 X3))))))))
  (ax_step : Prf (∀ (λ X1 : El ι, (p X1 (s (f X1))))))
  (ax_congr : Prf (∀ (λ X1 : El ι, ∀ (λ X2 : El ι,
     (p X1 X2) ⇒ (p (s X1) (s X2)))))))
  (ax_goal : Prf (¬ (∃ (λ X4 : El ι, ((p a (s (s X4))))))))
  : Prf ⊥
 ≔  ax_goal  (∃I (λ X4 : El ι, p a (s (s X4))) (f (f a))
   (ax_tran a (s (f a)) (s (s (f (f a))))
     (ax_step a)
     (ax_congr (f a) (s (f (f a))) (ax_step (f a)))));
```

```
symbol goal
  (ax_tran : Prf (∀ (λ X1 : El ι, ∀ (λ X2 : El ι, ∀ (λ X3 : El
     (p X1 X2) ⇒ ((p X2 X3) ⇒ (p X1 X3)))))))
  (ax_step : Prf (∀ (λ X, ∃ (λ Y, (p X (s Y))))))
  (ax_congr : Prf (∀ (λ X1 : El ι, ∀ (λ X2 : El ι,
     (p X1 X2) ⇒ (p (s X1) (s X2))))))
  (ax_goal : Prf (¬ (∃ (λ X4 : El ι, ((p a (s (s X4))))))))
  : Prf ⊥
≔ ax_goal (λ r h, ∃E (λ z, p a (s z)) (ax_step a) r
         (λ z a1, ∃E (λ z0, p z (s z0)) (ax_step z) r
         (λ z0 a2, h z0 (ax_tran a (s z) (s (s z0)) a1
             (ax_congr z (s z0) a2)))));
```

Outline

Conclusion

Instrumenting a prover to produce Dedukti proofs

- good if you start your prover from scratch

Reconstructing proofs

- more adapted for existing provers
- cannot reconstruct all proofs
- also for proof assistants
    - PVS, Atelier B

Linking together