Representation of automated proofs in lambdapi, using the case of GDV-LP

Deducteam Meeting

Guillaume Burel

Friday September 27th, 2024

Samovar, ENSIIE

In this talk

- Presentation of GDV-LP [Sutcliffe, Blanqui, Burel]
- Some reflections about what lambdapi proofs should automated theorem provers produce
- Some words on Skolemization

TSTP Proof Format

Generic format to express proof from automated theorem provers

List of declaration of formulæ:

fof(*name* , *role* , *formula* , *source*).

role: axiom, plain, conjecture, negated_conjecture, ...

source: file(*file*, *name*)

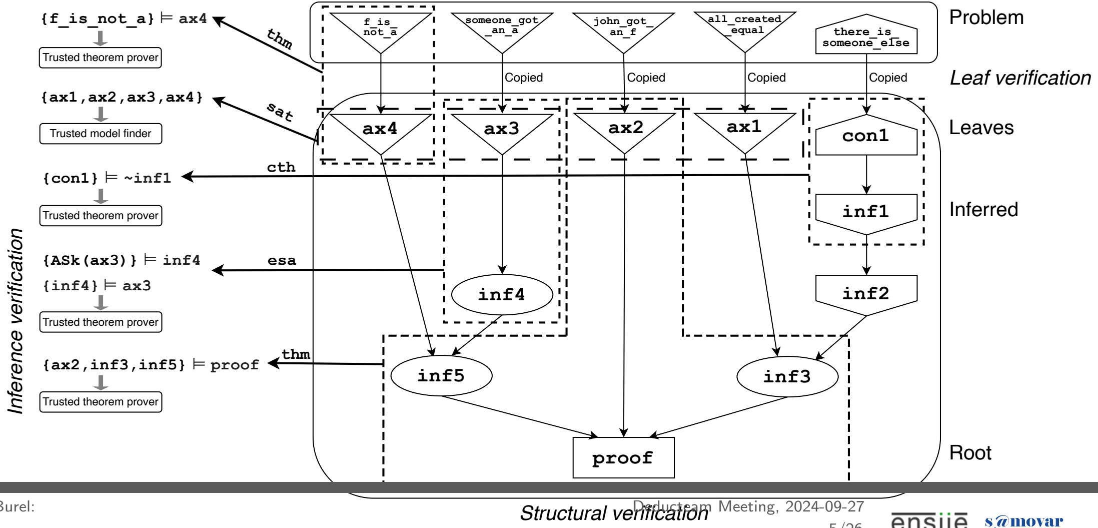
name

inference(*name*, [status(*status*)], *list of sources*)

Hopefully, represent the DAG of the inferred formulæ

```
fof(c_0_0, axiom, (?[X1]:p(f(X1))), file('input.p', hyp)).
fof(c_0_1, conjecture, (?[X1]:p(X1)), file('input.p', goal)).
fof(c_0_3, negated_conjecture, (~(?[X1]:p(X1))),
    inference(assume_negation,[status(cth)],[c_0_1])).
fof(c_0_4, plain, (p(f(esk1_0))),
    inference(skolemize,[status(esa)],[
        inference(variable_rename,[status(thm)],[c_0_0])])).
fof(c_0_5, negated_conjecture, (![X2]:~p(X2)),
    inference(variable_rename,[status(thm)],[
        inference(fof_nnf,[status(thm)],[c_0_3])])).
cnf(c_0_6, plain, (p(f(esk1_0))),
    inference(split_conjunct,[status(thm)],[c_0_4])).
cnf(c_0_7, negated_conjecture, (~p(X1)),
    inference(split_conjunct,[status(thm)],[c_0_5])).
cnf(c_0_12, plain, ($false),
    inference(sr,[status(thm)],
        [c_0_6, c_0_7, theory(equality)]), ['proof']).
```

GDV Architecture



From GDV to GDV-LP

Original GDV:

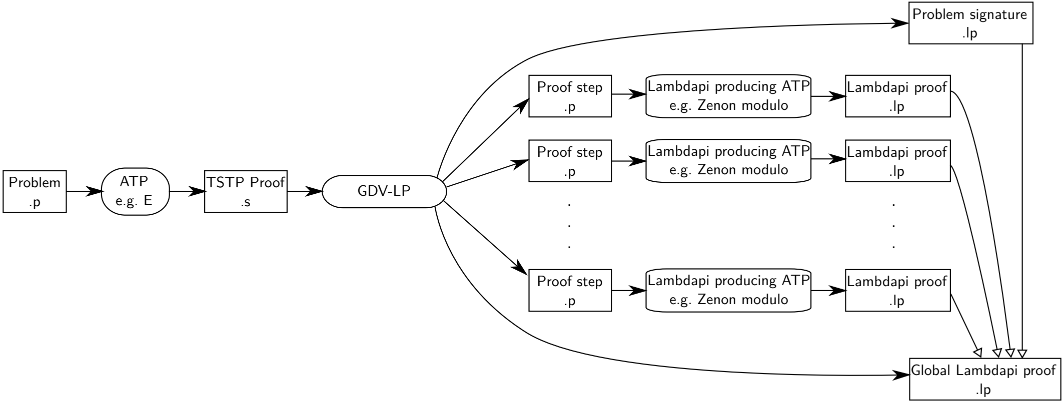
- uses Otter as a trusted verifier
- old automated theorem prover
- stable code, thoroughly tested

GDV-LP:

- uses lambdapi to check steps
- lambdapi proofs are produced by Zenon Modulo

Obtaining a global proof

Combine proof of each steps to get a lambdapi proof of the original problem



Interoperability

We would like to use any ATP instead of Zenon Modulo provided it produces lambdapi proofs

Problem:

- there is no consensus on what lambdapi proofs of ATP should be

- Theory for embedding logics?
- Axioms?
- Conjecture in proof by refutation?

Theory

Each provers comes with its own embedding of FOF in Dedukti/Lambdapi

Not really a problem because it is mostly the same theory, but with different symbol names

Use theory U?

- lacks built-in equality

Problem presentation

Axioms (hypothesis, assumptions, definitions, . . .) $A_1, \ldots, A_n$

Conjecture (conclusion, goal, . . .) C

What should Dedukti prove?

```
symbol proof: Prf A1 → ... → Prf An → Prf C ≔ ...
```

or

```
constant symbol a1: Prf A1;
...
constant symbol an: Prf An;
symbol c: Prf C ≔ ...;
```

λ-lifting

Proof presentation

Often, ATP proofs are sequences of inferred formulas

- resolution proofs
- TSTP files

with extra info to form a DAG
(premises used to infer the formula)

Should the Dedukti proof be a term representing a translation of the whole DAG?

Or should we add new symbols for each of the inferred formulas, and define them in term of other ones?

Example

Axioms

$a_1 : A \qquad a_2 : A \Rightarrow B \qquad a_3 : B \Rightarrow B \Rightarrow C$

Conjecture

$c : C$

Inferred formulas

$i_1 : B$ from a_2 and a_1

$i_2 : C$ from a_3, i_1 and i_1

$$\dfrac{A \Rightarrow B \Rightarrow C \qquad \dfrac{A \Rightarrow B \qquad A}{B} \qquad \dfrac{A \Rightarrow B \qquad A}{B}}{C}$$

In lambdapi, whole DAG

```
constant symbol a1: Prf A;
constant symbol a2: Prf A → Prf B;
constant symbol a3: Prf B → Prf B → Prf C;
symbol c: Prf C ≔
  a3 (a2 a1) (a2 a1);
```

Not the DAG but a tree

- no sharing

```
symbol c: Prf C ≔
  let i1 : Prf B ≔ a2 a1 in
  let i2 : Prf C ≔ a3 i1 i1 in
  i2;
```

Scaling up?

In lambdapi, symbols for inferred formulas

```
constant symbol a1: Prf A;
constant symbol a2: Prf A → Prf B;
constant symbol a3: Prf B → Prf B → Prf C;

symbol i1: Prf B ≔ a2 a1;
symbol i2: Prf C ≔ a3 i1 i1;

symbol c: Prf C ≔ i2;
```

Separating declarations and definitions

Signature.lp

```
constant symbol a1: Prf A;
constant symbol a2: Prf A → Prf B;
constant symbol a3: Prf B → Prf B → Prf C;
symbol i1: Prf B;
symbol i2: Prf C;
symbol c: Prf C;
```

i1.lp

```
rule i1 ↪ a2 a1;
```

i2.lp

```
rule i2 ↪ a3 i1 i1;
```

proof.lp

```
rule c ↪ i2;
```

Why separating?

Pros:

- Useful when subproofs are generated independently
  - GDV-LP
- Cleaner signature

Cons:

- Acyclicity is no longer guaranteed!

```
constant symbol a: Prf A → Prf ⊥;
symbol i1: Prf A;
symbol i2: Prf A;
symbol c: Prf ⊥;
rule i1 ↪ i2;     rule i2 ↪ i1;     rule c ↪ a i1;
```

Acyclicity

However, acyclicity can be checked:

- ask for the normal form of c
- if terminates, acyclic
- but expands the DAG into a tree

Proofs by refutation

In most ATP, do not deduce C from $A_1, \ldots, A_n$
but deduce $\perp$ from $A_1, \ldots, A_n, \neg C$

How to reflect this when using new symbols for inferred clauses?

Naive approach

Add the negation of C as an axiom

Deduce `Prf ⊥`

```
constant symbol a1: Prf A1;
...
constant symbol an: Prf An;
constant symbol neg_c: Prf C → Prf ⊥;

symbol i1: Prf I1 ≔ ...;
...
symbol ik: Prf In ≔ ...;
symbol ik+1: Prf ⊥ ≔ ...;
```

But this is not a proof of C

Adding the conjecture in the context

Define a new predicate

```
symbol Prf_c p ≔ Prf (¬C) → Prf p;
```

All inferred formulas are now proved in a context where $\neg C$ is assumed

New approach

```
constant symbol a1: Prf A1;
...
constant symbol an: Prf An;
symbol c: Prf C;

symbol i1: Prf_c I1 ≔ λ neg_c, ...;
...
symbol ik: Prf_c Ik ≔ λ neg_c, ...;
symbol ik+1: Prf_c ⊥ ≔ λ neg_c, ...;
rule c ↪ nnpp C ik+1;
```

$$(\texttt{Prf\_c}\ \bot) \equiv (\texttt{Prf}\ (\neg\texttt{C}) \to \texttt{Prf}\ \bot) \equiv (\texttt{Prf}\ (\neg\neg\texttt{C}))$$

Skolemization in GDV-LP

Problem: Skolemization steps are not provable

$$\forall X,\ \exists Y,\ p(X,Y) \quad \not\Rightarrow \quad \forall X,\ p(X, sk(X))$$

Futhermore: Skolemization hidden in more complex inferences

```
fof(c_0_4, plain, (p(f(esk1_0))),
    inference(skolemize,[status(esa)],[
        inference(variable_rename,[status(thm)],[c_0_0])]))
```

Solution

```
fof(out, plain, OUT, inference(...skolemize...[in]...)).
```

- Use a trusted tool to Skolemize the formula `in` into a formula `sk_in`
- Use Zenon Modulo to prove `sk_in` $\Rightarrow$ `out`
- Define the Skolem symbol as a Hilbert ϵ-term to be able to prove
  `in` $\Rightarrow$ `sk_in`
  $(\exists x,\ P[x]) \Rightarrow P[\epsilon(\lambda x,\ P[x])]$

Defining Hilbert ϵ

```
require open Logic.U.Prop Logic.U.Set Logic.U.Quant;

symbol ϵ [a : Set] : (El a → Prop) → El a;

symbol Hilbert_epsilon (a : Set) (p : El a → Prop) (x : El a)
  Prf (p x) → Prf (p (ϵ p));
```

Proving Skolemization

```
symbol p : El ι → El ι → Prop;

symbol hyp : Prf (‘∀ x, ‘∃ y, p x y);

symbol sk (x : El ι) ≔ ϵ (λ y, p x y);

symbol conclusion : Prf (‘∀ x, p x (sk x)) ≔
begin
  assume x;
  refine hyp x (p x (sk x)) _;
  assume y unsk;
  apply Hilbert_epsilon ι (λ y, p x y) y unsk;
end;
```

Issues

- The trusted Skolemizer must agree with how Skolem symbols are named, and of which variables they depend
  - need to modify ATPs so that they provide this information in the TSTP inference
- ϵ-terms are not first-order
  - in TSTP, need a stronger logic (TXF)
  - theoretically, ϵ-terms can be eliminated
    but in practice?