Proof Reconstruction in Classical Propositional Logic

(Work in Progress)

Jonathan Prieto-Cubides
(Joint work with Andrés Sicard-Ramírez)

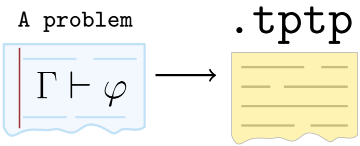
Universidad EAFIT
Medellín, Colombia

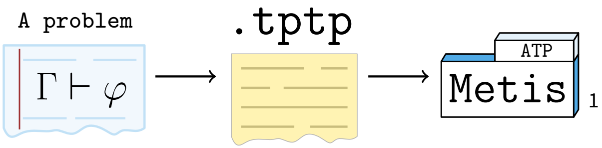
Agda Implementors' Meeting XXV
May 9-15th

`A problem`

$$\Gamma \vdash \varphi$$

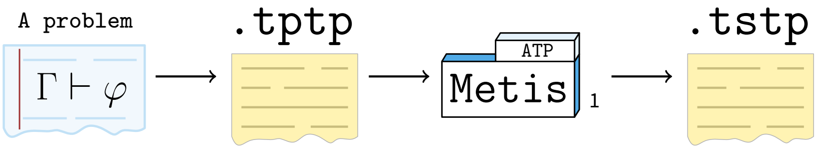
Proof Reconstruction: Overview





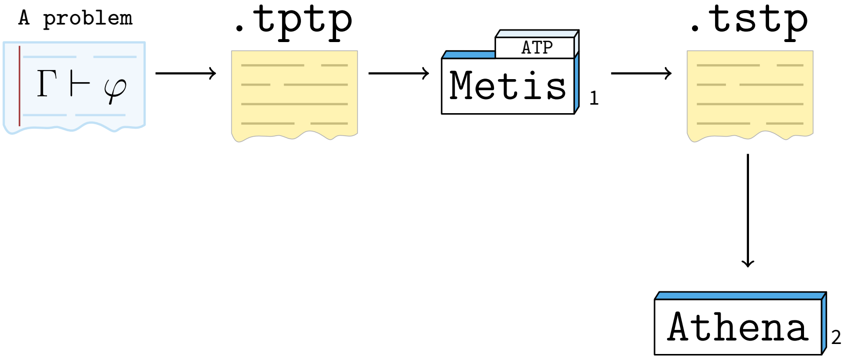
[1] It is available at `http://www.gilith.com/software/metis`

Proof Reconstruction: Overview



[1] It is available at `http://www.gilith.com/software/metis`

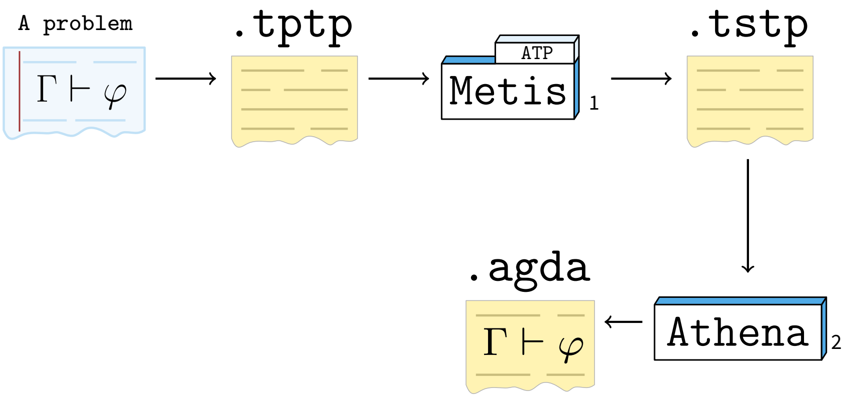
Proof Reconstruction: Overview



[1] It is available at `http://www.gilith.com/software/metis`

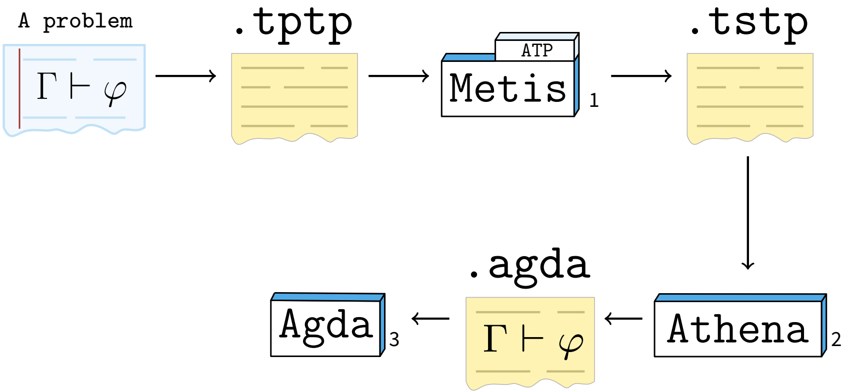
[2] It is available at `http://github.com/jonaprieto/athena`

Proof Reconstruction: Overview



[1] It is available at `http://www.gilith.com/software/metis`

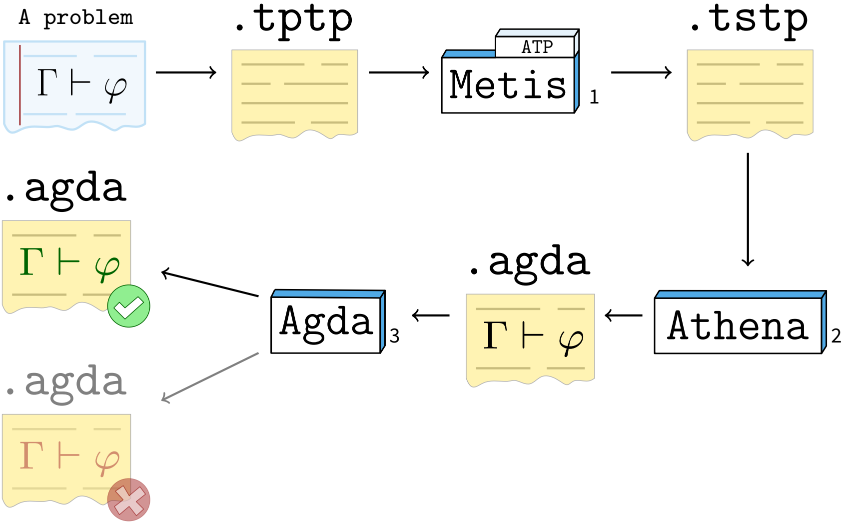
[2] It is available at `http://github.com/jonaprieto/athena`



[1]It is available at `http://www.gilith.com/software/metis`

[2]It is available at `http://github.com/jonaprieto/athena`

[3]It is available at `http://github.com/agda/agda`



[1] It is available at `http://www.gilith.com/software/metis`

[2] It is available at `http://github.com/jonaprieto/athena`

[3] It is available at `http://github.com/agda/agda`

SledgeHammer

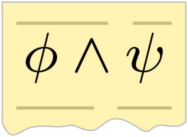
- Isabelle/HOL tool
- Metis ported within Isabelle
- Reconstruct proofs of well-known ATPs: EProver, Vampire, among others

Integrating Waldmeister and Agda

- Source code not available
- Equational Logic
- Reflection Layers

At the moment, the communication between Agda and the ATPs is unidirectional because the ATPs are being used as oracles (Sicard-Ramírez, 2015).

.agda



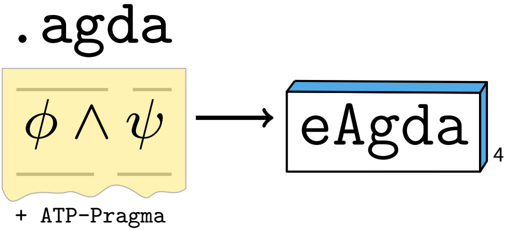
+ ATP-Pragma

```
$ cat Or.agda
module Or where

data _or_ (A B : Set) : Set where
  inj1 : A -> A or B
  inj2 : B -> A or B

postulate
  A B     : Set
  or-comm : A or B -> B or A
{-# ATP prove or-comm #-}
```

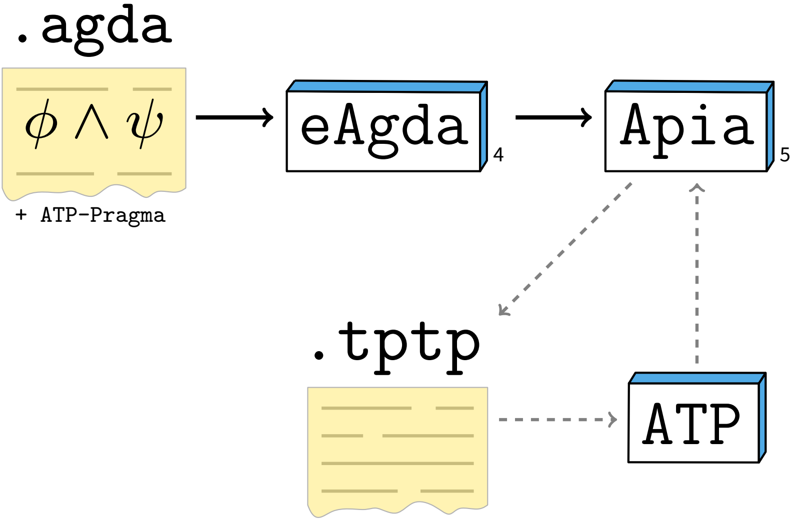
At the moment, the communication between Agda and the ATPs is unidirectional because the ATPs are being used as oracles (Sicard-Ramírez, 2015).



[4] Development version of Agda in order to handle a new built-in ATP-pragma. `https://github.com/asr/eagda`

Related Work: Apia

Proving first-order theorems written in Agda using automatic theorem provers for first-order logic



[4] Development version of Agda in order to handle a new built-in ATP-pragma. `https://github.com/asr/eagda`

[5] Haskell program for proving first-order theorems written in Agda using ATPs. `https://github.com/asr/apia`

Bonus Slides

TPTP Syntax

Thousands of Problems for Theorem Provers

.tptp

- Is a language[6] to encode problems
- Is the input of the ATPs
- Annotated formulas with the form

```
language(name, role, formula).
```

language FOF or CNF
name to identify the formula within the problem
role axiom, definition, hypothesis, conjecture, among others
formula version in TPTTP format

[6] Is available at `http://www.cs.miami.edu/~tptp/TPTP/SyntaxBNF.html`

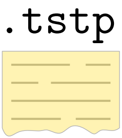
- $p \vdash p$

```
fof(a, axiom, p).
fof(goal, conjecture, p).
```

- $\vdash \neg(p \wedge \neg p) \vee (q \wedge \neg q)$

```
fof(goal, conjecture, ~ ((p & ~ p) | (q & ~ q))).
```

[7] Is available at `http://github.com/jonaprieto/pro-pack`



A TSTP derivation [8]

- Is a **D**irected **A**cyclic **G**raph where
  - `leaf` is a formula from the TPTP input
  - `node` is a formula inferred from parent formula
  - `root` the final derived formula
- Is a list of annotated formulas with the form:

```
language(name, role, formula, source [,useful info]).
```

where **source** typically is an inference record:

```
inference(rule, useful info, parents)
```

[8] `http://www.cs.miami.edu/~tptp/TPTP/QuickGuide/Derivations.html`

Go Back

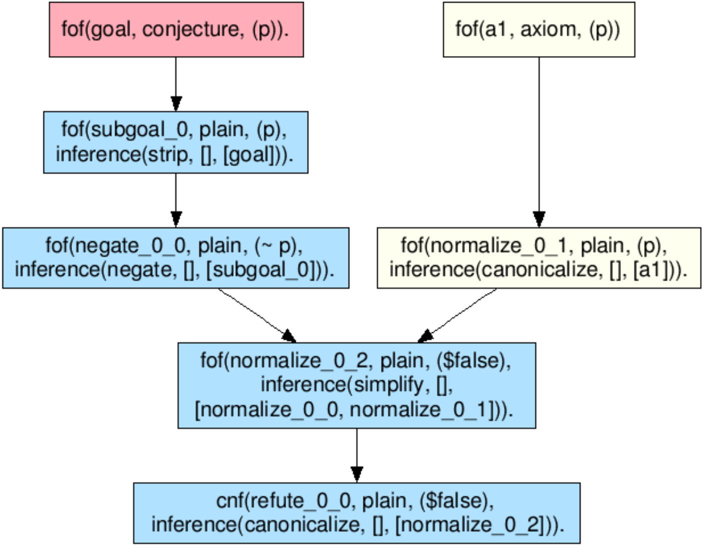
- Proof found by **Metis** ATP for the problem $p \vdash p$

ATP
Metis

```
$ metis --show proof basic-4.tptp
fof(a, axiom, (p)).
fof(goal, conjecture, (p)).
fof(subgoal_0, plain, (p),
  inference(strip, [], [goal])).
fof(negate_0_0, plain, (~ p),
  inference(negate, [], [subgoal_0])).
fof(normalize_0_0, plain, (~ p),
  inference(canonicalize, [], [negate_0_0])).
fof(normalize_0_1, plain, (p),
  inference(canonicalize, [], [a])).
fof(normalize_0_2, plain, ($false),
  inference(simplify, [],
    [normalize_0_0, normalize_0_1])).
cnf(refute_0_0, plain, ($false),
    inference(canonicalize, [], [normalize_0_2])).
```

Go Back

DAG for the previous TSTP derivation found by `Metis` ATP



Go Back

Athena

Athena

Is a Haskell program that translates proofs given by Metis Prover in TSTP format to Agda code.

It depends on:

- **`agda-prop`** Classical Logic within Agda: Axioms + Theorems
- **`agda-metis`** Theorems of the inference rules of Metis Prover

Go Back

Athena

Haskell

- Parsing
- AST construction
- Creation and analysis of DAG derivations
- Analysis of inference rules used
- Generation of Agda code of the proof

Go Back

Agda

Agda

Is a dependently typed functional programming language and it also a proof assistant.

We used it to type-check the proofs found by Metis Prover

- Agda-Prop Libary: Logic framework for Classical Propositional Logic
- Agda-Metis Library: theorems based on the inference rules of Metis Prover

Metis is an automatic theorem prover for First-Order Logic with equality

Why Metis?

- Open source implemented in Standard ML
- Reads problem in **TPTP** format
- Outputs detailed proofs in **TSTP** format
- Each refutation step is one of **6 simple rules**

Go Back

TSTP derivations exhibit these inferences:

Rule	Task
`canonicalize`	transforms formulas to CNF, DNF or NNF
`clausify`	performs clausification
`conjunct`	extracts a formula from a conjunction
`negate`	applies negation to the formula
`resolve`	applies theorems of resolution
`simplify`	applies over a list of formula to simplify them
`strip`	splits a formula into subgoals

Go Back

canonicalize

clausify

Conjunct

Inference Rule

conjunct

```
$ cat ~/agda-metis/src/ATP/Metis/Rules/Conjunct.agd
...
conjunct : Prop -> Prop -> Prop
conjunct φ( ⊠ψ ) ω with ⊠ eq φω⊠    | ⊠ eq ψω⊠
... | true  | _     = φ
... | false | true  = ψ
... | false | false = conjunct φω
conjunct φω          = φ

atp-conjunct
  : ⊠Γ {} φ{}
  -> ω( : Prop)
  -> Γ⊠φ
  -> Γ⊠  conjunct φω
```

Negate

Inference Rule

negate

resolve

simplify

Strip

Inference Rule

strip

Example goes here

Go Back

Type-checked Proof

Verified Example goes here

Go Back

Failure Example goes here

Go Back

Future Work

- Add shallow embedding in order to work with `Apia`
- Support First-Order Logic with Equality
- Support another prover like `EProver`

Sicard-Ramírez, Andrés (2015). *Reasoning about functional programs by combining interactive and automatic proofs.* PEDECIBA Informática, Universidad de la República.