

Proof Reconstruction in Classical Propositional Logic

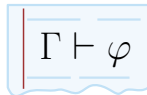
(Work in Progress)

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

Universidad EAFIT
Medellín, Colombia

Agda Implementors' Meeting XXV
May 9-15th

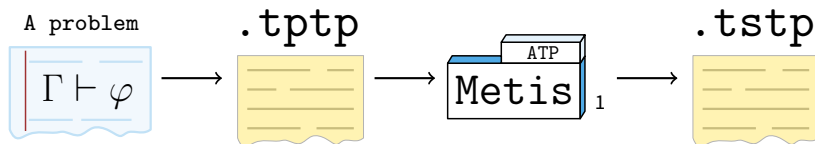
A problem


$$\Gamma \vdash \varphi$$

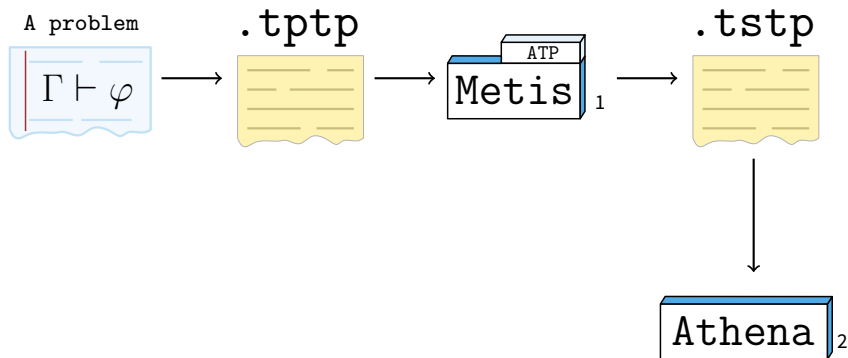




¹It is available at <http://www.gilith.com/software/metis>

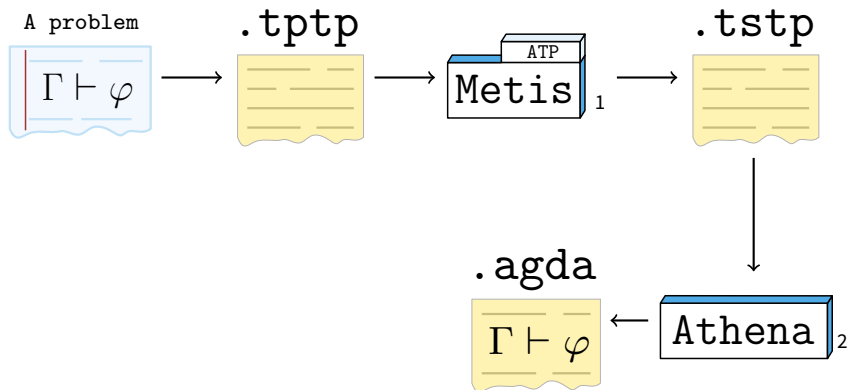


¹It is available at <http://www.gilith.com/software/metis>



¹ It is available at <http://www.gilith.com/software/metis>

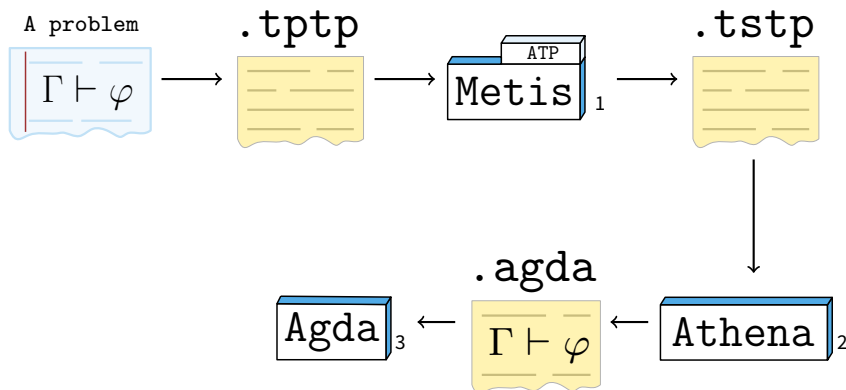
² It is available at <http://github.com/jonaprieto/athena>



¹ It is available at <http://www.gilith.com/software/metis>

² It is available at <http://github.com/jonaprieto/athena>

Proof Reconstruction: Overview

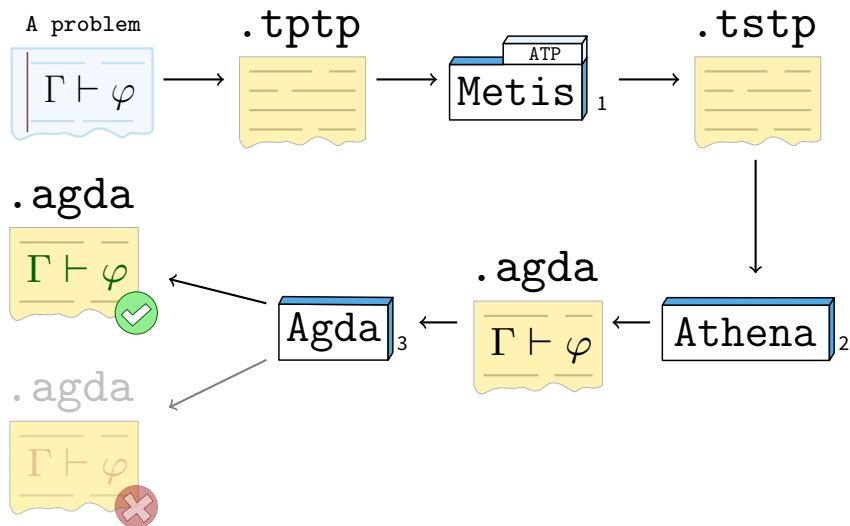


¹ It is available at <http://www.gilith.com/software/metis>

² It is available at <http://github.com/jonaprieto/athena>

³ It is available at <http://github.com/agda/agda>

Proof Reconstruction: Overview



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

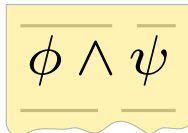
- ▶ Source code not available
- ▶ Equational Logic
- ▶ Reflection Layers

Related Work: Apia

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

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

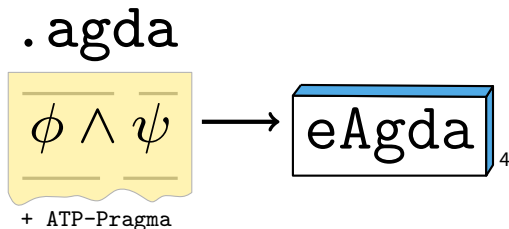
data _or_ (A B : Set) : Set where
  inj1 : A -> A or B
  inj2 : B -> A or B

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

Related Work: Apia

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

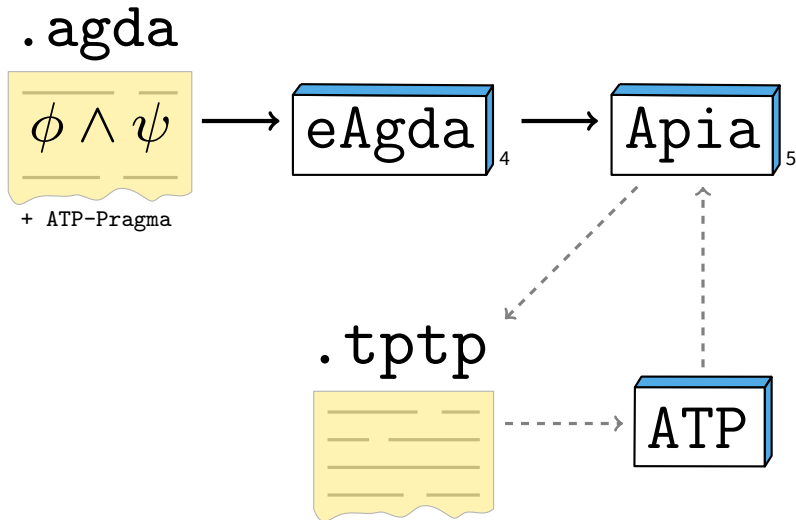
At the moment, the communication between Agda and the ATPs is unidirectional because the ATPs are being used as oracles (Sicard-Ramírez, 2015).



⁴Development 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



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

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

.tptp



- ▶ Is a language⁶ 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

⁶Is 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))).
```

⁷Is available at <http://github.com/jonaprieto/pro-pack>

.tstp



A TSTP derivation⁸

- ▶ Is a Directed Acyclic Graph 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:

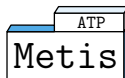
```
language(name, role, formula, source [,useful info]).
```

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

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

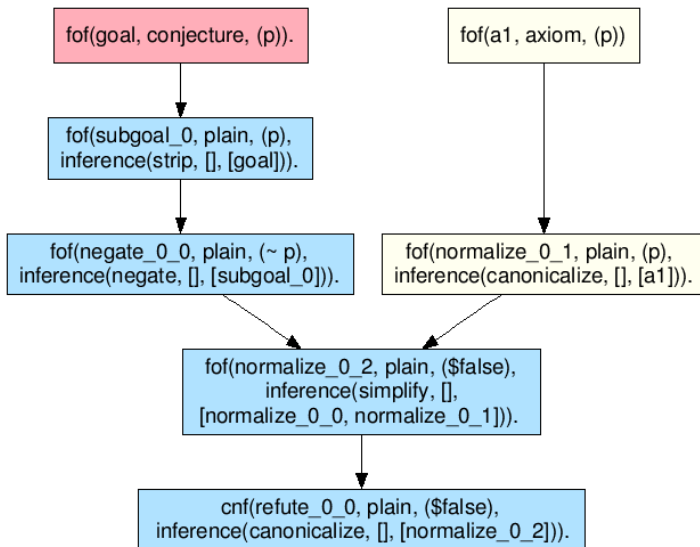
⁸<http://www.cs.miami.edu/~tptp/TPTP/QuickGuide/Derivations.html>

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



```
$ 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])).
```

DAG for the previous TSTP derivation found by Meti's ATP

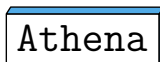


Athena

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



Haskell

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



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 Library: 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**

TSTP derivations exhibit these inferences:

Rule	Task
canonicalize	transforms formulas to CNF or NNF
clausify	performs clausification
conjoinct	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

canonicalize

clausify

conjunct

```
$ cat ~/agda-metis/src/ATP/Metis/Rules/Conjunct.agd
```

```
...
```

```
conjunct : Prop → Prop → Prop
```

```
conjunct  $\phi$  (  $\omega \psi$  )  $\omega$  with  $\omega$  eq  $\phi \omega$  |  $\omega$  eq  $\psi \omega$ 
```

```
... | true | _ =  $\phi$ 
```

```
... | false | true =  $\psi$ 
```

```
... | false | false = conjunct  $\phi \omega$ 
```

```
conjunct  $\phi \omega$  =  $\phi$ 
```

```
atp-conjunct
```

```
:  $\omega \Gamma \{ \} \phi \{ \}$ 
```

```
→  $\omega$  ( : Prop )
```

```
→  $\Gamma \omega \phi$ 
```

```
→  $\Gamma \omega$  conjunct  $\phi \omega$ 
```

Negate

Inference Rule

negate

resolve

simplify

strip

Example goes here

Verified Example goes here

Failure Example goes here

- ▶ 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.