Proof Reconstruction in Classical Propositional Logic

(Work in Progress)

Jonathan Prieto-Cubides
Joint work with Andrés Sicard-Ramírez

Universidad EAFIT
Medellín, Colombia

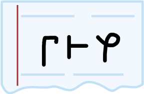
Agda Implementors' Meeting XXV
May 9-15th

UNIVERSIDAD EAFIT®

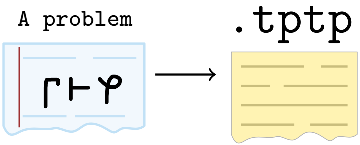
CHALMERS
UNIVERSITY OF TECHNOLOGY

Proof Reconstruction: Overview

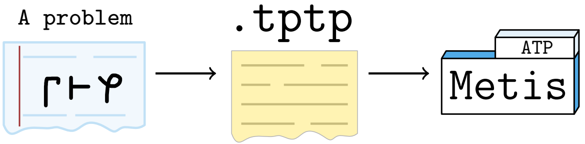
A problem



Proof Reconstruction: Overview

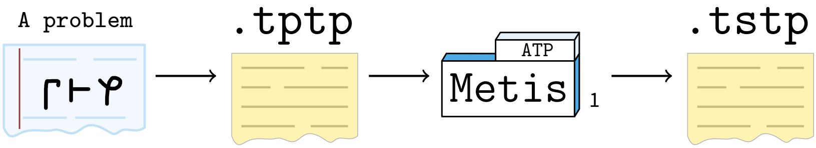


Proof Reconstruction: Overview

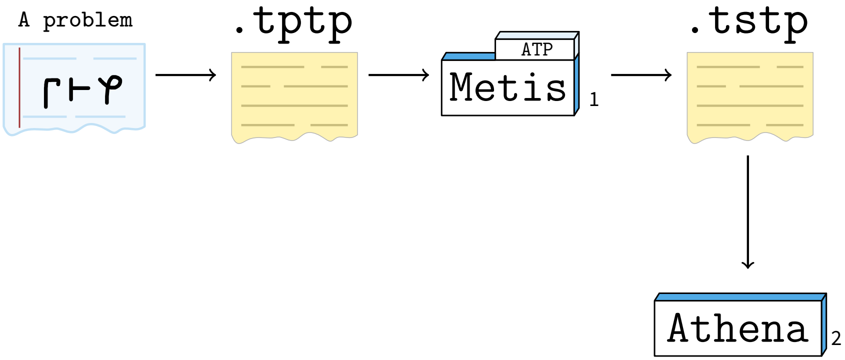


[1]

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



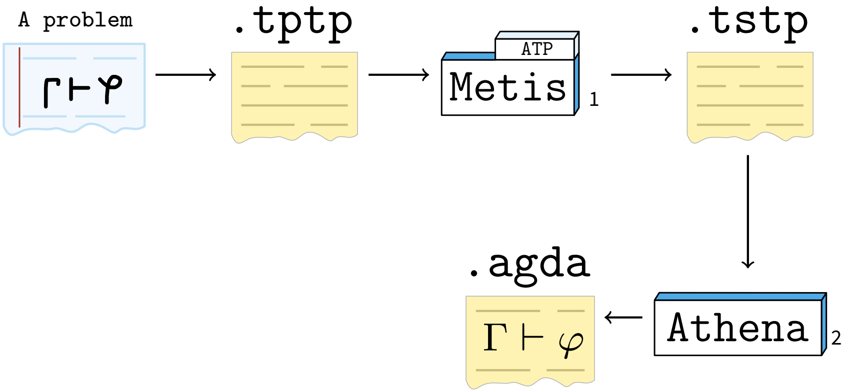
[1] It is available at `http://www.gilith.com/software/metis`



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

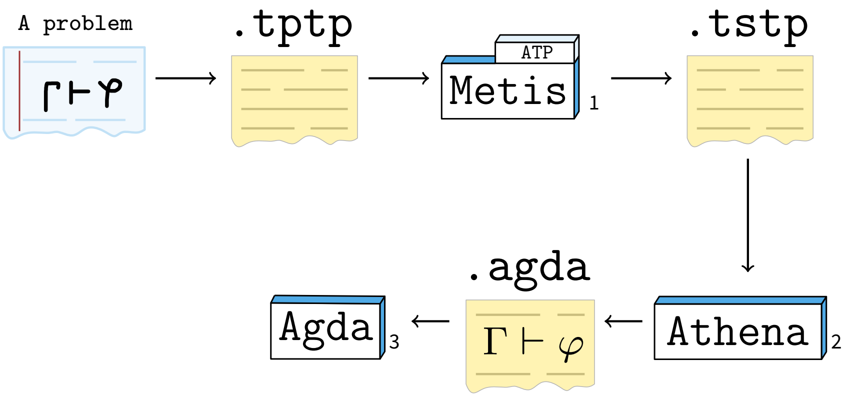
[2] It is available at `http://github.com/jonaprieto/athena`

Proof Reconstruction: Overview



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

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

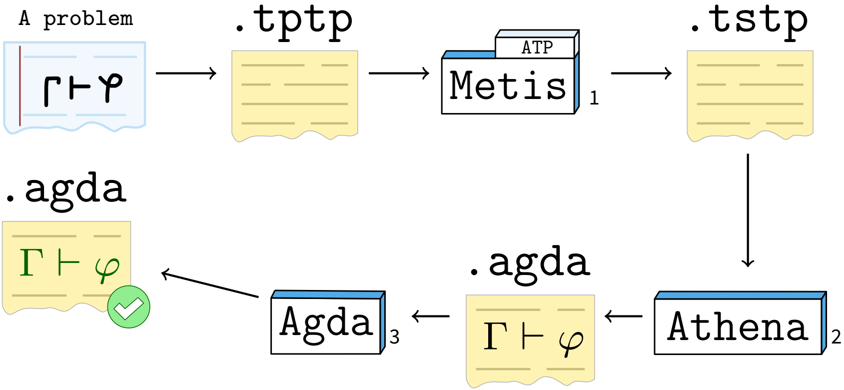


[1] It is available at `http://www.gilith.com/software/metis`

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

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

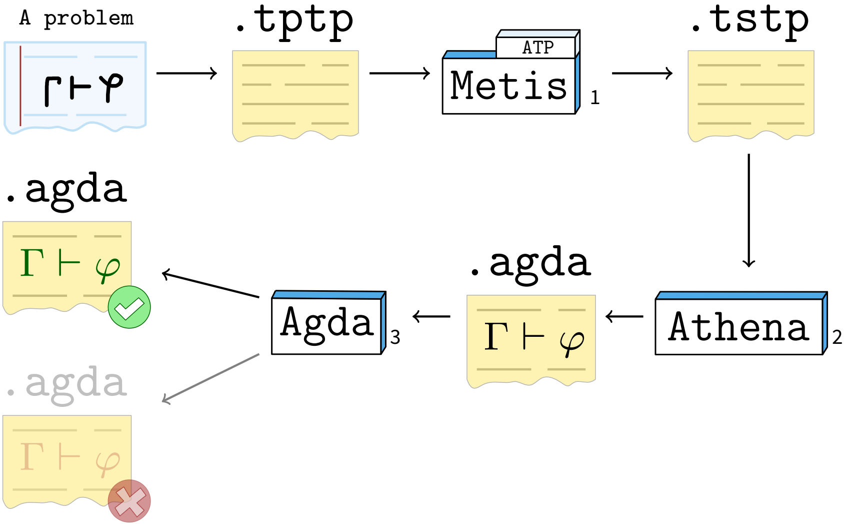
Proof Reconstruction: Overview



[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`

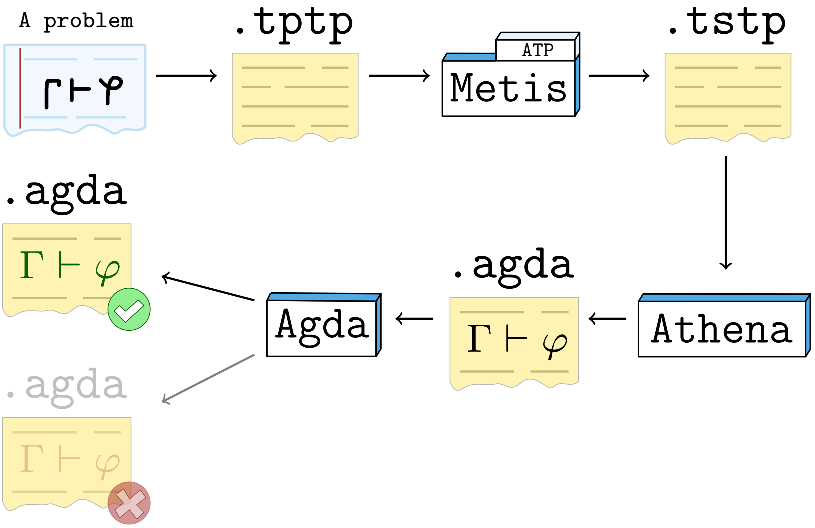


[1] It is available at `http://www.gilith.com/software/metis`

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

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

Proof Reconstruction: Overview



Bonus Slides

.tptp

- Is a language[4] to encode problems (Sutcliffe, 2009)
- 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

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

Go Back

TPTP Examples

From Prop-Pack repository[5]

- $p \vdash p$

```
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))).
```

[5] Is available at `http://github.com/jonaprieto/prop-pack`

Go Back

Metis Theorem Prover

http://www.gilith.com/software/metis/

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

Why Metis?

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

Go Back

TSTP derivations by Metis exhibit these inferences [6]:

Rule	Purpose
`canonicalize`	transforms formulas to CNF, DNF or NNF
`clausify`	performs clausification
`conjunct`	takes 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

[6] Inference rules found in proofs of Propositional Logic theorems

Go Back

TSTP Syntax

.tstp

A TSTP derivation [7]

- 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)
```

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

Go Back

- Proof found by **Metis** Prover 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

By refutation, we proved $p \vdash p$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p}{p}\,\texttt{assume}}{p}\,\texttt{strip}}{\neg p}\,\texttt{negate} \qquad \dfrac{p}{p}\,\texttt{canonicalize}}{\bot}\,\texttt{simplify}}{\bot}\,\texttt{canonicalize}$$

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

- Parsing of **TSTP** language
- Creation and analysis of **DAG** derivations
- Analysis of inference rules used in the **TSTP** derivation
- Agda code generation

Library	Purpose
Agda-Prop	axioms and theorems of Classical Propositional Logic
Agda-Metis	versions of the inference rules used by Metis

Go Back

- Propositional Logic + PEM ($\Gamma \vdash \phi \vee \neg\phi$)
- A data type for formulas

```
data Prop : Set where
  Var   : Fin n → Prop               -- Variables.
  ⊤     : Prop                       -- Top (truth).
  ⊥     : Prop                       -- Bottom (falsum).
  _∧_   : (φ ψ : Prop) → Prop        -- Conjunction.
  _∨_   : (φ ψ : Prop) → Prop        -- Disjunction.
  _⇒_   : (φ ψ : Prop) → Prop        -- Implication.
  _⇔_   : (φ ψ : Prop) → Prop        -- Biimplication.
  ¬_    : (φ : Prop) → Prop          -- Negation.
```

- A data type for theorems

```
data _⊢_ : (Γ : Ctxt)(φ : Prop) → Set
```

- Constructors

```
assume, axiom, weaken, ⊤-intro, ⊥-elim, ¬-intro,
¬-elim, ∧-intro, ∧-proj₁, ∧-proj₂, ∨-intro₁,
∨-intro₂, ∨-elim, ⇒-intro, ⇒-elim, ⇔-intro,
⇔-elim₁, ⇔-elim₂
```

- Natural deduction proofs for more than 71 theorems

```
⇔-equiv, ⇔-assoc, ⇔-comm, ⇒-⇔-¬∨, ⇔-¬-to-¬,
¬⇔-to-¬, ¬¬-equiv, ⇒⇒-⇔-∧⇒, ⇔-trans, ∧-assoc,
∧-comm, ∧-dist, ¬∧-to-¬∨¬, ¬∨¬-to-¬∧, ¬∨¬-⇔-¬∧,
subst⊢∧₁, subst⊢∧₂, ∨-assoc, ∨-comm, ∨-dist,
∨-equiv, ¬∨-to-¬∧¬, ¬∧¬-to-¬∨, ∨-dmorgan,
¬¬∨¬¬-to-∨, cnf, nnf, dnf, RAA,∘..
```

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

- Definition

$$conjunct(\phi_1 \land \phi_2 \land \cdots \land \boldsymbol{\varphi}_\mathbf{i} \land \cdots \land \phi\_n, \boldsymbol{\varphi}_\mathbf{i}) \longrightarrow \boldsymbol{\varphi}_\mathbf{i}$$

- Function[8]:

```
conjunct : Prop → Prop → Prop
conjunct (φ ∧ ψ) ω with ⌊ eq φ ω ⌋ | ⌊ eq ψ ω ⌋
... | true  | _     = φ
... | false | true  = ψ
... | false | false = conjunct φ ω
conjunct φ ω       = φ
```

- Theorem[8]

```
atp-conjunct
  : ∀ {Γ} {φ}
  → (ω : Prop)
  → Γ ⊢ φ
  → Γ ⊢ conjunct φ ω
```

[8] Excerpt from the `Agda-Metis` library available in `ATP.Metis.Rules.Conjunct` module.

A proof of atp-conjunct theorem

```
atp-conjunct
  : ∀{Γ}{φ}
  → (ω : Prop)
  → Γ ⊢ φ
  → Γ ⊢ conjunct φ ω

atp-conjunct {Γ} {φ ∧ ψ} ω Γ⊢φ with ⌊ eq φ ω ⌋ | ⌊ eq ψ ω ⌋
... | true  | _     = ∧-proj₁ Γ⊢φ
... | false | true  = ∧-proj₂ Γ⊢φ
... | false | false =
    atp-conjunct {Γ = Γ} {φ = φ} ω (∧-proj₁ Γ⊢φ)
atp-conjunct {_} {Var x} _ = id
atp-conjunct {_} {⊤} _     = id
atp-conjunct {_} {⊥} _     = id
atp-conjunct {_} {φ ∨ φ₁} _ = id
atp-conjunct {_} {φ ⇒ φ₁} _ = id
atp-conjunct {_} {φ ⇔ φ₁} _ = id
atp-conjunct {_} {¬φ} _    = id
```

Agda Code Example

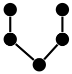
Generated by Athena Tool

- The problem is $p \wedge q \vdash q \wedge p$
- In TPTP format

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

- A natural deduction proof

$$\dfrac{\dfrac{\phi \wedge \psi}{\phi}\ \wedge\text{-}\mathtt{proj}_1 \qquad \dfrac{\phi \wedge \psi}{\psi}\ \wedge\text{-}\mathtt{proj}_2}{\psi \wedge \phi}\ \wedge\text{-}\mathtt{intro}$$



Go Back

TSTP derivation I

```
fof(a, axiom, p & q).
fof(goal, conjecture, q & p).
fof(subgoal_0, plain, q,
    inference(strip, [], [goal])).
fof(subgoal_1, plain, q => p,
    inference(strip, [], [goal])).
fof(negate_0_0, plain, ~ q,
inference(negate, [], [subgoal_0])).
fof(normalize_0_0, plain, (~ q),
    inference(canonicalize, [], [negate_0_0])).
fof(normalize_0_1, plain, p & q,
    inference(canonicalize, [], [a])).
fof(normalize_0_2, plain, q,
    inference(conjunct, [], [normalize_0_1])).
fof(normalize_0_3, plain, $false,
    inference(simplify, [],
        [normalize_0_0, normalize_0_2])).
cnf(refute_0_0, plain, $false,
    inference(canonicalize, [], [normalize_0_3])).
```

TSTP derivation II

```
fof(negate_1_0, plain, ~ (q => p),
    inference(negate, [], [subgoal_1])).
fof(normalize_1_0, plain, ~ p & q,
    inference(canonicalize, [], [negate_1_0])).
fof(normalize_1_1, plain, p & q,
     inference(canonicalize, [], [a])).
fof(normalize_1_2, plain, p,
    inference(conjunct, [], [normalize_1_1])).
fof(normalize_1_3, plain, q,
    inference(conjunct, [], [normalize_1_1])).
fof(normalize_1_4, plain, $false,
    inference(simplify, [],
      [normalize_1_0, normalize_1_2, normalize_1_3])).
cnf(refute_1_0, plain, ($false),
    inference(canonicalize, [], [normalize_1_4])).
```

Definitions

```
p : Prop
p = Var (# 0)
q : Prop
q = Var (# 1)
-- Axiom.
a : Prop
a = (p ∧ q)
-- Premise.
Γ : Ctxt
Γ = [ a ]
-- Conjecture.
goal : Prop
goal = (q ∧ p)
-- Subgoals.
subgoal₀ : Prop
subgoal₀ = q
subgoal₁ : Prop
subgoal₁ = (q ⇒ p)
```

```
-- Axiom.
a : Prop
a = (p ∧ q)
-- Subgoal.
subgoal₀ : Prop
subgoal₀ = q

proof₀ : Γ ⊢ subgoal₀
proof₀ =
  (RAA
    (atp-canonicalize
      (atp-simplify
        (atp-canonicalize
          (atp-strip
            (assume {Γ = Γ} (atp-negate subgoal₀))))
        (atp-conjunct (q)
          (atp-canonicalize
            (weaken (atp-negate subgoal₀)
              (assume {Γ = ∅} a)))))))
```

```
subgoal₁ : Prop
subgoal₁ = (q ⇒ p)

proof₁ : Γ ⊢ subgoal₁
proof₁ =
  (RAA
    (atp-canonicalize
      (atp-simplify
        (atp-conjunct (q)
          (atp-canonicalize
            (weaken (atp-negate subgoal₁)
              (assume {Γ = ∅} a))))
        (atp-simplify
          (atp-canonicalize
            (atp-strip
              (assume {Γ = Γ} (atp-negate subgoal₁))))
          (atp-conjunct (p)
            (atp-canonicalize
              (weaken (atp-negate subgoal₁)
                (assume {Γ = ∅} a))))))))
```

```
proof : Γ ⊢ goal
proof =
  ⇒-elim
    atp-splitGoal
    (∧-intro proof₀ proof₁)
```

Go Back

Bug in the Printing of the Proof

Metis' Issue: *https://github.com/gilith/metis/issues/2*

```
$ metis --version
metis 2.3 (release 20161108)
$ metis --show proof problem.tptp
...
fof(normalize_2_0, plain,
  (~ p & (~ q <=> ~ r) & (~ p <=> (~ q <=> ~ r))),
  inference(canonicalize, [], [negate_2_0])).
fof(normalize_2_1, plain, ~ p <=> (~ q <=> ~ r),
    inference(conjunct, [], [normalize_2_0])).
fof(normalize_2_2, plain, ~ q <=> ~ r,
    inference(conjunct, [], [normalize_2_0])).
fof(normalize_2_3, plain, ~ p,
  inference(conjunct, [], [normalize_2_0])).
fof(normalize_2_4, plain, $false,
    inference(simplify, [],
      [normalize_2_1, normalize_2_2, normalize_2_3])).
...
```

SledgeHammer

(Paulson and Susanto, 2007)

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

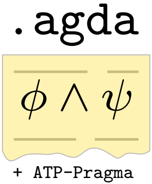
Integrating Waldmeister into Agda

(Foster and Struth, 2011)

- Framework for a integration between Agda and ATPs
  - Equational Logic
  - Reflection Layers
- Source code is not available[9]

[9] `http://simon-foster.staff.shef.ac.uk/agdaatp`

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

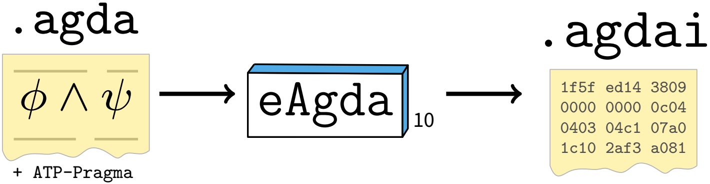



```
module Or where

data _∨_ (A B : Set) : Set where
  inj₁ : A → A ∨ B
  inj₂ : B → A ∨ B

postulate
  A B    : Set
  ∨-comm : A ∨ B → B ∨ A
{-# ATP prove ∨-comm #-}
```

Related Work: Apia

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

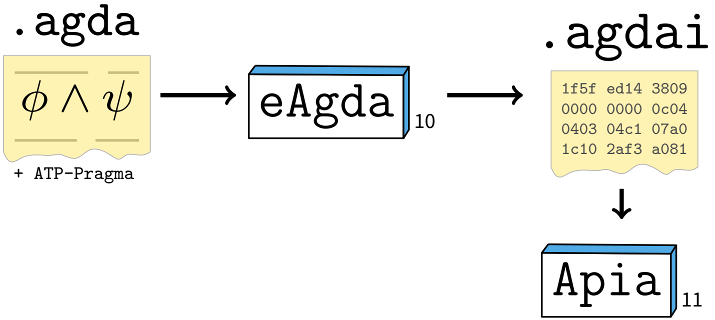
At the moment, the communication between Agda and the ATPs is unidirectional because the ATPs are being used as oracles (Sicard-Ramírez, 2015).



[10] 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

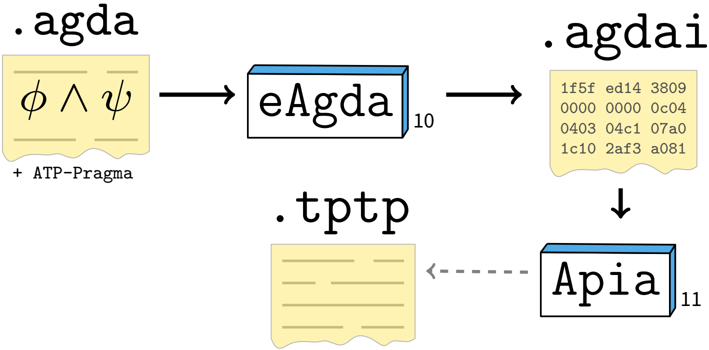


[10] Development version of Agda in order to handle a new built-in ATP-pragma. `https://github.com/asr/eagda`

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

Related Work: Apia

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

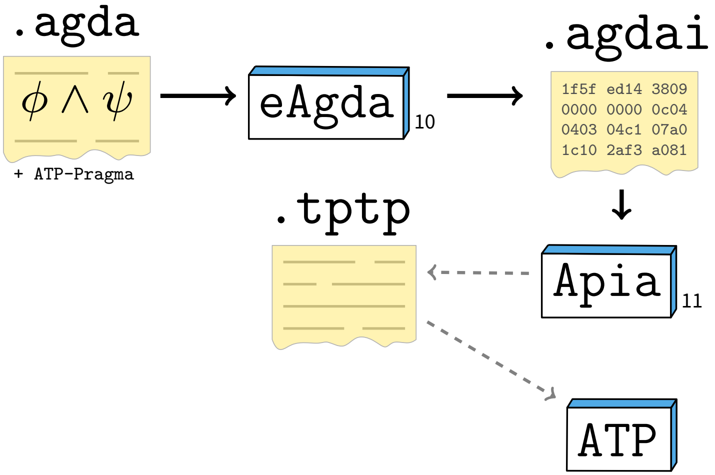


[10] Development version of Agda in order to handle a new built-in ATP-pragma. `https://github.com/asr/eagda`

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

Related Work: Apia

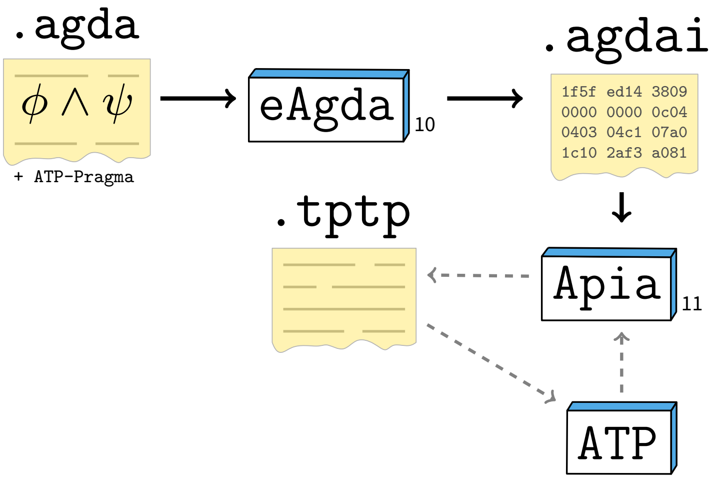
Proving first-order theorems written in Agda using automatic theorem provers for first-order logic



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

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

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

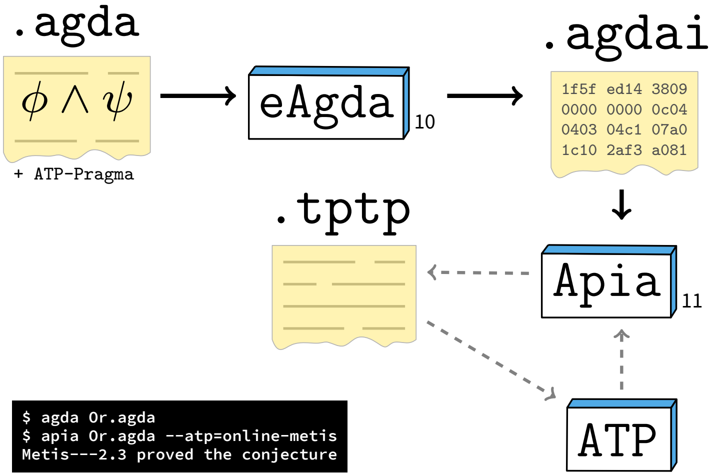


10 Development version of Agda in order to handle a new built-in ATP-pragma. `https://github.com/asr/eagda`

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

Related Work: Apia

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




```
$ agda Or.agda
$ apia Or.agda --atp=online-metis
Metis---2.3 proved the conjecture
```

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

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

Pending Work

- There are missing cases with the **`simplify`** inference
- Is not clear, how **`canonicalize`** inference choose what normal form use to transform the formulas
- When Metis prove a goal, it first splits that goal in a list of subgoal, then we are working on verification of such splitting.

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

References

Foster, Simon and Georg Struth (2011). "Integrating an Automated Theorem Prover into Agda". In: *NASA Formal Methods: Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*. Ed. by Mihaela Bobaru et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 116–130.

Paulson, Lawrence C. and Kong Woei Susanto (2007). "Source-Level Proof Reconstruction for Interactive Theorem Proving". In: *Theorem Proving in Higher Order Logics: 20th International Conference, TPHOLs 2007, Kaiserslautern, Germany, September 10-13, 2007. Proceedings*. Ed. by Klaus Schneider and Jens Brandt. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 232–245.

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

Sutcliffe, G. (2009). "The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0". In: *Journal of Automated Reasoning* 43.4, pp. 337–362.