Proof Reconstruction in Classical Propositional Logic

(work in progress)

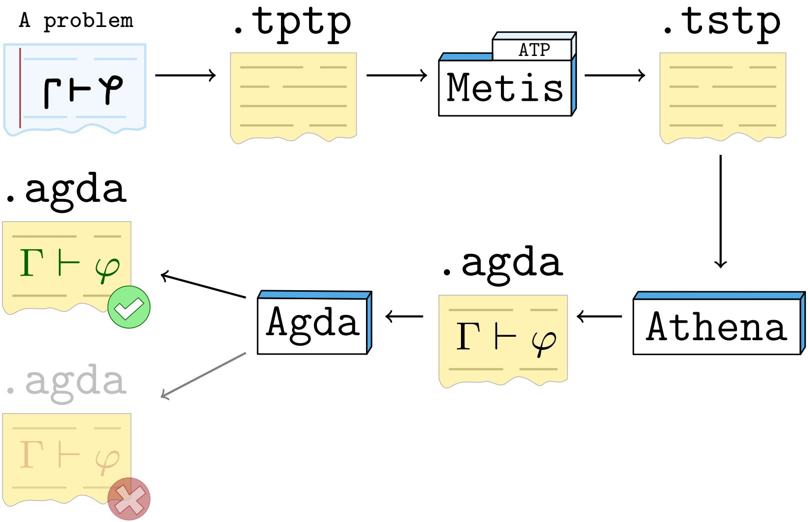
Jonathan Prieto-Cubides
(joint work with Andrés Sicard-Ramírez)

Master in Applied Mathematics
Universidad EAFIT
Medellín, Colombia

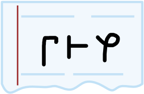
Agda Implementors’ Meeting XXV
Teknikparken, Chalmers, Gothenburg, Sweden
May 11 2017

Updated: November 16, 2017

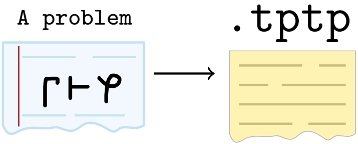
Proof Reconstruction: Overview



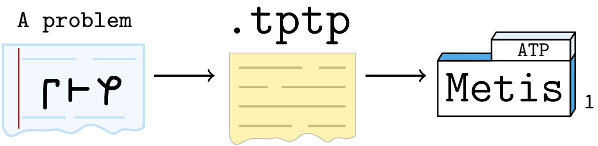
A problem



Proof Reconstruction: Overview

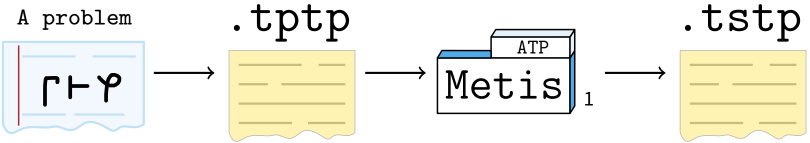


Proof Reconstruction: Overview



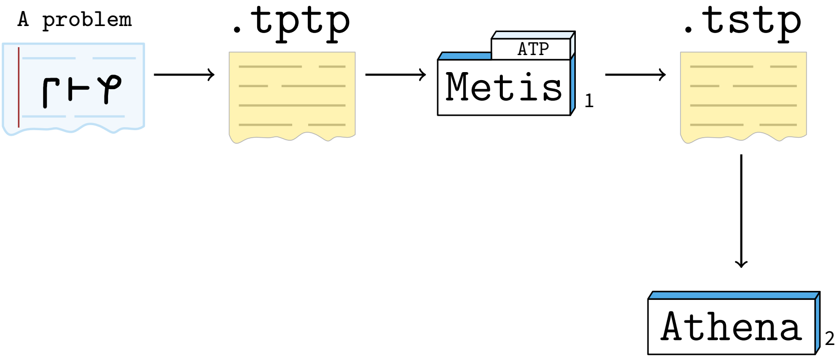
[1] http://www.gilith.com/software/metis.

Proof Reconstruction: Overview



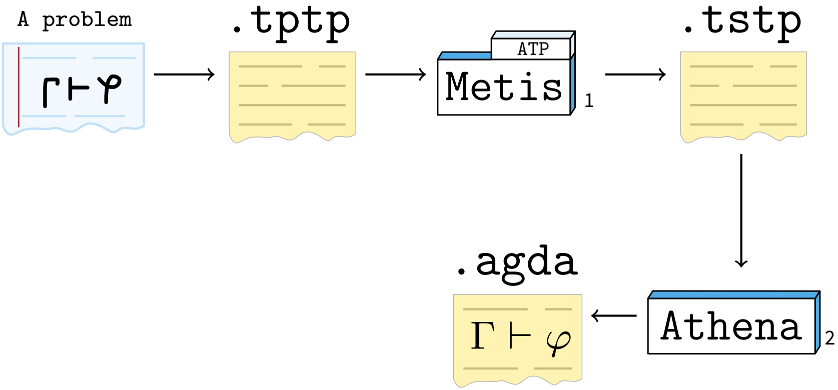
[1] http://www.gilith.com/software/metis.

Proof Reconstruction: Overview



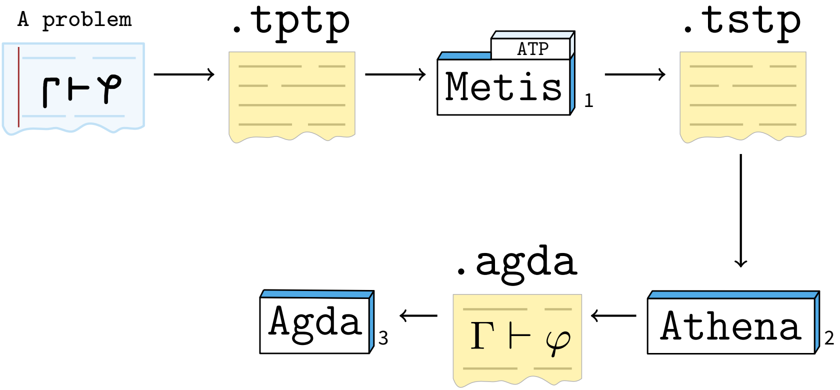
[1] http://www.gilith.com/software/metis.
[2] http://github.com/jonaprieto/athena.

Proof Reconstruction: Overview



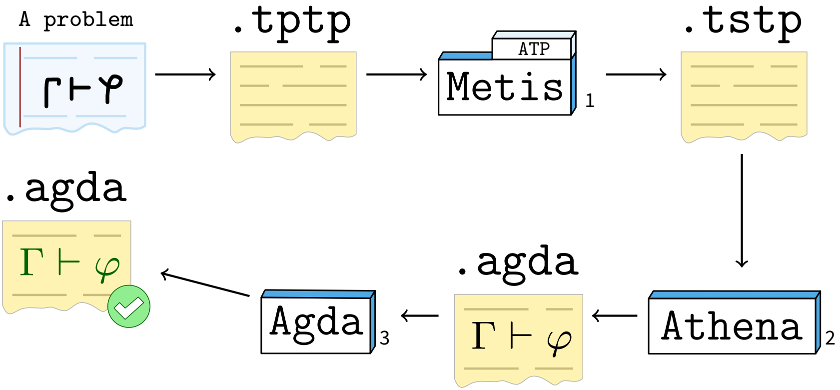
[1] http://www.gilith.com/software/metis.

[2] http://github.com/jonaprieto/athena.



[1] http://www.gilith.com/software/metis.
[2] http://github.com/jonaprieto/athena.
[3] http://github.com/agda/agda.

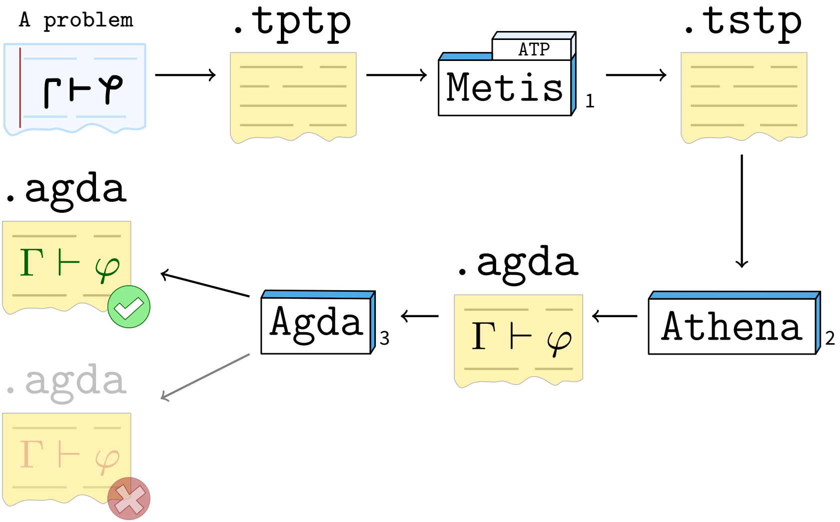
Proof Reconstruction: Overview



[1] http://www.gilith.com/software/metis.

[2] http://github.com/jonaprieto/athena.

[3] http://github.com/agda/agda.



[1] http://www.gilith.com/software/metis.

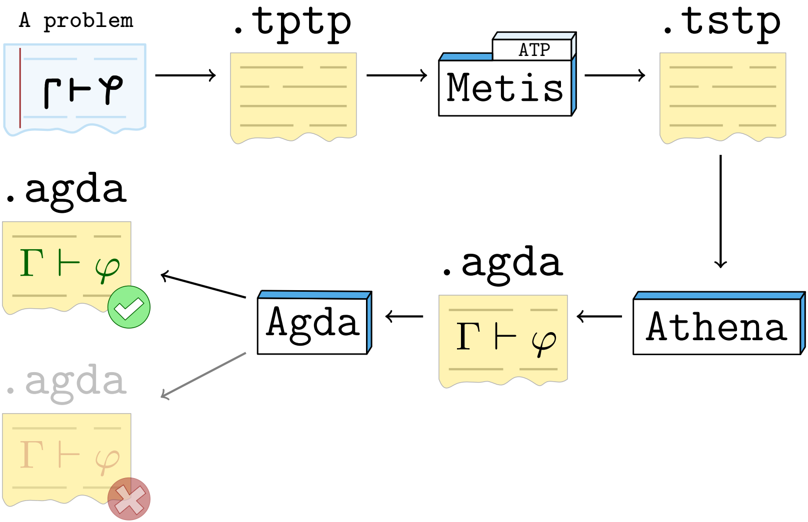
[2] http://github.com/jonaprieto/athena.

[3] http://github.com/agda/agda.

Obstacles to Reconstruction with Automatic Provers (Paulson2007)

- *Ambiguities*: their output typically omits crucial information, such as which term is affected by rewriting.
- *Lack of standards*: automatic provers generate different output formats and employ a variety of inference systems
- *Complexity*: a single automatic prover may use numerous inference rules with complicated behaviors
- *Problem transformations*: ATPs re-order literals and make other changes to the clauses they are given

Proof Reconstruction: Overview



TPTP Syntax
Thousands of Problems for Theorem Provers

.tptp

- Is a language[4] to encode problems (**Sut09**)
- 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
`formula` formula in TPTP format

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

TPTP Examples

- $p \vdash p$

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

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

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

Go Back

ATP Metis

`Metis` is an automatic theorem prover for First-Order Logic with Equality (Hurd, 2003)

Why Metis?

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

[5] `http://www.gilith.com/software/metis/`.

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

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).
```

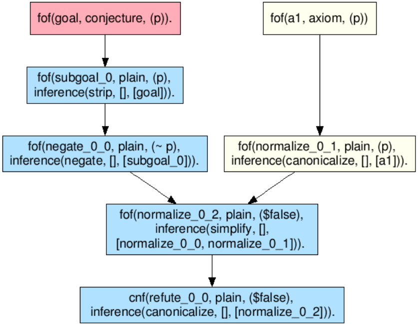
[7] http://www.cs.miami.edu/~tptp/TPTP/QuickGuide/Derivations.html.

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

```
$ metis --show proof problem.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])).
```

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

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



Go Back

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

- Parsing of TSTP language[8]
- Creation[8] 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

[8] https://github.com/agomezl/tstp2agda.

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

```
data PropFormula : Set where
  Var : Fin n → Prop
  ⊤   : Prop
  ⊥   : Prop
  _∧_ : (φ ψ : Prop) → Prop
  _∨_ : (φ ψ : Prop) → Prop
  _⇒_ : (φ ψ : Prop) → Prop
  _⇔_ : (φ ψ : Prop) → Prop
  ¬_  : (φ : Prop)   → Prop
```

[9] https://github.com/jonaprieto/agda-prop.

Agda-Prop Library (Prieto-Cubides, 2017a)

- 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`	append negation symbol 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	`atp-strip`

- Definition

$$\text{conjunct}(\overbrace{\varphi_1 \wedge \cdots \wedge \varphi_n}^{\varphi}, \psi) = \begin{cases} \varphi_i & \text{if } \psi \text{ is equal to some } \varphi_i \\ \varphi & \text{otherwise} \end{cases}$$

[10] https://github.com/jonaprieto/agda-metis.

- Definition

$$\text{conjunct}(\overbrace{\varphi_1 \land \cdots \land \varphi_n}^{\varphi}, \psi) = \begin{cases} \varphi_i & \text{if } \psi \text{ is equal to some } \varphi_i \\ \varphi & \text{otherwise} \end{cases}$$

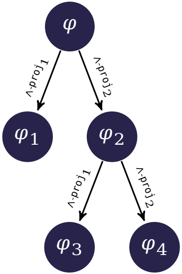
- Inference rules involved

$$\frac{\varphi_1 \land \varphi_2}{\varphi_1}\ \land\text{-}\mathtt{proj}_1 \qquad \frac{\varphi_1 \land \varphi_2}{\varphi_2}\ \land\text{-}\mathtt{proj}_2$$

- Example

$$\varphi := \varphi_1 \land \overbrace{(\varphi_3 \land \varphi_4)}^{\varphi_2}$$

  - $\mathtt{conjunct}(\varphi, \varphi_3 \land \varphi_1) \equiv \varphi$
  - $\mathtt{conjunct}(\varphi, \varphi_3) \equiv \varphi_3$
  - $\mathtt{conjunct}(\varphi, \varphi_2) \equiv \varphi_2$



[10] https://github.com/jonaprieto/agda-metis.

```agda
data ConjView : Prop → Set where
  conj  : (φ₁ φ₂ : Prop) → ConjView (φ₁ ∧ φ₂)
  other : (φ : Prop)     → ConjView φ

conj-view : (φ : Prop) → ConjView φ
conj-view (φ ∧ ψ) = conj _ _
conj-view φ       = other _

data Step : Set where
  pick  : Step
  proj₁ : Step
  proj₂ : Step

Path : Set
Path = List Step

conjunct-path : (φ ψ : Prop) → Path → Path
conjunct-path φ ψ path with ⌊ eq φ ψ ⌋
... | true  = path ∷ʳ pick
... | false with conj-view φ
...   | other _ = []
...   | conj φ₁ φ₂ with conjunct-path φ₁ ψ []
...     | subpath@(_ ∷ _) = (path ∷ʳ proj₁) ++ subpath
...     | [] with conjunct-path φ₂ ψ []
...          | subpath@(_ ∷ _) = (path ∷ʳ proj₂) ++ subpath
...          | []              = []
```

The conjunct function and its theorem, atp-conjunct

```
conjunct : Prop → Prop → Prop
conjunct φ ψ with conj-view φ | conjunct-path φ ψ []
... | _          | []         = φ
... | conj _ _   | pick  ∷ _  = φ
... | conj φ₁ _  | proj₁ ∷ _  = conjunct φ₁ ψ
... | conj _ φ₂  | proj₂ ∷ _  = conjunct φ₂ ψ
... | other .φ   | _          = φ

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

atp-conjunct {Γ} {φ} ψ Γ⊢φ
  with conj-view φ | conjunct-path φ ψ []
...  | _          | []         = Γ⊢φ
...  | conj _ _   | pick  ∷ _  = Γ⊢φ
...  | conj _ _   | proj₁ ∷ _  = atp-conjunct ψ (∧-proj₁ Γ⊢φ)
...  | conj _ _   | proj₂ ∷ _  = atp-conjunct ψ (∧-proj₂ Γ⊢φ)
...  | other _    | (_ ∷ _)    = Γ⊢φ
```

Go Back

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

```
$ cat problem.tptp
fof(a, axiom, p & q).
fof(goal, conjecture, q & p).
```

- How to use Athena with your problem

```
$ metis --show proof problem.tptp > problem.tstp
$ athena problem.tstp
$ agda problem.tstp
```

TSTP Derivation given by Metis 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])).
fof(negate_1_0, plain, ~ (q => p),
    inference(negate, [], [subgoal_1])).
fof(normalize_1_0, plain, ~ p & q,
```

TSTP Derivation given by `Metis` II

```
    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])).
```

```
p, q, a, goal, subgoal₀, subgoal₁ : Prop

-- Axiom.
a = p ∧ q

-- Premise.
Γ : Ctxt
Γ = [ a ]

-- Conjecture.
goal = q ∧ p

-- Subgoals.
subgoal₀ = q
subgoal₁ = q ⇒ p
```

Problem $p \wedge q \vdash q \wedge p$

Reconstructed Proof

```
a : Prop
a = p ∧ q

subgoal₀ : Prop
subgoal₀ = q

proof₀ : Γ ⊢ subgoal₀
proof₀ =
  (RAA
    (atp-simplify ⊥
      (assume {Γ = Γ}
        (¬ subgoal₀))
      (atp-conjunct q
        (atp-canonicalize (p ∧ q)
          (weaken (¬ subgoal₀)
            (assume {Γ = ∅} a))))))
```

```
subgoal₁ : Prop
subgoal₁ = q ⇒ p

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

```
-- Premise.
Γ = [ a ]

-- Conjecture.
goal = q ∧ p

-- Subgoals.
subgoal₀ = q
subgoal₁ = q ⇒ p

-- Proof
proof₀ : Γ ⊢ subgoal₀
proof₁ : Γ ⊢ subgoal₁

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

Go Back

```
$ cat problem.tptp
fof(goal, conjecture,
  ((p <=> q) <=> r) <=> (p <=> (q <=> r))).
```

```
$ 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])).
...
```

[11] https://github.com/gilith/metis/issues/2.

$$\varphi := \neg p \wedge (\neg q \Leftrightarrow \neg r) \wedge (\neg p \Leftrightarrow (\neg q \Leftrightarrow \neg r))$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\varphi}\,\text{canonicalize}}{\neg p \Leftrightarrow (\neg q \Leftrightarrow \neg r)}\,\text{conjunct} \qquad \dfrac{\dfrac{\vdots}{\varphi}\,\text{canonicalize}}{\neg q \Leftrightarrow \neg r}\,\text{conjunct} \qquad \dfrac{\dfrac{\vdots}{\varphi}\,\text{canonicalize}}{\neg p}\,\text{conjunct}}{\bot}\,\text{simplify}$$

Bug in the Printing of the Proof

`Metis` v2.3 (release 20161108)

$$\varphi := \neg p \wedge (\neg q \Leftrightarrow \neg r) \wedge (\neg p \Leftrightarrow (\neg q \Leftrightarrow \neg r))$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\varphi}\,\text{canonicalize}}{\neg p \Leftrightarrow (\neg q \Leftrightarrow \neg r)}\,\text{conjunct} \qquad \dfrac{\dfrac{\vdots}{\varphi}\,\text{canonicalize}}{\neg q \Leftrightarrow \neg r}\,\text{conjunct} \qquad \dfrac{\dfrac{\vdots}{\varphi}\,\text{canonicalize}}{\neg p}\,\text{conjunct}}{\bot}\,\text{simplify}$$

The bug was caused by the conversion of `Xor` sets to `Iff` lists. After reporting this, Hurd fixed the printing of `canonicalize` inference rule

$$\varphi := \neg p \wedge (\neg q \Leftrightarrow \neg r) \wedge (\neg p \Leftrightarrow (\neg q \Leftrightarrow r))$$

Go Back

SledgeHammer

(**Paulson2007**)

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

Integrating `Waldmeister` into `Agda`

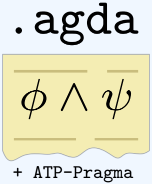
(**Foster2011**)

- Framework for a integration between `Agda` and ATPs
  - Equational Logic
  - Reflection Layers
- Source code is not available[12]

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

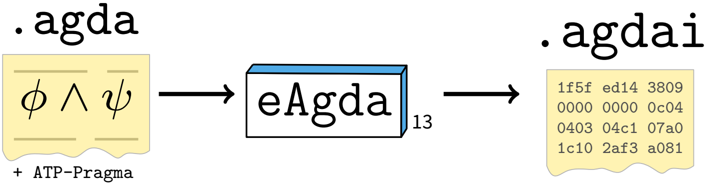
At the moment, the communication between Agda and the ATPs is unidirectional because the ATPs are being used as oracles (Sicard-Ramírez, Bove, and Dybjer, 2015)

```
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

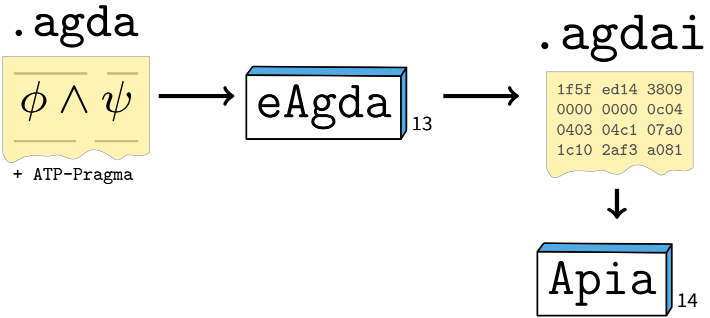
Proving First-Order theorems written in Agda using automatic theorem provers for First-Order Logic



[13] https://github.com/asr/eagda.

Related Work: Apia

Proving First-Order theorems written in Agda using automatic theorem provers for First-Order Logic

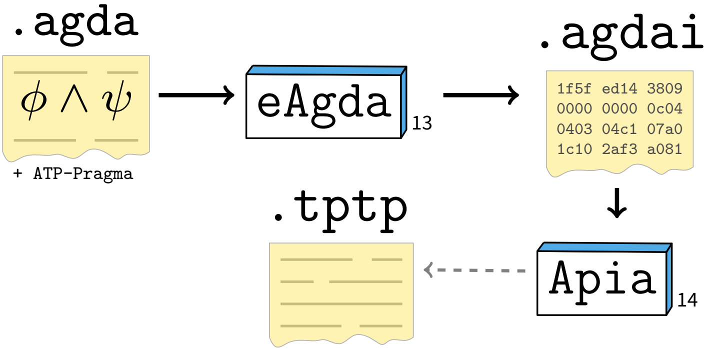


[13] https://github.com/asr/eagda.

[14] https://github.com/asr/apia.

Related Work: Apia

Proving First-Order theorems written in Agda using automatic theorem provers for First-Order Logic

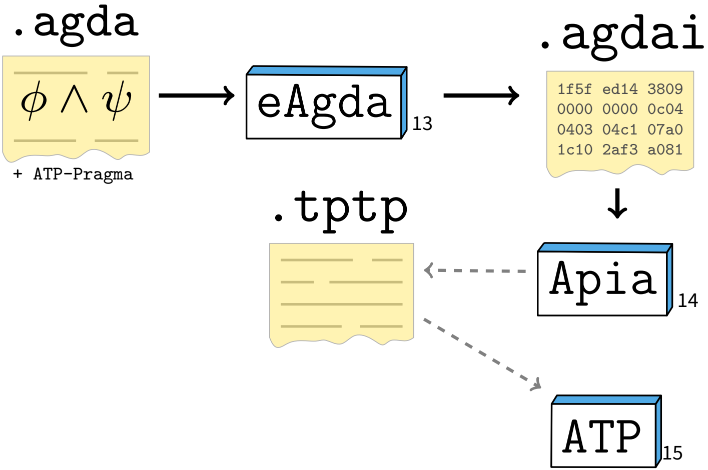


[13] https://github.com/asr/eagda.

[14] https://github.com/asr/apia.

Related Work: Apia

Proving First-Order theorems written in Agda using automatic theorem provers for First-Order Logic



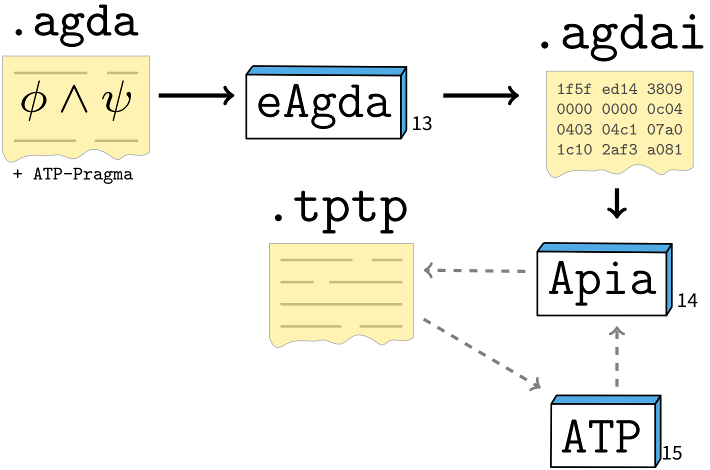
13 https://github.com/asr/eagda.

14 https://github.com/asr/apia.

15 http://github.com/jonaprieto/online-atps.

Related Work: Apia

Proving First-Order theorems written in Agda using automatic theorem provers for First-Order Logic



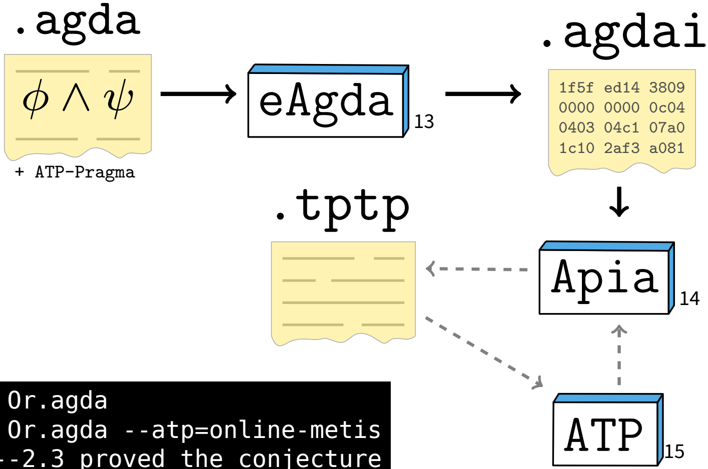
[13] https://github.com/asr/eagda.

[14] https://github.com/asr/apia.

[15] http://github.com/jonaprieto/online-atps.

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
```

[13] https://github.com/asr/eagda.

[14] https://github.com/asr/apia.

[15] http://github.com/jonaprieto/online-atps.

TODO

- Complete implementation for `simplify` inference[16]
- Complete implementation for `canonicalize` inference: what normal form use to transform the formulas
- Complete implementation for Splitting a goal in a list of subgoals

[16] `https://github.com/gilith/metis/issues/3`.

Contributions

Name	References
`Agda-Metis`	(Prieto-Cubides, 2017c)
`Agda-Prop`	(Prieto-Cubides, 2017a)
`Athena`	(Prieto-Cubides, 2017b)
`OnlineATPs`	(**OnlineATPs**)
`Prop-Pack`	(**ProPack**)

Future Work

- Integration with `Apia`
- Support First-Order Logic with Equality
- Support another prover like `EProver` or `Vampire`

References I

- Hurd, Joe (2003). "First-order Proof Tactics In Higher-order Logic Theorem Provers". In: *Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports*, pp. 56–68. URL: http://www.gilith.com/research/papers.
- Prieto-Cubides, Jonathan (2017a). *A Library for Classical Propositional Logic in Agda*. URL: https://doi.org/10.5281/zenodo.398852.
- – (2017b). *A Translator Tool for Metis Proofs in Haskell*. URL: https://doi.org/10.5281/zenodo.437196.
- – (2017c). *Metis Prover Reasoning for Propositional Logic in Agda*. URL: https://doi.org/10.5281/zenodo.398862.
- Sicard-Ramírez, Andrés, Ana Bove, and Peter Dybjer (2015). "Reasoning about Functional Programs by Combining Interactive and Automatic Proofs". PhD thesis. Universidad de la Rep{ú}blica. URL: https://www.colibri.udelar.edu.uy/handle/123456789/4715.

`Metis` Inference Rules

$$\frac{}{C}\ \text{axiom}$$

$$\frac{}{L \vee \neg L}\ \text{assume}\ L$$

$$\frac{C}{\sigma C}\ \text{subst}\ \sigma$$

$$\frac{L \vee C \qquad \neg L \vee D}{C \vee D}\ \text{resolve}\ L$$

$$\frac{}{t = t}\ \text{refl}\ t$$

$$\frac{}{\neg(L[p] = t) \vee \neg L \vee L[p \mapsto t]}\ \text{eq}\ L\ p\ t$$

Go Back