Proof Reconstruction in Classical Propositional Logic

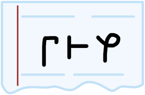
(work in progress)

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

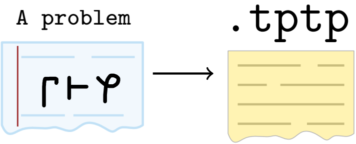
Master in Applied Mathematics
Universidad EAFIT
Medellín, Colombia

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

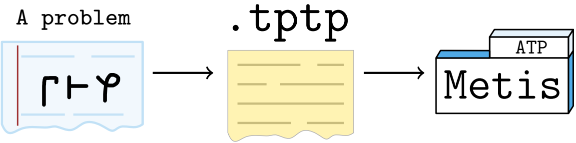
A problem



Proof Reconstruction: Overview



Proof Reconstruction: Overview

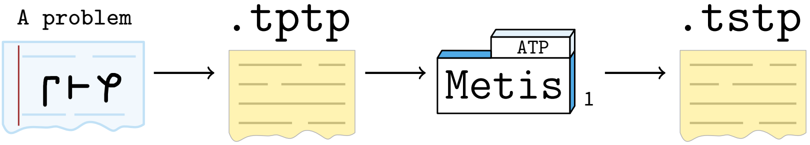


[1]

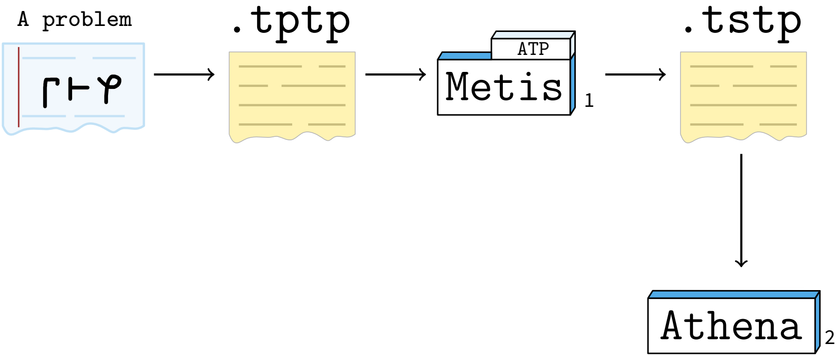
---

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

Proof Reconstruction: Overview

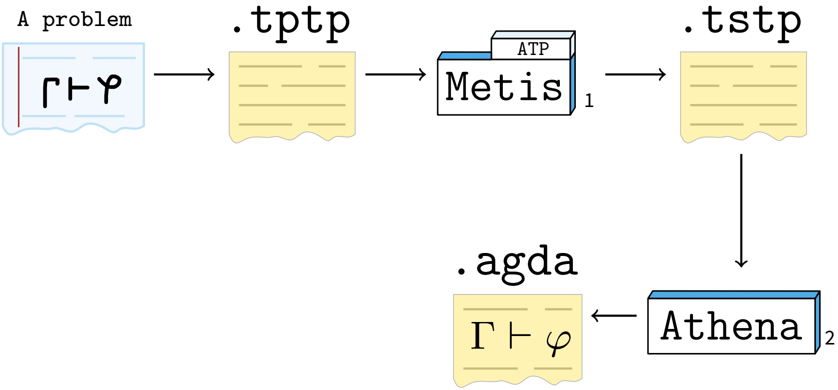


[1] http://www.gilith.com/software/metis.



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

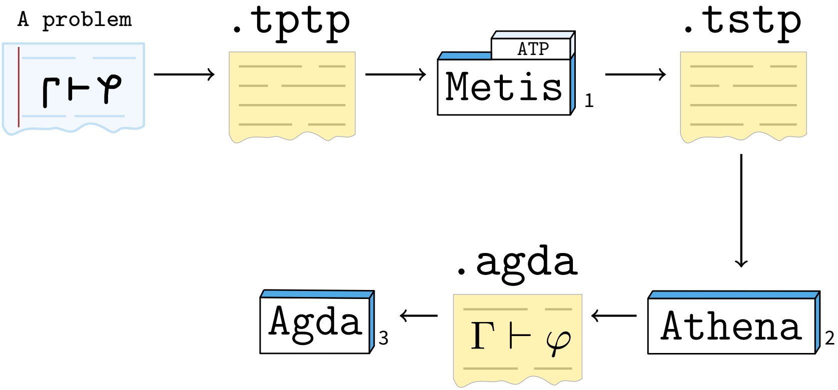
[2] http://github.com/jonaprieto/athena.



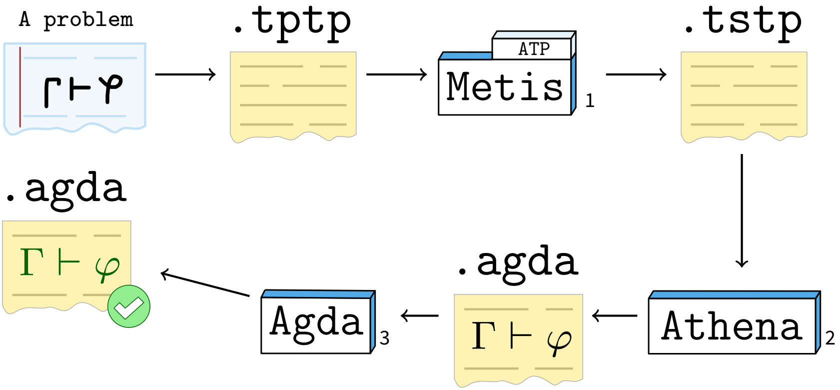
[1]http://www.gilith.com/software/metis.

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

Proof Reconstruction: Overview



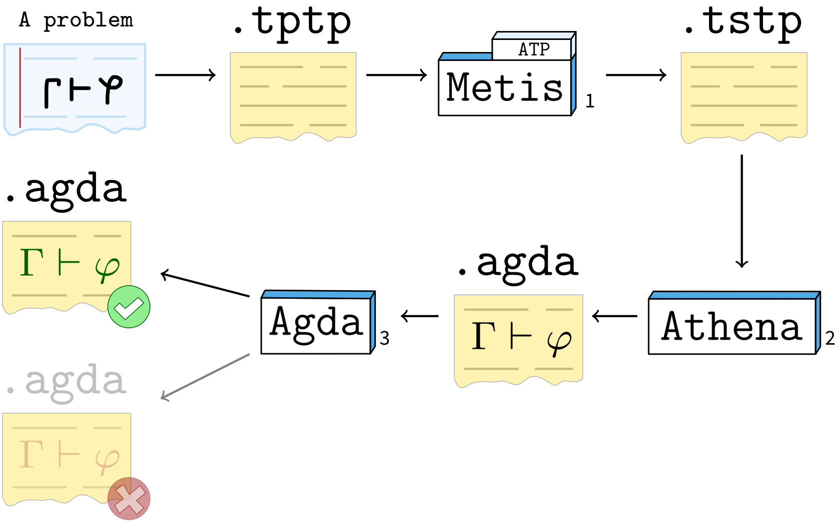
[1] http://www.gilith.com/software/metis.
[2] http://github.com/jonaprieto/athena.
[3] http://github.com/agda/agda.



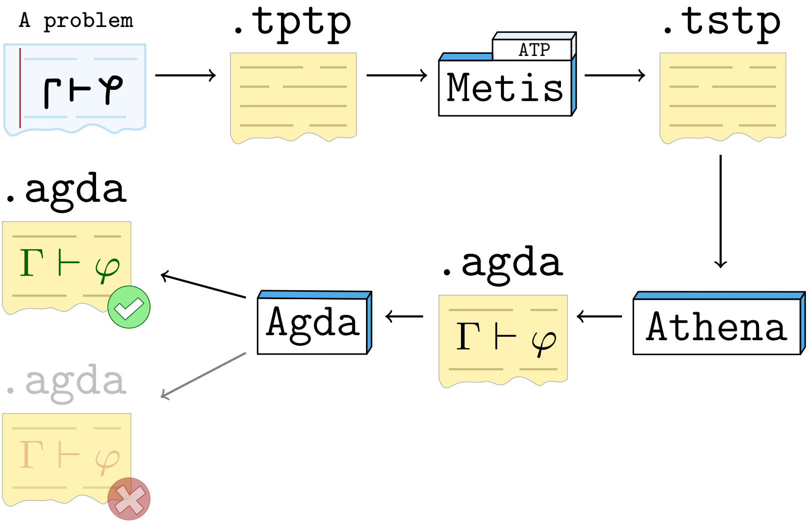
[1] http://www.gilith.com/software/metis.

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

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



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


A problem
$\Gamma \vdash \varphi$
.tptp
ATP
Metis
.tstp
.agda
$\Gamma \vdash \varphi$
Agda
.agda
$\Gamma \vdash \varphi$
Athena
.agda
$\Gamma \vdash \varphi$

TPTP Syntax

Thousands of Problems for Theorem Provers

.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
`formula` formula in TPTP format

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

Go Back

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

`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/`.

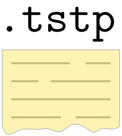
TSTP derivations by `Metis` exhibit these inferences[6]

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 Syntax



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

TSTP Example

- 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])).
```

Go Back

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

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

Go Back

Athena [10]

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

- Parsing of TSTP language[8,9]
- 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.

[9] https://github.com/jonaprieto/tstp2agda.

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

Go Back

Agda-Prop Library [11]

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

```
data Prop : 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
```

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

- 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, ...
```

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

`Agda-Metis` Library [13]

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	`atp-strip`

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

- Definition

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

- Function

```
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

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

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

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 ∨ ψ}  _ = id
atp-conjunct {_} {φ ⇒ ψ}  _ = id
atp-conjunct {_} {φ ⇔ ψ}  _ = id
atp-conjunct {_} {¬ φ}    _ = id
```

Go Back

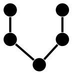
Agda Code Example
Generated by Athena Tool

- 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

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



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])).
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])).
```

```
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)
```

```
a : Prop
a = (p ∧ q)

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))))))))
```

```
-- 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

Bug[15] in the Printing of the Proof

Metis v2.3 (release 20161108)

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

15 https://github.com/gilith/metis/issues/2.

Bug in the Printing of the Proof

`Metis` v2.3 (release 20161108)

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

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

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

(Paulson and Susanto, 2007)

- `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`

(Foster and Struth, 2011)

- Framework for a integration between `Agda` and ATPs
  - Equational Logic
  - Reflection Layers
- Source code is not available[16]

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

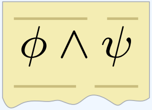
Related Work: Apia

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

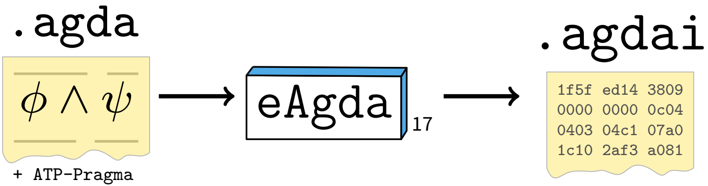
At the moment, the communication between Agda and the ATPs is unidirectional because the ATPs are being used as oracles (Sicard-Ramírez, Bove, and Dybjer, 2015)

```
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 #-}
```

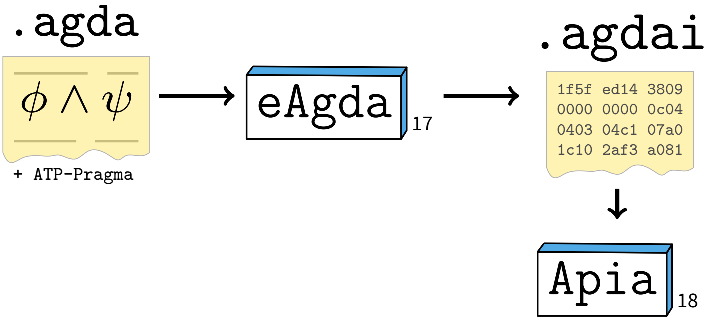
.agda



+ ATP-Pragma

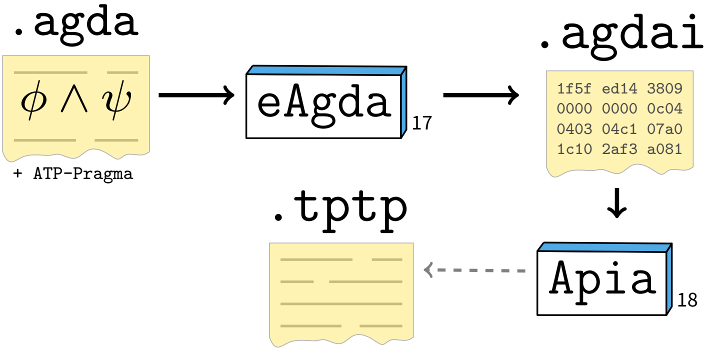


[17] https://github.com/asr/eagda.



[17] https://github.com/asr/eagda.
[18] https://github.com/asr/apia.

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

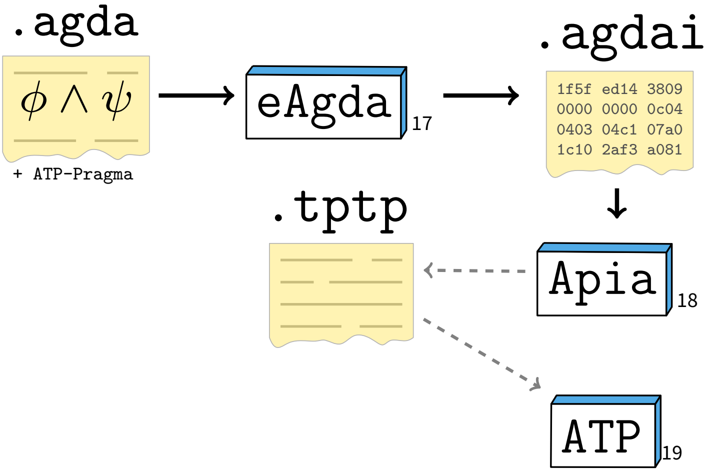


[17] https://github.com/asr/eagda.

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

Related Work: Apia

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



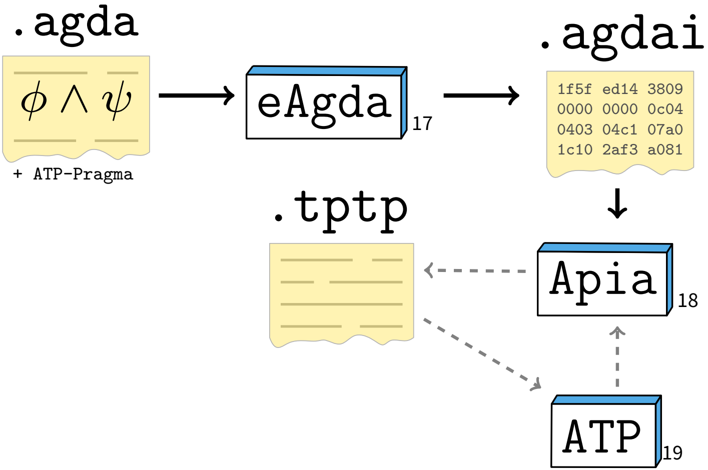
[17] https://github.com/asr/eagda.

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

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

Related Work: Apia

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



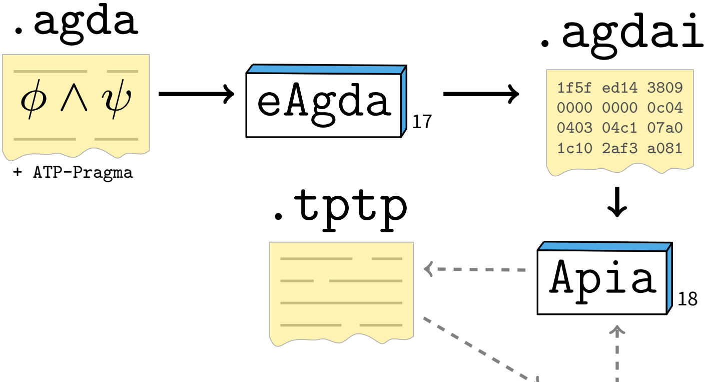
17 https://github.com/asr/eagda.

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

19 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
```

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

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

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

- Complete implementation for `simplify` inference[20]
- Complete implementation for `canonicalize` inference: what normal form use to transform the formulas
- Complete implementation for Splitting a goal in a list of subgoals

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

Contributions

Name	Purpose
`Agda-Metis`	Implementation of `Metis` inference rules
`Agda-Prop`	Syntax and theorems of Classical Propositional Logic
`Athena`	Translator for `Metis` `TSTP` files to `Agda`
`OnlineATPs`	Client to use ATPs from `SystemOnTPTP` of `TPTP` World
`Prop-Pack`	Collection of `TPTP` problems to test `Athena`

Future Work

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

- 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.
- 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.
- 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.

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

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

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

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

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

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

Go Back