The Alethe Proof Format

An Overview

Hans-Jörg Schurr

May 12 2023

Structure

1. Small number of slides.
2. Interactive exploration.
   - I only have slides for $t < t_{allocated}$

What is Alethe?

Alethe is …

…a format to represent derivations of the empty clause from an SMT problem.

- A language (think TSTP)
  and a collection of proof rules.
- Ongoing work, but there are multiple users!

`Catability`

- Alethe is a language for machines, but
- when a human runs `cat` on an Alethe file they should not be shocked.

Catability

- Alethe is a language for machines, but
- when a human runs `cat` on an Alethe file they should not be shocked.

How?

- Follow SMT-LIB ideas.
- Formulas are SMT-LIB formulas + choice.
- Proof-appropriate commands.
- Reuse other ideas, such as annotations.

Some History

A long time ago

- For veriT: EUF, LIRA, QF_
- First: Ad-hoc 📅 2006
- Later: Redesigned 📅 2011
- Syntax changed over time

Soon after

- SMTCoq one of the first users
- Verified checker 📅 2011
- Base for automation in Coq 📅 2017, now

Some History

A long time ago

- For veriT: EUF, LIRA, QF_
- First: Ad-hoc 📅 2006
- Later: Redesigned 📅 2011
- Syntax changed over time

Soon after

- SMTCoq one of the first users
- Verified checker 📅 2011
- Base for automation in Coq 📅 2017, now

Recently

- Support for reasoning with bound variables 📅 2017, 2020
- Isabelle/HOL integration 📅 2021, now
- cvc5 support 📅 2021
- Proof checker 📅 2022

💀 Proofonomicon

Now!

😇 Speculative Specification

🪶 It's now Alethe!

Users

Producers

- veriT
  - \+ Stable
  - \+ Well documented
  - – Exposes internals
  - – Limited
- cvc5
  - \+ Powerful
  - \+ Principled
  - – Undocumented
  - o Rewrites

Users

Producers

- veriT
  - \+ Stable
  - \+ Well documented
  - – Exposes internals
  - – Limited
- cvc5
  - \+ Powerful
  - \+ Principled
  - – Undocumented
  - o Rewrites

Consumers

- Carcara
  - Proof checker and elaborator
  - Fast
  - Good feature coverage
- Isabelle/HOL
  - Alethe powered tactic
  - excelent veriT support
  - ongoing for cvc5
- SMTCoq
  - translates to an internal format
  - ongoing

Resources

- Material on `https://schurr.io`
- **Documentation – `https://gitlab.uliege.be/verit/alethe`**
- Checker – `https://github.com/ufmg-smite/carcara`
- veriT – `http://www.verit-solver.org`
- cvc5 – `https://cvc5.github.io`

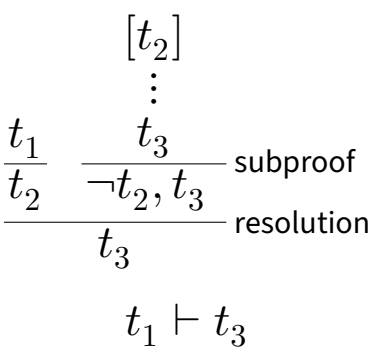
Alethe Proofs: Basic Structure

$$\dfrac{\dfrac{\dfrac{t_2}{t_3}}{\vdots}\quad t_1 \quad \neg t_1}{\bot}\ \text{resolution}$$

$$t_1, t_2 \vdash \bot$$

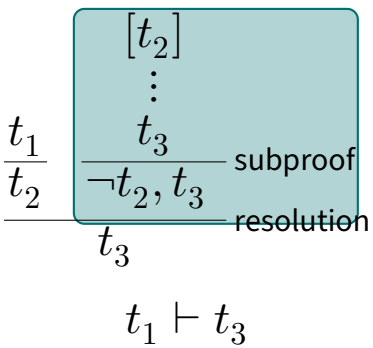
```
(assume a0 t1)
(assume a1 t2)
(step  s1 (cl t3)
      :premises (a1)      :rule rule1)
...
(step s20 (cl (not t1))
      :premises (s19)     :rule rule2)
(step s21 (cl )
      :premises (a0 s20) :rule resolution)
```

Alethe Proofs: Subproofs With Assumptions




```
(assume a0   t1)
(step s1 (cl t2)
      :premises (a0)    :rule rule1)
(anchor :step s2)
  (assume s2.a1      t2)
  ...
  (step   s2.s10 (cl t3)
      :premises (s2.s9) :rule rule2)
(step s2 (cl (not t2) t3) :rule subproof)
(step s3 (cl t3)
      :premises (s1 s2) :rule resolution)
```

Alethe Proofs: Subproofs With Assumptions




```
(assume a0   t1)
(step s1 (cl t2)
      :premises (a0)    :rule rule1)
(anchor :step s2)
  (assume s2.a1      t2)
  ...
  (step   s2.s10 (cl t3)
      :premises (s2.s9) :rule rule2)
(step s2 (cl (not t2) t3) :rule subproof)
(step s3 (cl t3)
      :premises (s1 s2) :rule resolution)
```

Alethe Grammar

⟨proof⟩ := ⟨proof_command⟩*

⟨proof_command⟩ := (assume ⟨symbol⟩ ⟨proof_term⟩)
| (step ⟨symbol⟩ ⟨clause⟩ :rule ⟨symbol⟩
⟨premises_annotation⟩?
⟨context_annotation⟩? ⟨attribute⟩*)
| (anchor :step ⟨symbol⟩
⟨args_annotation⟩? ⟨attribute⟩*)
| (define-fun ⟨function_def⟩)

⟨clause⟩ := (cl ⟨proof_term⟩*)

⟨proof_term⟩ := ⟨term⟩ extended with
(choice (⟨sorted_var⟩) ⟨proof_term⟩)

⟨premises_annotation⟩ := :premises (⟨symbol⟩+)

⟨args_annotation⟩ := :args (⟨step_arg⟩+)

⟨step_arg⟩ := ⟨symbol⟩|(⟨symbol⟩ ⟨proof_term⟩)

⟨context_annotation⟩ := :args (⟨context_assignment⟩+)

⟨context_assignment⟩ := (⟨sorted_var⟩)
| (:= ⟨symbol⟩ ⟨proof_term⟩)

Alethe Proofs: Reasoning With Binders

$$\dfrac{\dfrac{\dfrac{}{y, x \mapsto y \rhd \quad x = y}\ \text{refl}}{y, x \mapsto y \rhd f(x) = f(y)}\ \text{cong}}{\forall x.\, f(x) = \forall y.\, f(y)}\ \text{bind}$$

$$\vdash \forall x.\, f(x) = \forall y.\, f(y)$$

```
(anchor :step s2 :args ((:= (x S) y)))
  (step s2.s1 (cl (= x y)) :rule refl)
  (step s2.s2 (cl (= (f x) (f y)))
                              :rule cong)
(step s2 (cl (= (forall ((x S)) (f x))
                (forall ((y S)) (f y)))
                              :rule bind)
```

Contexts

Definition

Context A possibly empty list $c_1, \ldots, c_l$.
Each element is either a variable-term tuple denoted $x_i \mapsto t_i$ ora variable x_i.

Contexts

Definition

Context A possibly empty list $c_1, \ldots, c_l$.
Each element is either a variable-term tuple denoted $x_i \mapsto t_i$ ora variable x_i.

- The first case is a mapping.
- The second case shadows the mapping for x_i.
- Every context Γ induces a capture-avoiding substitution $\mathit{subst}(\Gamma)$.

Contexts

Definition

Context A possibly empty list $c_1, \ldots, c_l$.
Each element is either a variable-term tuple denoted $x_i \mapsto t_i$ ora variable x_i.

- The first case is a mapping.
- The second case shadows the mapping for x_i.
- Every context Γ induces a capture-avoiding substitution $\mathit{subst}(\Gamma)$.

1. if $\Gamma = \epsilon$, then $\mathit{subst}(\Gamma)$ is identity.
2. $\mathit{subst}(c_1, \ldots, c_{n-1}, x_n \mapsto t_n) = \mathit{subst}(c_1, \ldots, c_{n-1}) \circ \{x_n \mapsto t_n\}$.
3. $\mathit{subst}(c_1, \ldots, c_{n-1}, x_n)$ is $\mathit{subst}(c_1, \ldots, c_{n-1})$, but x_n maps to x_n.

Things We Do With Contexts

$$\frac{\mathit{subst}(\Gamma)(t) \text{ equal to } u \text{ up to } \alpha\text{-eq.}}{\Gamma \rhd \quad t = u}\ \text{refl}$$

$$\frac{y, x \mapsto y \rhd \varphi = \psi}{\forall x.\, \varphi = \forall y.\, \psi}\ \text{bind}$$

$$\frac{x \mapsto \epsilon x.\, \varphi \rhd \varphi = \psi}{\exists x.\, \varphi = \psi}\ \text{sko\_ex}$$

Alethe Grammar

⟨proof⟩ := ⟨proof_command⟩*

⟨proof_command⟩ := (assume ⟨symbol⟩ ⟨proof_term⟩)
| (step ⟨symbol⟩ ⟨clause⟩ :rule ⟨symbol⟩
⟨premises_annotation⟩?
⟨context_annotation⟩? ⟨attribute⟩*)
| (anchor :step ⟨symbol⟩
⟨args_annotation⟩? ⟨attribute⟩*)
| (define-fun ⟨function_def⟩)

⟨clause⟩ := (cl ⟨proof_term⟩*)

⟨proof_term⟩ := ⟨term⟩ extended with
(choice (⟨sorted_var⟩) ⟨proof_term⟩)

⟨premises_annotation⟩ := :premises (⟨symbol⟩+)

⟨args_annotation⟩ := :args (⟨step_arg⟩+)

⟨step_arg⟩ := ⟨symbol⟩|(⟨symbol⟩ ⟨proof_term⟩)

⟨context_annotation⟩ := :args (⟨context_assignment⟩+)

⟨context_assignment⟩ := (⟨sorted_var⟩)
| (:= ⟨symbol⟩ ⟨proof_term⟩)

Where We Are Now

Now

🪛 You can build things with it!

🪨 The language is stable.

💅 The proof rules need polish.

Where We Are Now

Now

🪛 You can build things with it!

🪨 The language is stable.

💅 The proof rules need polish.

Soon

📈 How to handle rule growth?

👫 Better way for Skolemization and friends?

🧐 What about SMT-LIB 3?