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 …

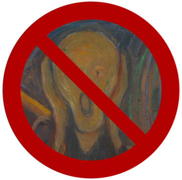
…a format to represent derivations of the empty clause from an SMT problem.

- A language (think TSTP)

  and a collection of proof rules.
- Ongoing work, but there are multiple users!

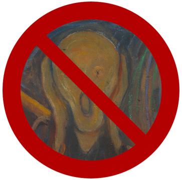
Catability

- 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 `https://homepage.cs.uiowa.edu/~hschrr/alethe.tar.gz`
- **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

$$\dfrac{t_2}{t_3}$$

$$\vdots$$

$$\dfrac{t_1 \qquad \neg t_1}{\bot}\ \text{resolution}$$

$$t_1, t_2 \vdash \bot$$

```
(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

$$\dfrac{\dfrac{t_1}{t_2} \qquad \dfrac{\begin{array}{c}[t_2]\\ \vdots \\ t_3\end{array}}{\neg t_2, t_3}\ \text{subproof}}{t_3}\ \text{resolution}$$

$$t_1 \vdash t_3$$

```
(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

$$
\dfrac{\dfrac{t_1}{t_2} \qquad \dfrac{\begin{array}{c}[t_2]\\ \vdots \\ t_3\end{array}}{\neg t_2, t_3}\ \text{subproof}}{t_3}\ \text{resolution}
$$

$$t_1 \vdash t_3$$

```
(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

$$
\begin{array}{rcl}
\langle\texttt{proof}\rangle & := & \langle\texttt{proof\_command}\rangle^{*} \\
\langle\texttt{proof\_command}\rangle & := & \texttt{(assume}\ \langle\texttt{symbol}\rangle\ \langle\texttt{proof\_term}\rangle\ \texttt{)} \\
& \mid & \texttt{(step}\ \langle\texttt{symbol}\rangle\ \langle\texttt{clause}\rangle\ \texttt{:rule}\ \langle\texttt{symbol}\rangle \\
& & \quad \langle\texttt{premises\_annotation}\rangle^{?} \\
& & \quad \langle\texttt{context\_annotation}\rangle^{?}\ \langle\texttt{attribute}\rangle^{*}\ \texttt{)} \\
& \mid & \texttt{(anchor :step}\ \langle\texttt{symbol}\rangle \\
& & \quad \langle\texttt{args\_annotation}\rangle^{?}\ \langle\texttt{attribute}\rangle^{*}\ \texttt{)} \\
& \mid & \texttt{(define-fun}\ \langle\texttt{function\_def}\rangle\ \texttt{)} \\
\langle\texttt{clause}\rangle & := & \texttt{(cl}\ \langle\texttt{proof\_term}\rangle^{*}\ \texttt{)} \\
\langle\texttt{proof\_term}\rangle & := & \langle\texttt{term}\rangle\ \text{extended with} \\
& & \texttt{(choice (}\ \langle\texttt{sorted\_var}\rangle\ \texttt{)}\ \langle\texttt{proof\_term}\rangle\ \texttt{)} \\
\langle\texttt{premises\_annotation}\rangle & := & \texttt{:premises (}\ \langle\texttt{symbol}\rangle^{+}\ \texttt{)} \\
\langle\texttt{args\_annotation}\rangle & := & \texttt{:args (}\ \langle\texttt{step\_arg}\rangle^{+}\ \texttt{)} \\
\langle\texttt{step\_arg}\rangle & := & \langle\texttt{symbol}\rangle \mid \texttt{(}\ \langle\texttt{symbol}\rangle\ \langle\texttt{proof\_term}\rangle\ \texttt{)} \\
\langle\texttt{context\_annotation}\rangle & := & \texttt{:args (}\ \langle\texttt{context\_assignment}\rangle^{+}\ \texttt{)} \\
\langle\texttt{context\_assignment}\rangle & := & \texttt{(}\ \langle\texttt{sorted\_var}\rangle\ \texttt{)} \\
& \mid & \texttt{(:=}\ \langle\texttt{symbol}\rangle\ \langle\texttt{proof\_term}\rangle\ \texttt{)}
\end{array}
$$

Alethe Proofs: Reasoning With Binders

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

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

```
(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 *subst*(Γ).

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

Things We Do With Contexts

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

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

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

Alethe Grammar

$$
\begin{array}{rcl}
\langle \texttt{proof} \rangle & := & \langle \texttt{proof\_command} \rangle^{*} \\
\langle \texttt{proof\_command} \rangle & := & \texttt{(assume } \langle \texttt{symbol} \rangle\ \langle \texttt{proof\_term} \rangle \texttt{)} \\
& \mid & \texttt{(step } \langle \texttt{symbol} \rangle\ \langle \texttt{clause} \rangle \texttt{ :rule } \langle \texttt{symbol} \rangle \\
& & \quad \langle \texttt{premises\_annotation} \rangle^{?} \\
& & \quad \langle \texttt{context\_annotation} \rangle^{?}\ \langle \texttt{attribute} \rangle^{*} \texttt{)} \\
& \mid & \texttt{(anchor :step } \langle \texttt{symbol} \rangle \\
& & \quad \langle \texttt{args\_annotation} \rangle^{?}\ \langle \texttt{attribute} \rangle^{*} \texttt{)} \\
& \mid & \texttt{(define-fun } \langle \texttt{function\_def} \rangle \texttt{)} \\
\langle \texttt{clause} \rangle & := & \texttt{(cl } \langle \texttt{proof\_term} \rangle^{*} \texttt{)} \\
\langle \texttt{proof\_term} \rangle & := & \langle \texttt{term} \rangle \text{ extended with} \\
& & \texttt{(choice (} \langle \texttt{sorted\_var} \rangle \texttt{) } \langle \texttt{proof\_term} \rangle \texttt{)} \\
\langle \texttt{premises\_annotation} \rangle & := & \texttt{:premises (} \langle \texttt{symbol} \rangle^{+} \texttt{)} \\
\langle \texttt{args\_annotation} \rangle & := & \texttt{:args (} \langle \texttt{step\_arg} \rangle^{+} \texttt{)} \\
\langle \texttt{step\_arg} \rangle & := & \langle \texttt{symbol} \rangle \mid \texttt{(} \langle \texttt{symbol} \rangle\ \langle \texttt{proof\_term} \rangle \texttt{)} \\
\langle \texttt{context\_annotation} \rangle & := & \texttt{:args (} \langle \texttt{context\_assignment} \rangle^{+} \texttt{)} \\
\langle \texttt{context\_assignment} \rangle & := & \texttt{(} \langle \texttt{sorted\_var} \rangle \texttt{)} \\
& \mid & \texttt{(:= } \langle \texttt{symbol} \rangle\ \langle \texttt{proof\_term} \rangle \texttt{)}
\end{array}
$$

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?