

Reconstruction of TLAPS proofs solved by SMT in Lambdapi

Alessio Coltellacci

Univ. Lorraine, CNRS, Inria, Loria

ICSPA



Outline

SMT translation at a glance

Formalisation overview (experimental notation)

Evaluation

Unresolved points

Future perspectives

TLA⁺ at a glance

- ▶ Specification language to design and verify reactive systems
- ▶ Systems are described as state machines

VARIABLE x
CONSTANT N
ASSUME $N \in \text{Nat}$

$$\text{Init} \triangleq \quad \wedge x = 0$$

$$\text{Next} \triangleq \quad \wedge x < N \\ \quad \wedge x' = x + 1$$

$$\text{Spec} \triangleq \text{Init} \wedge \Box[\text{Next}]_{\langle x \rangle}$$

TLAPS proof example

----- MODULE Cantor1 -----

THEOREM cantor ==

$\forall S :$

$\forall f \in [S \rightarrow \text{SUBSET } S] :$

$\exists A \in \text{SUBSET } S :$

$\forall x \in S :$

$f[x] \# A$

PROOF

<1> 1. TAKE S

<1> 2. TAKE $f \in [S \rightarrow \text{SUBSET } S]$

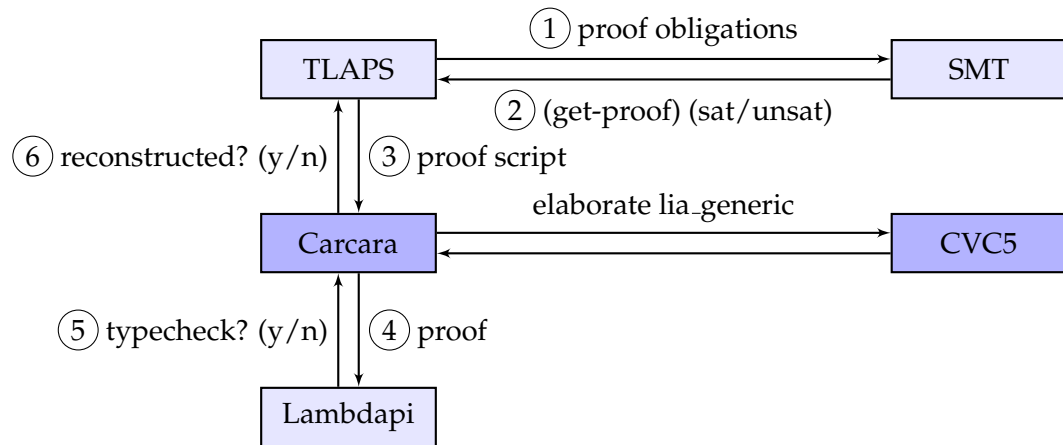
<1> 3. DEFINE $T == \{ z \in S : z \notin f[z] \}$

<1> 4. WITNESS $T \in \text{SUBSET } S$

<1> 5. TAKE $x \in S$

<1> 6. QED BY $x \in T \vee x \notin T$

Proposed solution



Simple example

```
(set-logic QF_UF)
(declare-sort U 0)
(declare-fun a () U)
(declare-fun b () U)
(declare-fun p (U) Bool)
(assert (p a))
(assert (= a b))
(assert (not (p b)))
(get-proof)
```

Its Alethe SMT proof

```
(assume a0 (p a))  
(assume a1 (= a b))  
(assume a2 (not (p b)))  
(step t1 (cl (not (= (p a) (p b))) (not (p a)) (p b)) :rule equiv_pos2)  
(step t2 (cl (= (p a) (p b))) :rule cong :premises (a1))  
(step t3 (cl (p b)) :rule resolution :premises (t1 t2 a0))  
(step t4 (cl) :rule resolution :premises (a2 t3))
```

Alethe format

Definition (Alethe step)

A proof in the Alethe language is an indexed list of step following the format:

$$j. \quad \Delta \quad \vdash \quad \varphi \quad (R; p_1 \dots p_n)[a_1, \dots, a_n]$$

With $i \in \mathbb{I}$ where \mathbb{I} is a countable infinite set of valid indices,

a formula φ ,

a rule name \mathcal{R} from a set of possible rules,

a possible empty sets $\{p_1 \dots p_n\} \subseteq \mathbb{I}$ of premises (previous steps),

a possible empty list of arguments $[a_1 \dots a_n]$ where $a_i = (x_i, t_i)$
with x_i a variable and t_i a term,

and a context Δ .

Overview of rules

1. Special rules

- * $\vdash \varphi$ (asssume)
- * $\vdash \varphi$ (hole; $p_1 \dots p_n$)[$a_1 \dots a_n$]
- * $\varphi_1 \dots \varphi_n, \psi \vdash \neg \varphi_1 \dots \neg \varphi_n \psi$
(subproof; $p_1 \dots p_n$)

2. Resolution rules

- * `th_resolution, resolution`
- * `contraction, reordering`

3. Introducing tautologies

- * $\vdash \neg(\neg\neg\varphi), \varphi$ (`not_not`)
- * $\vdash \neg(\varphi_1 \approx \varphi_2), \neg\varphi_1, \varphi_2$ (`equiv_pos2`)
- * $\vdash \neg(\varphi_1 \wedge \dots \wedge \varphi_n), \varphi_k$ (`and_pos`)

4. Linear arithmetic

- * `lia_generic, la_generic`
- * $\vdash t_1 \leq t_2 \vee t_2 \leq t_1$ (`la_totality`)

5. Quantifier handling

- * $j. \Delta, x_i \mapsto y_i \vdash \varphi \approx \varphi'$
- * $i. \vdash \forall x_1 \dots x_n, \varphi \approx \forall y_1 \dots y_n, \varphi'$ (`bind`)
- * `forall_inst`

6. Skolemization

- * `ske_ex`
- * `ske_forall`

7. Clausification rules

- * `let`
- * `distinct_elim`

8. Simplification rules

- * `and_simplify`
- * `bool_simplify`
- * `eq_simplify`
- * `sum_simplify`

Alethe proof as derivation tree

$$\begin{array}{c} \vdots \\ t_j \frac{}{\Delta'' \vdash p_1} \text{Rule}(\dots) \quad \dots \quad t_k \frac{}{\Delta' \vdash p_n} \text{Rule}(\dots) \\ t_i \frac{}{\Delta \cup \{p_1 \dots p_n\} \vdash c_1, \dots, c_n} \text{Rule}(a_1 \dots a_n) \end{array}$$

Loop back on the SMT proof example

```
(assume a0 (p a))  
(assume a1 (= a b))  
(assume a2 (not (p b)))  
(step t1 (cl (not (= (p a) (p b))) (not (p a)) (p b)) :rule equiv_pos2)  
(step t2 (cl (= (p a) (p b))) :rule cong :premises (a1))  
(step t3 (cl (p b)) :rule resolution :premises (t1 t2 a0))  
(step t4 (cl) :rule resolution :premises (a2 t3))
```

This proof can not be reconstructed directly due to coarse-grained steps e.g: pivots are not given for t3 and t4.

Carcara

- ▶ Carcara is an efficient and independent proof checker and elaborator for Alethe proofs.
- ▶ Carcara is written in Rust, a high performance language,
- ▶ implements elaboration procedures for a few important rules (ex: inferring pivots),
- ▶ it remove implicit transformations (ex: reordering clause).

Elaborated proof with Carcara

Make pivot and resolution order explicit

```
(assume a0 (p a))
(assume a1 (= a b))
(assume a2 (not (p b)))
(step t1 (cl (not (= (p a) (p b))) (not (p a)) (p b)) :rule equiv_pos2)
(step t2 (cl (= (p a) (p b))) :rule cong :premises (a1))
(step t3 (cl (p b)) :rule resolution :premises (t1 t2 a0)
  :args ((= (p a) (p b)) false (p a) false))
(step t4 (cl) :rule resolution :premises (a2 t3)
  :args ((p b) false))
```

Corresponding proof tree

$$\begin{array}{c}
 \begin{array}{c} a_0 \\ t_3 \end{array} \frac{}{\Delta \vdash p(a)} \quad \begin{array}{c} t_1 \\ t_3' \end{array} \frac{\frac{}{\Delta \vdash \neg(p(a) = p(b)), \neg p(a), p(b)} \text{equiv_pos2} \quad \frac{}{\Delta \vdash p(a) = p(b)} \text{cong}(a_1)}{\Delta \vdash \neg p(a), p(b)} \text{Resolution}(t_1, t_2) \\
 \hline
 \begin{array}{c} t_4 \end{array} \frac{\frac{}{\Delta \vdash p(b)} \text{Resolution}(a_0, t_3') \quad \frac{}{\Delta \vdash \neg p(b)} \text{Resolution}(a_2, t_3)}{\Delta \vdash \perp}
 \end{array}$$

Translate prelude

```
1 (declare-sort U 0)
2 (declare-fun a () U)
3 (declare-fun b () U)
4 (declare-fun p (U) Bool)
```

~

```
1 symbol U : TYPE;
2 rule U  $\hookrightarrow$   $\tau$  o;
3
4 symbol a : U;
5 symbol b : U;
6 symbol p : U  $\rightarrow$  Prop;
```

Translate assert/assume

```
1 (assert (p a))  
2 (assert (= a b))  
3 (assert (not (p b)))
```

```
1 (assume a0 (p a))  
2 (assume a1 (= a b))  
3 (assume a2 (not (p b)))
```

```
1 constant symbol a0 :  $\pi$  (p a  $\vee$   $\Box$ );  
2 constant symbol a1 :  
3    $\pi$  (a  $\stackrel{c}{\iff}$  b  $\vee$   $\Box$ );  
4 constant symbol a2 :  
5    $\pi$  ( $\neg^c$  (p b)  $\vee$   $\Box$ );
```


Translation of step t1 and t2

```
1 (step t1
2   (cl (not (= (p a) (p b)))
3       (not (p a))
4         (p b))
5   :rule equiv_pos2)
6 (step t2 (cl (= (p a) (p b)))
7   :rule cong :premises (a1))
8 ...
```

↔

```
1 opaque symbol pb :
2 begin
3 have t1:  $\pi$  (
4    $\neg^c ((p\ a) \iff^c (p\ b))$ 
5    $\forall (p\ a)$ 
6    $\forall (p\ b)$ 
7    $\forall \square$ )
8 {
9   apply equiv_pos2;
10 };
11 have t2:  $\pi$  (
12    $p\ a \iff^c p\ b$ 
13    $\forall \square$ )
14 {
15   apply  $\forall_{i1}^c$ ;
16   apply cong p a1;
17 };
```

Translation of step t3

```
1 (step t3 (cl (p b))
2   :rule resolution
3   :premises (t1 t2 a0))
4   :args (
5     (= (p a) (p b)) false
6     (p a) false
7   ))
```

~>

```
1 ...
2 have t3 :  $\dot{\pi}$  ((p b)  $\vee$   $\Box$ ) {
3   have t1_t2 :  $\dot{\pi}$  (
4     ( $\neg^c$  ((p a)))
5      $\vee$  (p b)  $\vee$   $\Box$ )
6   {
7     apply resolution t1 t2;
8   };
9   have t1_t2_a0 :  $\dot{\pi}$  ((p b)  $\vee$   $\Box$ )
10  {
11    apply resolution t1_t2 a0;
12  };
13  apply t1_t2_a0;
14};
```

Translation of step t4

```
1 (step t4 (c1)
2   :rule resolution
3   :premises (a2 t3)
4   :args ((p b) false))
```

~

```
1 ...
2 have t4 :  $\neg$  □ {
3   have a2_t3 :  $\neg$  □ {
4     apply resolution a2 t3;
5   };
6   apply a2_t3;
7 };
8 apply t4;
9 proofterm;
10 end;
```

Proof term obtained

```
1 resolution_r (p b) _ _  
2   a2  
3   (resolution_r (p a) _ _  
4     (resolution_r (p a  $\iff^c$  p b) _ _  
5       equiv_pos2  
6       ( $\forall_{i1}^c$  (cong p ( $\pi$  a1))))  
7   )  
8   a0  
9 )
```

Obtained with the `Lambdapi` tactic `proofterm`.

Supported rules overview

1. Special rules ✓

- * $\vdash \varphi$ `asssume`
- * $\vdash \varphi$ `(hole;p1...pn)[a1...an]`
- * $\varphi_1 \dots \varphi_n, \psi \text{ i. } \vdash \neg \varphi_1 \dots \neg \varphi_n \psi$
`(subproof; p1...pn)`

2. Resolution rules ✓

- * `th_resolution, resolution`
- * ~~`contraction, reordering`~~

3. Introducing tautologies ✓

- * $\vdash \neg(\neg\neg\varphi) \vee \varphi$ `(not_not)`
- * $\vdash \neg(\varphi_1 \approx \varphi_2) \vee \neg\varphi_1 \vee \varphi_2$ `(equiv_pos2)`
- * $\vdash \neg(\varphi_1 \wedge \dots \wedge \varphi_n) \vee \varphi_k$ `(and_pos)`

4. Linear arithmetic ×

- * `lia_generic, la_generic`
- * $\vdash t_1 \leq t_2 \vee t_2 \leq t_1$ `(la_totality)`

5. Quantifier handling (WIP)

- * $j.\Delta, x_i \mapsto y_i \vdash \varphi \approx \varphi'$
- * $i. \vdash \forall x_1 \dots x_n, \varphi \approx \forall y_1 \dots y_n, \varphi'$ `(bind)`
- * `forall_inst`

6. Skolemization (WIP)

- * `ske_ex`
- * `ske_forall`

7. Clausification rules (WIP)

- * `let`
- * `distinct_elim`

8. Simplification rules ×

- * `and_simplify`
- * `bool_simplify`
- * `eq_simplify`
- * `sum_simplify`

SMT translation at a glance

Formalisation overview (experimental notation)

Evaluation

Unresolved points

Future perspectives

Contexts in Alethe

Definition (Proof context)

We denote Δ_c as the Alethe proof context. It is use to reason about bound variable and store previous proved steps.

If $j. \quad x_1 \mapsto y_1, \dots, x_n \mapsto y_n \quad \vdash \quad \varphi \ y_1 \dots y_n \quad (R, p_1 \dots p_n)[x_1, \dots, x_n] \quad \text{proved}$

Then $j(\Delta \vdash \varphi \ y_1 \dots y_n \ (R; p_1 \dots p_n)[x_1 \dots x_n]) \in \Delta$,
and $x_1 \mapsto y_1, \dots, x_n \mapsto y_n \in \Delta$

Definition (Prelude context)

We denote Δ_{def} as the Alethe definition context part. In other words, it store user declare-sort and declare-fun.

Definition (Alethe context)

We set $\Delta = \Delta_c \cup \Delta_{def}$

Alethe encoding in Lambdapi

We denote $\Gamma_{\mathcal{A}}$ as the Lambdapi context with Alethe definitions.

Encoding of classical logic in $\Gamma_{\mathcal{A}}$ ¹

- ▶ The set of terms `Set` : **TYPE**
function symbol `Set → ... → Set`,
- ▶ the set of propositions `Prop` : **TYPE**
predicate symbol `Set → ... → Prop`,
- ▶ and the classical connectives $\forall^c \mid \exists^c \mid \wedge^c \mid \neg^c \mid \vee^c \mid \Rightarrow^c \mid \Longleftrightarrow^c \mid \epsilon$.
- ▶ $\pi^c := \neg\neg p$ the definition of classical proofs of proposition p ,
- ▶ the axioms of classical natural deduction system \mathcal{NK} and some lemmas.
- ▶ We have quantification on propositions/impredicativity (e.g. $\forall p, p \Rightarrow p$) :
symbol `o` : `Set`;
rule `to` \hookrightarrow `Prop`;

¹Classical logic definitions is based on Lambdapi Stdlib.

Alethe encoding in Lambdapi

We encode Alethe rule \mathcal{R} as corresponding **symbol** \mathcal{R} .

Clause

Alethe treats clause as Set causing a canonical representation issue. We define clause as list in a Church encoding style to solve it:

```
1 constant symbol Clause : TYPE;  
2 symbol  $\square$  : Clause; // Nil  
3 injective symbol  $\vee$  : Prop  $\rightarrow$  Clause  $\rightarrow$  Clause; // Cons x l  
4 sequential symbol ++ : Clause  $\rightarrow$  Clause  $\rightarrow$  Clause;  
5 rule (  $\$x \vee \$l$  ) ++  $\$m \hookrightarrow \$x \vee ( \$l ++ \$m )$   
6 with  $\square ++ \$m \hookrightarrow \$m$ ;  
7  
8  
9 symbol Clause_ind:  $\prod P: (Clause \rightarrow Prop), \prod l,$   
10  $\pi (P \square) \rightarrow (\prod x: Prop, \prod l: Clause, \pi (P l) \rightarrow$   
11  $\pi (P (x \vee l))) \rightarrow \pi (P l);$ 
```

Clause concatenation has a unique canonical form

With clause as disjunction

$$((x_1 \vee x_2) \vee x_3) \vee (y_1 \vee y_2 \vee y_3) \rightsquigarrow (((x_1 \vee x_2) \vee x_3) \vee (y_1 \vee y_2 \vee y_3))$$

With clause with type Clause

$$((x_1 \vee x_2) \vee x_3 \vee \square) \text{ ++ } (y_1 \vee y_2 \vee y_3 \vee \square) \rightsquigarrow x_1 \vee x_2 \vee x_3 \vee y_1 \vee y_2 \vee y_3 \vee \square$$

Proof of clause

Proof of clause $(\text{cl } \varphi_1, \dots, \varphi_n)$ with $(\mathcal{R}; p_1 \dots p_n)[a_1 \dots a_n]$ in a step such as

$$j. \quad \Delta \quad \vdash \quad (\text{cl } \varphi_1, \dots, \varphi_n) \quad (\mathcal{R}; p_1 \dots p_n)[a_1, \dots, a_n]$$

are encoded as proof:

```
1 injective symbol  $\pi$  c: TYPE :=  $\pi$  (E c);  
2  
3 sequential symbol E: Clause  $\rightarrow$  Prop;  
4 rule E ( $x  $\vee$  $y)  $\hookrightarrow$  $x  $\vee^c$  (E $y)  
5 with E  $\square \hookrightarrow \perp$ ;
```

Alethe rule encoding example

The rule *not_implies1* in Alethe set of rules

$$\begin{array}{ll} i. & \vdash \neg(\varphi_1 \rightarrow \varphi_2) \quad (\dots) \\ j. & \vdash \varphi_1 \quad (\text{not_implies1}; i) \end{array}$$

is translated into:

```
1 opaque symbol not_implies_1 [ $\varphi_1$   $\varphi_2$ ] :  $\pi(\neg^c(\varphi_1 \Rightarrow^c \varphi_2)) \rightarrow \dot{\pi}(\varphi_1 \vee \Box) :=$   
2 begin  
3   assume  $\varphi_1$   $\varphi_2$   $H$ ;  
4   apply  $\vee_{i1}^c$ ;  
5   apply  $\wedge_{e1}^c$  (imply_to_and  $H$ );  
6 end;
```

Clause conversion lemma

Lemma (Equivalence between clause and disjunctions)

For any two Clause a b , we have the equivalence:

$$\llbracket a \text{ ++ } b \rrbracket \iff^c \llbracket a \rrbracket \vee^c \llbracket b \rrbracket$$

Clause resolution rule translation

Lemma (Resolution)

Given a b : Clause and a pivot x : Prop, then a premise of $(x \vee a) \in \Gamma_{\mathcal{A}}$ and a premise of $(\neg^c x \vee b) \in \Gamma_{\mathcal{A}}$ implies a clause $a ++ b$.

Lambdapi encoding:

opaque symbol resolution x a b : $\pi (x \vee a) \rightarrow \pi (\neg^c x \vee b) \rightarrow \pi (a ++ b) :=$

Translation functions

The embedding uses four functions:

- ▶ \mathcal{F} which translates first order formulas to $\Gamma_{\mathcal{A}}$ -propositions,
- ▶ \mathcal{S} which translates SMTLib sort from theory to $\Gamma_{\mathcal{A}}$ -type,
- ▶ \mathcal{T} which translates first order individual terms to $\Gamma_{\mathcal{A}}$ -terms,
- ▶ $\mathcal{C}(\Gamma, c_1 \dots c_n)$ which translates a non-empty set of commands $c_1 \dots c_n$ to typing goals $\Gamma \vdash M : N$ and terms of type $M : N$.

Function \mathcal{F}

translates first order formulas to $\Gamma_{\mathcal{A}}$ -propositions

Definition (\mathcal{F})

The definition of $\mathcal{F}(f)$ is as follows.

- ▶ if $f = cl\ x_1 \dots x_n$, then $\mathcal{F}_{\Delta}(cl\ x_1 \dots x_n) = x_1 \vee \dots \vee x_n \vee \Box$,
- ▶ if $f = a_1 \wedge \dots \wedge a_n$, then $\mathcal{F}_{\Delta}(a_1 \wedge \dots \wedge a_n) = a_1 \wedge^c \dots \wedge^c a_n \wedge^c \top$,
- ▶ if $f = a_1 \vee \dots \vee a_n$, then $\mathcal{F}_{\Delta}(a_1 \vee \dots \vee a_n) = a_1 \vee^c \dots \vee^c a_n \vee^c \perp$,
- ▶ if $f = a \approx b$ and $a\ b \in \mathbf{Bool}$, then $\mathcal{F}_{\Delta}(a \approx b) = a \iff^c b$,
- ▶ if $f = a \approx b$ and $a\ b \notin \mathbf{Bool}$, then $\mathcal{F}_{\Delta}(a \approx b) = (a = b)$,
- ▶ otherwise we are in the case $\mathcal{F}_{\Delta}(f) = f$ with all connector changes for their Corresponding classical connector \star^c .

Why this \mathcal{F} definitions for conjunctions and disjunctions ?

N-ary rules are proved by "reflexivity" proof. For example

i. $\Delta \vdash \neg(\varphi_1 \wedge \cdots \wedge \varphi_n), \varphi_k$ (*and_pos*)

with $1 \leq k \leq n$

where we have the reflexivity proof:

```
1 sequential symbol In_ $\wedge^c$ : Prop  $\rightarrow$  Prop  $\rightarrow$   $\mathbb{B}$ ;  
2 rule In_ $\wedge^c$  $x ( $h  $\wedge^c$  $tl)  $\hookrightarrow$  (eq $x $h) Bool.or (In_ $\wedge^c$  $x $tl)  
3 with In_ $\wedge^c$  $x T  $\hookrightarrow$  false;  
4  
5 symbol and_pos [ $\varphi_1 \dots \varphi_n \varphi_k$ ]:  
6    $\pi$  ((In_ $\wedge^c$   $\varphi_k$   $\varphi_1 \dots \varphi_n$ ) = true)  $\rightarrow$   $\pi$  ( $\neg^c \varphi_1 \dots \varphi_n \ \forall \varphi_k \ \forall \square$ );
```

Function $\mathcal{S}(s)$

translates SMTLib sort from theory to $\Gamma_{\mathcal{A}}$ -type

The definition of $\mathcal{S}(s)$ is as follows.

- ▶ if $s = \mathbf{Bool}$, then $\mathcal{S}(\mathbf{Bool}) = \mathit{Prop}$,
- ▶ if $s \neq \mathbf{Bool}$, then $\mathcal{S}(s) = \tau \ o$,
- ▶ if $s = f(a_1 \dots a_n)$ and $\mathit{codomain}(f) = \mathbf{Bool}$, then $\mathcal{S}(f(a_1 \dots a_n)) = f : \mathcal{S}(a_1) \rightarrow \dots \rightarrow \mathcal{S}(a_n) \rightarrow \mathbf{Prop}$,
- ▶ if $s = f(a_1 \dots a_n)$ and $\mathit{codomain}(f) \neq \mathbf{Bool}$, then $\mathcal{S}(f(a_1 \dots a_n)) = f : \mathcal{S}(a_1) \rightarrow \dots \rightarrow \mathcal{S}(a_n) \rightarrow \mathbf{Set}$,

Function $\mathcal{T}(t)$

Definition

The definition of $\mathcal{T}(t)$ is a direct shallow embedding of t in corresponding term t in $\Gamma_{\mathcal{A}}$ (variables, functions, constants).

Function $\mathcal{C}(\Gamma, -)$

translates Alethe commands

Definition

The function $\mathcal{C}(\Gamma, i. \Delta \vdash \varphi \quad (R; p_1 \dots p_n)[a_1 \dots a_n]) \rightarrow \Gamma'$ translates, in a given context Γ , a step i into a judgement (typing goal) $\Gamma \vdash i : \mathcal{F}(\varphi)$ with a term M along satisfying the goal. It returns a new context Γ' with $i \in \Gamma'$. The definition of \mathcal{C} is defined recursively on R .

Notation

The notation $tac(A)[\Gamma \vdash \varphi]$ means applying the `Lambdapi` tactic `tac` (with argument A) to the judgement $\Gamma \vdash \varphi$ and making the judgements (subgoals) generated by the tactic be the premises of the rule.

Example 1: *equiv_pos2* case

We translate a step i using the alethe rule *equiv_pos2*:

$$i. \quad \vdash \neg(\varphi_1 \approx \varphi_2), \neg\varphi_1, \varphi_2 \qquad (equiv_pos2)$$

with given a context Γ as

$$\mathcal{C}(\Gamma, i. \vdash \neg(\varphi_1 \approx \varphi_2), \neg\varphi_1, \varphi_2 (equiv_pos2)) = apply(equiv_pos2)[\Gamma \vdash i : \mathcal{F}(\neg(\varphi_1 \approx \varphi_2), \neg\varphi_1, \varphi_2)]$$

Example 2: *cong* case

We translate a step k using the alethe rule *cong*

$$\begin{array}{ll} i. & \Delta \quad \vdash t_1 \approx u_1 \quad (\dots) \\ & \vdots \\ j. & \Delta \quad \vdash t_n \approx u_n \quad (\dots) \\ k. & \Delta \quad \vdash f t_1 \dots t_n \approx f u_1 \dots u_n \quad (cong; p_1 \dots p_n) \end{array}$$

as:

if $\text{codomain}(f) \in \mathbf{Bool}$, then $\mathcal{C}(\Gamma, i. \Delta \vdash f t_1 \dots t_n \approx f u_1 \dots u_n (cong; p_1 \dots p_n))$
= $cong2_f(f p_1 \dots cong2_f(f p_{n-1} p_n))[\Gamma \vdash k : \mathcal{F}(f t_1 \dots t_n \approx f u_1 u_n)]$

otherwise $f_equal_n(f p_1 \dots p_n)[\Gamma \vdash k : \mathcal{F}(f t_1 \dots t_n \approx f u_1 u_n)]$,

with $\mathcal{C}(\Gamma \setminus \{i\}, i. \Delta \vdash t_1 \approx u_1 (\dots)) \in \Gamma$ and

\vdots

with $\mathcal{C}(\Gamma \setminus \{j\}, j. \Delta \vdash t_n \approx u_n (\dots)) \in \Gamma$

Soundness argument

Theorem (soundness)

We define the soundness as for any: Alethe context Δ and a first order formula φ in a step i . $\Delta \vdash \varphi(\mathcal{R}; p_1 \dots p_n)[a_1 \dots a_n]$ proved by a rule \mathcal{R} , the translation $\mathcal{C}(\Gamma, i. \Delta \vdash_{FOL} \varphi(\mathcal{R}; p_1 \dots p_n)[a_1 \dots a_n])$ give a term $M : \mathcal{F}(\varphi)$ such that goal $\Gamma_{\mathcal{A}} \vdash i : \mathcal{F}(\varphi)$ is valide.

Proof.

(proof intuition) By induction on \mathcal{R} .



SMT translation at a glance

Formalisation overview (experimental notation)

Evaluation

Unresolved points

Future perspectives

Evaluation with a TLA+ example

Stephan Merz. TLA+ Case Study: A Resource Allocator.

- ▶ Use set theory only,
- ▶ no skolemization,
- ▶ 25 proofs obligations,
- ▶ size of the Alethe proofs varies between 4 and 288 steps.

Evaluation results

Proofs obligations

- ▶ 16 on the 25 proofs obligations passed,
- ▶ some proofs obligations do not passed due to rules not supported yet.

Bug founds

- ▶ 2 importants bugs in Carcara elaboration process found,
- ▶ 1 bug found in CVC5 with `and_not` rule,
- ▶ and a related bug found in the new TLAPS SMT encoding.

SMT translation at a glance

Formalisation overview (experimental notation)

Evaluation

Unresolved points

Future perspectives

Issues for reconstructing simplification rules

Transformation cases are implicit, and multiple transformations in a step could occur.

Rules example

$j. \Delta \vdash \varphi_1 \vee \dots \vee \varphi_n$ (**or_simplify**)

► $\perp \vee \dots \vee \perp \Rightarrow \perp$

► $\varphi_1 \vee \dots \vee \varphi_n \Rightarrow \varphi_1 \vee \dots \vee \varphi_n'$ where all \perp literals removed

► $\varphi_1 \vee \dots \vee \top \vee \dots \vee \varphi_n \Rightarrow \top$

► ...

$j. \Delta \vdash (\varphi_1 \approx \varphi_2) \approx \psi$ (**equiv_simplify**)

► $(\neg \varphi_1 \approx \neg \varphi_2) \Rightarrow \varphi_1 \approx \varphi_2$

► $(\varphi \approx \varphi) \Rightarrow \top$

► $(\varphi \approx \perp) \Rightarrow \perp$

...

$j. \Delta \vdash \varphi \approx \psi$ (**bool_simplify**)

► $\neg(\varphi_1 \rightarrow \neg \varphi_2) \Rightarrow \varphi_1 \wedge \neg \varphi_2$

► $\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3) \Rightarrow (\varphi_1 \wedge \varphi_2) \rightarrow \varphi_3$

► $\neg(\varphi_1 \rightarrow \neg \varphi_2) \Rightarrow \varphi_1 \wedge \neg \varphi_2$

► ...

$j. \Delta \vdash \varphi_1 \bowtie \varphi_n \approx \psi$ (**comp_simplify**)

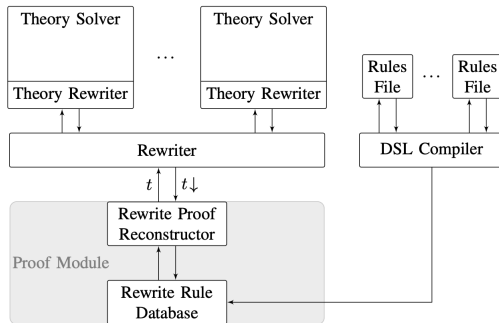
► $t < t \Rightarrow \perp$

► $t_1 < t_2 \Rightarrow \neg(t_2 \leq t_1)$

► $t_1 \leq t_2 \Rightarrow t_2 \leq t_1$

► ...

Reconstruction of transformation rules with RARE



[RARE] Schurr, HJ., et.al. *Reliable Reconstruction of Fine-grained Proofs in a Proof Assistant*. CADE 2021. Springer, Cham.

Checking linear arithmetic steps

- ▶ The `la_generic` rule models linear arithmetic reasoning
- ▶ For example, consider this `la_generic` step:

```
(step t1
  (cl (<= (- x) 1) (<= (+ (* 2 x) (* (- 3) y)) 2) (<= y (- 1))))
:rule la_generic :args (2 1 3))
```

- ▶ It introduces the following tautology:

$$(-x \leq 1) \vee (2x - 3y \leq 2) \vee (y \leq -1)$$

Checking linear arithmetic steps

```
(step t1
  (cl (<= (- x) 1) (<= (+ (* 2 x) (* (- 3) y)) 2) (<= y (- 1)))
  :rule la_generic :args (2 1 3))
```

- ▶ Checking that this clause is true is equivalent to proving that its negation, the following three inequalities, are contradictory
- ▶ Since `la_generic` steps provide the needed coefficients as arguments, checking them is simple
- ▶ Computing $2 \cdot (a) + 1 \cdot (b) + 3 \cdot (c)$, we get $0 > 1$, so the step must be true

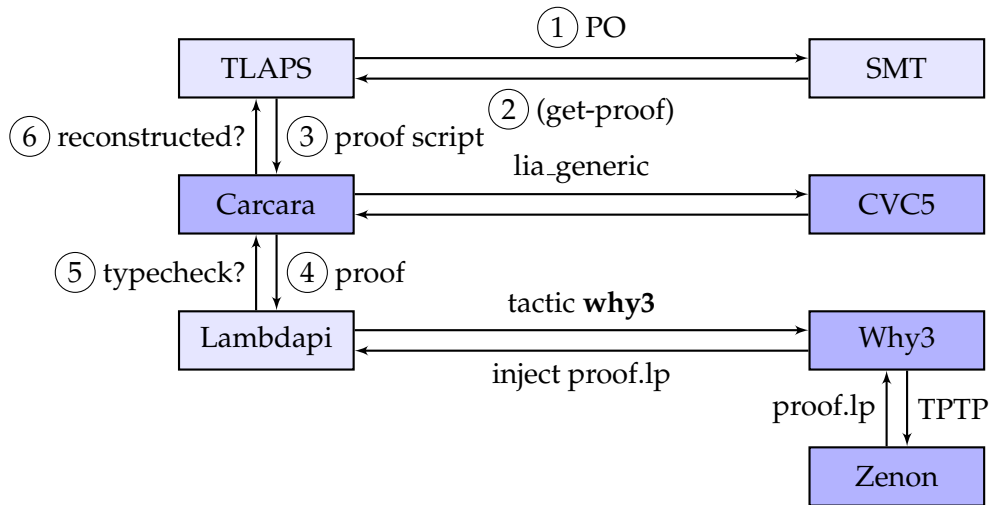
Checking linear arithmetic steps

- ▶ The `lia_generic` rule is very similar to `la_generic`, but it does not provide the coefficients as arguments:

```
(step t1
  (cl (<= (- x) 1) (<= (+ (* 2 x) (* (- 3) y)) 2) (<= y (- 1)))
  :rule lia_generic)
```

- ▶ In this case, the checker would need to search for the coefficients, which is an NP-hard problem
- ▶ Instead, occurrences of this rule are not checked, and are considered holes by Carcara

Proposal for reconstructing arithmetic steps



SMT translation at a glance

Formalisation overview (experimental notation)

Evaluation

Unresolved points

Future perspectives

Future perspectives

- ▶ Finish to validate *Allocator.tla*,
- ▶ add support for arithmetic steps,
- ▶ support for simplicification steps,
- ▶ connect `lambdapi` TLA^+ encoding (*tla-lambdapi*) with TLA^+ SMT encoding,
- ▶ link Event-B encoding with *tla-lambdapi* (shared set theory library).