Checking Linear Integer Arithmetic Proofs in Lambdapi

Alessio Coltellacci

Univ. Lorraine, CNRS, Inria, Loria

FroCoS 2025

Can We Trust SMT Linear Integer Arithmetic?

- SMT solvers are essential for verification, but their large codebases make bugs unavoidable.
- Certifying solvers (e.g. cvc5) directly is herculean.
- Developping a certified SMT solver is complex.
- Instead, solvers can produce *proofs certificates* of their results.
- Proofs can be checked independently.

Key question: *how do we check SMT arithmetic proofs?*

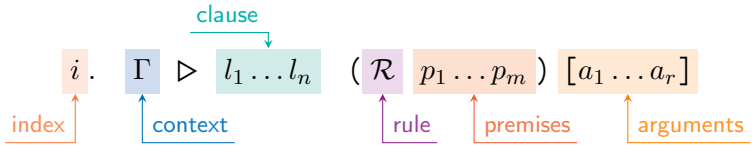
Example of an Arithmetic Alethe SMT Proof

```
(set-logic LIA)
(declare-const x Int)
(declare-const y Int)
(assert (= x 2))
(assert (= 0 y))
(assert (or (< (+ x y) 1) (< 3 x)))
(check-sat)
(get-proof)
```



```
(assume a0 (or (< (+ x y) 1) (< 3 x)))
(assume a2 (= 0 y))
(assume a1 (= x 2))
(step t1 (cl (< (+ x y) 1) (< 3 x)) :rule or :premises (a0))
(step t2 (cl (not (< 3 x)) (not (= x 2))) :rule la_generic :args (1/1 -1/1))
(step t3 (cl (not (< 3 x))) :rule resolution :premises (a1 t2))
(step t4 (cl (< (+ x y) 1)) :rule resolution :premises (t1 t3))
(step t5 (cl (not (< (+ x y) 1)) (not (= x 2)) (not (= 0 y)))
      :rule la_generic :args (1/1 1/1 -1/1))
(step t6 (cl) :rule resolution :premises (t5 t4 a1 a2))
```

The Modern Alethe Proof Format



(1)

- Many-Sorted First-Order Logic of SMT-LIB
- The proof forms a directed acyclic graph
- Proof rules $\mathcal{R}$ include theory lemmas

Example of an Arithmetic Alethe SMT Proof

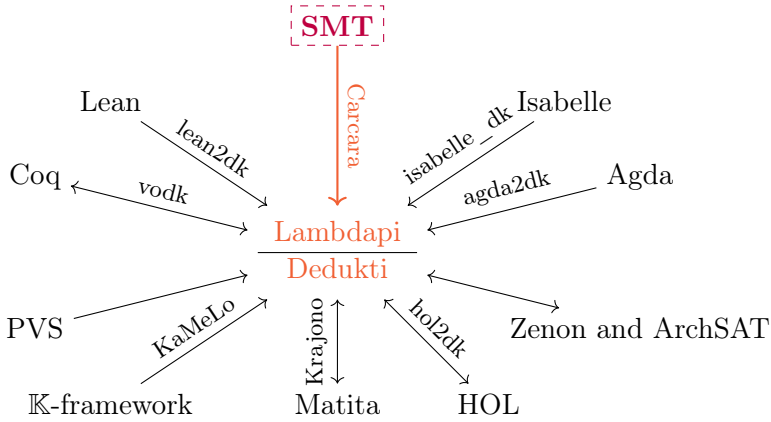
```
(set-logic LIA)
(declare-const x Int)
(declare-const y Int)
(assert (= x 2))
(assert (= 0 y))
(assert (or (< (+ x y) 1) (< 3 x)))
(check-sat)
(get-proof)
```

```
(assume a0 (or (< (+ x y) 1) (< 3 x)))
(assume a2 (= 0 y))
(assume a1 (= x 2))
(step t1 (cl (< (+ x y) 1) (< 3 x)) :rule or :premises (a0))
(step t2 (cl (not (< 3 x)) (not (= x 2))) :rule la_generic :args (1/1 -1/1))
(step t3 (cl (not (< 3 x))) :rule resolution :premises (a1 t2))
(step t4 (cl (< (+ x y) 1)) :rule resolution :premises (t1 t3))
(step t5 (cl (not (< (+ x y) 1)) (not (= x 2)) (not (= 0 y)))
      :rule la_generic :args (1/1 1/1 -1/1))
(step t6 (cl) :rule resolution :premises (t5 t4 a1 a2))
```

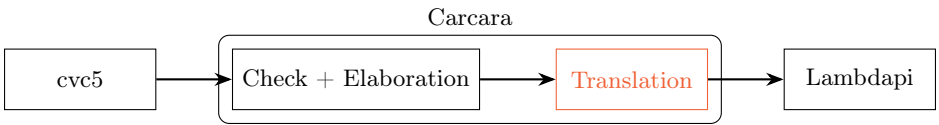
Challenges in Validating Alethe Proofs

- The ordering of clauses $l_1, \ldots, l_n$ is unspecified, which affects proof interpretation.
- Solvers may implicitly reorder equalities, introducing nondeterminism in proof structure.
- Proofs are often coarse-grained, lacking detailed justification for individual steps.
- Key information is sometimes omitted, especially for reasoning over linear integer arithmetic (`LIA`).

Verifying SMT Proofs in Lambdapi



Complete Verification Pipeline for Alethe Proof



Lambdapi Syntax

Based on the Edinburgh Logical Framework (LF):

Universes	$u ::= \texttt{TYPE} \mid \texttt{KIND}$
Terms	$t, v, A, B, C ::= c \mid x \mid u \mid \Pi\, x : A,\, B \mid \lambda\, x : A,\, t \mid t\ v$
Contexts	$\Gamma ::= \langle\rangle \mid \Gamma, x : A$
Signatures	$\Sigma ::= \langle\rangle \mid \Sigma, c : C \mid \Sigma, c := t : C \mid \Sigma, t \hookrightarrow v$

- Rewriting rules must be confluent and preserve typing (subject reduction).
- Confluence is not guaranteed and must be proved separately.
- Supports higher-order rewriting.
- Lacks meta-programming features such as type classes and records.
- Simple set of tactic i.e. no automatic solvers and programmable tactics

Lambdapi Typing Rules

Similar to Edinburgh Logical Framework (LF) but it uses $\equiv_{\beta\Sigma}$.

$$\frac{}{\vdash_\Sigma \langle\,\rangle}\ \text{(Empty)} \qquad \frac{\vdash_\Sigma \Gamma \qquad \Gamma \vdash_\Sigma A : s}{\vdash_\Sigma \Gamma, x : A}\ \text{(Decl)} \qquad \frac{\vdash_\Sigma \Gamma \qquad \Gamma \vdash_\Sigma A : s}{\Gamma \vdash_\Sigma c : A}\ \text{(Const)}$$

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \texttt{TYPE} : \texttt{KIND}}\ \text{(Sort)} \qquad \frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma x : A}\ \text{(Var)}\ x : A \in \Sigma$$

$$\frac{\Gamma, \vdash_\Sigma A : \texttt{TYPE} \qquad \Gamma, x : A \vdash_\Sigma B : s \qquad \Gamma, x : A \vdash_\Sigma t : B}{\Gamma \vdash_\Sigma \lambda x : A, t : \Pi x : A, B}\ \text{(Abs)}$$

$$\frac{\Gamma \vdash_\Sigma A : \texttt{TYPE} \qquad \Gamma, x : A \vdash_\Sigma B : s}{\Gamma \vdash_\Sigma \Pi x : A, B : s}\ \text{(Prod)} \qquad \frac{\Gamma \vdash t : \Pi x : A, B \qquad \Gamma \vdash u : A}{\Gamma \vdash_\Sigma t\ u : B[u \leftarrow x]}\ \text{(App)}$$

$$\frac{\Gamma, \vdash_\Sigma B : u \qquad \Gamma \vdash_\Sigma t : A \qquad A \equiv_{\beta\Sigma} B}{\Gamma \vdash_\Sigma t : B}\ \text{(Conv)}$$

Lambdapi Prelude Encoding Example

$$\texttt{Set} : \texttt{TYPE}$$
$$\texttt{El} : \texttt{Set} \to \texttt{TYPE}$$
$$\rightsquigarrow : \texttt{Set} \to \texttt{Set} \to \texttt{Set}$$
$$\texttt{El}\,(x \rightsquigarrow y) \hookrightarrow \texttt{El}\,x \to \texttt{El}\,y$$
$$= : \Pi[a : \texttt{Set}], \texttt{El}\,a \to \texttt{El}\,a \to \texttt{Prop}$$

$$\texttt{Prop} : \texttt{TYPE}$$
$$\texttt{Prf} : \texttt{Prop} \to \texttt{TYPE}$$
$$o : \texttt{Set}$$
$$\texttt{El}\,o \hookrightarrow \texttt{Prop}$$

$$\texttt{Clause} : \texttt{TYPE}$$
$$\blacksquare : \texttt{Clause}$$
$$\vee : \texttt{Prop} \to \texttt{Clause} \to \texttt{Clause}$$
$$\texttt{Prf}^{\bullet} : \texttt{Clause} \to \texttt{TYPE}$$

$$+\!+ : \texttt{Clause} \to \texttt{Clause} \to \texttt{Clause}$$
$$\blacksquare +\!+ x \hookrightarrow x$$
$$(x \vee y) +\!+ z \hookrightarrow x \vee (y +\!+ z)$$

Encoding Alethe Rules in Lambdapi

Alethe rule	**Lambdapi encoding**
R = equiv_pos2	
$i.\ \Gamma\ \triangleright\ \neg(a \approx b), \neg a, b \quad (\texttt{R})[\,]$	$i : \texttt{Prf}^{\bullet}(\neg(a = b) \vee \neg a \vee b \vee \blacksquare)$
R = resolution	
$i_1.\ \triangleright\ \ l_1^1, \ldots, l_{k^1}^1 \quad (\ldots)$ $i_n.\ \triangleright\ \ l_1^n, \ldots, l_{k^n}^n \quad (\ldots)$ $\vdots$ $j.\ \triangleright\ \ l_{s_1}^{r_1}, \ldots, l_{s_m}^{r_m} \quad (\texttt{R}\ i_1 \ldots i_n)[\,]$	$\texttt{resolution}\ (ps\ qs : \texttt{Clause})\ (i\ j : \mathbb{N})$ $(hps : \texttt{Prf}^{\bullet} ps)$ $(hqs : \texttt{Prf}^{\bullet} qs)$ $(hi : \texttt{Prf}(i < \texttt{size}\ ps))$ $(hj : \texttt{Prf}(j < \texttt{size}\ qs))$ $(hij : \texttt{Prf}((\texttt{nth}\ ps\ i) = \neg(\texttt{nth}\ qs\ j)))$ $: \texttt{Prf}^{\bullet}(\texttt{remove}\ ps\ i +\!\!+ \texttt{remove}\ qs\ j)$

Linear Arithmetic Rules in Alethe Supported in Our Encoding

Rule	Description
la_generic	Tautologous disjunction of linear inequalities
lia_generic	Tautologous disjunction of linear integer inequalities
la_disequality	$t_1 \approx t_2 \vee \neg(t_1 \geq t_2) \vee \neg(t_2 \geq t_1)$
la_totality	$t_1 \geq t_2 \vee t_2 \geq t_1$
la_mult_pos	$t_1 > 0 \wedge (t_2 \bowtie t_3) \rightarrow t_1 * t_2 \bowtie t_1 * t_3$ and $\bowtie \in \{<, >, \geq, \leq, \approx\}$
la_mult_neg	$t_1 < 0 \wedge (t_2 \bowtie t_3) \rightarrow t_1 * t_2 \bowtie_{inv} t_1 * t_3$
la_rw_eq	$(t \approx u) \approx (t \geq u \wedge u \geq t)$
comp_simplify	Simplification of arithmetic comparisons
arith-int-eq-elim	$(t \approx s) \rightarrow t \geq s \wedge t \leq s$
arith-leq-norm	$t \leq s \rightarrow \neg(t \geq s + 1)$
arith-geq-norm1	$t \geq s \rightarrow (t - s) \geq 0$
arith-geq-norm2	$t \geq s \rightarrow -t \leq -s$
arith-geq-tighten	$\neg(t \geq s) \rightarrow s \geq t + 1$
arith-poly-norm	polynomial normalization
evaluate	evaluate constant terms

The Refactored `la_generic` Description

$$i. \quad \triangleright \qquad \varphi_1, \ldots, \varphi_n \qquad \texttt{la\_generic} \quad [a_1, \ldots, a_n]$$

1. If φ_i is of the form $s_1 \geq s_2$ or $\neg(s_1 < s_2)$, then let $\psi_i = s_2 > s_1$. If φ_i is of the form $s_1 > s_2$ or $\neg(s_1 \leq s_2)$, then let $\psi_i = s_2 \geq s_1$. If φ_i is of the form $s_1 < s_2$ or $\neg(s_1 \geq s_2)$, then let $\psi_i = s_1 \geq s_2$. If φ_i is of the form $s_1 \leq s_2$ or $\neg(s_1 > s_2)$, then let $\psi_i = s_1 > s_2$. If φ_i is of the form $\neg(s_1 \approx s_2)$, then let $\psi_i = s_1 \approx s_2$. This step produces a positive literal that is equivalent to $\neg\varphi_i$ and that only contains the operators $>$, $\geq$, and $\approx$.
2. Replace $\psi_i = \sum_{j=0}^{k_i} c_j^i \times t_j^i + d_1^i \bowtie \sum_{j=k_i+1}^{m_i} c_j^i \times t_j^i + d_2^i$ by the literal $\left(\sum_{j=0}^{k_i} c_j^i \times t_j^i\right) - \left(\sum_{j=k_i+1}^{m_i} c_j^i \times t_j^i\right) \bowtie d_2^i - d_1^i$.
3. Now ψ_i has the form $s_1^i \bowtie d^i$. If all variables in s_1^i are integer-sorted then replace $s_1^i > d^i$ by $s_1^i \geq \lfloor d^i \rfloor + 1$, respectively, replace $s_1^i \geq d^i$ by $s_1^i \geq \lfloor d^i \rfloor + 1$ if d is not an integer.
4. If all variables of ψ_i are integer-sorted and the coefficients $a_1 \ldots a_n$ are in $\mathbb{Q}$, then $a_i := a_i \times lcd(a_1 \ldots a_n)$ where lcd is the least common denominator of $\{a_1, \ldots, a_n\}$.
5. If $\bowtie$ is $\approx$, then replace ψ_i by $\sum_{j=0}^{m_i} a_i \times c_j^i \times t_j^i = a_i \times d^i$, otherwise replace ψ_i by $\sum_{j=0}^{m_i} |a_i| \times c_j^i \times t_j^i \bowtie |a_i| \times d^i$.
6. Finally, the sum of the resulting literals is trivially contradictory,

$$\sum_{i=1}^{n} \sum_{j=1}^{m_i} c_j^i * t_j^i \bowtie \sum_{i=1}^{n} d^i$$

An Example of `la_generic`

Consider the following `la_generic` step in the logic `QF_UFLIA` with the uninterpreted function symbol `(f Int)`:

```
1 (step t11 (cl (not (<= f 0)) (<= (+ 1 (* 4 f)) 1))
2   :rule la_generic :args (1/1 1/4))
```

The algorithm then performs the following steps:

$-f \geq 0,\ 4 \times f > 0$ (Steps 1 and 2)

$-f \geq 0,\ 4 \times f \geq 1$ (Step 3)

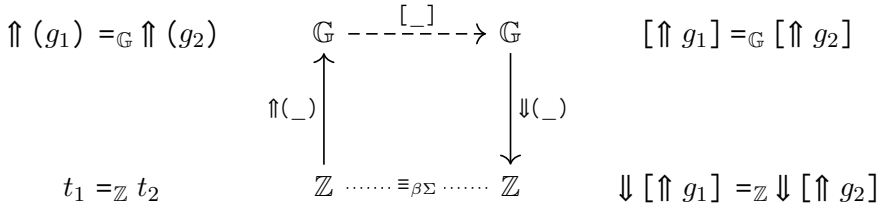
Replace arguments $[\frac{1}{1}, \frac{1}{4}]$ by $[4, 1]$ due to clearing denominators (Step 4)

$|4| \times (-f) \geq |4| \times 0,\ |1| \times 4 \times f \geq |1| \times 1)$ (Step 5)

$-4 \times f + 4 \times f \geq 1$ (Step 6)

Which simplifies to the contradiction $0 \geq 1$.

A Scheme for Proof by Reflection



with:

- $\mathbb{G}$ an Algebra to represent integers
- $\Uparrow : \mathbb{Z} \to \mathbb{G}$ the reify function
- $\Downarrow : \mathbb{G} \to \mathbb{Z}$ the denotation function
- $[\_] : \mathbb{G} \to \mathbb{G}$ the normalization function
- $=_{\mathbb{G}} : \mathbb{G} \to \mathbb{G} \to \mathbb{B}$ a (decidable) equivalence relation

Two Methods for Normalising Terms

1. **Inner normalisation** of terms performed in the Lambdapi *kernel* using `associative commutative`

2. **Outer normalisation** function with user-defined rewrite rules and symbolic execution.

The $\mathbb{G}$ Algebra to Represent Integers - Inner Version

$\mathbb{G}$: TYPE
| $\oplus : \mathbb{G} \to \mathbb{G} \to \mathbb{G}$
| var : $\mathbb{Z} \to \mathbb{Z} \to \mathbb{G}$
| mul : $\mathbb{Z} \to \mathbb{G} \to \mathbb{G}$
| opp : $\mathbb{G} \to \mathbb{G}$
| cst : $\mathbb{Z} \to \mathbb{G}$
grp : Set
El grp $\hookrightarrow \mathbb{G}$

$\Uparrow : \mathbb{Z} \to \mathbb{G}$
$\Uparrow$ Z0 $\hookrightarrow$ (cst Z0)
$\Uparrow$ ZPos $c \hookrightarrow$ (cst (ZPos c))
$\Uparrow$ ZNeg $c \hookrightarrow$ (cst (ZNeg c))
$\Uparrow (x + y) \hookrightarrow (\Uparrow x) \oplus (\Uparrow y)$
$\Uparrow (\sim x) \hookrightarrow$ opp $\Uparrow x$
$\Uparrow ((\text{ZPos}\, c) * x) \hookrightarrow$ mul (ZPos c) ($\Uparrow x$)
$\Uparrow ((\text{ZNeg}\, c) * x) \hookrightarrow$ mul (ZNeg c) ($\Uparrow x$)
$\Uparrow (x * (\text{ZPos}\, c)) \hookrightarrow$ mul (ZPos c) ($\Uparrow x$)
$\Uparrow (x * (\text{ZNeg}\, c)) \hookrightarrow$ mul (ZNeg c) ($\Uparrow x$)
$\Uparrow (x * \text{Z0}) \hookrightarrow$ (cst 0)
$\Uparrow (\text{Z0} * x) \hookrightarrow$ (cst 0)
$\Uparrow x \hookrightarrow$ (var 1 x)

$\Downarrow : \mathbb{G} \to \mathbb{Z}$
$\Downarrow$ (cst c) $\hookrightarrow c$
$\Downarrow$ opp $x \hookrightarrow \sim (\Downarrow x)$
$\Downarrow$ mul $c\ x \hookrightarrow c \times (\Downarrow x)$
$\Downarrow x \oplus y \hookrightarrow (\Downarrow x) + (\Downarrow y)$
$\Downarrow$ (var $c\ x$) $\hookrightarrow c \times x$

with $\oplus$ declared associative commutative, and $\Uparrow$ sequential.

Normalisation with Associative Commutative Modifier

Definition

The $\leq$ builtin total order on $\mathbb{G}$-terms is defined as follows: Terms are ordered such that $\texttt{cst}(c_1) \leq \texttt{cst}(c_2) < (\texttt{var}\ c\ x)$ for any constants $c_1 \leq c_2$ and any variable term $(\texttt{var}\ c\ x)$. For variable terms, $(\texttt{var}\ c\ x) \leq (\texttt{var}\ d\ y)$ if either $x < y$, or $x = y$ and $c \leq d$.

Example

Consider the term below not in normal form with variables $\mathbf{X}$ and $\mathbf{Y}$ of type $\mathbb{Z}$:

$$(\texttt{var}\ c_1\ \mathbf{X}) \oplus (\texttt{cst}\ k_1) \oplus (\texttt{var}\ c_2\ \mathbf{Y}) \oplus (\texttt{cst}\ k_m) \oplus (\texttt{var}\ c_3\ \mathbf{X}) \oplus (\texttt{var}\ c_4\ \mathbf{Y})$$

It will be then normalise into:

$$(\texttt{cst}\ k_1) \oplus (\texttt{cst}\ k_2) \oplus (\texttt{var}\ c_1\ \mathbf{X}) \oplus (\texttt{var}\ c_3\ \mathbf{X}) \oplus (\texttt{var}\ c_2\ \mathbf{Y}) \oplus (\texttt{var}\ c_4\ \mathbf{Y})$$

Theory Rules for $\mathbb{G}$

Group theory axioms

$$(\texttt{var}\ c_1\ x) \oplus (\texttt{var}\ c_2\ x) \hookrightarrow (\texttt{var}\ (c_1 + c_2)\ x) \tag{1}$$

$$(\texttt{var}\ c_1\ x) \oplus ((\texttt{var}\ c_2\ x) \oplus y) \hookrightarrow (\texttt{var}\ (c_1 + c_2)\ x) \oplus y \tag{2}$$

$$(\texttt{cst}\ c_1) \oplus (\texttt{cst}\ c_2) \hookrightarrow (\texttt{cst}\ (c_1 + c_2)) \tag{3}$$

$$(\texttt{cst}\ c_1) \oplus ((\texttt{cst}\ c_2) \oplus y) \hookrightarrow (\texttt{cst}\ (c_1 + c_2)) \oplus y \tag{4}$$

$$(\texttt{cst}\ 0) \oplus x \hookrightarrow x \tag{5}$$

$$x \oplus (\texttt{cst}\ 0) \hookrightarrow x \tag{6}$$

$$\texttt{opp}\ (\texttt{var}\ c\ x) \hookrightarrow (\texttt{var}\ (-c)\ x) \tag{7}$$

$$\texttt{opp}\ (\texttt{cst}\ c) \hookrightarrow (\texttt{cst}\ (-c)) \tag{8}$$

$$\texttt{opp}\ (\texttt{opp}\ x) \hookrightarrow x \tag{9}$$

$$\texttt{opp}\ (x \oplus y) \hookrightarrow (\texttt{opp}\ x) \oplus (\texttt{opp}\ y) \tag{10}$$

$$\texttt{opp}\ (\texttt{mul}\ k\ x) \hookrightarrow \texttt{mul}\ (-k)\ x \tag{11}$$

$$\texttt{mul}\ k\ (\texttt{var}\ c\ x) \hookrightarrow (\texttt{var}\ (k * c)\ x) \tag{12}$$

$$\texttt{mul}\ k\ (\texttt{opp}\ x) \hookrightarrow \texttt{mul}\ (-k)\ x \tag{13}$$

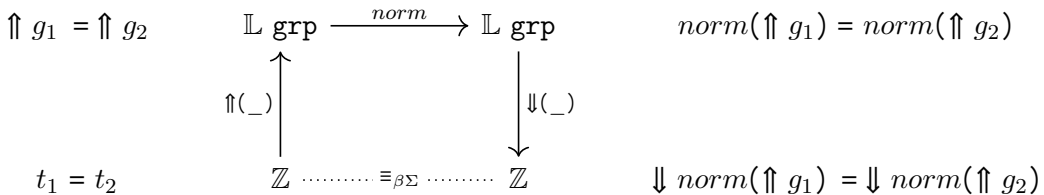
$$\texttt{mul}\ k\ (x \oplus y) \hookrightarrow (\texttt{mul}\ k\ x) \oplus (\texttt{mul}\ k\ y) \tag{14}$$

$$\texttt{mul}\ k\ (\texttt{cst}\ c) \hookrightarrow (\texttt{cst}\ (k * c)) \tag{15}$$

$$\texttt{mul}\ c_1\ (\texttt{mul}\ c_2\ x) \hookrightarrow \texttt{mul}\ (c_1 * c_2)\ x \tag{16}$$

```
opaque symbol t2: Prf (¬ (3 < x) ∨ ¬ (x = 2)) {
    rewrite Zinv_lt_eq;
    rewrite Z_diff_gt_Z0_eq;
    rewrite Z_diff_eq_Z0_eq;
    rewrite Zgt_le_succ_r_eq;
    rewrite Zmult_ge_compat_eq 1;
    rewrite Zmult_eq_compat_eq (- 1);
    rewrite imp_eq_or; apply ⇒ᵢ; assume H0; apply ¬ᵢ; assume H1;
    set H0l' ≔ ...; set H0r' ≔ ...;
    set H1l' ≔ ...; set H1r' ≔ ...;
    have H1': π (H1l' ≥ H1r') { refine Z_eq_implies_ge H1 };
    have contra : π ((⇓ (⇑ (H0l' + H1l'))) ≥ (⇓ (⇑ (H0r' + H1r')))){
        rewrite reify_correct; rewrite reify_correct;
        apply Zsum_geq_s;
    };
    apply contra; apply ⊤ᵢ;
};
```

A Scheme for Proof by Reflection - Outer Version



with:

- $\mathbb{G}$ an Algebra to represent integers
- $\Uparrow\colon \mathbb{Z} \to \mathbb{G} \times (\mathbb{L}\ \texttt{int})$ the reify function
- $\Downarrow\colon \mathbb{G} \times (\mathbb{L}\ \texttt{int}) \to \mathbb{Z}$ the denotation function
- $norm\colon \mathbb{G} \to \mathbb{G}$ the normalization function

Normalization in Outer Version: An Overview

We redefine the type $\mathbb{G}$ and we reify into a $\mathbb{L}$ `int`:

$$\begin{array}{l} \mathbb{G} : \texttt{TYPE} \\ \mid \texttt{var} : \mathbb{N} \to \mathbb{Z} \to \mathbb{G} \\ \mid \texttt{cst} : \mathbb{Z} \to \mathbb{G} \\ \mid \texttt{mul} : \mathbb{Z} \to \mathbb{G} \to \mathbb{G} \end{array} \qquad \begin{array}{r} \texttt{grp} : \texttt{Set} \\ \texttt{El grp} \hookrightarrow \mathbb{G} \end{array}$$

We then define the normalization function as follows:

$$norm\,(x : \mathbb{L}\ \texttt{grp}) \coloneqq \underbrace{remove0}_{\text{cancel and removes neutral elements}}\Big(\underbrace{mergesort}_{\text{sorts a list of grp}}\ x\Big) \tag{2}$$

Current Evaluation of the Two Methods

- The inner approach is easier to implement but requires trusting the Lambdapi kernel.

- The outer approach is still slow on large examples and poses major challenges for further optimisation.

Evaluation

Table: Benchmark results with format: success - fail (- timeout).

Logic	Benchmark	Samples	Proofs	Elaborate	Translate	Check
LIA	tptp	36	36 - 0 - 0	36 - 0	36 - 0 - 0	28 - 8 - 0
	Ultimate	153	120 - 0 - 33	73 - 47	68 - 5 - 0	50 - 18 - 0
	Svcomp'19	27	27 - 0 - 0	25 - 2	0 - 25 - 0	0
	psyco	50	48 - 0 - 2	48 - 0	43 - 0 - 5	0 - 37 - 6
QFLIA	SMPT	1568	1529 - 39 - 0	1497 - 32	1476 - 0 - 21	804 - 638 - 34
	rings	294	70 - 1 - 223	49 - 21	49 - 0 - 0	7 - 0 - 42
	CAV2009	85	85 - 0 - 0	19 - 66	19 - 0 - 0	19 - 0 - 0
UFLIA	sledgeh	1521	1343 - 0 - 178	1278 - 65	1258 - 13 - 7	711 - 467 - 80
	tokeneer	1732	1732 - 0 - 0	1689 - 43	1689 - 0 - 0	1482 - 197 - 10

Timeout settings:

- 30s for cvc5
- 30s for Carcara
- 20s for Lambdapi

Checking Times

Table: Lambdapi checking times in milliseconds.

Bench	**Min**	**Q1**	**Mean**	**Q3**	**Max**
tptp	240	262	263	267	336
Ultimate	82	96	100	111	346
SMPT	52	55	55	56	293
rings	670	704	773	826	1197
CAV2009	54	296	403	498	683
sledgeh	53	166	354	552	1594
tokeneer	55	60	61	248	753

Conclusion

Contributions

- Automated verification of Alethe SMT proof expressed in **LIA** and **UF** logics (+ **QF**).
- Converting Alethe SMT proof with Lambdapi to other format (e.g. Rocq, Lean).

Future works

- Add support for bitvectors (**BV**).
- Add support for rational and real number (**LRA**).