Hint-Based SMT Proof Reconstruction

Joshua Clune[1] Haniel Barbosa[2] Jeremy Avigad[1]

[1]Carnegie Mellon University, [2]Universidade Federal de Minas Gerais

TACAS 2026

2026–04–13, Torino, IT

Established Proof Reconstruction Strategies

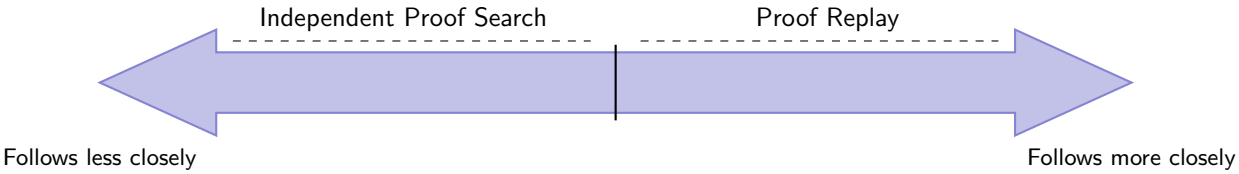
- Independent proof search on minimized facts
  - Used e.g. for superposition theorem provers and SMT solvers in Sledgehammer (Isabelle/HOL)
  - Used e.g. for superposition theorem provers in LeanHammer (Lean)
  - Used e.g. for superposition theorem provers and SMT solvers in CoqHammer (Rocq)

- Proof replay
  - Used e.g. for z3, veriT, and cvc5 in Sledgehammer (Isabelle/HOL)
  - Used e.g. for cvc5 in smt-lean (Lean)
  - Used e.g. for veriT in SMTCoq (Rocq)

- "The difference between walking and running"

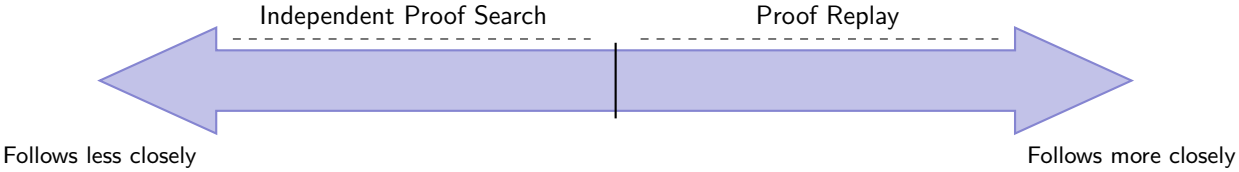
Tradeoffs

These strategies differ in how closely they follow the original solvers' proofs.



Tradeoffs

These strategies differ in how closely they follow the original solvers' proofs.



Following proofs less closely:

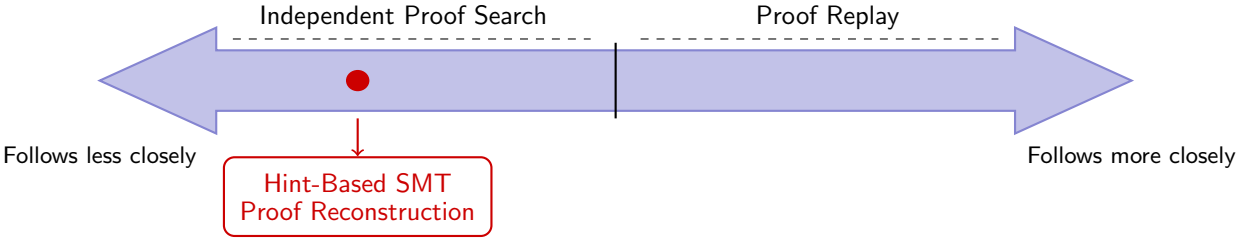
- ▷ Requires ITP automation to rediscover more
- ▷ Makes it possible to drop dependencies on external solvers

Following proofs more closely:

- ▷ Puts less burden on the ITP automation
- ▷ Requires retaining a dependence on the external solvers

Tradeoffs

These strategies differ in how closely they follow the original solvers' proofs.



Following proofs less closely:

- ▷ Requires ITP automation to rediscover more
- ▷ Makes it possible to drop dependencies on external solvers

Following proofs more closely:

- ▷ Puts less burden on the ITP automation
- ▷ Requires retaining a dependence on the external solvers

Hint-Based SMT Proof Reconstruction

- SMT proofs consist both of `theory content` and `logical content`.
  - Theory content refers to reasoning on arithmetic, data structures, and so on.
  - Logical content refers to reasoning on connectives and quantifiers.

- Our reconstruction approach is to use hints output by the solver
  - reconstruct the theory content independently (e.g. via grind)
  - use independent proof search (duper) to reconstruct the logical content while `aided` by the theory content

- This is implemented as a Lean tactic called QuerySMT

What is QuerySMT?

QuerySMT is a tactic that:

- Translates Lean goals to SMT-LIB (using lean-auto)
- Receives hints on how to solve the goals from cvc5
- Translates those hints back to Lean
- Produces a standalone proof script (that doesn't depend on cvc5)

Example

```
example (f : ℤ → ℤ) (h1: ∀ x y, f x = f y → x = y)
  (h2 : ∃ x, ∀ y, f x ≤ f y) :
  ∃ x, ∀ (y : ℤ), x ≠ y → f x < f y :=by querySMT
```

Example

```
example (f : ℤ → ℤ) (h1: ∀ x y, f x = f y → x = y)
  (h2 : ∃ x, ∀ y, f x ≤ f y) :
  ∃ x, ∀ (y : ℤ), x ≠ y → f x < f y :=by querySMT
```

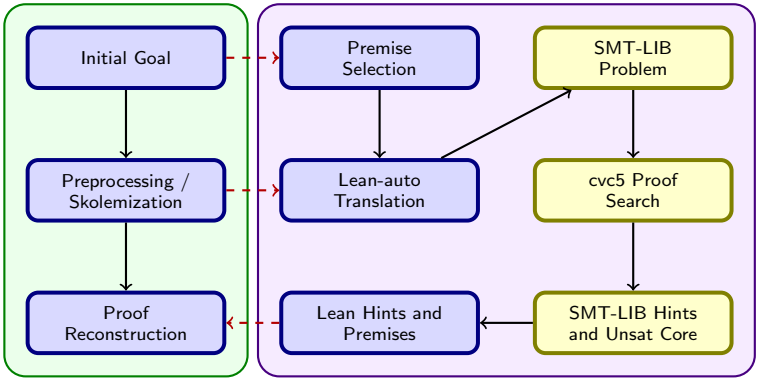
QuerySMT produces a standalone proof:

```
  @[classical] byContradiction
  intro negGoal
  skolemizeAll
  have smtLemma0 : ∀ (_i_0 : ℤ), ¬f sk0 ≤ -1 * f _i_0 ≥ 1 :=by grind
  have smtLemma1 : ∀ (bv0 : ℤ), f bv0 + -1 * f (sk1 bv0) ≥ 0 :=by grind
  have smtLemma2 : ∀ (bv0 : ℤ), ¬bv0 = sk1 bv0 :=by grind
  have smtLemma3 : (¬f sk0 + -1 * f (sk1 sk0) ≥ 0 ∨ f sk0 = f (sk1 sk0))∨
     f sk0 + -1 * f (sk1 sk0) ≥ 1 :=by grind
  duper [h1, smtLemma0, smtLemma1, smtLemma2, smtLemma3] []
```

QuerySMT Design

QuerySMT consists of five primary components:

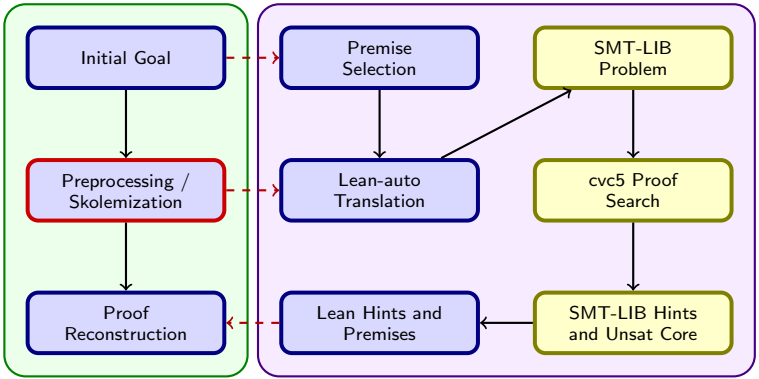
1. Preprocessing
2. Translation
3. Hint generation
4. Hint interpretation
5. Proof reconstruction



QuerySMT Design

QuerySMT consists of five primary components:

1. Preprocessing
2. Translation
3. Hint generation
4. Hint interpretation
5. Proof reconstruction



Preprocessing

```
example (f : ℤ → ℤ) (h1: ∀ x y, f x = f y → x = y)
(h2 : ∃ x, ∀ y, f x ≤ f y) :
∃ x, ∀ (y : ℤ), x ≠ y → f x < f y := by
```

▼ Tactic state

1 goal

```
f : ℤ → ℤ
h1 : ∀ (x y : ℤ),
f x = f y → x = y
h2 : ∃ x, ∀ (y :
ℤ), f x ≤ f y
⊢ ∃ x, ∀ (y : ℤ), x
≠ y → f x < f y
```

Preprocessing

```
example (f : ℤ → ℤ) (h1: ∀ x y, f x = f y → x = y)
(h2 : ∃ x, ∀ y, f x ≤ f y) :
∃ x, ∀ (y : ℤ), x ≠ y → f x < f y := by
@[classical] byContradiction
intro negGoal
```

▼ Tactic state
1 goal

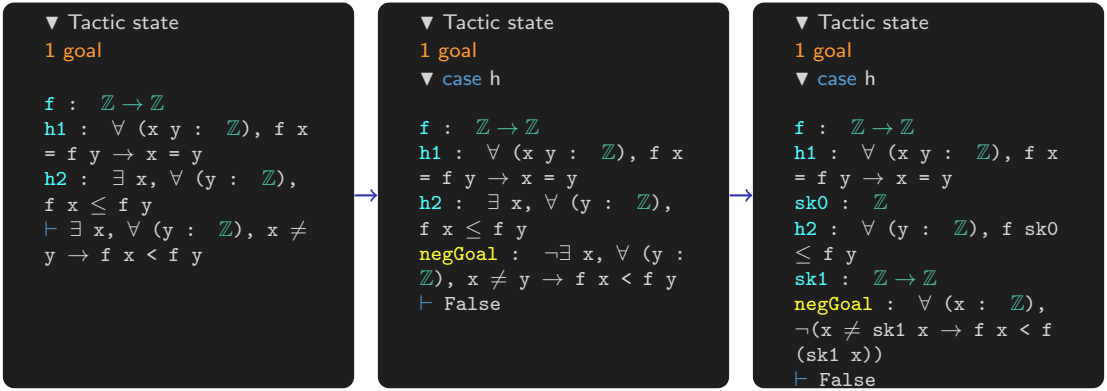
```
f : ℤ → ℤ
h1 : ∀ (x y : ℤ), f x
= f y → x = y
h2 : ∃ x, ∀ (y : ℤ),
f x ≤ f y
⊢ ∃ x, ∀ (y : ℤ), x ≠
y → f x < f y
```

▼ Tactic state
1 goal
▼ case h

```
f : ℤ → ℤ
h1 : ∀ (x y : ℤ), f x
= f y → x = y
h2 : ∃ x, ∀ (y : ℤ),
f x ≤ f y
negGoal : ¬∃ x, ∀ (y :
ℤ), x ≠ y → f x < f y
⊢ False
```

Preprocessing

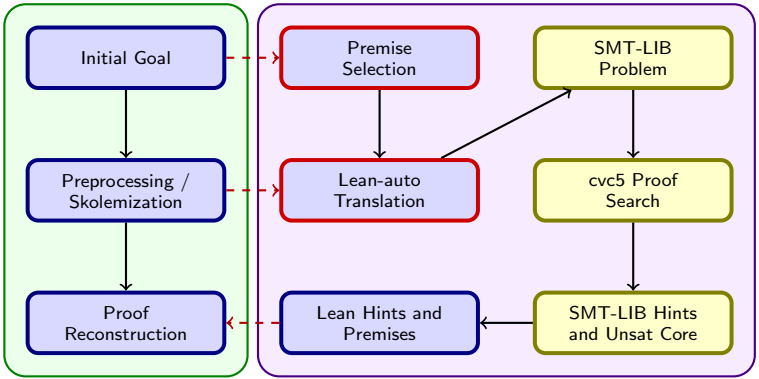
```
example (f : ℤ → ℤ) (h1: ∀ x y, f x = f y → x = y)
(h2 : ∃ x, ∀ y, f x ≤ f y) :
∃ x, ∀ (y : ℤ), x ≠ y → f x < f y := by
@[classical] byContradiction
intro negGoal
skolemizeAll
```



QuerySMT Design

QuerySMT consists of five primary components:

1. Preprocessing
2. Translation
3. Hint generation
4. Hint interpretation
5. Proof reconstruction



Translation

```
▼ Tactic state
1 goal
▼ case h

f : ℤ → ℤ, h1 : ∀ (x y : ℤ), f x = f y → x = y
sk0 : ℤ, h2 : ∀ (y : ℤ), f sk0 ≤ f y, sk1 : ℤ → ℤ
negGoal : ∀ (x : ℤ), ¬(x ≠ sk1 x → f x < f (sk1 x))
⊢ False
```

⇓ Lean-auto

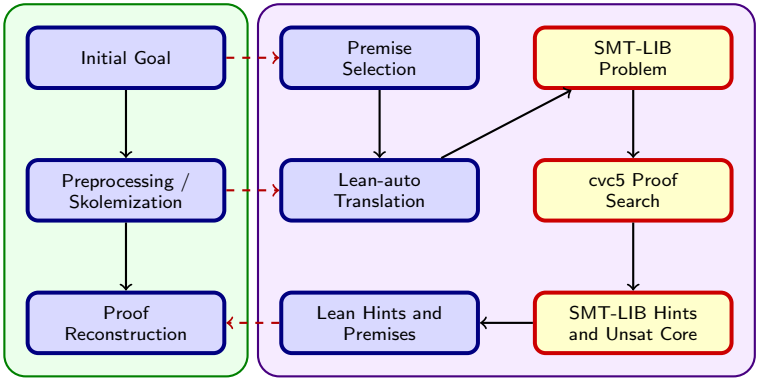
SMT-LIB

```
(declare-sort Empty 0)
(define-fun nsub ((x Int) (y Int)) Int (ite (>= x y) (- x y) 0))
(define-fun itdiv ((x Int) (y Int)) Int (ite (>= y 0) x (ite (>= x 0) (div y (- (div (- x) y))))))
(define-fun itmod ((x Int) (y Int)) Int (ite (>= y 0) x (ite (>= x 0) (mod y (mod (- x) y))))))
(define-fun iediv ((x Int) (y Int)) Int (ite (>= y 0) (itdiv x y) (- (itdiv (- x) y))))
(declare-fun _sk1_ (Int) Int)
(declare-fun _f (Int) Int)
(assert (forall ((_i_0 Int)) (not (=> (not (= _i_0 (_sk1_ _i_0))) (< (_f _i_0) (_f (_sk1_ _i_0)))))) :named valid_fact_0)
(declare-fun _sk0_ () Int)
(assert (forall ((_i_0 Int)) (<= (_f _sk0_) (_f _i_0))) :named valid_fact_1)
(assert (forall ((_i_0 Int)) (not (=> (not (= _i_0 (_sk1_ _i_0))) (< (_f _i_0) (_f (_sk1_ _i_0)))))) :named valid_fact_2)
(assert (forall ((_i_0 Int) (_i_1 Int)) (=> (= (_f _i_0) (_f _i_1)) (= _i_0 _i_1))) :named valid_fact_3)
(assert (<= (- 5) (_f (_i_0 Int)) (_f (_i_1 Int)) (<= (_f _i_0) (- 5 (_f _i_1)))) :named valid_fact_4)
```

QuerySMT Design

QuerySMT consists of five primary components:

1. Preprocessing
2. Translation
3. Hint generation
4. Hint interpretation
5. Proof reconstruction



Hint Generation

- cvc5 is invoked with special flags:

```
cvc5 --tlimit=10000 --produce-models --enum-inst
--dump-hints --proof-granularity=dsl-rewrite
--hints-only-rw-insts --produce-proofs temp.smt2
```

Hint Generation

- cvc5 is invoked with special flags:

```
cvc5 --tlimit=10000 --produce-models --enum-inst
--dump-hints --proof-granularity=dsl-rewrite
--hints-only-rw-insts --produce-proofs temp.smt2
```

- cvc5 generates hints: preprocessing facts and theory lemmas

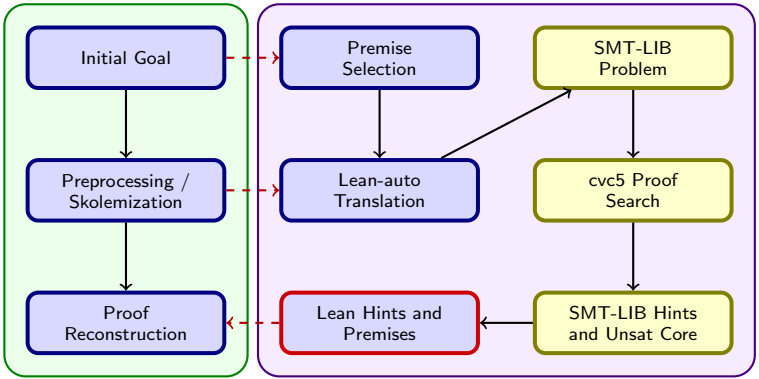
8 hints generated by cvc5

```
unsat
preprocessing facts:
(=> (forall ((_i_0 Int)) (<= (_f _sk0_) (_f _i_0))) (forall ((_i_0 Int)) (not (=> (+ (_f _sk0) (* (- 1) (_f _i_0))) 1))))
(=> (forall ((_i_0 Int)) (forall ((_i_2 Int)) (=> (= (_f _i_1) (_f _i_2)) (= _i1 _i_2))))
(forall ((_i_1 Int) (_i_2 Int)) (or (not (= _f _i_1) (_f _i_2)) (= _i_1 _i_2))))
(=> (forall ((_i Int)) (let ((_let_1 (_sk1_ _i))) (not (=> (not (= _i _let_1) (< (_f _1) (_f _let_1)))))))
(forall ((BOUND_VARIABLE_4960 Int)) (>= (+ (_f BOUND_VARIABLE_4960) (* (- 1) (_f (_sk1_ BOUND_VARIABLE_4960)))) 0)))
(=> (forall ((_i Int)) (let ((_let_1 (_sk1_ _i))) (not (=> (not (= _i _let_1)) (< (_f _i) (_f _let_1)))))))
(forall ((BOUND_VARIABLE_4953 Int)) (not (= BOUND_VARIABLE_4953 (_sk1_ BOUND_VARIABLE_4953)))))
theory lemmas:
(let ((_let_1 (_f (_sk1_ _sk0_)))) (let ((_let_2 (_f _sk0_)))
(let ((_let_3 (+ _let_2 (* (- 1) _let_1)))) (or (not (>= _let_3 0)) (= _let_2 _let_1) (>= _let_3 1)))))
(let ((_let_1 (+ (_f _sk0_) (* (- 1) (_f (_sk1_ _sk0_)))))) (=> (< _let_1 1) (<= _let_1 0)))
...
```

QuerySMT Design

QuerySMT consists of five primary components:

1. Preprocessing
2. Translation
3. Hint generation
4. Hint interpretation
5. Proof reconstruction



Hint Interpretation

Solver output (cvc5 hints)

```
preprocessing facts:    (=> (forall ((_i_0 Int)) ...)  ...)    ...      theory lemmas:    (let ((_let_1 ...))  ...)    ...
```

⇓

Initial Lean interpretation

```
have smtLemma0 :  ∀ (_i_0 :  ℤ), f sk0 ≤ f _i_0 → ¬f sk0 + -1 * f _i_0 ≥ Int.ofNat 1 := by grind
have smtLemma1 :  (∀ (_i_1 _i_2 :  ℤ), f _i_1 = f _i_2 → _i_1 = _i_2) → ∀ (_i_1 _i_2 :  ℤ), ¬f _i_1 = f _i_2 ∨ _i_1 = _i_2 := by grind
have smtLemma2 :  ∀ (bv0 :  ℤ), have _let_1 := sk1 ... _let_1 + -1 * f _let_1 ...
∀ (BOUND_VARIABLE_4962 :  ℤ), f BOUND_VARIABLE_4962 +.ofNat 1 ≤ f (sk1 BOUND_VARIABLE_4962) ≥ Int.ofNat 0 := by grind
have smtLemma3 :  ∀ (BOUND_VARIABLE_4953 :  ℤ), ¬BOUND_VARIABLE_4953 = sk1 BOUND_VARIABLE_4953 := by grind
have smtLemma4 :  have _let_1 := f sk0; have _let_2 := f (sk1 sk0);
(¬_let_1 + -1 * _let_2 ≥ Int.ofNat 0 ∨ _let_1 = _let_2) ∨ _let_1 + -1 * _let_2 ≥ Int.ofNat 1 := by grind
have smtLemma5 :  have _let_1 := f sk0; have _let_2 := f (sk1 sk0); have _let_3 := _let_1 + -1 * _let_2;
_let_3 < Int.ofNat 1 → (¬_let_3 ≥ Int.ofNat 0 ∨ _let_1 = _let_2) ∨ _let_3 ≥ Int.ofNat 1 := by grind
have smtLemma6 :  (sk1 sk0 = sk0) = (sk0 = sk1 sk0) := by grind
have smtLemma7 :  (f (sk1 sk0) = f sk0) = (f sk0 = f (sk1 sk0)) := by grind
```

Hint Interpretation

Solver output (cvc5 hints)

```
preprocessing facts:    (=> (forall ((_i_0 Int)) ...)  ...)    ...
theory lemmas:    (let ((_let_1 (_f _sk0_...)))  ...)    ...
```

⇓

Initial Lean interpretation

```
have smtLemma0 :  ∀ (_i_0 :  ℤ), f sk0 ∈ f _i_0 → ¬f sk0 → Int.ofNat ... := by grind
have smtLemma1 :  (∀ (_i_1 _i_2 :  ℤ), f _i_1 = f _i_2 → _i_1 = _i_2) → ... := by grind
have smtLemma2 :  ∀ (bv0 :  ℤ), have _let_1 := sk1 ... ∀ (BV_4962 :  ℤ), f BV_4962 +.ofNat 1 ... := by grind  ...
...
```

⇓

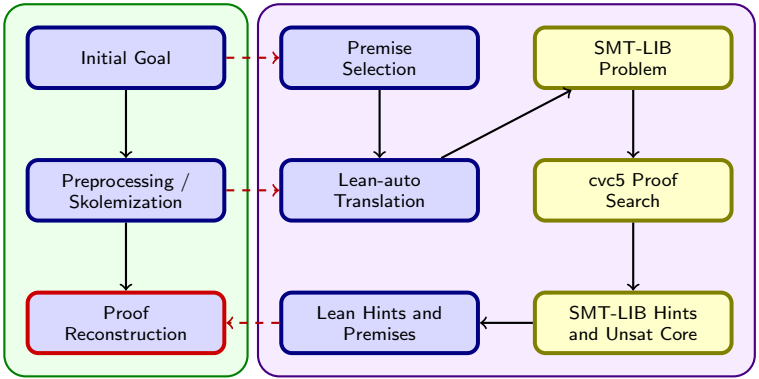
Refined Lean interpretation

```
have smtLemma0 :  ∀ (_i_0 :  ℤ), ¬ f sk0 + -1 * f _i_0 ≥ 1 := by grind
have smtLemma1 :  (∀ (_i_1 _i_2 :  ℤ), f _i_1 = f _i_2 → _i_1 = _i_2) → ... := by grind
have smtLemma2 :  ∀ (bv0 :  ℤ), f bv0 + -1 * f (sk1 bv0) ≥ 0 := by grind
have smtLemma3 :  ∀ (bv0 :  ℤ), ¬bv0 = sk1 bv0 := by grind
have smtLemma4 :  (¬f sk0 + -1 * f (sk1 sk0) ≥ 0 ∨ f sk0 = f (sk1 sk0)) ∨ f sk0 + -1 * f (sk1 sk0) ≥ 1 := by grind
have smtLemma5 :  f sk0 + -1 * f (sk1 sk0) < 1 → ... := by grind
have smtLemma6 :  (sk1 sk0 = sk0) = (sk0 = sk1 sk0) := by grind
have smtLemma7 :  (f (sk1 sk0) = f sk0) = (f sk0 = f (sk1 sk0)) := by grind
```

QuerySMT Design

QuerySMT consists of five primary components:

1. Preprocessing
2. Translation
3. Hint generation
4. Hint interpretation
5. Proof reconstruction



Proof Reconstruction

```
example (f : ℤ → ℤ) (h1: ∀ x y, f x = f y → x = y)
(h2 : ∃ x, ∀ y, f x ≤ f y) :
∃ x, ∀ (y : ℤ), x ≠ y → f x < f y := by
@[classical] byContradiction
intro negGoal
skolemizeAll
```

```
[querySMT.debug] getDuperCoreSMTLemmas ::
Calling runDuperPortfolioMode with formulas:
[∀ (x : ℤ), ¬(x ≠ sk1 x → f x < f (sk1 x)),
∀ (y : ℤ), f sk0 ≤ f y,
∀ (x y : ℤ), f x = f y → x = y,
∀ (_i_0 : ℤ), ¬f sk0 + -1 * f _i_0 ≥ 1,
∀ (_i_1 _i_2 : ℤ), ¬f _i_1 = f _i_2 ∨ _i_1 = _i_2,
∀ (bv0 : ℤ), f bv0 + -1 * f (sk1 bv0) ≥ 0,
∀ (bv0 : ℤ), ¬bv0 = sk1 bv0,
(¬f sk0 + -1 * f (sk1 sk0) ≥ 0 ∨ f sk0 = f (sk1
sk0))
∨ f sk0 + -1 * f (sk1 sk0) ≥ 1,
f sk0 + -1 * f (sk1 sk0) < 1 → ...,
(sk1 sk0 = sk0) = (sk0 = sk1 sk0),
(f (sk1 sk0) = f sk0) = (f sk0 = f (sk1 sk0))]
```

Proof Reconstruction

```
example (f : ℤ → ℤ) (h1: ∀ x y, f x = f y → x = y)
(h2 : ∃ x, ∀ y, f x ≤ f y) :
∃ x, ∀ (y : ℤ), x ≠ y → f x < f y := by
@[classical] byContradiction
intro negGoal
skolemizeAll
```

```
[querySMT.debug] getDuperCoreSMTLemmas ::
Calling runDuperPortfolioMode
with formulas:
[∀ (x : ℤ), ¬(x ≠ sk1 x → ...),
∀ (y : ℤ), f sk0 ≤ f y,
∀ (x y : ℤ), f x = f y → x = y,
...]
```

→

```
[querySMT.debug] ∀ (_i_0 : ℤ), ...
appears in the proof (index: 0)
[querySMT.debug] ∀ (_i_1 _i_2 : ℤ), ...
does not appear in the proof (index: 1)
[querySMT.debug] ∀ (bv0 : ℤ), ... appears (2)
[querySMT.debug] ∀ (bv0 : ℤ), ... appears (3)
[querySMT.debug] (¬f sk0 ...) appears (4)
[querySMT.debug] f sk0 ... does not appear (5)
[querySMT.debug] (sk1 sk0 = sk0) ...not appear (6)
[querySMT.debug] (f (sk1 sk0) = f sk0) ... not (7)
```

Proof Reconstruction

```
example (f : ℤ → ℤ) (h1: ∀ x y, f x = f y → x = y)
(h2 : ∃ x, ∀ y, f x ≤ f y) :
∃ x, ∀ (y : ℤ), x ≠ y → f x < f y := by
@[classical] byContradiction
intro negGoal
skolemizeAll
have smtLemma0 : ∀ (_i_0 : ℤ), ¬f sk0 + -1 * f _i_0 ≥ 1 := by grind
have smtLemma2 : ∀ (bv0 : ℤ), f bv0 + -1 * f (sk1 bv0) ≥ 0 := by grind
have smtLemma3 : ∀ (bv0 : ℤ), ¬bv0 = sk1 bv0 := by grind
have smtLemma4 : (¬f sk0 + -1 * f (sk1 sk0) ≥ 0 ∨ f sk0 = f (sk1 sk0))
∨ f sk0 + -1 * f (sk1 sk0) ≥ 1 := by grind
duper [h1, smtLemma0, smtLemma2, smtLemma3, smtLemma4] []
```

Conclusion

- New hint-based approach to leveraging SMT solvers for ITP automation
  - Translates Lean goals to SMT-LIB, extracts preprocessing and theory lemmas from cvc5, produces self-contained proof scripts after minimization
- Evaluated on Int, Nat, and List benchmarks from Mathlib
  - QuerySMT compares favorably to existing SMT-related Lean automation
  - Hints produce a clear improvement in the underlying proof automation
  - Only ∼5% of hints fail to be certified by `grind`
- Future work
  - Support more SMT theories beyond integers and algebraic datatypes
  - Collect quantifier instances from cvc5 proofs to speed up Duper
  - Better instrumentation for AC operator normalization
  - Port to other proof assistants / SMT solvers

Hint-Based SMT Proof Reconstruction

Joshua Clune[1] Haniel Barbosa[2] Jeremy Avigad[1]

[1]Carnegie Mellon University, [2]Universidade Federal de Minas Gerais

TACAS 2026

2026–04–13, Torino, IT

Experimental Results

- **lean-auto + cvc5**: translates to SMT-LIB and trusts cvc5 proofs (theoretical upper bound)
- **querysmt**: full pipeline with hints from cvc5
- **querysmt$^-$**: like querysmt but without cvc5 hints (only unsat core minimization)
- **lean-smt**: proof replay approach for cvc5 in Lean
- **grind**: built-in Lean tactic inspired by SMT solvers (no external solver)

	Int Theorems	**Nat Theorems**	**List Theorems**
Total	2058	3270	4576
lean-auto + cvc5	1137	1486	891
querysmt	840	892	749
querysmt$^-$	472	627	708
lean-smt	333	35	445
grind	541	812	–