Dependently typed superposition in Lean

Gabriel Ebner

Matryoshka 2018
2018-06-27

TU Wien

Lean

- Proof assistant based on dependent type theory
  - terms, types, formulas, proofs are all expressions
  - small kernel (unlike Coq)
- uses axiom of choice and classical logic
  - but avoided outside of proofs
- Tactics/metaprograms are defined in the object language

Syntax

```
c               -- constants
x               -- variables
t s
λ x : t, s
Π x : t, s      -- often written ∀ or →
Sort u          -- Prop, Type
```

- e.g.

```
∀ α : Type, ∀ β : Type, ∀ r : ring α,
∀ m : module α β r, ∀ x : β,
```

$1 \cdot x = x$

Syntax

```
c              -- constants
x              -- variables
t s
λ x : t, s
Π x : t, s     -- often written ∀ or →
Sort u         -- Prop, Type
```

- e.g.

```
∀ α : Type, ∀ β : Type, ∀ r : ring α,
∀ m : module α β r, ∀ x : β,
smul α β r m (one α r) x = x
```

`super`

- Superposition prover
- Implemented 100% in Lean
  - First "large" metaprogram
- Uses Lean expressions, unification, proofs
- 2200 lines of code
  - (including toy SAT solver)
- Think of metis in Isabelle

Intended logic

- Complete for first-order logic with equality
- Higher-order not a focus
  - but don’t fail on lambdas
  - no encoding for applicative FOL
- Some inferences for inductive data types

Introduction

Calculus

Implementation

Conclusion

Clauses

- How to represent a clause $a, b \vdash c, d$?

1. $\neg a \vee \neg b \vee c \vee d$
2. $a \to b \to c \vee d$
3. $a \to b \to \neg c \to \neg d \to \text{false}$
   - also used in Coq by Bezem, Hendriks, de Nivelle 2002

Intuitionistic reasoning

- Actually: $a \to b \to (c \to F) \to (d \to F) \to F$
    - (F is a definition for the original goal)

$\to$ often intuitionistic proofs

- e.g. for assumptions like $\forall x, \bigwedge A \to \bigvee B$
- want to avoid classical reasoning on types

- also makes use of `decidable` instances

Quantifiers

```
∀ α, ∀ β, ∀ r : ring α,
∀ m : module α β r, ∀ x : β,
(smul α β r m (one α r) x = x → F) → F
```

- only perform inferences on non-dependent literals
- when literals are resolved away,
  we get new non-dependent literals

Quantifiers

```
∀ α, ∀ β, ∀ r : ring α,
module α β r → β → F
```

- only perform inferences on non-dependent literals
- when literals are resolved away,
  we get new non-dependent literals

Quantifiers

```
∀ α, ∀ β,
ring α → β → F
```

- only perform inferences on non-dependent literals
- when literals are resolved away,
  we get new non-dependent literals

Skolemization

- Skolemization not sound in general
  - requires non-empty domain

$\rightarrow$ add extra (implicit) argument to Skolem function

  - $\forall x, P \rightarrow \exists y : \alpha, Q\, x\, y$ becomes
    $\forall x, \forall z : \alpha, P \rightarrow \forall x, Q\, x\, (fzx)$

- automatically discharged for `nonempty` instances

Refinements

- Subsumption
- Term ordering
- Literal selection
- Demodulation
- Splitting (Avatar)

Term ordering

- Standard lexicographic path order
- Curried applications $fabc$ are treated as $f(a, b, c)$
- Type parameters, type-class instances
  are also arguments!
  - does not seem to be a performance problem
- λ and Π expressions are treated as (unknown) variables

Avatar-style splitting

- Splits clause into variable-disjoint components

$$\frac{\vdash Px, Qy}{\vdash Px} \qquad \frac{\vdash Px, Qy}{\vdash Qy}$$

Avatar-style splitting

- Splits clause into variable-disjoint components

$$\frac{\vdash Px, Qy}{s_1 \vdash Px} \quad \frac{\vdash Px, Qy}{s_2 \vdash Qy} \quad \frac{}{\vdash s_1, s_2}$$

- $s_1 := \forall x\, Px$
- $s_2 := \forall y\, Qy$

- No inferences are performed on these splitting atoms

$\rightarrow$ sent to SAT solver instead

Lean-specific rules

$$\frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta} \text{ if } A \text{ has a type-class instance}$$

- such as `inhabited`, `is_associative`, ...

$$\frac{\Gamma \vdash \Delta, \mathtt{cons}\ a\ b = \mathtt{cons}\ c\ d}{\Gamma \vdash \Delta, a = c \wedge b = d} \qquad \cancel{\mathtt{cons}\ a\ b = \mathtt{nil}, \Gamma \vdash \Delta}$$

Introduction

Calculus

Implementation

Conclusion

General approach

- reuse built-in data structures
    - expressions
    - proofs
    - unifier
- Lean's unifier essentially does:
    - pattern unification
    - definitional reduction
    - some heuristics
- rewriting only at first-order argument positions

Proof construction

- unification, type inference only work with local context

→ clauses are stored in the local context:

```
cls_0: A → (B → F) → F
cls_1: (A → F) → F
cls_2: B → F
```

$\vdash$ F

⚠ does not work with universe polymorphism

Proof construction

- unification, type inference only work with local context

$\rightarrow$ clauses are stored in the local context:

```
cls_0: A → (B → F) → F
cls_1: (A → F) → F
cls_2: B → F
cls_3: (B → F) → F
```

$\vdash$ F

⚠ does not work with universe polymorphism

Proof construction

- unification, type inference only work with local context

$\rightarrow$ clauses are stored in the local context:

```
cls_0: A → (B → F) → F
cls_1: (A → F) → F
cls_2: B → F
cls_3: (B → F) → F
cls_4: F
⊢ F
```

⚠ does not work with universe polymorphism

Proof construction

- unification, type inference only work with local context

→ clauses are stored in the local context:

```
cls_0: A → (B → F) → F
cls_1: (A → F) → F
cls_2: B → F
cls_3: (B → F) → F
cls_4: F
⊢ F
```

- avoids exponential blowup
- post-processing step to remove unused subproofs

⚠ does not work with universe polymorphism

SAT proof construction

- For each literal on the trail, store proof of:
  $A \qquad A \to F \qquad (A \to F) \to F$
  - decision literals have fresh local constants
  - propagated literals have actual proofs
- on conflict, add lambdas for the decision literals
- produces intuitionistic proofs

State transformer

```
meta structure prover_state :=
(active : rb_map clause_id derived_clause)
(passive : rb_map clause_id derived_clause)
(newly_derived : list derived_clause)
(prec : list expr)
-- ...

meta def prover := state_t prover_state tactic
```

Universe polymorphism

- Hypothesis in the local context cannot be universe polymorphic
  - You can't even have `∀ x, x ++ [] = x` for all types
- Possible workaround: create a new environment
  - extra type-checking
  - not intended use (errors and warnings are printed directly)

→ manually implement proof handling

Performance: metavariables

- For unification, we instantiate $\forall x\, Px \to F$ as $P\,?\mathtt{m\_1} \to F$
  - built-in unification uses metavariables
- Afterwards, we quantify over the free metavariables
- Pretty slow

$\to$ Lean 4 will expose temporary metavariables

Performance: unifier

- unpredictable performance
- performance problem even with few clauses

$\rightarrow$ do some prefiltering

$\rightarrow$ implement term indexing

  - non-trivial, idiomatic code relies on definitional equality

Other future work

- Simplifier integration
  - different term order for $x \cdot y = y \cdot x$
- AC redundancy checks
- Heterogeneous equality, congruence lemmas

Conclusion

- Not yet production-ready
    - performance subpar
    - missing support for universe polymorphism
- Lean 4 should bring useful APIs
    - temporary metavariables
- long term: proof reconstruction for “leanhammer”