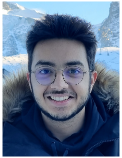
LISA – A Set-Theory Based Proof System

Simon Guilloud   Sankalp Gambhir   Viktor Kunčak

EPFL
School of Computer and Communication Sciences
Lausanne, Switzerland
{firstname.lastname}@epfl.ch

September 21, 2023

Introduction

LISA is a proof assistant in continuous development.

- Based on FOL and set theory
- LCF/de Bruijn model with a trusted kernel (but explicit proofs)
- Developed in Scala

Introduction

LISA is a proof assistant in continuous development.

- Based on FOL and set theory
- LCF/de Bruijn model with a trusted kernel (but explicit proofs)
- Developed in Scala

LISA's ultimate goal is to serve both as prover for mathematical theorems and program correctness.

Writing Proofs in LISA: Example

```
val x = variable
val P = predicate(1)
val f = function(1)

val fixedPointDoubleApplication = Theorem(
        ∀(x, P(x) ⟹ P(f(x))) ⊢ P(x) ⟹ P(f(f(x)))
    ) {
    assume(∀(x, P(x) ⟹ P(f(x))))
    val step1 = have(P(x) ⟹ P(f(x))) by InstantiateForall
    val step2 = have(P(f(x)) ⟹ P(f(f(x)))) by InstantiateForall
    have(thesis) by Tautology.from(step1, step2)
    }
```

LISA's Logic: FOL

LISA uses First Order Logic as its foundational language.

- It has schematic predicate and function symbols (free second-order variables).
- This enables expression of axiom and theorem schemas.

$$'P(0) \land \forall x. \left('P(x) \implies 'P(x+1)\right) \vdash \forall x. 'P(x)$$

LISA's Logic: FOL

LISA uses First Order Logic as its foundational language.

- It has schematic predicate and function symbols (free second-order variables).
- This enables expression of axiom and theorem schemas.

$$'P(0) \land \forall x. \, ('P(x) \implies 'P(x+1)) \vdash \forall x. 'P(x)$$

- Those symbols can be instantiated, but cannot be bound and behave otherwise like uninterpreted symbols.
- Does not change provability of non-schematic formulas!

LISA's Proof System: LK

LISA uses a variation of Sequent Calculus LK.

- Sequents $\Gamma \vdash \Delta$, with Γ and Δ sets of formulas

LISA's Proof System: LK

LISA uses a variation of Sequent Calculus LK.

- Sequents $\Gamma \vdash \Delta$, with Γ and Δ sets of formulas
- Introduction rule for each logical symbol on each side + Cut, Weakening

LISA's Proof System: LK

LISA uses a variation of Sequent Calculus LK.

- Sequents $\Gamma \vdash \Delta$, with Γ and Δ sets of formulas
- Introduction rule for each logical symbol on each side + Cut, Weakening
- Deduced rules for efficiency:
    - Instantiation of schematic symbols
    - Substitution of equal terms/formulas.

LISA's Proof System: LK

LISA uses a variation of Sequent Calculus LK.

- Sequents $\Gamma \vdash \Delta$, with Γ and Δ sets of formulas
- Introduction rule for each logical symbol on each side + Cut, Weakening
- Deduced rules for efficiency:
  - Instantiation of schematic symbols
  - Substitution of equal terms/formulas.

$$\frac{\Gamma \vdash \phi[s/'f], \Delta}{\Gamma, s = t, \vdash \phi[t/'f], \Delta}\ \texttt{SubstEq} \qquad \frac{\Gamma \vdash \phi[a/'p], \Delta}{\Gamma, a \leftrightarrow b \vdash \phi[b/'p], \Delta}\ \texttt{SubstIff}$$

Proofs

- Proofs in Sequent Calculus are Directed Acyclic Graphs.
- In LISA, serialized into lists of proof steps.
- Dependence on axioms, definitions and previous theorems stated explicitly.

Proofs

- Proofs in Sequent Calculus are Directed Acyclic Graphs.
- In LISA, serialized into lists of proof steps.
- Dependence on axioms, definitions and previous theorems stated explicitly.

0	`Hypothesis`	$\phi \vdash \phi$
1	`Weakening(0)`	$\phi \vdash \phi, \psi$
2	`RightImplies(1)`	$\vdash \phi, (\phi \to \psi)$
3	`LeftImplies(2, 0)`	$(\phi \to \psi) \to \phi \vdash \phi$
4	`RightImplies(3)`	$\vdash ((\phi \to \psi) \to \phi) \to \phi$

Built-in Automation: Ortholattices

Dealing with visually obvious syntactic equivalence, such has commutativity, is frustrating, and makes proofs longer.

$$\frac{\vdash b \wedge a \qquad a \wedge b \vdash c}{\vdash c}\ \text{Cut}$$

Who wants a proof rejected because $a \wedge b \not\equiv b \wedge a$?

Built-in Automation: Ortholattices

Dealing with visually obvious syntactic equivalence, such has commutativity, is frustrating, and makes proofs longer.

$$\frac{\vdash b \wedge a \qquad a \wedge b \vdash c}{\vdash c}\ \text{Cut}$$

Who wants a proof rejected because $a \wedge b \not\equiv b \wedge a$?

- Solution: Heuristic? No

Built-in Automation: Ortholattices

Dealing with visually obvious syntactic equivalence, such has commutativity, is frustrating, and makes proofs longer.

$$\frac{\vdash b \wedge a \qquad a \wedge b \vdash c}{\vdash c}\ \text{Cut}$$

Who wants a proof rejected because $a \wedge b \not\equiv b \wedge a$?

- Solution: Heuristic? No
- Solution: replace syntactic equality checking by a more powerful equivalence
- But still *efficiently* decidable
- Sound approximation of Boolean algebra

Ortholattices

Commutativity	$x \vee y = y \vee x$
Associativity	$x \vee (y \vee z) = (x \vee y) \vee z$
Idempotence	$x \vee x = x$
Constants laws	$x \vee 1 = 1$
Double negation	$\neg\neg x = x$
Excluded middle	$x \vee \neg x = 1$
De Morgan's law	$\neg(x \vee y) = \neg x \wedge \neg y$
Absorption	$x \vee (x \wedge y) = x$

$\rightarrow$ Ortholattices

Ortholattices

Commutativity	$x \vee y = y \vee x$
Associativity	$x \vee (y \vee z) = (x \vee y) \vee z$
Idempotence	$x \vee x = x$
Constants laws	$x \vee 1 = 1$
Double negation	$\neg\neg x = x$
Excluded middle	$x \vee \neg x = 1$
De Morgan's law	$\neg(x \vee y) = \neg x \wedge \neg y$
Absorption	$x \vee (x \wedge y) = x$

$\rightarrow$ Ortholattices

Boolean Algebra without distributivity

Distributivity:	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

Equivalence Checker

LISA's Kernel contains an algorithm to decide:

Given two formulas ϕ and ψ, does $\phi \sim_{OL} \psi$ hold?

Equivalence Checker

LISA's Kernel contains an algorithm to decide:

Given two formulas ϕ and ψ, does $\phi \sim_{OL} \psi$ hold?

- Worst case $\mathcal{O}(n^2)$ time
- Also alpha-equivalence, symmetry and reflexivity of equality...

Equivalence Checker

LISA's Kernel contains an algorithm to decide:

Given two formulas ϕ and ψ, does $\phi \sim_{OL} \psi$ hold?

- Worst case $\mathcal{O}(n^2)$ time
- Also alpha-equivalence, symmetry and reflexivity of equality...
- Proof Checker uses it instead of syntactic equality.
- Works particularly well in combination with substitution rules.

Other example:

$$\frac{\vdash [(a \lor b) \land (a \lor c)] \lor b}{\vdash a \lor b} \text{ Restate}$$

Ortholattices

Ortholattice-based reasoning has potential way beyond proof assistants

- Core part of Stainless (program verifier) now:
  - Simplify formulas before passing them to SMT solver
  - Normalization used for caching solved formulas

Why Set Theory

Most theorem provers are based on higher order logic or type theory

- HOL familiy, Isabelle, Coq, Lean...

But set theory has seen successful use too!

- Mizar, Isabelle/ZF, Isabelle/HOL/TG, TLA$^+$

And it is the most widely accepted foundation of mathematics among mathematician studying foundations.

Prototypical foundational set theory.

- $\in, \subseteq, \bigcup, \{a, b\}, \mathcal{P}(\cdot)$

Prototypical foundational set theory.

- $\in, \subseteq, \bigcup, \{a, b\}, \mathcal{P}(\cdot)$
- Comprehension: $\{x | x \in A \wedge P(x)\}$

ZFC

Prototypical foundational set theory.

- $\in, \subseteq, \bigcup, \{a, b\}, \mathcal{P}(\cdot)$
- Comprehension: $\{x | x \in A \wedge P(x)\}$
- Replacement: $\{f(x) | x \in A\}$

ZFC

Prototypical foundational set theory.

- $\in, \subseteq, \bigcup, \{a, b\}, \mathcal{P}(\cdot)$
- Comprehension: $\{x | x \in A \wedge P(x)\}$
- Replacement: $\{f(x) | x \in A\}$
- Choice

- Functions as represented as their graph: $f = \{(x, y) | f(x) = y\}$

- Functions as represented as their graph: $f = \{(x, y)|f(x) = y\}$
- Function spaces: for A and B sets, $A \to B$ is a definable set

Set Theory

- Functions as represented as their graph: $f = \{(x, y) | f(x) = y\}$
- Function spaces: for A and B sets, $A \to B$ is a definable set
- Encode dependant function spaces too.
- (Medium term goal: embed HOL, inductive data types)

Why Set Theory (ZFC+TG)

- Built-in functions and inductive definitions: Easier early game
- Set theory foundations are lower-level,
  - With an initial effort in development, automation and presentation, can make it arbitrarily familiar.
- All usual formalism can be simulated.

LISA's Implementation

- LISA is developed as a Scala library
- Kernel is in a restricted subset of Scala, $\rightarrow$ future formal verification.

LISA's Implementation

- LISA is developed as a Scala library
- Kernel is in a restricted subset of Scala, $\rightarrow$ future formal verification.
- Everything else is in Scala 3:
  - DSL for proof writing
  - Strong type safety via precise types

- A tactic = A scala function that produce a proof
- Can use all features and library of Scala
- Can mix programming with DSL for proofs
- A Propositional solver is 20 loc

6 Virtues of Modern Proof Systems

LISA strives to follow these key design features

- Efficiency
- Trust
- Usability
- Predictability
- Interoperability
- Programmability

Conclusions

- FOL with schematic symbols and set theory
- Equivalence Checker modulo Ortholattices for formulas
- Explicit and self-contained proofs
- Expresive DSL

Find LISA on GitHub: github.com/epfl-lara/lisa