Introduction to CSLib: The Lean Computer Science Library

Leaning In! 2026

Presenter: Sorrachai Yingchareonthawornchai (ETH Zurich)

A convergence of ambitions in 2025

➔ Scale up formal computer science and programming by building a common infrastructure. CSLib – www.cslib.io

Clark Barrett

Stanford & Amazon

Swarat Chaudhuri

Google DeepMind & UT Austin

Jim Grundy

Amazon

Pushmeet Kohli

Google DeepMind

Fabrizio Montesi

FORM, University of Southern Denmark

Leonardo de Moura

Lean FRO & Amazon

*Slides thanks to Fabrizio Montesi

What is CSLib?

An open source repository (github.com/leanprover/cslib) of

1. computer science definitions and results,
2. verified software components, and
3. verification infrastructure.

*Slides thanks to Fabrizio Montesi

What is CSLib?

An open source repository (github.com/leanprover/cslib) of

1. computer science definitions and results,
2. verified software components, and
3. verification infrastructure.

★ Written in Lean, with mathlib as dependency.

*Slides thanks to Fabrizio Montesi

What is CSLib?

An open source repository (github.com/leanprover/cslib) of

1. computer science definitions and results,
2. verified software components, and
3. verification infrastructure.

★ Written in Lean, with mathlib as dependency.

★ Curated by experts.

*Slides thanks to Fabrizio Montesi

What is CSLib?

An open source repository (github.com/leanprover/cslib) of

1. computer science definitions and results,
2. verified software components, and
3. verification infrastructure.

★ Written in Lean, with mathlib as dependency.

★ Curated by experts.

★ Done in collaboration with the Lean community and FRO.

*Slides thanks to Fabrizio Montesi

Governance

- Steering Group
  - (strategy, funding)

Clark Barrett

STANFORD UNIVERSITY AND AMAZON

Swarat Chaudhuri

GOOGLE DEEPMIND AND UT AUSTIN

Jim Grundy

AMAZON

Pushmeet Kohli

GOOGLE DEEPMIND

Leo de Moura

LEAN CHIEF ARCHITECT, CO-FOUNDER OF LEAN FRO, AND AMAZON

Fabrizio Montesi

UNIVERSITY OF SOUTHERN DENMARK AND DANISH INSTITUTE FOR ADVANCED STUDY

Governance

- Maintainer Group

  (technical leadership and supervision)

Fabrizio Montesi

UNIVERSITY OF SOUTHERN DENMARK AND DANISH INSTITUTE FOR ADVANCED STUDY

Alexandre Rademaker

ATLAS COMPUTING AND GETULIO VARGAS FOUNDATION

Sorrachai Yingchareonthawornchai

ETH ZURICH

Chris Henson

DREXEL UNIVERSITY. LAMBDA CALCULUS, METAPROGRAMMING

Kim Morrison

LEAN FRO. CONTINUOUS INTEGRATION AND DEPLOYMENT (CI/CD) WITH UPSTREAM (LEAN, MATHLIB)

Governance

- Discussions on GitHub, Zulip, Email, …
  - https://github.com/leanprover/cslib/graphs/contributors

Who is CSLib for?

- Researchers
  - Verify your claims.
  - Speed up your development.
  - Consolidate and offer your findings through our APIs.

*Slides thanks to Fabrizio Montesi

Who is CSLib for?

- Researchers
- Educators & Learners
  - Explore CS through a unified language.
  - Interact with the tool.

*Slides thanks to Fabrizio Montesi

Who is CSLib for?

- Researchers
- Educators & Learners
- Programmers
  - Use CSLib components to write reliable software.
  - Use CSLib's languages and logics to model and implement systems.
  - Use CSLib's verification infrastructure to verify new and old code.

*Slides thanks to Fabrizio Montesi

Who is CSLib for?

- Researchers
- Educators & Learners
- Programmers
- AI Developers
  - Train AI on CSLib.
  - Increase the production of specs, programs, and proofs.

*Slides thanks to Fabrizio Montesi

Who is CSLib for?

- Researchers
- Educators & Learners
- Programmers
- AI Developers
- Interdisciplinary Researchers
  - Explore the application of formal methods to provide actionable information to users.

*Slides thanks to Fabrizio Montesi

CSLib vision

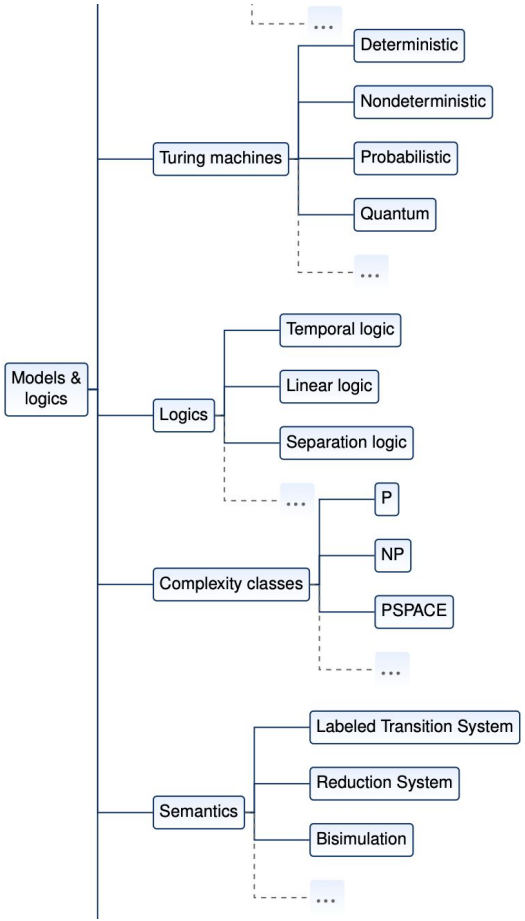
- Pillar 1: Formalizing theoretical foundation of CS
  - Computational Models & Logics
  - Verified the Analysis of Algorithms and Data Structures
- Pillar 2: Infrastructure for Reasoning about Everyday Code
  - Will be based on a new IVL called Boole – Imperative language + Lean specification
  - With machinery for generating Lean-language verification conditions from Boole code

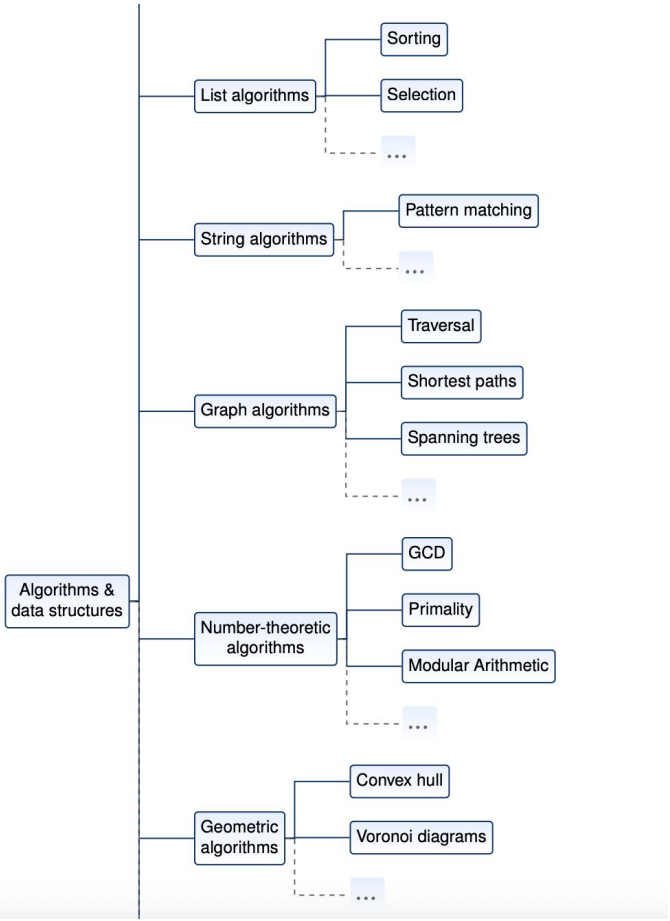
Pillar 1 : Formalizing theoretical foundation of CS

```
Cslib/
    Algorithms/
    Computability/
    Foundations/ # General, reusable foundations
    Languages/       # Modelling and programming
    Logics/          # Various logics for reasoning
    [more to come...]
```

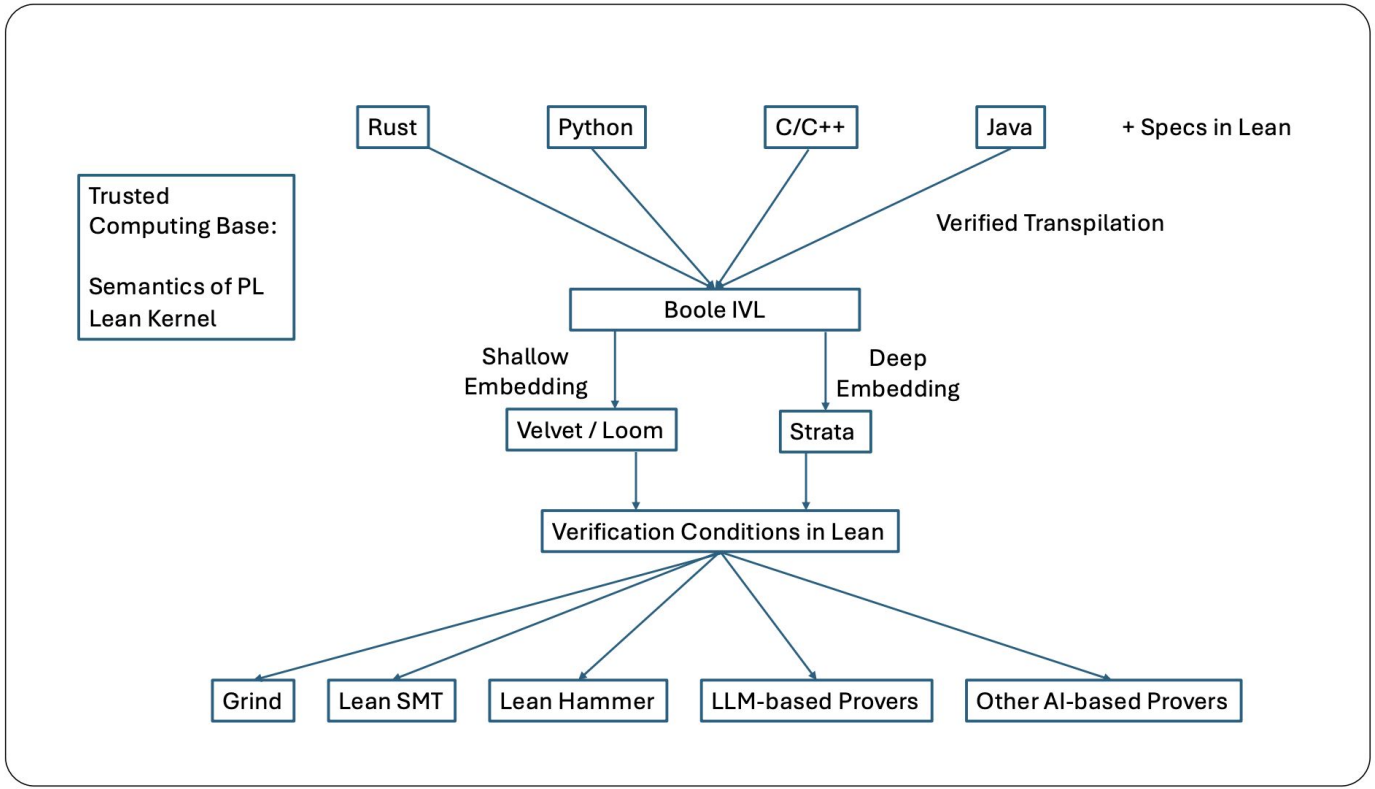
See Fabrizio Montesi's talk (Lean together 2026) for more discussion

Pillar 1 Topics of Interests

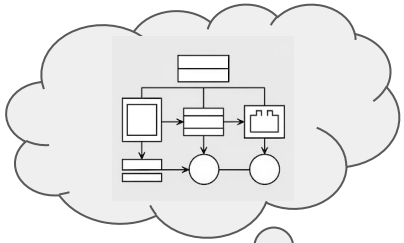




Pillar 2 : Infrastructure for Reasoning about Everyday Code

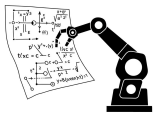


Usage scenario: Reasoning about code



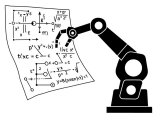
CSLib API

Pillar 1: Data structures, algorithms, theorems about correctness, theorems about resource use,...



Lean specifications

Boole code



Lean proof

QED

Programming & specification

Theorem-proving

Analyzing Algorithms using CSLib

- Framework for writing algorithms and analyze time complexity
- **Key design principle:** Balancing robustness with ergonomics
- Example,

TimeM (= writer monad with a lot of simp lemmas)

  - **Annotate your code**
    - **Insert** “tick” as +1 on this line
  - Treat the time complexity as side-effect of the program
  - separation of concern:
    - Correctness proofs and running time become **orthogonal**

```
structure TimeM (α : Type*) where
  ret : α
  time : ℕ

namespace TimeM

protected def pure {α} (a : α) : TimeM α :=
  ⟨a, 0⟩

def bind {α β} (m : TimeM α) (f : α → TimeM β) : TimeM β :=
  let r := f m.ret
  ⟨r.ret, m.time + r.time⟩

instance : Monad TimeM where
  pure := pure
  bind := bind

/-- Advances the time cost by `c` (default 1) without changing the value. -/
def tick (c : ℕ := 1) : TimeM PUnit := ⟨.unit, c⟩

macro "✓[" c:term "]" body:doElem : doElem => `(doElem| do TimeM.tick c; body:doElem)
/-- `✓ x` is `pure x`, adding one tick. -/
macro "✓" body:doElem : doElem => `(doElem| ✓[1] body)
```

How to use TimeM

1. Write a normal Lean code
2. **Annotate** your cost using **TimeM**
3. State theorems about correctness and time complexity
    a. Simp lemmas will do heavy lifting !

```
import ComplexityAPI
variable {α : Type} [LinearOrder α]

def merge :  List α → List α  → TimeM (List α)
  | [], ys => return ys
  | xs, [] => return xs
  | x::xs', y::ys' => do
    ✓ let c := x ≤ y
    if c then
      let rest ← merge xs' (y::ys')
      return (x :: rest)
    else
      let rest ← merge (x::xs') ys'
      return (y :: rest)

/-- Merge sort algorithm that returns a `TimeM (List α)`
    number of comparisons. -/
def mergeSort (xs : List α) : TimeM (List α) :=  do
  if xs.length < 2 then return xs
  else
    let half := xs.length / 2
    let L := xs.take half
    let R := xs.drop half
    let L' ← mergeSort L
    let R' ← mergeSort R
    let result ←  merge L' R'
    return result
```

```
      return (y :: rest)

  /-- Merge sort algorithm that returns a `TimeM (List α)`
      number of comparisons. -/
  def mergeSort (xs : List α) : TimeM (List α) :=  do
    if xs.length < 2 then return xs
    else
      let half := xs.length / 2
      let L := xs.take half
      let R := xs.drop half
      let L' ← mergeSort L
      let R' ← mergeSort R
      let result ←  merge L' R'
      return result


/-- MergeSort is functionally correct. -/
theorem mergeSort_correct (xs : List α) :
  IsSorted (mergeSort xs).ret ∧ List.Perm (mergeSort xs).ret xs  := by
  -- Proof omitted

/-- Time complexity of mergeSort counted as the number of comparisons. -/
theorem mergeSort_time (xs : List α) :
  let n := xs.length
  (mergeSort xs).time ≤ n * clog2 n := by
  -- Proof omitted
```

TimeM Demo

See also the tutorial/lecture notes: https://github.com/sorrachai/FAA2025

- TimeM
- Powered by *grind* and *functional induction (*see also *Markus Himmel's* talk*)*

Ongoing activities

- New contributions are welcome (see also, https://github.com/leanprover/cslib/blob/main/CONTRIBUTING.md)
- Pillar 1
  - Complexity APIs; TimeM, Query models,
  - Graph algorithms and graph foundations
  - Automata (on finite and infinite words)
  - Lambda calculi
- Pillar 2
  - Boole specifications: specifications that reference arbitrary Lean concepts
  - Front ends for Boole: translations from real-world programming languages to Boole
  - Back ends for Boole: Boole back ends that take as input Boole programs with formal specifications and construct proof obligations in Lean

Questions?