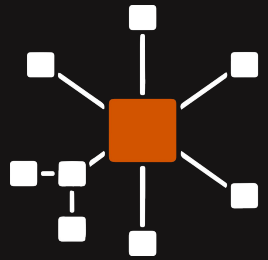


CSLib
The Lean Computer Science Library

Lean Together, 20 January 2026

Presenter:

Fabrizio Montesi
Director, Professor, FORM
CSLib Lead Maintainer

fabriziomontesi.com
linkedin.com/in/fmontesi

University of Southern Denmark
Centre for Formal Methods and Future Computing
D·IAS
DANISH INSTITUTE FOR ADVANCED STUDY

Computing is under pressure

Computing is under pressure

More quantity

More systems!

More features!

Cheaper!

Faster!

Computing is under pressure

More quantity	More quality
More systems!	More secure!
More features!	More robust!
Cheaper!	More sustainable!
Faster!	More privacy!

A convergence of ambitions in 2025

A convergence of ambitions in 2025

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

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

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.

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.

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.

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.

Governance

- Steering Group (strategy, funding)
  - Clark Barrett, Swarat Chaudhuri, Jim Grundy, Pushmeet Kohli, Fabrizio Montesi, Leonardo de Moura.
- Maintainer Group (technical leadership and supervision)
  - Lead Maintainer: Fabrizio Montesi
  - Tech Leads: Alexandre Rademaker, Sorrachai Yingchareonthawornchai
  - Area Maintainers: Chris Henson, Kim Morrison
- 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.

Who is CSLib for?

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

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.

Who is CSLib for?

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

Who is CSLib for?

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

Where are we now?

Credits:
https://github.com/leanprover/cslib/graphs/contributors

Current structure

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

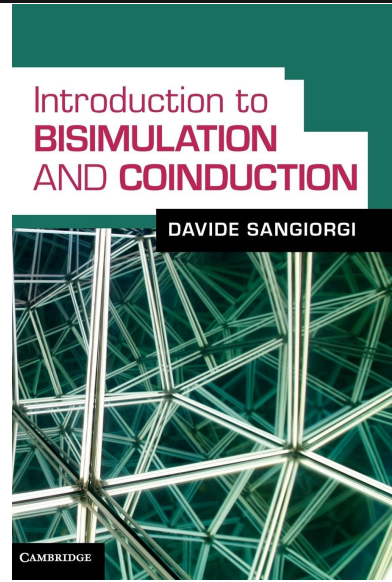
Current structure

Many answers found!

A lot more questions lie ahead…

Foundations

- ★ Labelled Transition Systems.
  - Classes of LTSs (image-finite, deterministic, etc.)
  - Bisimilarity, weak bisimilarity.
  - Similarity.
  - Trace equivalence (w/ inclusion in bisimilarity).
- ★ Reduction systems.
  - Lots about relations (confluence, etc.).
- ★ Freer monads.



Grind-first approach. (See Chris' talk tomorrow!)

Foundations

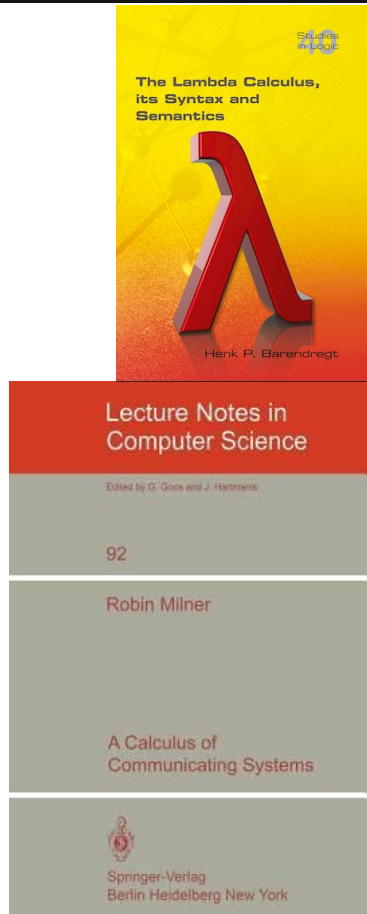
💡 Grind-first approach.

```
117  130    /-- The inverse of a bisimulation is a bisimulation. -/
118       - theorem Bisimulation.inv (h : Bisimulation lts r) :
119       -   Bisimulation lts (flip r) := by
120       -   simp only [Bisimulation] at h
121       -   simp only [Bisimulation]
122       -   intro s1 s2 hrinv μ
123       -   constructor
124       -   case left =>
125       -     intro s1' htr
126       -     specialize h s2 s1 hrinv μ
127       -     have h' := h.2 s1' htr
128       -     obtain ⟨ s2', h' ⟩ := h'
129       -     exists s2'
130       -   case right =>
131       -     intro s2' htr
132       -     specialize h s2 s1 hrinv μ
133       -     have h' := h.1 s2' htr
134       -     obtain ⟨ s1', h' ⟩ := h'
135       -     exists s1'
          131 + @[grind]
          132 + theorem Bisimulation.inv (h : lts.IsBisimulation r) :
          133 +   lts.IsBisimulation (flip r) := by grind [flip]
```

Languages

- ★ λ-calculus.
  - Untyped, typed (STLC, System Fsub).
  - Alternative formalisations of α-equivalence: named (standard), locally nameless, well-scoped (WIP).
- ★ Calculus of Communicating Systems (CCS).
  - Behavioural theory, w/ equivalences and proof that bisimilarity is a congruence.
- ★ Combinatory Logic (SKI).
  - Rice's theorem.

💡 Generality; reusable APIs.



Languages

💡 Generality; reusable APIs.

```
/-- A type `α` has a computable `fresh` function if it is always possible, for any finite set
of `α`, to compute a fresh element not in the set. -/
class HasFresh (α : Type u) where
  /-- Given a finite set, returns an element not in the set. -/
  fresh : Finset α → α
  /-- Proof that `fresh` returns a fresh element for its input set. -/
  fresh_notMem (s : Finset α) : fresh s ∉ s
```

Logics

- ★ Linear Logic.
  - ○ Sequent calculus.
  - ○ Many logical equivalences.
    - ■ $a \otimes 0 \equiv 0$, $a \otimes (b \oplus c) \equiv (a \otimes b) \oplus (a \otimes c)$, etc.
  - ○ η-expansion (proven correct).
  - ○ Phase semantics.
- ★ WIP: Propositional Logic.

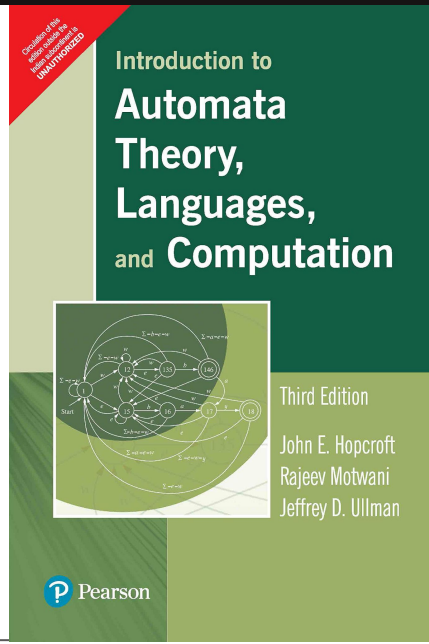
💡 Derivations are in Type; notation can be controversial; ‘pray to grind’ for dealing with Multiset rewriting.

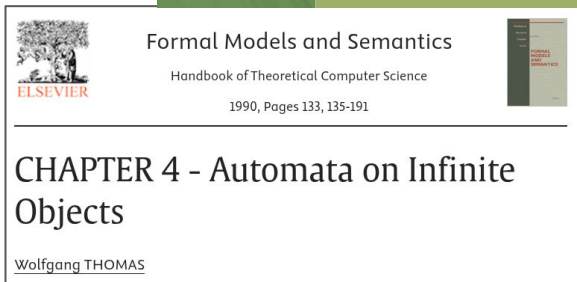
Logics

Automata and Their Languages

- ★ Deterministic and nondeterministic automata based on labelled transition systems.
- ★ Acceptors for finite and infinite words (generalisations of DFA, NFA, εNFA, Buchi, Muller, etc.).
- ★ Theory of ω-languages and ω-regular languages.

💡 Interactions between Labelled Transition Systems and Automata.





Automata and Their Languages

Interactions between Labelled Transition Systems and Automata.

Automata and Their Languages

Interactions between Labelled Transition Systems and Automata.

```
/-- A nondeterministic automaton extends an `LTS` with a set of initial states. -/
structure NA (State Symbol : Type*) extends LTS State Symbol where
  /-- The set of initial states of the automaton. -/
  start : Set State
```

Automata and Their Languages

Interactions between Labelled Transition Systems and Automata.

```
/-- A nondeterministic automaton extends an `LTS` with a set of initial states. -/
structure NA (State Symbol : Type*) extends LTS State Symbol where
  /-- The set of initial states of the automaton. -/
  start : Set State
```

```
/-- A nondeterministic automaton that accepts finite strings (lists of symbols). -/
structure FinAcc (State Symbol : Type*) extends NA State Symbol where
  /-- The set of accepting states. -/
  accept : Set State
```

Automata and Their Languages

Interactions between Labelled Transition Systems and Automata.

```
/-- A nondeterministic automaton extends an `LTS` with a set of initial states. -/
structure NA (State Symbol : Type*) extends LTS State Symbol where
  /-- The set of initial states of the automaton. -/
  start : Set State
```

```
/-- A nondeterministic automaton that accepts finite strings (lists of symbols). -/
structure FinAcc (State Symbol : Type*) extends NA State Symbol where
  /-- The set of accepting states. -/
  accept : Set State
```

```
/-- Nondeterministic Muller automaton. -/
structure Muller (State Symbol : Type*) extends NA State Symbol where
  /-- The set of sets of accepting states. -/
  accept : Set (Set State)
```

Automata and Their Languages

Interactions between Labelled Transition Systems and Automata.

LTS images:

```
/-- The `DA.FinAcc` constructed from an `NA.FinAcc` has the same language. -/
@[scoped grind _=_]
theorem toDAFinAcc_language_eq {na : NA.FinAcc State Symbol} :
    language na.toDAFinAcc = language na := by
  ext xs
  grind
```

Automata and Their Languages

Interactions between Labelled Transition Systems and Automata.

LTS images:

```
/-- The `DA.FinAcc` constructed from an `NA.FinAcc` has the same language. -/
@[scoped grind _=_]
theorem toDAFinAcc_language_eq {na : NA.FinAcc State Symbol} :
    language na.toDAFinAcc = language na := by
  ext xs
  grind
```

Saturation:

```
theorem toNAFinAcc_language_eq {ena : εNA.FinAcc State Symbol} :
    language ena.toNAFinAcc = language ena := by
  ext xs
  have : ∀ s s', ena.saturate.MTr s (xs.map some) s' = ena.saturate.noε.MTr s xs s' := by
    simp [LTS.noε_saturate_mTr]
  grind
```

Automata and Their Languages

💡 Interactions between Labelled Transition Systems and Automata.

LTS images:

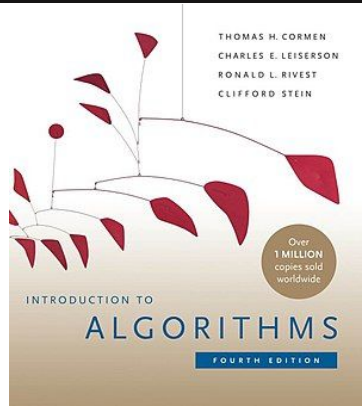
```
/-- The `DA.FinAcc` constructed from an `NA.FinAcc` has the same language. -/
@[scoped grind _=_]
theorem toDAFinAcc_language_eq {na : NA.FinAcc State Symbol} :
    language na.toDAFinAcc = language na := by
  ext xs
  grind
```

Saturation:

```
theorem toNAFinAcc_language_eq {ena : εNA.FinAcc State Symbol} :
    language ena.toNAFinAcc = language ena := by
  ext xs
  have : ∀ s s', ena.saturate.MTr s (xs.map some) s' = ena.saturate.noε.MTr s xs s' := by
    simp [LTS.noε_saturate_mTr]
  grind
```

Plus general results on infinite executions and saturation (not shown).

Algorithms

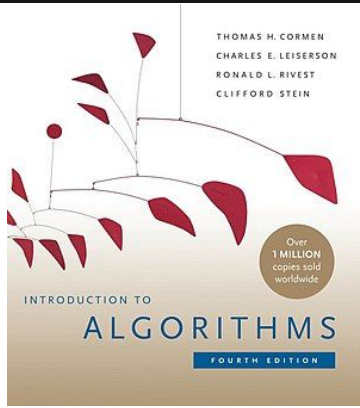


- ★ Framework for writing algorithms and proving their time complexity in terms of atomic operations.
  - MergeSort (functional correctness, time complexity on number of comparisons).
  - WIP: more algorithms, space complexity, more automation.
- ★ WIP: Framework based on query models.
  - Separate definition of query model.
  - Algorithms perform queries.

💡 Balancing robustness with ergonomics.

Algorithms

Balancing robustness with ergonomics.

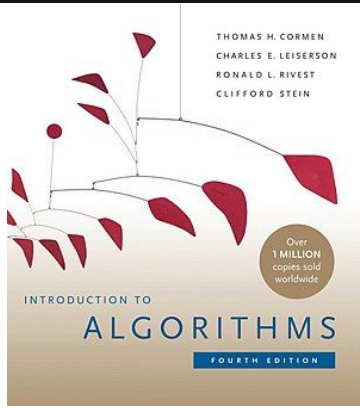



```
def merge :  List α → List α → TimeM (List α)
  | [], ys => return ys
  | xs, [] => return xs
  | x::xs', y::ys' => do
    let c ← ✓ (x ≤ y : Bool)
    if c then
      let rest ← merge xs' (y::ys')
      return (x :: rest)
    else
      let rest ← merge (x::xs') ys'
      return (y :: rest)
```

```
/-- Time complexity of mergeSort -/
theorem mergeSort_time (xs : List α) :
    let n := xs.length
    (mergeSort xs).time ≤ n * clog2 n:= by
  grind [mergeSort_time_le, timeMergeSortRec_le]
```

Algorithms

Balancing robustness with ergonomics.



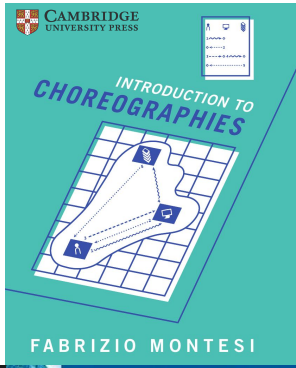

```
def merge : List α → List α → TimeM (List α)
  | [], ys => return ys
  | xs, [] => return xs
  | x::xs', y::ys' => do
    let c ← ✓ (x ≤ y : Bool)
    if c then
      let rest ← merge xs' (y::ys')
      return (x :: rest)
    else
      let rest ← merge (x::xs') ys'
      return (y :: rest)
```

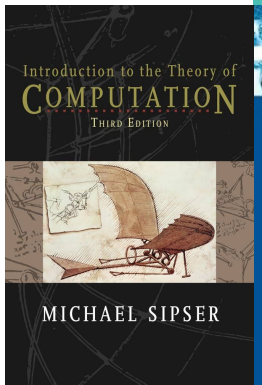
```
/-- Time complexity of mergeSort -/
theorem mergeSort_time (xs : List α) :
    let n := xs.length
    (mergeSort xs).time ≤ n * clog2 n:= by
  grind [mergeSort_time_le, timeMergeSortRec_le]
```

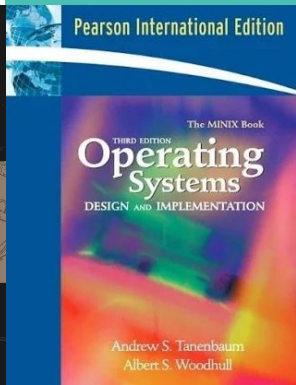
What next?

Next steps

- ➔ More Computer Science.
  - Database theory.
  - Cost|Probabilistic|Denotational|* semantics.
  - Concurrency theory (Petri nets, pi-calculus, etc.).
  - Choreographic languages (security protocols, choreographic programming, business processes, multiparty session types, etc.).
  - Logics: modal logics, separation logic, etc.
  - …





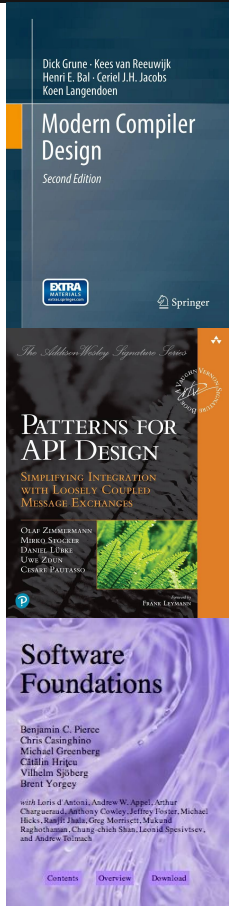


Next steps

- ➔ More Computer Science.
- ➔ Verification infrastructure.
  - ◆ Development of IRs for verification (Boole).
  - ◆ Compilers from mainstream languages to Boole.
  - ◆ Integration of logics with verification infrastructure.

Next steps

- ➔ More Computer Science.
- ➔ Verification infrastructure.
- ➔ Programming infrastructure.
  - ◆ Design and implementation of high-level languages with certified compilers.
  - ◆ Bridge to SE: Architecture Description Languages, Design Patterns, Provably-Correct Refactorings, etc.
  - ◆ Integration w/ system middleware (microservices, serverless, etc.).



Thank you for the amazing contributions!

Thank you for listening!

And thanks for funding my time and our group to:
:-)

Funded by the European Union

European Research Council
Established by the European Commission

Co-funded by the European Union (ERC, CHORDS, 101124225). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

University of Southern Denmark
Centre for Formal Methods and Future Computing

VILLUM FONDEN