Different runs may return different optimal trees.

Parallel branch-and-bound

deterministic assume (data == data') {
  // Parallel branch-and-bound
  Tree t = min_phylo_tree(N, data);
  assert (t.cost == t'.cost);
}

Goal: Decompose effort of verifying parallelism and verifying functional correctness.

- Prove parallel correctness simply — not entangled in complex sequential functional correctness.
- Verify functional correctness in a sequential way.

Question: What is parallel correctness?

SPECIFYING DETERMINISM

- Previous work: Deterministic specifications.
  [Burnim and Sen, FSE 2009]
  - Idea: Parallel correctness means every thread schedule gives semantically equivalent results.
  - Internal nondeterminism, but deterministic output.
  - Assert that parallel code yields semantically equivalent outputs for equivalent inputs.

Lightweight spec of parallel correctness.

- Independent of complex functional correctness.
- Great for testing (with, e.g., active testing).
- Can automatically infer likely specifications [Burnim and Sen, ICSE 2010].

Not a complete spec of parallel correctness.

- Specification ignores tree t in Figure 1.
- For complex programs, determinism proof attempts get entangled in details of sequential correctness.

OVERVIEW

Verifying parallel programs is very challenging.

- Painful to reason simultaneously about correctness of parallelism and about functional correctness.
- Functional correctness often largely sequential.

Goal: Decompose effort of verifying parallelism and verifying functional correctness.

- Prove parallel correctness simply — not entangled in complex sequential functional correctness.
- Verify functional correctness in a sequential way.

OUR APPROACH

For a parallel program, use a sequential but nondeterministic version as a specification.

- User annotates intended algorithmic nondeterminism
- We interpret parallel constructs as nondeterministic and sequential.

Parallelism is correct if it adds no unintended nondeterminism.

- I.e., if parallel and nondeterministic sequential versions of the program are equivalent.

PROOF BY REDUCTION

Reduction: Method for proving atomicity.
[Lipton, CACM 1974]

- Program operations classified as right-movers and left-movers if they commute to the right/left with all operations that can run in parallel with them.
- Code block is atomic if a sequence of right-movers, one non-mover, and a sequence of left-movers.
- Implies all parallel runs equivalent to ones where atomic code block is run serially.

PREVIOUS WORK

- Formal proof rules for parallel and nondeterministic sequential equivalence.
- Automated proofs of parallel correctness.
- Combine with verification tools for sequential programs with nondeterminism.
  - Model checking with predicate abstraction (CEGAR).
  - Can verify functional correctness on sequential code!
- Apply above to real parallel benchmarks.
- Applications to debugging?
  - Allow programmer to sequentially debug a parallel execution by mapping a parallel trace to a nondeterministic sequential one.

SPECIFYING DETERMINISM

- Previous work: Deterministic specifications.
  [Burnim and Sen, FSE 2009]
  - Idea: Parallel correctness means every thread schedule gives semantically equivalent results.
  - Internal nondeterminism, but deterministic output.
  - Assert that parallel code yields semantically equivalent outputs for equivalent inputs.

Lightweight spec of parallel correctness.

- Independent of complex functional correctness.
- Great for testing (with, e.g., active testing).
- Can automatically infer likely specifications [Burnim and Sen, ICSE 2010].

Not a complete spec of parallel correctness.

- Specification ignores tree t in Figure 1.
- For complex programs, determinism proof attempts get entangled in details of sequential correctness.

OVERVIEW

Verifying parallel programs is very challenging.

- Painful to reason simultaneously about correctness of parallelism and about functional correctness.
- Functional correctness often largely sequential.

Goal: Decompose effort of verifying parallelism and verifying functional correctness.

- Prove parallel correctness simply — not entangled in complex sequential functional correctness.
- Verify functional correctness in a sequential way.

Question: What is parallel correctness?

SPECIFYING DETERMINISM

- Previous work: Deterministic specifications.
  [Burnim and Sen, FSE 2009]
  - Idea: Parallel correctness means every thread schedule gives semantically equivalent results.
  - Internal nondeterminism, but deterministic output.
  - Assert that parallel code yields semantically equivalent outputs for equivalent inputs.

Lightweight spec of parallel correctness.

- Independent of complex functional correctness.
- Great for testing (with, e.g., active testing).
- Can automatically infer likely specifications [Burnim and Sen, ICSE 2010].

Not a complete spec of parallel correctness.

- Specification ignores tree t in Figure 1.
- For complex programs, determinism proof attempts get entangled in details of sequential correctness.

OVERVIEW

Verifying parallel programs is very challenging.

- Painful to reason simultaneously about correctness of parallelism and about functional correctness.
- Functional correctness often largely sequential.

Goal: Decompose effort of verifying parallelism and verifying functional correctness.

- Prove parallel correctness simply — not entangled in complex sequential functional correctness.
- Verify functional correctness in a sequential way.

Question: What is parallel correctness?

SPECIFYING DETERMINISM

- Previous work: Deterministic specifications.
  [Burnim and Sen, FSE 2009]
  - Idea: Parallel correctness means every thread schedule gives semantically equivalent results.
  - Internal nondeterminism, but deterministic output.
  - Assert that parallel code yields semantically equivalent outputs for equivalent inputs.

Lightweight spec of parallel correctness.

- Independent of complex functional correctness.
- Great for testing (with, e.g., active testing).
- Can automatically infer likely specifications [Burnim and Sen, ICSE 2010].

Not a complete spec of parallel correctness.

- Specification ignores tree t in Figure 1.
- For complex programs, determinism proof attempts get entangled in details of sequential correctness.

OVERVIEW

Verifying parallel programs is very challenging.

- Painful to reason simultaneously about correctness of parallelism and about functional correctness.
- Functional correctness often largely sequential.

Goal: Decompose effort of verifying parallelism and verifying functional correctness.

- Prove parallel correctness simply — not entangled in complex sequential functional correctness.
- Verify functional correctness in a sequential way.

Question: What is parallel correctness?

SPECIFYING DETERMINISM

- Previous work: Deterministic specifications.
  [Burnim and Sen, FSE 2009]
  - Idea: Parallel correctness means every thread schedule gives semantically equivalent results.
  - Internal nondeterminism, but deterministic output.
  - Assert that parallel code yields semantically equivalent outputs for equivalent inputs.

Lightweight spec of parallel correctness.

- Independent of complex functional correctness.
- Great for testing (with, e.g., active testing).
- Can automatically infer likely specifications [Burnim and Sen, ICSE 2010].

Not a complete spec of parallel correctness.

- Specification ignores tree t in Figure 1.
- For complex programs, determinism proof attempts get entangled in details of sequential correctness.

OVERVIEW

Verifying parallel programs is very challenging.

- Painful to reason simultaneously about correctness of parallelism and about functional correctness.
- Functional correctness often largely sequential.

Goal: Decompose effort of verifying parallelism and verifying functional correctness.

- Prove parallel correctness simply — not entangled in complex sequential functional correctness.
- Verify functional correctness in a sequential way.

Question: What is parallel correctness?

SPECIFYING DETERMINISM

- Previous work: Deterministic specifications.
  [Burnim and Sen, FSE 2009]
  - Idea: Parallel correctness means every thread schedule gives semantically equivalent results.
  - Internal nondeterminism, but deterministic output.
  - Assert that parallel code yields semantically equivalent outputs for equivalent inputs.

Lightweight spec of parallel correctness.

- Independent of complex functional correctness.
- Great for testing (with, e.g., active testing).
- Can automatically infer likely specifications [Burnim and Sen, ICSE 2010].

Not a complete spec of parallel correctness.

- Specification ignores tree t in Figure 1.
- For complex programs, determinism proof attempts get entangled in details of sequential correctness.