Our proposal:

For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our technique: Improves traditional conflict serializability by flipping default value of *

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.

Parallelism correctness. Prove independently of complex & sequential functional properties.

Functional correctness. Reason about sequentially without thread interleavings.

Question: What is parallelism correctness?

Previous work: Deterministic specifications. [Burnim and Sen, FSE 2009]

- Idea: Parallel correctness means every thread schedule gives semantically equivalent results.

Deterministic assume (data == data’) {
// Parallel branch-and-bound search
Solution best = parallel-search(data);
assert (cost(best) == cost(best’));
}

Generating a consistent thread interleaving from the parallel branch-and-bound search is a classical nondeterministic speculation problem.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our technique: Improves traditional conflict serializability by flipping default value of *

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.

Parallelism correctness. Prove independently of complex & sequential functional properties.

Functional correctness. Reason about sequentially without thread interleavings.

Question: What is parallelism correctness?

Previous work: Deterministic specifications. [Burnim and Sen, FSE 2009]

- Idea: Parallel correctness means every thread schedule gives semantically equivalent results.

Deterministic assume (data == data’) {
// Parallel branch-and-bound search
Solution best = parallel-search(data);
assert (cost(best) == cost(best’));
}

Example: Flip *true to false
- Skip body of if(*)

Figure 1. Deterministic specification for a generic parallel branch-and-bound search.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.

Not a complete spec of parallel correctness! For complex programs, determinism proof attempts get entangled in details of sequential correctness.

Our proposal: For a parallel program, use a sequential but nondeterministic (NDSeq) version as the intermediate specification.

Lightweight annotations: Specify intended nondeterminism without referring to an explicit functional specification.