SEPARATING PARALLEL AND FUNCTIONAL CORRECTNESS

JACOB BURNIM, GEORGE NECULA, KOUSHIK SEN

BERKELEY PARLAB

HOTPAR 2010 – BERKELEY, CA – JUNE 14, 2010

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.

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

Figure 1. Deterministic spec for parallel branch-and-bound search to find minimum-cost phyogenetic trees. Different runs may return different optimal trees.

- 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.

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.

```
parallel-for (w in queue):
  if (lower_bnd(w) >= best):
    continue
  if (size(w) < T):
    (soln, cost) = find_min(w)
    atomic:
      if cost < best:
        best = cost
        best_soln = soln
  else:
    queue.addAll(split(w))
```

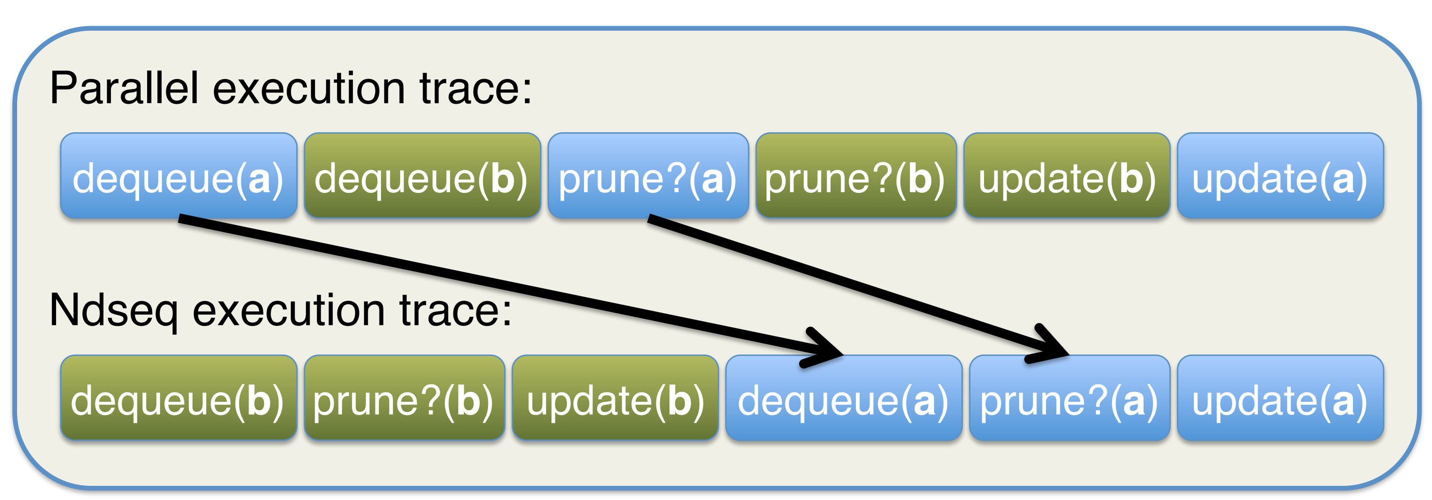
Figure 2. Generic parallel branch-and-bound search.

```
nondet-for (w in queue):
  if (lower_bnd(w) >= best && *):
    continue
  if (size(w) < T):
    (soln, cost) = find_min(w)
    atomic:
      if cost < best:
        best = cost
        best_soln = soln
  else:
    queue.addAll(split(w))
```

Figure 3. Nondeterministic but sequential branch-and-bound.

PROVING PARALLEL CORRECTNESS

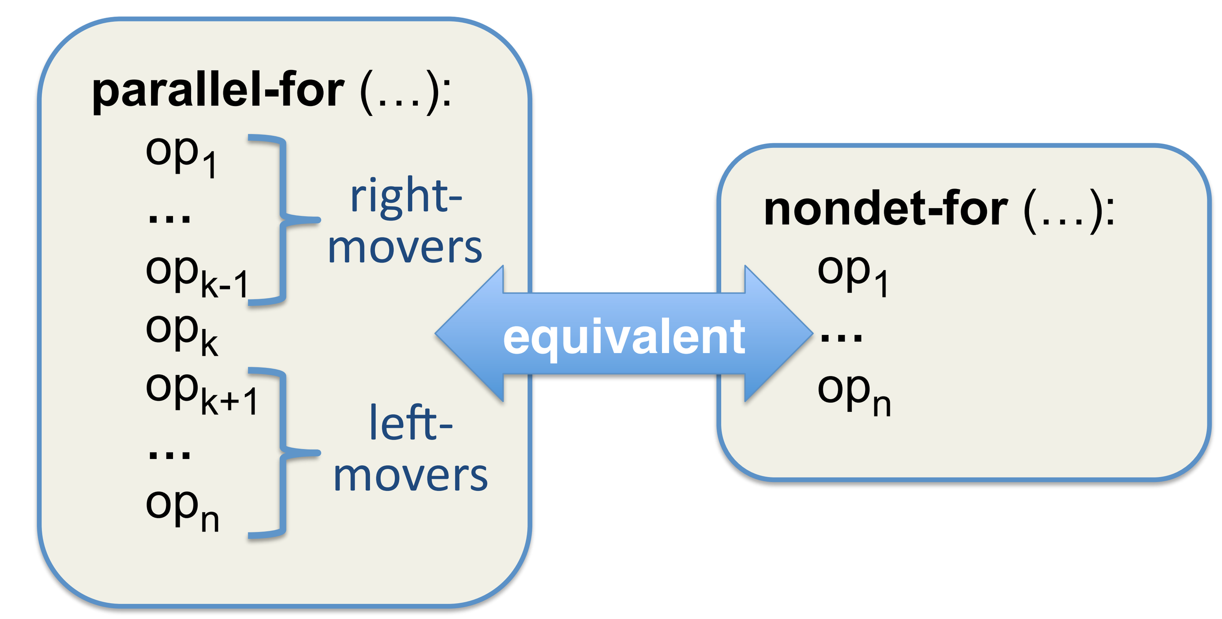
- **Goal**: Prove each execution of a parallel program is equivalent to a nondeterministic sequential (**ndseq**) execution.



- Added nondeterminism allows `prune?(a)` to be moved past `update(b)` without changing the program's behavior.

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.



- **Idea**: Statically prove that operations are right- and left-movers using SMT solving.
  - Encode: Are all behaviors of $op_1 ; op_2$ also behaviors of $op_2 ; op_1$?
  - Like [Elmas, Qadeer, and Tasiran, POPL 2009].

FUTURE 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.