Determinism for Multithreaded Code

Asserting and Checking Determinism for Parallel Programs

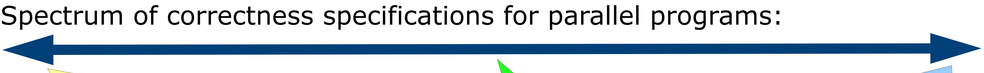
Jacob Burnim ▪ Koushik Sen ▪ Par Lab, University of California, Berkeley

A Case for Determinism

- Increasing cores-per-chip requires parallel software
- Parallel software is more difficult to write/debug than sequential counterpart
  - Unexpected thread interleavings may yield unintended results
- To make parallel programs easy to write and debug, we try to make them behave as sequential programs, that is
  - Same input should yield semantically same output (i.e. **deterministic** output)
  - Non-determinism from thread scheduling should not yield different outputs

Specifying Parallel Correctness

- Spectrum of correctness specifications for parallel programs:



No source of non-determinism (e.g. no race)

- Pro: No programmer specification
- Con: Lack of precision
  - Races could be benign or harmful

Full Functional Correctness

- Pro: Very precise.
- Con: Difficult to write.
  - May requires reimplementing code sequentially, possibly in an assertion language

Deterministic Specification: A Sweet Spot?

- Proposal: specify deterministic behavior
  - A parallel program behaves as if it is sequential
- Easy for programmers
  - Identify outputs that should be the same
- More precise than implicit specs
  - Can distinguish benign data races from races leading to unintended output
  - Also for high-level races, atomicity, etc.

Basic Example

```
deterministic {
  img = par_mandelbrot(params);
}
```

Specifies that, for any two executions with identical initial state (`params`), the resulting states (`img`) should be identical.

Semantic Determinism

```
deterministic {
    float C[][];
    C = par_matrix_mult(A, B);
} assert (|C – C'| < ε);
```

Specifies that, for any two executions from the same initial state, the resulting matrices must have entries equal to within tolerance ε.

Preconditions for Determinism

```
Set set = new RedBlackTreeSet();
...
deterministic assume (set.equals(set')) {
    cobegin {
       set.add(5);
       set.add(7);
    }
} assert (set.equals(set'));
```

Specifies that, for any two executions where `sets` initially contains the same elements, the resulting `sets` will also contain the same elements.

Checking Determinism

- Built a Java library for determinism assertions.
  - Records initial and final state for each block.
  - Checks for every deterministic block:

```
deterministic assume (Pre(s0,s0')) {
    P
} assert (Post(s1,s1'));
```

that for every two runs (s_0,s_1) and (s_0',s_1'):

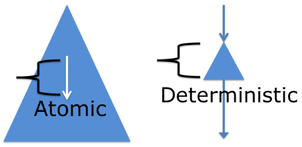
$$Pre(s0,s0') \Rightarrow Post(s0,s1')$$

- **Easy to use:** For benchmarks, writing assertions took only 5-10 minutes.
- **Effective:** For benchmarks, library automatically distinguished benign races from real bugs:

Benchmark		LOC	Threads	Data Races		High-Level Races	
				Found	Bugs	Found	Bugs
JGF	sor	300	10	2	0	0	0
	moldyn	1k	10	2	0	0	0
	lufact	2k	10	1	0	0	0
	raytracer	2k	10	3	1	0	0
	montecarlo	4k	10	1	0	2	0
PJ	pi	15k	4	9	0	1+	1
	keysearch3	15k	4	3	0	0+	0
	mandelbrot	15k	4	9	0	0+	0
	phylogeny	19k	4	4	0	0+	0
tsp		700	5	6	0	2	0

Atomicity vs Determinism

- Atomicity: *Sequential* code not harmed by its *parallel/non-deterministic* environ.
- Determinism: *Parallel* code is essentially sequential despite its *internal* non-determinism.



Use in Verification

Can treat blocks as sequential once determinism verified, avoiding exponential # paths: