Efficient Checking of Thread Refinement

Daniel Poetzl

University of Oxford

Refinement

- Compilers must guarantee refinement between an optimized thread T’ and an original thread T
- We assume the sequential consistency (SC) for data-race-free (DRF) programs execution model (e.g. pthreads)
- Current theories specify the valid optimizations in terms of allowed reorderings, eliminations, and introductions
- Contribution: New way of specifying refinement
  - Based on comparing the state of T’ and T at synchronization operations
  - Supports the implementation of efficient refinement checkers
  - Supports new compiler optimizations

Event-Based Refinement

Original

```
lock   L
write  x 1
write  y 3
read   z 8
read   z 8
unlock L
```

Optimized

```
lock   L
write  y 3
write  x 1
read   z 8
unlock L
```

Event-Based Refinement

Original

```
lock   L
write  x 1
write  y 3
read   z 8
read   z 8
unlock L
```

Optimized

```
lock   L
write  y 3
write  x 1
read   z 8
unlock L
```

Event-Based Refinement

Original

```
lock   L
write  x 1
write  y 3
read   z 8
read   z 8
unlock L
```

Optimized

```
lock   L
write  y 3
write  x 1
read   z 8
unlock L
```

State-Based Refinement

Original

```
lock   L
write  x 1
write  y 3
read   z 8
read   z 8
unlock L
```

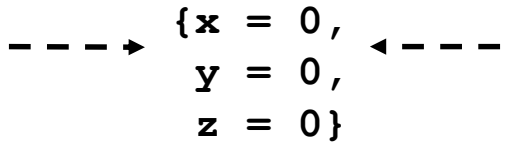
Optimized

```
lock   L
write  y 3
write  x 1
read   z 8
unlock L
```

State-Based Refinement

Original

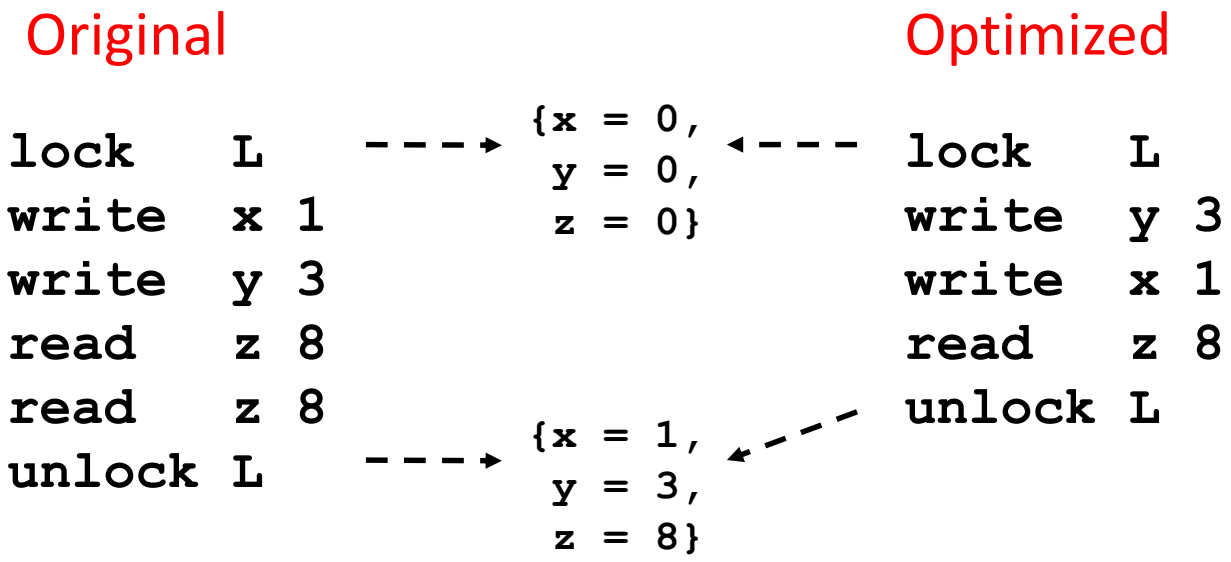
```
lock   L
write  x 1
write  y 3
read   z 8
read   z 8
unlock L
```



Optimized

```
lock   L
write  y 3
write  x 1
read   z 8
unlock L
```

State-Based Refinement



Evaluation: Checking Traces

- Compared our tool *tracecheck* to *cmmtest* (Morisset et al. PLDI ‘13)
- *tracecheck* outperforms *cmmtest* on every program we tried
  - On average ~3 orders of magnitude faster

Evaluation: Checking Traces

- Compared our tool *tracecheck* to *cmmtest* (Morisset et al. PLDI ‘13)
- *tracecheck* outperforms *cmmtest* on every program we tried
  - On average ~3 orders of magnitude faster

Thank You!