Delay-Bounded Scheduling

A canonical characterization of scheduling nondeterminism

Michael Emmi
Shaz Qadeer
Zvonimir Rakamarić
Abstract

• Explore less-nondeterministic concurrent program semantics
• fewer interleavings
• still find bugs
• handle dynamic task-creation
Concurrent Programs

Task Buffer

Executing Task

- dispatch
- yield
- newly-created tasks
- global state
- procedure stack
A General Model

- Preemptive multithreaded programs
- Asynchronous “event-driven” programs
- Distributed multiprocessor programs
- ...
Bug Reachability

bugs exhibited in certain task-interleavings

(we’re looking for certain reachable global-state valuations)

.safe!
Challenges

• Unbounded # of tasks
  • dynamically created
• Unbounded procedure-stack per task
• Exponentially many task-interleavings
  ...for fixed number of tasks
Undecidability

unbounded # finite-state tasks\(^1\)

\[\ldots\]

at least 2 tasks w/recursive procedures\(^2\)

\[\text{two stacks}\]

\[\ldots\text{ we have both }\;-;\]

[1] Apt & Kozen ’86
Interleaving Explosion

- non-det scheduling choice made at each preemption
- each time choose from $N$ tasks

* supposing # tasks is bounded

\[ \# \text{ scheduler invocations} = O(N^I) \]
Fewer Interleavings

Restrict scheduling nondeterminism

• Bound number of choices at each preemption?

• Bound number of preemptions?
Parameterized Underapproximation

- bounding parameter $K$
- prioritized search: target low-$K$ bugs first
- for fixed $K$
  - fewer interleavings
  - decidable reachability*

* in cases where sequential reachability is decidable...
Context Bounding

[Qadeer & Rehof '05]

$O(I^KN^K)$ interleavings

$N$ tasks

K "context-switches"

# scheduler invocations = $I$
# Context Bounding

[Qadeer & Rehof ’05]

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>unrestricted schedule, K interruptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interleavings</td>
<td>$O(I^KN^K)$ — exponential in $K$?</td>
</tr>
<tr>
<td>Complexity*</td>
<td>NP-complete in $K$</td>
</tr>
<tr>
<td>Drawbacks</td>
<td>• only $K$ tasks explored</td>
</tr>
<tr>
<td></td>
<td>— incomplete executions</td>
</tr>
<tr>
<td></td>
<td>• $K$ may depend on $N$</td>
</tr>
<tr>
<td></td>
<td>— interleavings effectively exp. in $N$</td>
</tr>
</tbody>
</table>

* for programs with finite data-domains
Bounded Round-Robin

[Lal & Reps ’08]

$N$ tasks

$O(I^{KN})$ interleavings

$K$ rounds
# Bounded Round-Robin

[Lal & Reps ’08]

<table>
<thead>
<tr>
<th>mechanism</th>
<th>restricted schedule, NK interruptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>interleavings</td>
<td>$O(I^{KN})$ — exponential in $N$</td>
</tr>
<tr>
<td>complexity*</td>
<td>NP-complete in $K$</td>
</tr>
<tr>
<td>features</td>
<td>✓ sequentialization</td>
</tr>
<tr>
<td>drawbacks</td>
<td>• # interleavings depends on # tasks</td>
</tr>
<tr>
<td></td>
<td>— complexity exponential in $N$</td>
</tr>
<tr>
<td></td>
<td>• dynamic task-creation?</td>
</tr>
</tbody>
</table>

* for programs with finite data-domains
Idea

• Restricted schedule (like BRR)
  • but for unbounded # of dynamically-created tasks

• Bound interruptions globally (like CB)
  • rather than per-task (like BRR)
Deterministic Schedule

1  2  3  4
5  6  7  8
9

next-scheduled task

delay?
(i) execute now
(ii) execute later
Delay Bounding

deterministic schedule + K delays

\[ O(I^K) \] interleavings
- pick K out of I to delay

e.g. reschedule delayed tasks after all non-delayed tasks

round 2

round 3
Which Interleavings?

reorder two tasks: 3; 2

reorder three tasks: 4; 3; 2

Will catch bugs exhibited with few reorderings between unbounded # tasks
Application to Testing

Comparison w/ context-bounding in CHESS (explicit-state model checker)

• same, or smaller, delay bound
• all existing bugs found
• new bug in 5-task program
• (OoM) fewer
• tester can use any det. scheduling policy
Dynamic Task Scheduling
DFS Sequentialization

Re-use call-stack for posted tasks

translate "post" into "call"

but be careful with passed global-state...
Multi-round Seq.
Multi-round Seq.
(like [Lal & Reps ’08])

compute all rounds at once

non-det. advance to next global
K-DFS Sequentialization

carry K rounds of global state
... using K extra global variables

ensure $g_2 = g_1$

compute all rounds at once
... by non-det. advancing rounds
Application to Verification

Implemented in STORM (verification engine)

- 20 device drivers, 1-30K LOC
  - dynamic task-creation enables precise modeling...
- Simple implementation
- many new bugs found with 2 delays
Results

- Parameterized concurrent exploration
- fewer interleavings in testing
- sequentialization for verification
- unbounded \# dynamically-created tasks
### Comparison

<table>
<thead>
<tr>
<th>interceptings</th>
<th>context bound</th>
<th>round robin</th>
<th>delay bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma(I^K) / \sigma(I^{NN})$</td>
<td>$\sigma(I^K)$</td>
<td>$\sigma(I^K)$</td>
<td></td>
</tr>
<tr>
<td>complexity*</td>
<td>NP-complete</td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>features</td>
<td>- incomplete executions - effectively exp. in $N$ - bounded tasks</td>
<td>- bounded tasks + sequentialization</td>
<td>- fewer behaviors + unbounded tasks + sequentialization</td>
</tr>
</tbody>
</table>

* for programs with finite data-domains