Lecture 12
Context Bounding Checkers I

Zvonimir Rakamarić
University of Utah

Announcements

- Dafny homework assignment due on Wed (before class)
- Project proposal due on Wed (before class)
- Message me if you would like to discuss your project before Wed
- Posting papers on Canvas accompanying the material covered in lectures
  - Try to read them

Huge Number of Thread Schedules

- Concurrent program with *n* threads where each thread has *k* instructions has

$$(n*k)! \,/\, (k!)^n \geq (n!)^k$$

  interleavings
- Exponential in both n and k!
- Example: 5 threads with 5 instruction each

$$25! \,/\, 5!^5 = 6.2336074e+14$$

$$= 623 \text{ trillion interleavings}$$

Java Path Finder (JPF)

- Program checker for Java
- Properties to be verified
  - Program assertions
  - LTL properties
- Depth-first and breadth-first search, heuristics
  - Uses static analysis techniques to improve the efficiency of the search
- Requires a complete Java program
  - Cannot handle native code

Combating State Space Explosion

- Symmetry reduction
  - Search equivalent states only once
- Partial order reduction
  - Do not search thread interleavings that generate equivalent behavior
- Static analyses
  - Reduce state space using static analyses
- User-provided restrictions
  - Manually bound variable domains, array sizes,…

Symmetry Reduction in JPF

- Order in which classes are loaded shouldn’t effect the state
  - There is a canonical ordering of classes
- Location of dynamically allocated heap objects shouldn’t effect the state
  - If we store the memory location as the state, then we can miss equivalent states which have different memory layouts
  - Store some information about the new statements and the number of times they are executed

Simple Symmetry Example

```
int x, y;
Foo a, b;
```

Thread 1:

```
1) a = new Foo();
2) x = 1;
3) y = 2;
4) x++;
5) y++;
```

Thread 2:

```
5) b = new Foo();
6) y = 3;
7) x = 2;
8) y++;
9) x++;
```

Partial Order Reduction

- Statements of concurrently executing threads can generate many different interleavings
  - All these different interleavings are allowable behavior of the program
- Model checker checks all possible interleavings for errors
  - But different interleavings may generate equivalent behaviors
- Partial order reduction
  - It is sufficient to check just one representative interleaving

Simple POR Example

```
int x, y;
```

Thread 1:

```
int a;
1) a = 5;
2) a++;
3) x = 1;
4) y = 2;
5) x++;
6) y++;
```

Thread 2:

```
int b;
5) b = 10;
6) b--;
7) y = 3;
8) x = 2;
9) y++;
10) x++;
```

Static Analysis in JPF

- Using static analysis techniques to reduce the state space
  - Slicing
  - Partial evaluation

Static Analysis in JPF

- Slicing
  - Remove program parts with no effect on the slicing criterion
    - A slicing criterion could be a program point
  - Program slices are computed using dependency analysis
- Partial evaluation
  - Propagate constant values and simplify expressions

User-Provided Restrictions

- To improve scalability, users can restrict domains of variables, sizes of arrays,…
- Restrictions under-approximate program behaviors
  - May result in missed errors
  - Still useful in finding bugs

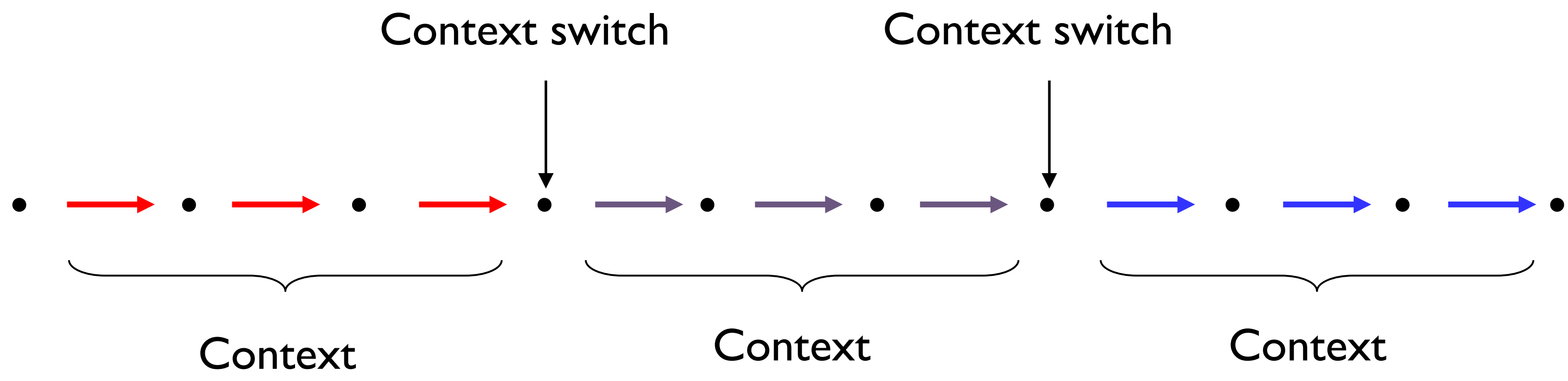
This Time

- Context-bounded verification of concurrent programs

Context-Bounded Verification

slides acknowledgements: Shaz Qadeer, Madan Musuvathi

Context-Bounded Verification



- Many subtle concurrency errors are manifested in executions with few context switches
- Analyze all executions with few context switches

Context-Bounded Reachability Problem

- An execution is c-bounded if every thread has at most c contexts
- Does there exist a c-bounded execution from a state *S* to a state *E*?

CB Reachability is NP-Complete

- Membership in NP
  - Witness is an initial state and n*c sequences each of length at most $|L \times G|$

    n = # of threads, c = # of contexts

    L = # of program locations, G = # of global states
- NP-hardness
  - Reduction from the CIRCUIT-SAT problem

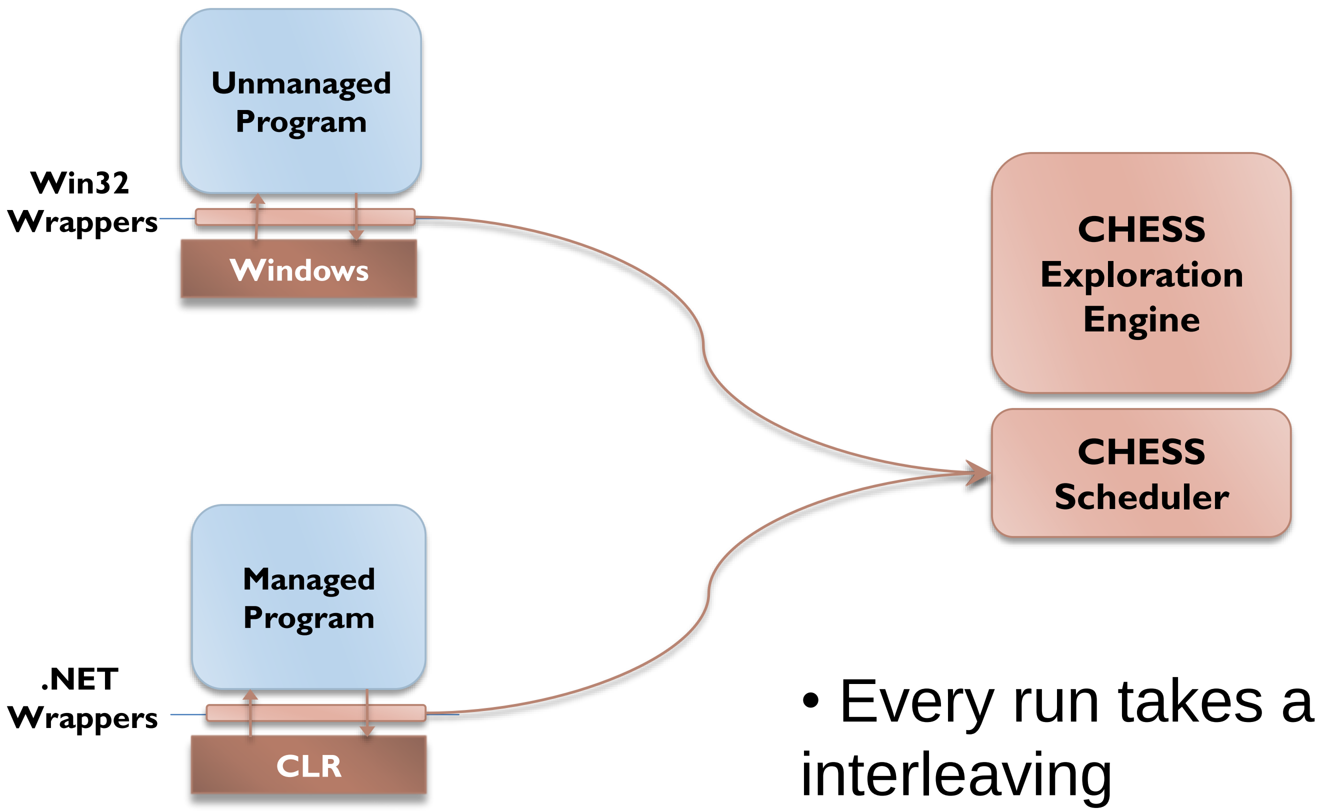
Complexity of Safety Verification

	Unbounded	Context-bounded
Finite-state systems	PSPACE complete	NP-complete
Pushdown systems	Undecidable	NP-complete

CHESS: Systematic Testing for Concurrency

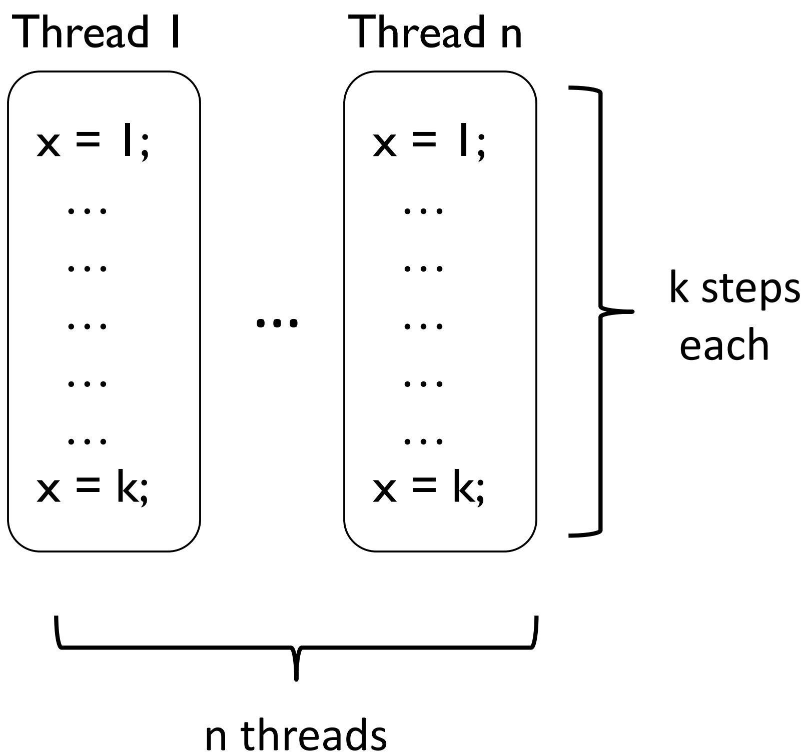
- CHESS is a user-mode scheduler
- Controls all scheduling nondeterminism
  - Replace the OS scheduler
- Guarantees:
  - Every program run takes a different thread interleaving
  - Reproduce the interleaving for every run

CHESS Architecture



- Every run takes a different interleaving
- Reproduce the interleaving for every run

State-Space Explosion

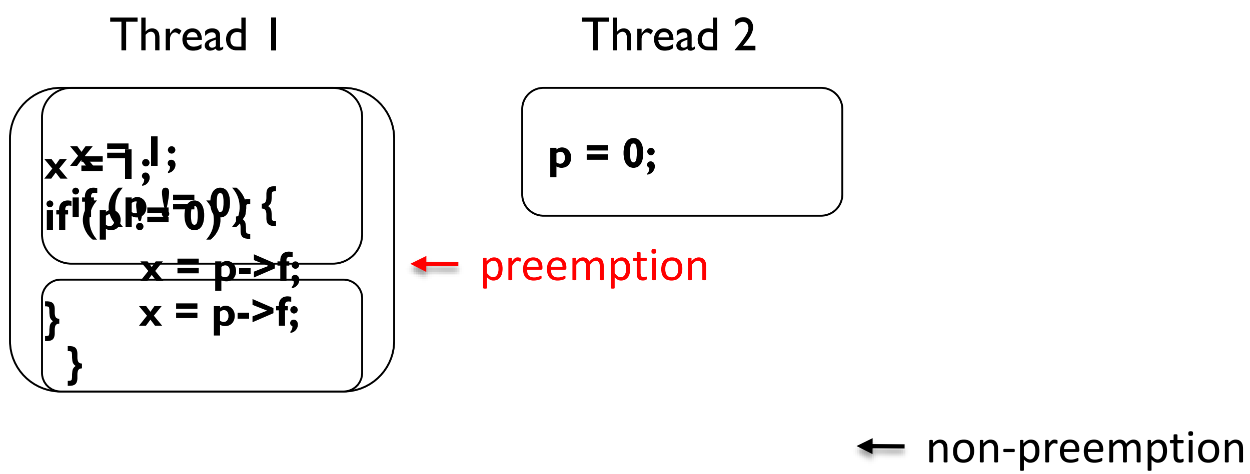


- Number of executions is $O(n^{nk})$
- Exponential in both n and k
  - Typically: n < 10, k > 1000
- Limits scalability to large programs

Goal: Scale CHESS to large programs (large k)

Preemption-Bounding

- Prioritize executions with small # of preemptions
- Two kinds of context switches:
  - Preemptions – forced by the scheduler
    - E.g., time-slice expiration
  - Non-preemptions – a thread voluntarily yields
    - E.g., blocking on an unavailable lock, thread end

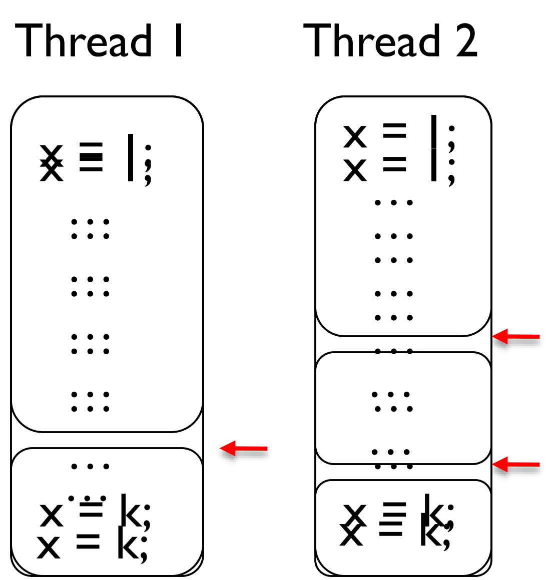


Preemption-Bounding in CHESS

- The scheduler has a budget of c preemptions
  - Nondeterministically choose the preemption points
- Resort to non-preemptive scheduling after c preemptions
- Once all executions explored with c preemptions
  - Try with c+1 preemptions

Property 1: Polynomial Bound

- Terminating program with fixed inputs and deterministic threads
  - n threads, k steps each, c preemptions
- Number of executions <= $_{nk}C_c * (n+c)!$
  $= O((n^2k)^c * n!)$
- Exponential in n and c, but not in k!



- Choose c preemption points
- Permute n+c atomic blocks

Property 2: Simple Error Traces

- Finds smallest number of preemptions to the error
- Number of preemptions better metric of error complexity than execution length

Property 3: Coverage Metric

- If search terminates with preemption-bound of c, then any remaining error must require at least c+1 preemptions
- Intuitive estimate for
  - The complexity of the bugs remaining in the program
  - The chance of their occurrence in practice

Property 4: Many Bugs with Few Preemptions

Program	kLOC	Threads	Preemptions	Bugs
Work-Stealing Queue	1.3	3	2	3
CDS	6.2	3	2	1
CCR	9.3	3	2	2
ConcRT	16.5	4	3	4
Dryad	18.1	25	2	7
APE	18.9	4	2	4
STM	20.2	2	2	2
PLINQ	23.8	8	2	1
TPL	24.1	8	2	9

Coverage vs Preemption-Bound