Concurrency Testing

Finding and Reproducing Heisenbugs in Concurrent Programs
Coverage Guided Systematic Concurrency Testing

Trevis Alleyne
November 4, 2011

Context

- Concurrency testing
  - Multithreading creates bugs that are difficult to find and reproduce

Motivation

- Naïve methods artificially stress test the system, e.g. increasing num. threads
  - Rely on OS to cause bug to appear
- Classic model checking was previously demonstrated, but storing states is difficult
  - Not scalable
- Classic model checking also not guided by coverage
  - Set of interleavings grows exponentially
  - DPOR doesn't scale because creates many redundant equivalence classes

Two Approaches: Chess and Fusion

- Common points:
  - Explore space of interleavings intelligently
  - Stateless model checking
  - Coverage-guided
- Chess (2008)
  - Guided by happens-before graphs
  - C/C++: Windows, .Net
  - Operates on binaries
- Fusion (2011)
  - Guided by history-aware predecessor-sets
  - Learns ordering constraints between shared objects
  - C/C++: Linux/PThreads
  - Needs to instrument source code

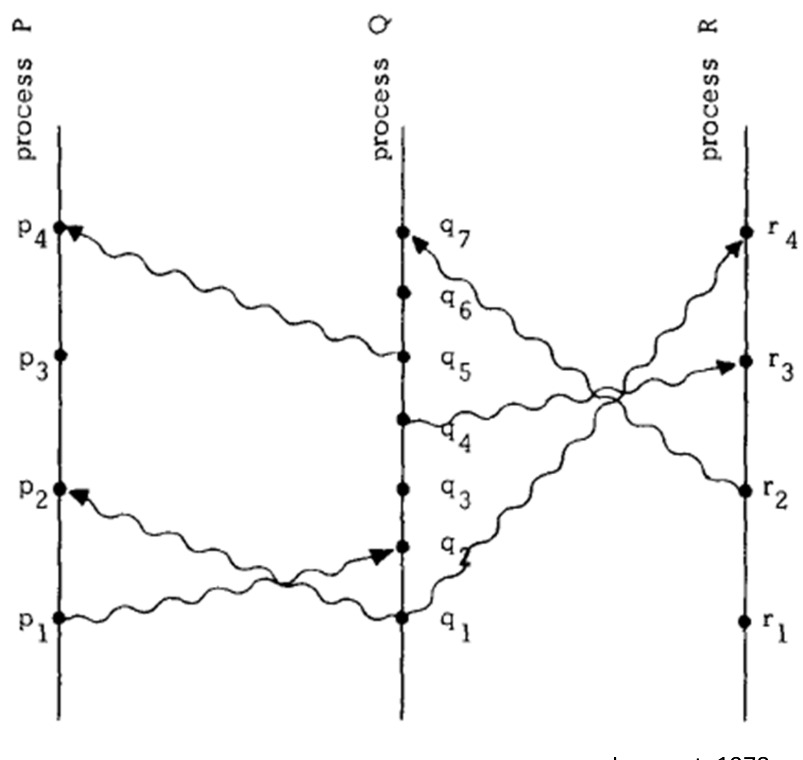
CHESS

Chess: Motivation

- Preemption Bounding: give priority to schedules with *fewer* preemptions
  - Instead, pay attention to the *places* preemption occur, e.g. access to synchronization primitives
- Stateless model checking: caching visited states of large systems is difficult
  - Instead, replay tests by capturing nondeterminism, while being robust to uncontrolled nondeterminism
  - Need way to avoid exploring same program states: happens-before graph
- Minimal perturbation: testing tool should test code as-is without special OS or virtual machine

Methods: Happens-Before Graph (HBG)

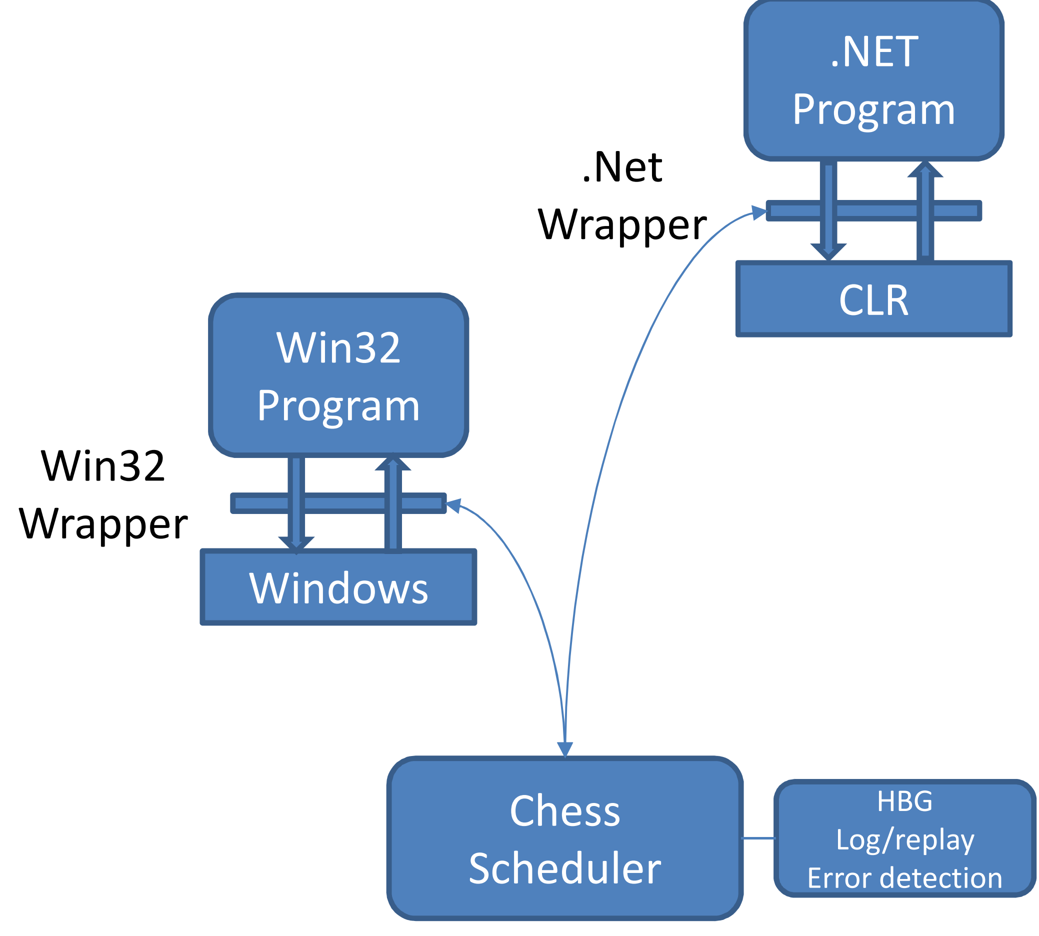
- Way to represent current state using execution trace
  - Same HBG, different order of independent operations
- Each node annotated with triple:
  1. Task
  2. Synchronization variable
  3. Operation: isWrite or isRelease



Lamport, 1978

Chess: Method

- Deterministic search strategy
  - Requires user to generate deterministic input
- Consists of:
  1. Wrapper
  2. Scheduler

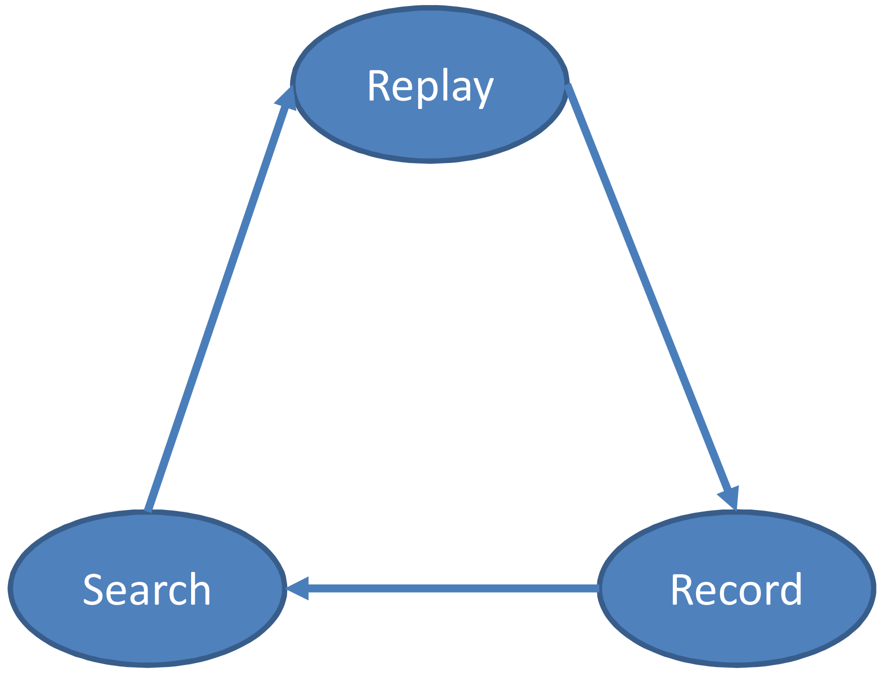


Chess: Wrapper

- Used to create HBG
- Needs to understand API well enough to:
  1. Determine when API calls disable/enable threads
  2. Create triple labels
  3. Inform scheduler about creation/termination of tasks
- Tracks states of threads and mapping of resource handles to synchronization variables
- Use various methods to intercept API calls, e.g. shimming

Chess: Scheduler

- Functions in 3 phases:
  1. Replay
  2. Record
  3. Search
- Builds HBG by monitoring interthread communication:
  1. Access to synchronization variables (wrapper)
  2. Access to shared memory
- Shared memory: with no data-races, watching SV access enough to capture order of communication
  - Real software has data races, prevents replay
  - Chess enables 1 thread at time, prevents concurrent access to memory locations



Scheduler (continued)

- Replay
  - Accounts for common sources of nondeterminism, e.g. lazy-initialization, interference from environment, nondeterministic calls
  - Deals with imperfect replay: rerecords test run or gives up and moves on (loss of coverage)
- Fair scheduling
  - OS schedules are fair, therefore Chess must be fair
  - Explore only fair schedules, prunes search space/tests livelocks
  - Prioritizes fair schedules by giving yielding threads low priority
- Search
  - Preemption bounding: given k steps, allow c places to preempt
  - Heuristically insert preemptions in system functions, avoid base libraries and volatile variables protected by synchronization
  - Caches HBG for each execution to prune search

Chess: Evaluation

- Validated on Win32, .NET, and Singularity
- CCR library
  - <1 hour to integrate Chess and reproduce rare race condition

A

Programs	LOC	max Threads	max Synch.	max Preemp.
PLINQ	23750	8	23930	2
CDS	6243	3	143	2
STM	20176	2	75	4
TPL	24134	8	31200	2
ConcRT	16494	4	486	3
CCR	9305	3	226	2
Dryad	18093	25	4892	2
Singularity	174601	14	167924	1

B

		Failure / Bug		
Programs	Total	Unk/Unk	Kn/Unk	Kn/Kn
PLINQ	1		1	
CDS	1		1	
STM	2			2
TPL	9	9		
ConcRT	4	4		
CCR	2	1	1	
Dryad	7	7		
Singularity	1		1	
Total	27	21	4	2

FUSION

Fusion: Motivation

- Existing methods still too slow
- Programs have implicit assumptions regarding concurrency control, but programmers fail to enforce this

Method: History-aware Predecessor-Sets (HaPSet)

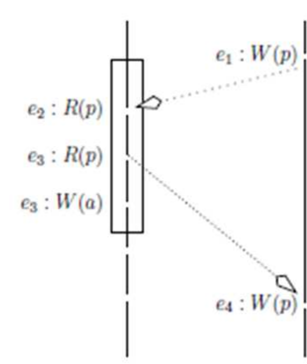
- Capture ordering constraints between statements common to set of interleavings
  - Prevent same concurrency scenarios from being retested
- Statement st´ ∊ HaPSet[st] if st <u>immediately</u> and <u>remotely</u> <u>dependent</u> on st´ in some interleaving
  - Similar to persistent sets except consider memory-access AND synchronization statements

Thread T_1	Thread T_2

	{ //alloc
	e_1 : $p := \&a$;
	}

{ //use
e_2 : if $(p \neq 0)$
e_3 : $*(p) := 10$
}	

	{ //free
	e_4 : $p := 0$;
	}

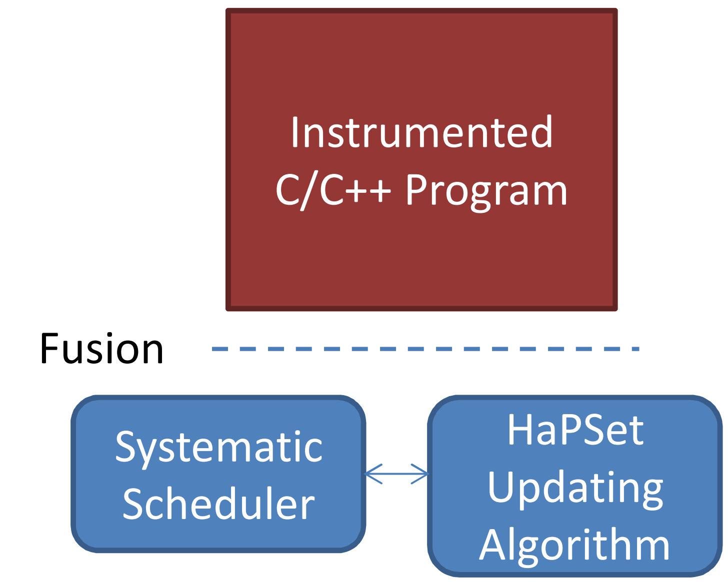


(file, line, thr, ctx)

HaPSet[e1]={}, HaPSet[e2]={e1},
HaPSet[e3]={}, HaPSet[e4]={e3}

Fusion: Method

- Only monitor shared memory access and synchronization statements
- Use HaPSet to capture ordering constraints from set of interleavings
  - Execute interleavings not covered by existing HaPSets

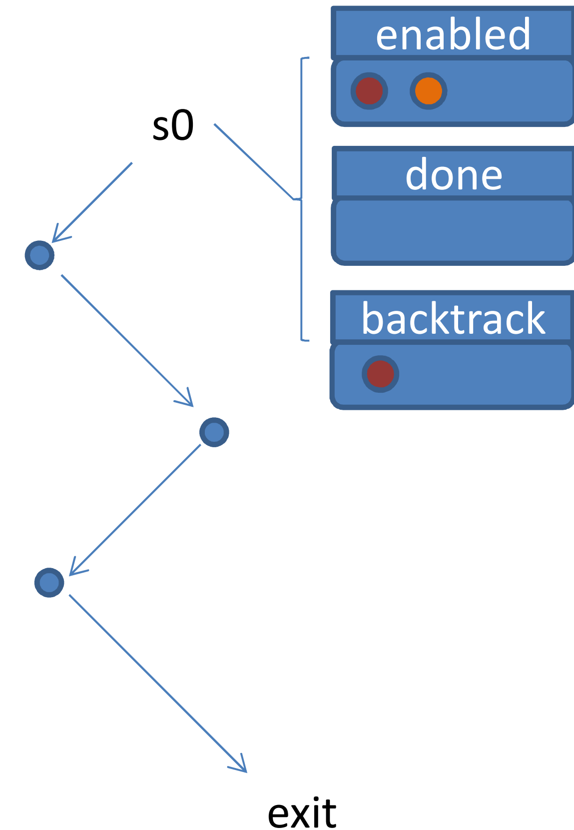


Fusion: HaPSet Learning

- Continuously update HaPSets from good runs
  - Execute instrumented program under control of scheduler
  - Maintain HaPSet[st] for each statement
  - At every execution step, for last executed statement of each thread, add last remote immediately dependent statement to HaPSet
- Perform an initial training period + continuously learn while testing

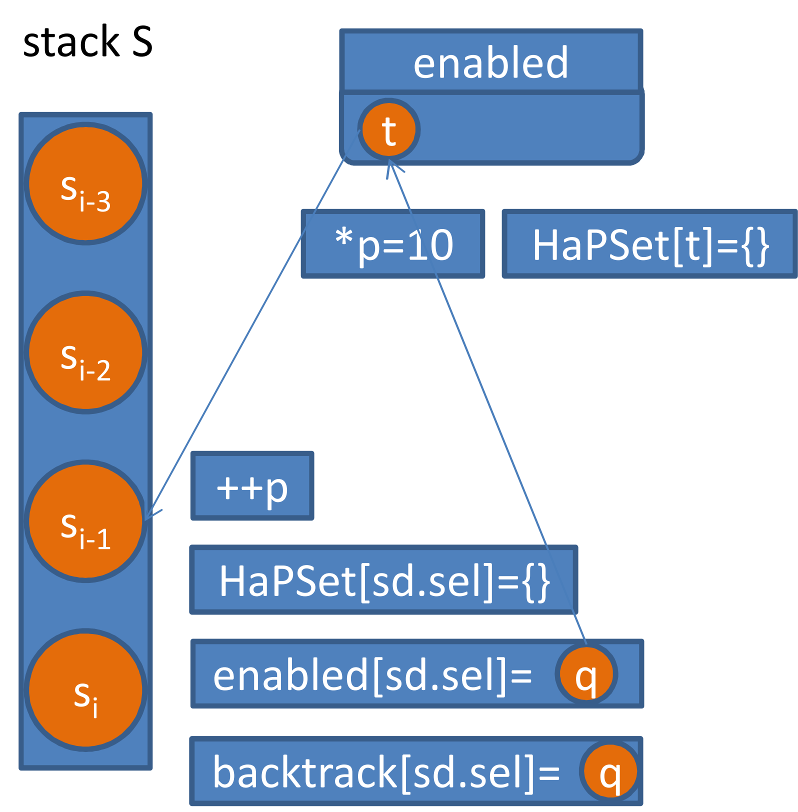
Fusion: Systematic Scheduler

- Start from initial statement s0
- Maintain 3 sets:
  1. Enabled: enabled statements
  2. Done: choices already made
  3. Backtrack: future scheduling options
- Recursively explore tree starting from s0 and selecting statements from backtrack set
  - Update backtrack set as we explore
  - Terminate when nothing left to explore



Scheduler: Guiding Interleaving Selection

- To update backtrack, for each enabled statement t, find last previously executed statement sd.sel such that:
  1. sd.sel is dependent on t
  2. sd.sel is co-enabled with t
  3. t ∉ HaPSet[sd.sel]
- Other methods of computing backtrack set yield many redundant interleavings, e.g. DPOR and PCB
  – Fusion scales better, but it is not safe (may miss errors)



Fusion: Evaluation

- Tested 3 C/C++ applications written for Linux/Pthreads
- Compared against DPOR and Chess

A

Test Program				HaPSet		DPOR		PCB0	
name	LoC	thrds	bug type	runs	time(s)	runs	time(s)	runs	time(s)
thrift-lib-w2-5t	18.5k	3	deadlk	14	27.8	23	18.6	512(no)	247.2
thrift-lib-w3-5t	18.5k	4	deadlk	18	27.5	733	TO	1301	TO
thrift-lib-w4-5t	18.5k	5	deadlk	22	33.7	665	TO	1111	TO
thrift-lib-w5-5t	18.5k	6	deadlk	25	38.1	572	TO	899	TO

B

2 threads		3 threads		4 threads		5 threads		6 threads	
runs	time	runs	time	runs	time	runs	time	runs	time
3	3.1	7	10.0	12	21.4	14	36.9	16	54.2
7 threads		**8 threads**		**9 threads**		**10 threads**			
runs	time	runs	time	runs	time	runs	time		
18	80.9	20	116.4	22	159.9	24	256.9		

C

Test Program		HaPSet		DPOR		PCB2	
name	bug	runs	time	runs	time	runs	time
MysqlLog	atom	5	0.3	22	1.2	12	0.7
NodeState	order	5	0.3	22	1.5	12	0.8
Loadscript	atom	5	1.0	79	6.0	27	2.1
SeekToItem	atom	3	0.2	127	9.8	22	1.6
UpdateTimer	atom	3	0.2	128	11.2	10	1.0
FileTransport	deadlk	5	0.2	22	1.2	12	0.9
CreateThread	order	5	0.2	22	1.1	12	0.6
ReadWriteProc	order	5	1.9	175	12.0	20	2.8
OpenInputStr	deadlk	5	0.3	1409	107.7	42	3.1
HttpConnect	order	5	3.3	37	28.5	16	12.0
TimerThread	deadlk	3	0.2	101	7.3	18	1.2

Summary/Conclusions

- Chess
  - Demonstrates applying model checking to large systems with little perturbation
  - Novel fair scheduler detects liveness violations
  - Deterministic replay without states
- Fusion
  - Coverage-guided systematic concurrency testing algorithm that uses HaPSets
  - Learns ordering constraints from good runs to guide selection of high-risk interleavings
  - Scales better than DPOR and Chess

Critique/Evaluation

- Chess
  - Not a single figure to explain their algorithm
  - Seems highly Windows-specific, use knowledge unique to that API
  - Could be difficult to implement complex primitives
  - Don’t quantify claimed increase in coverage
- Fusion
  - Livelocks?
  - Ensuring deterministic replay?
  - Their 10 minute time limit seems a bit restrictive
  - Did the other methods find other, non-planted bugs?