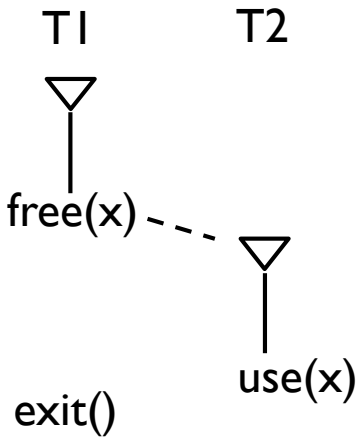
On interleaving space exploration of multi-threaded programs

Dongjie CHEN, Yanyan JIANG, Chang XU, Xiaoxing MA

Frontiers of Computer Science, DOI: 10.1007/s11704-020-9501-6

Problems & Ideas

- Interleaving space exploration (generating thread interleaving for testing/verification) of multi-threaded programs: A survey
- Ideas:
  - Concurrency bugs can manifest by small interleaving.

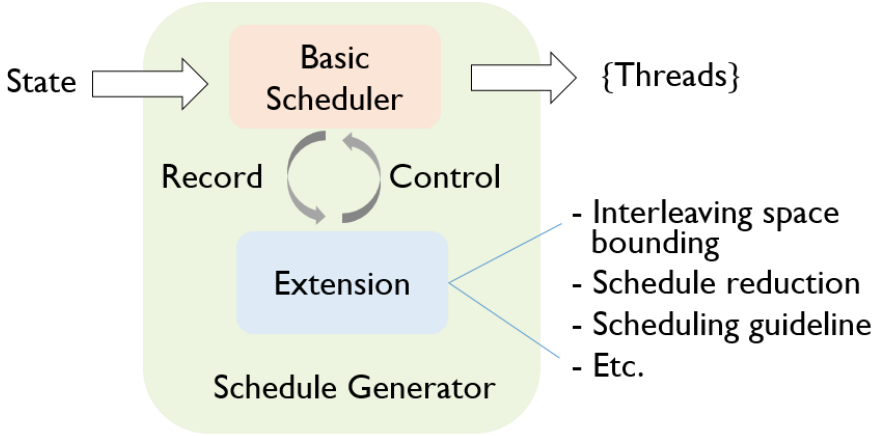


Thread beginning

Thread switch

A real bug from exiting work

  - Schedule Generator Φ = A Basic Scheduler Φ_{basic} (a simple scheduling policy) + An Extension $\Phi_{extension}$ (for bug manifestation)



$\tau = trace(s)$
$sched(\tau) = \langle t_1, t_2, t_2 \rangle$
$enable(s) = \{t_1, t_2, t_3\}$

$\langle t_1, t_2, t_2, t_1 \rangle$
$\langle t_1, t_2, t_2, t_2 \rangle$ ✖
$\langle t_1, t_2, t_2, t_3 \rangle$

$\Phi(s) = \{t_1, t_3\}$

Main Contributions

- We proposed the small interleaving hypothesis.

> SMALL INTERLEAVING HYPOTHESIS. *A high proportion of concurrency bugs can manifest by small thread interleaving.*

- We surveyed existing techniques and classified them into the framework.
  - #Basic schedule generator: 5
  - #Extension schedule generator: 25

- We proposed the framework of schedule generators.
  - $\Phi = \Phi_{basic} \times \Phi_{extension}$

Extension category	#Tech.
Interleaving Space Bounding	3
Schedule Reduction	4
Probability Promotion	4
Interleaving Guideline	8
Schedule Synthesis	4
Diverse Behaviors	2

Technique (Year)	Basic× Extension	Summary
Context/Preemption-bounded (2005)	$\frac{(\Phi_{enum}, \Phi_{rand}, \Phi_{prio}, \Phi_{scs}) \times}{\text{Bounding interleaving space}}$	It exercises schedules containing at most k contexts or preemptions [47, 53].
Delay-bounded (2011)	$\frac{(any) \times}{\text{Bounding interleaving space}}$	It transforms the traditional interleaving space into a binary-tree space, and bounds the number of invoking delay explorer [48].
Verisoft (1997)	$\frac{(any) \times}{\text{Schedule reduction}}$	It filters out threads that are independent with the `last`(s) when enumerating new states from state s [46].
DPOR (2005) Inspect (2008)	$\frac{(any) \times}{\text{Schedule reduction}}$	It inserts backtracking points where conflicted events can be executed forward to result in different schedules during explorations [63, 64].
Lapor (2019)	$\frac{(any) \times}{\text{Schedule reduction}}$	It regards lock acquisition events as unordered, identifies critical sections with conflicted read-write events, and just explores different schedules between conflict critical sections [65].

.....