

# COBET: 패턴 기반 동시성 버그 검출기 프레임워크

홍신 \* 김문주

Provable Software Laboratory  
CS Dept. KAIST

# Introduction

- OS kernel utilizes cutting-edge multi-threaded programming techniques

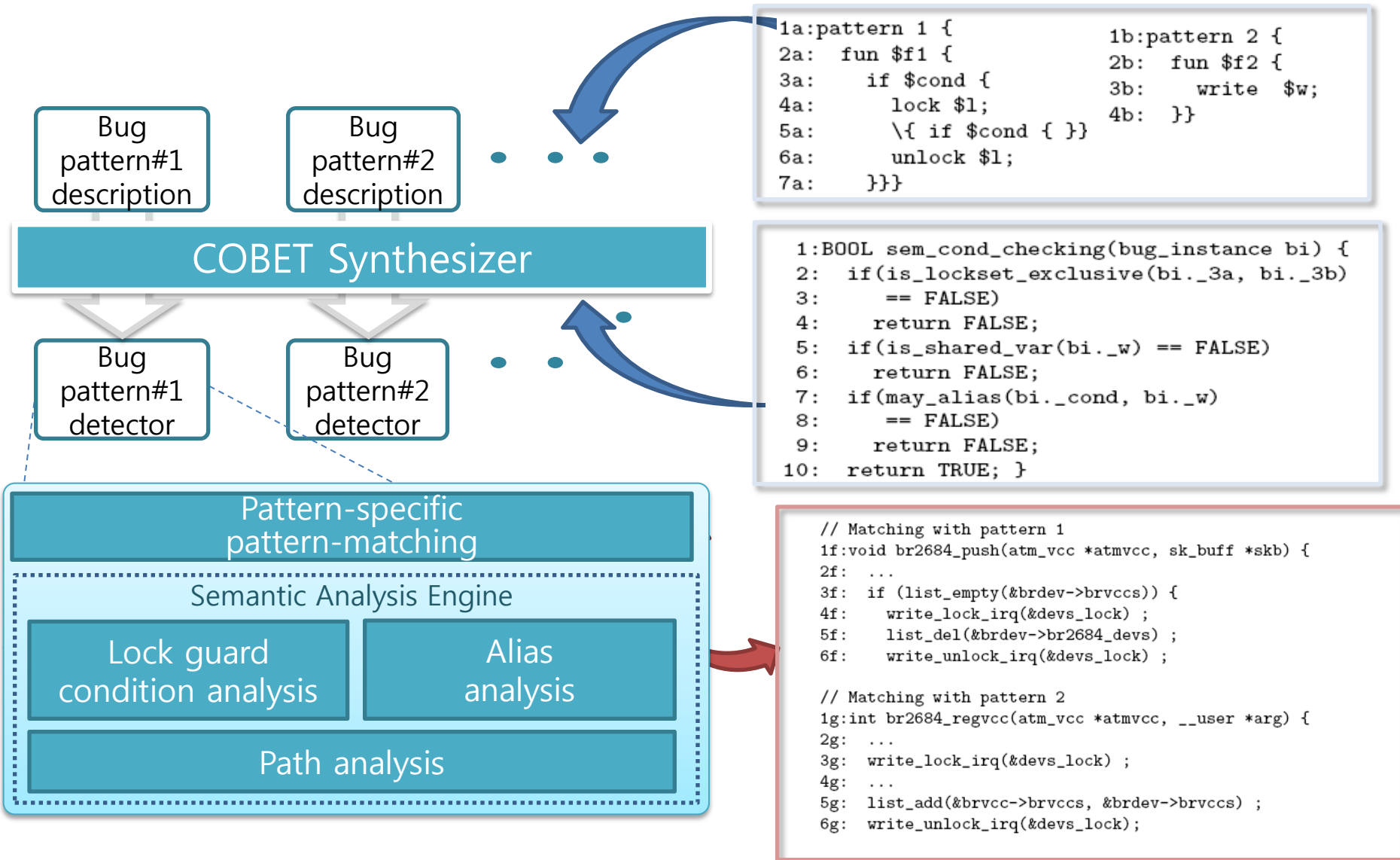
Ex. A function from Linux MTD/UBI device driver in ver. 2.6.27.22 (Simplified)

```
int ubi_thread(void * u) {
    for (;;) {
        if (kthread_should_stop())
            break ;
        spin_lock(&ubi->wl_lock) ;
        if (list_empty(&ubi->works) || ubi->ro_mode ||
            !ubi->thread_enabled) {
            spin_unlock(&ubi->wl_lock) ;
            schedule() ;
            continue ;
        }
        err = do_work(ubi) ;
        if (err) {
            if (failures++ > WL_MAX_FAILURES)
                break ;
        }
        cond_resched() ;
    }
}
```

	btrfs	ext4	nfs
LOC	41K	28K	29K
# of call seq.	2100M	1501M	3394M
Max/min call seq. length	88 / 60	54 / 53	57 / 39

➔ Pattern-driven bug detection reflecting concurrency characteristics

# COBET Framework



# Experiment Result

- Pattern-driven approach works?
  - Apply 5 pattern detectors to 7 Linux file systems
  - 42 warning and 4 confirmed.
- Semantic condition checking gives benefit ?
  - Reduce 44% false alarms consuming 2.6x more times.

	Syntactic (Single sub-pattern)		Syntactic (multiple-subpattern)		Syntactic + path analysis + lock analysis		Syntactic + path analysis + lock analysis + alias analysis	
	Detection	Time	Detection	Time	Detection	Time	Detection	Time
Total in five bug patterns	75	3.69 sec	52	6.10 sec	46	9.44 sec	42	9.56 sec

- For other domains ?
  - 5 pattern detectors from FS to Linux device drivers and Linux network modules
  - 11 warnings and 6 confirmed

# Discussions

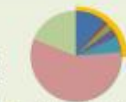
## COBET: Effective CONcurrency Bug dETector Framework for OS Kernel Hong,Shin (advised by Prof. Moonzoo Kim)

### Introduction

#### ✓ Motivation

As multi-core hardware becomes increasingly prevalent and program complexity grows, OSs such as Linux utilize the cutting-edge multithreaded techniques to enhance performance.

However, current analysis techniques do not provide enough to support OS developers in a practical manner due to the volume and complexity of the source programming. In particular, the following three obstacles hinder analysis of the concurrent behavior of OS:



Atomicity violation  
Atomicity Error  
Atomicity Error  
Atomicity Error  
Atomicity Error

Deadlock  
Starvation  
Lock Ordering

A recent analysis on the other hand, cannot analyze the OS code accurately, due to its complex code and size increase, the complexity structure of OS security-critical modules, and the intricate data synchronization mechanisms and complex runtime results.

OS developers sometimes implement their own synchronization primitive. Concurrency bug detection tools for standard synchronization mechanisms do not recognize these customized synchronization mechanisms and produce erroneous results.

• Limitations of lock-based bug detection  
Many existing bug detection mechanisms focus on lock-based data races through the analysis of binary lock content. However, OSs usually contain "spinlock/unspinlock" mechanisms for performance. In addition, spinlocks do not use any atomicity operations, and are more difficult to analyze than lock-based data races.

• Lack of scalability  
A significant analysis often fails to analyze the massive concurrency bugs due to the massive number of added parallelism threads.

### COBET Framework

✓ Overview  
Most concurrency errors are caused by unsynchronized interaction among concurrently executing multiple threads. Therefore, to detect a concurrency bug accurately, the concurrency bug pattern should be checked with multiple code patterns and/or atomic operations. These multiple code patterns are possible to be checked automatically at the source code or not.

For this purpose, COBET provides the pattern detection framework (PDL) to describe the specific features of multiple code patterns. And the COBET semantic analysis engine enables the feasibility test, which is a selected feature of COBET compared to other pattern-based bug detection techniques.

✓ Pattern Detector Construction  
The construction proceeds like the three steps:  
1. A user specifies the syntactic characteristics of a bug pattern in PDL.  
2. The COBET synthesizer generates the PDL description by the corresponding bug pattern detector template code. The template code performs the syntactic pattern-matching and calls the semantic analysis engine to check the semantic condition. In this step, semantic condition is specified with the semantic condition checking routine to complete the bug pattern detector.

✓ Approach  
We designed the Concurrency Bug dETector (COBET) framework that utilizes semantic analysis together with various tools.

A related contribution of COBET is that it utilizes semantic information to define and detect bug patterns in a more precise manner. In addition, COBET performs semantic analysis to check the semantic condition, which is a more precise manner than other tools. In addition, COBET provides a user interface to specify and analyze various bug patterns.

COBET log description of "Unbalanced Read and Write" bug pattern

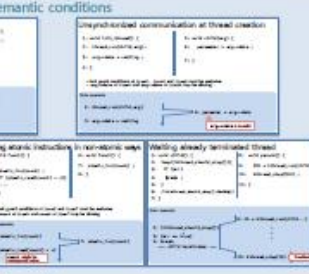
✓ Five bug patterns with semantic conditions

Unbalanced Read and Write  
Data Race  
Atomicity Violation  
Deadlock  
Starvation  
Lock Ordering

✓ Bug Detection Result on File Systems  
We applied the five COBET bug pattern detectors to the seven Linux file systems. The result shows the number of alarms was modest (3 bug patterns). In the systems, result of COBET is relatively more the systems itself have several bugs and COBET approach effectively identified these bugs.

✓ Evaluation of Semantic Analysis Techniques  
To demonstrate the effectiveness of COBET semantic analysis, we measured the false alarm reduction rate through the semantic analysis and the additional time cost.

✓ Bug Detection Results on Device Drivers and Network Modules  
To demonstrate applicability we applied 5 pattern detectors to Linux device drivers and network modules.



Further work  
✓ Patternize conventional data race and atomicity violations  
Many bug definitions used for standard tool-based concurrency bug detection techniques such as data race or atomicity violations can be represented or approximated by COBET patterns.  
✓ Apply to Application-level programs  
Large-scale application programs such as HTTP daemon or DBMS systems suffer similar difficulties to Linux kernel programming. We plan to apply COBET approach and COBET bug patterns to find bugs in these domain and compare the result to the case of Linux kernel.  
✓ Enhance PDL to include semantic conditions specification  
Currently, PDL only specifies syntactic aspects of bug patterns. We plan to extend PDL to associate essential semantic conditions to improve usability of COBET framework.

Answer to the reporting detected bug from the corresponding Linux maintainer  
... that occurs pretty frequently; i don't know how you would fix this one in practice given the evaluation from the user-space code for startup and shutdown, however, the user's report for the many months ...

### Experiments

#### ✓ Bug Detection Result on File Systems

File System	COBET	Linux	COBET	Linux	COBET	Linux	COBET	Linux	COBET	Linux
ext2	0	0	0	0	0	0	0	0	0	0
ext3	0	0	0	0	0	0	0	0	0	0
ext4	0	0	0	0	0	0	0	0	0	0
tmpfs	0	0	0	0	0	0	0	0	0	0
total	0	0	0	0	0	0	0	0	0	0

File System	COBET	Linux	COBET	Linux	COBET	Linux	COBET	Linux	COBET	Linux
ext2	0	0	0	0	0	0	0	0	0	0
ext3	0	0	0	0	0	0	0	0	0	0
ext4	0	0	0	0	0	0	0	0	0	0
tmpfs	0	0	0	0	0	0	0	0	0	0
total	0	0	0	0	0	0	0	0	0	0

File System	COBET	Linux	COBET	Linux	COBET	Linux	COBET	Linux	COBET	Linux
ext2	0	0	0	0	0	0	0	0	0	0
ext3	0	0	0	0	0	0	0	0	0	0
ext4	0	0	0	0	0	0	0	0	0	0
tmpfs	0	0	0	0	0	0	0	0	0	0
total	0	0	0	0	0	0	0	0	0	0

See you at poster session!

- And I am also interested in ...
- General concurrency bug detection techniques and their classifications
- Finding good test input for effective concurrent program testing
- Practices of finding bugs in Linux kernel programs