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

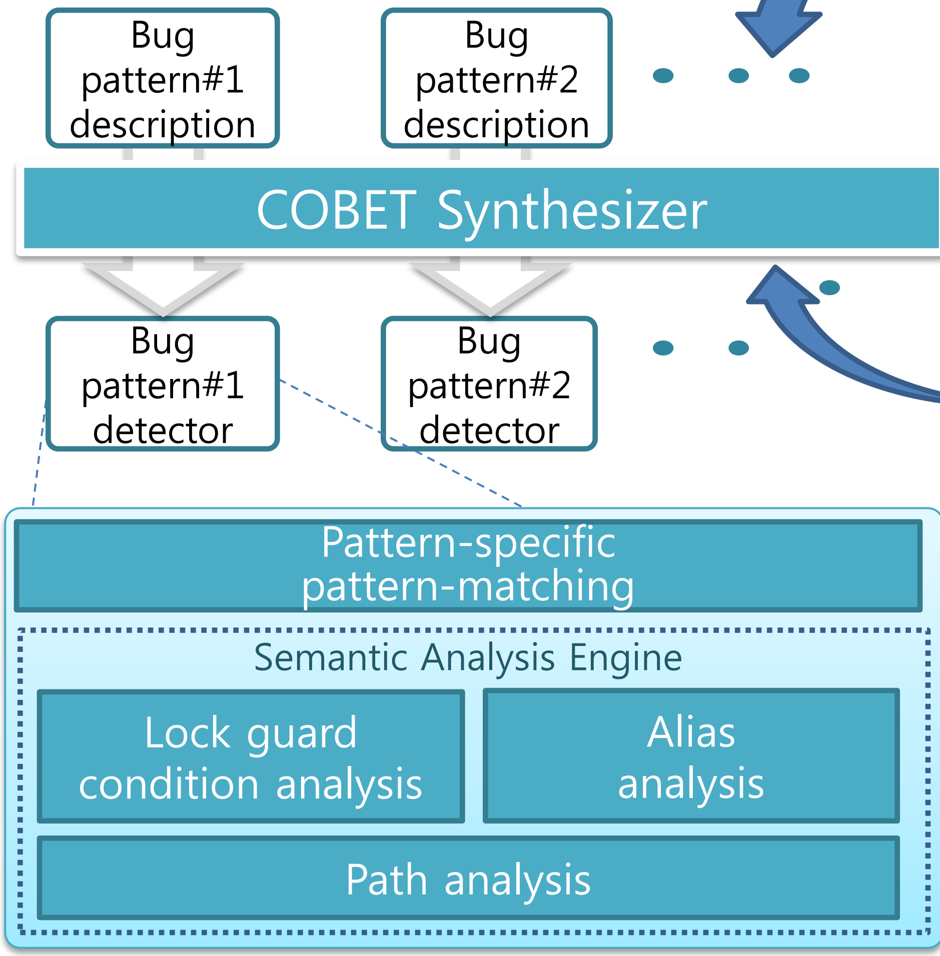
Ex. A function from Linux MTD/UBI device driver in ver. 2.6.27.22 (Simplified)

```
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




```
1a:pattern 1 {
2a:   fun $f1 {
3a:     if $cond {
4a:       lock $l;
5a:       \{ if $cond { }}
6a:       unlock $l;
7a:     }}}
```

```
1b:pattern 2 {
2b:   fun $f2 {
3b:     write  $w;
4b:   }}
```

```
 1:BOOL sem_cond_checking(bug_instance bi) {
 2:  if(is_lockset_exclusive(bi._3a, bi._3b)
 3:     == FALSE)
 4:    return FALSE;
 5:  if(is_shared_var(bi._w) == FALSE)
 6:    return FALSE;
 7:  if(may_alias(bi._cond, bi._w)
 8:     == FALSE)
 9:    return FALSE;
10:  return TRUE; }
```

```
// Matching with pattern 1
1f:void br2684_push(atm_vcc *atmvcc, sk_buff *skb) {
2f:  ...
3f:  if (list_empty(&brdev->brvccs)) {
4f:    write_lock_irq(&devs_lock) ;
5f:    list_del(&brdev->br2684_devs) ;
6f:    write_unlock_irq(&devs_lock) ;

// Matching with pattern 2
1g:int br2684_regvcc(atm_vcc *atmvcc, __user *arg) {
2g:  ...
3g:  write_lock_irq(&devs_lock) ;
4g:  ...
5g:  list_add(&brvcc->brvccs, &brdev->brvccs) ;
6g:  write_unlock_irq(&devs_lock);
```

Experiment Result

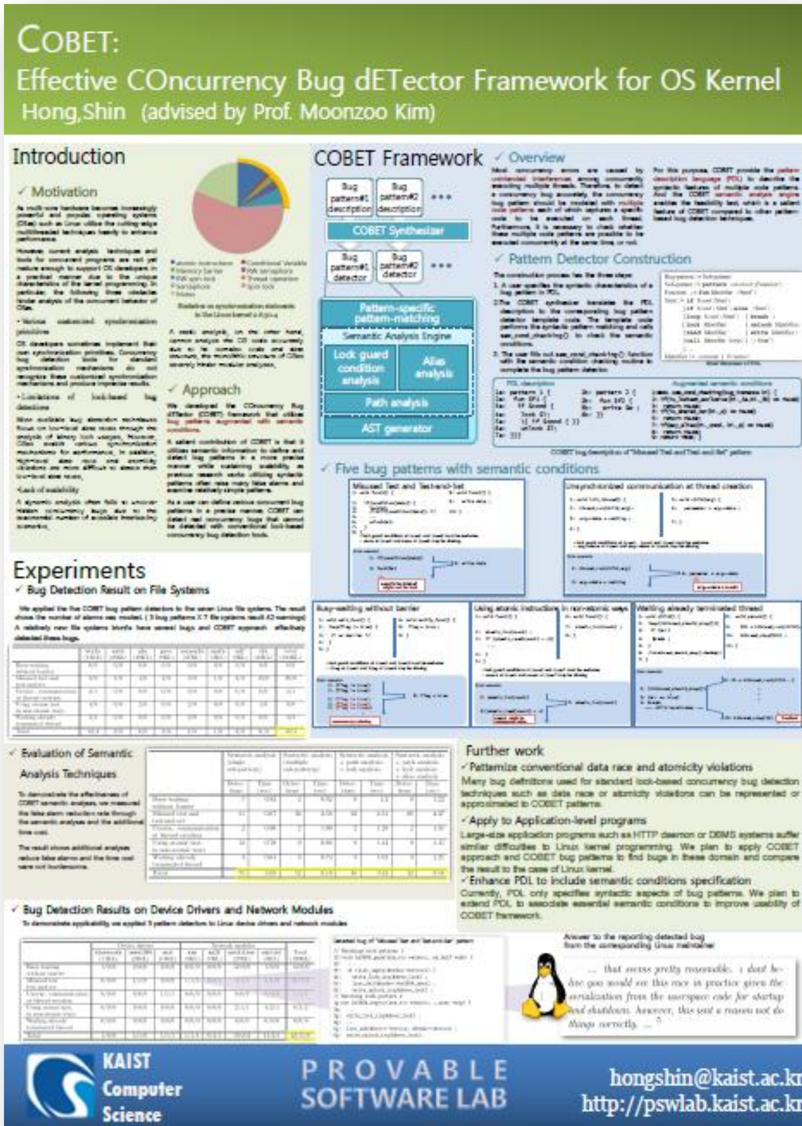
- Pattern-driven approach works?
  - Apply 5 pattern detectors to 7 Linux file systems
  - 42 warning and 4 confirmed.

- 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



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