Comparative Study on Software Model Checkers as Unit Testing Tools based on an Industrial Case Study

Moonzoo Kim, Yunho Kim
Provable Software Lab.
CS Dept. KAIST, South Korea

Hotae Kim
Samsung Electronics
Suwon, South Korea

Motivation

- **Software model checkers (SMC) can be used as great debugging tools rather than verification tools**
  - Slam, Blast: CEGAR based SMC
  - CBMC: Bounded analysis based SMC
- **Depending on the underlying theories of SMC, SMCs have different characteristics**
  - These differences matter greatly when applied to real application!!!
- **This talk compares those two different SMC techniques empirically based on a real-world case study**
  - Targeting on Unified Storage Platform for OneNAND flash memory

Overview

PART I: Background on CEGAR-based and SAT/SMT-based (CBMC) Software Model Checking

- **CEGAR approach**
- **SAT-based model checking**

PART II: Verification of Synchronization Behavior

- **Synchronization between Urgent Read and Generic Operations**
- **BML Semaphore**

PART III: Verification of Multi-sector Read

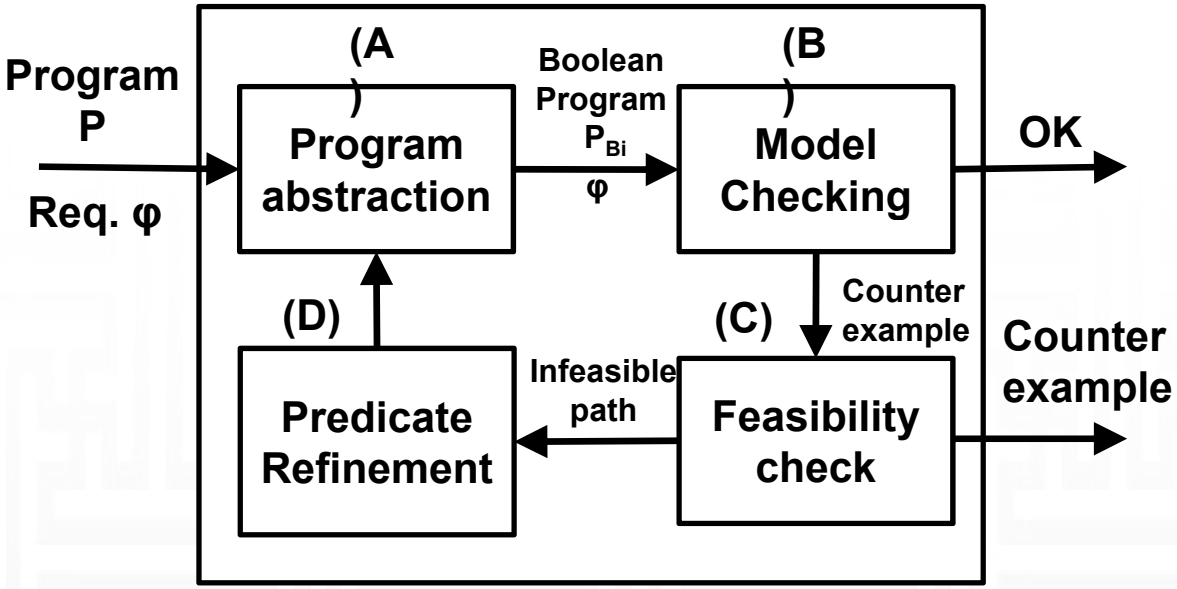
- **Overview of SM_ReadSectors()**
  - Modeling strategy
- **Multi-sector Read**
  - Verification by Blast : failed
  - Verification by CBMC-SAT

PART IV: Lessons Learned and Future Work

Model Checking History

1981	Clarke / Emerson: CTL Model Checking Sifakis / Quielle	10^5
1982	EMC: Explicit Model Checker Clarke, Emerson, Sistla	
1990	Symbolic Model Checking Burch, Clarke, Dill, McMillan	10^{100}
1992	SMV: Symbolic Model Verifier McMillan	
1998	Bounded Model Checking using SAT Biere, Clarke, Zhu	10^{1000}
2000	Counterexample-guided Abstraction Refinement Clarke, Grumberg, Jha, Lu, Veith	

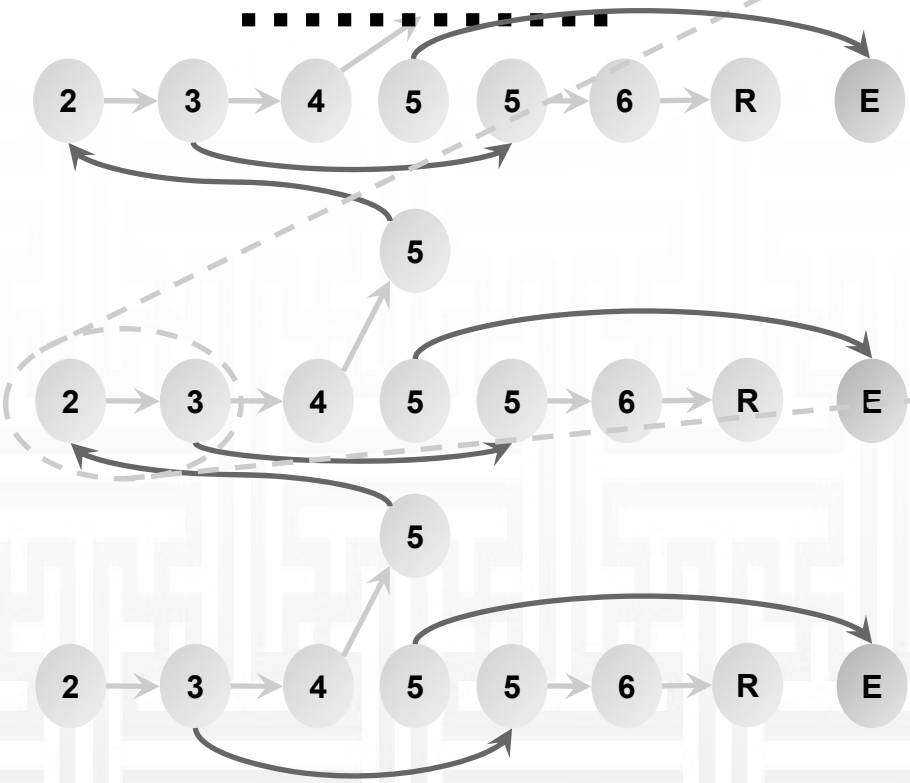
CounterExample Guided Abstract Refinement (CEGAR)

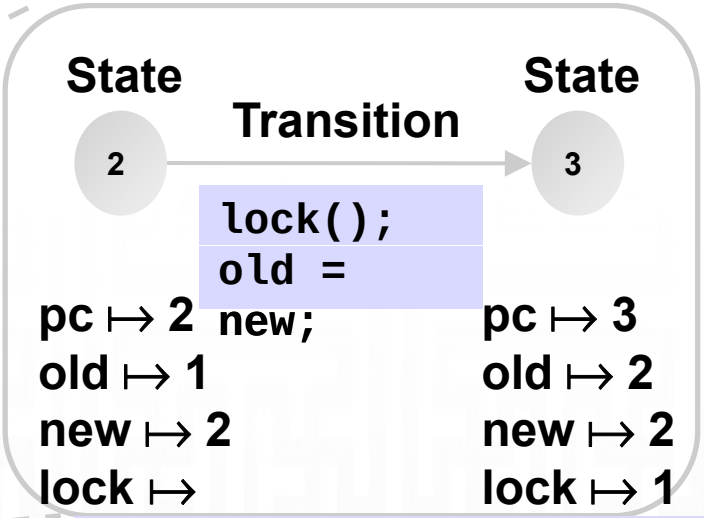


- Blast uses
  - Simplify as an internal decision procedure for computing next abstract reachable states
    - Handles linear arithmetics and uninterpreted functions only
  - FOCI anad CSIsat to calculate interpolants from infeasible paths
- Blast handles complex C operations that cannot be handled by these decision procedures as uninterpreted functions
  - Source of false alarms

Behavior of program

- Behavior of program can be modeled as a state transition graph

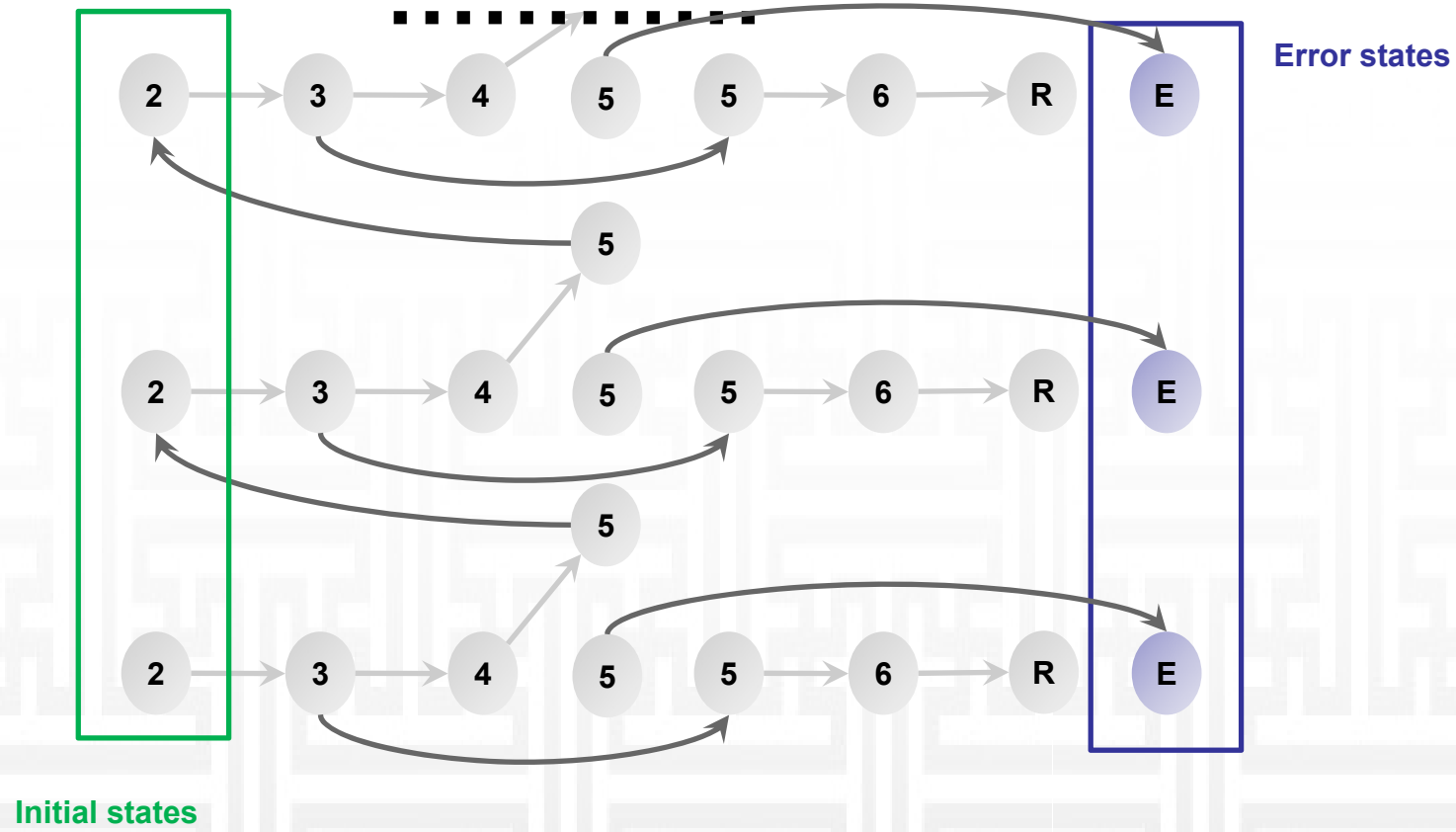





```
0  1:  Example() {
   2:  do {
         lock();
         old = new;
   3:    if (*) {
   4:      unlock();
           new++;
         }
   5:  } while (new != old);
   6:  unlock();
       return;
   }
```

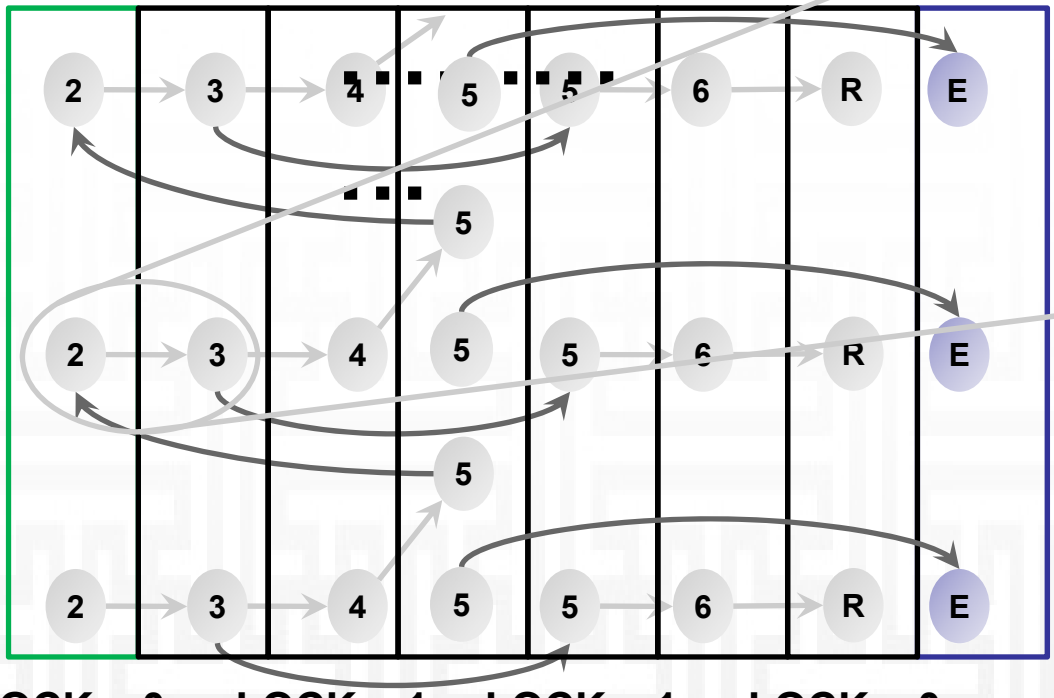
The safety verification

- Is there a **path** from an **initial** to an **error** state ?



Abstract behavior of program

- **Equivalent states satisfy same predicates and have same control location**
  - They are merged into one abstract state



State

Transition

2 → 3

```
lock();
old = new;
```

pc ↦ 2
old ↦ 1
new ↦ 2
lock ↦ 0

pc ↦ 3
old ↦ 2
new ↦ 2
lock ↦ 1

Abs. state

Abs. transition

pc ↦ 2
$p1$
$\neg p2$

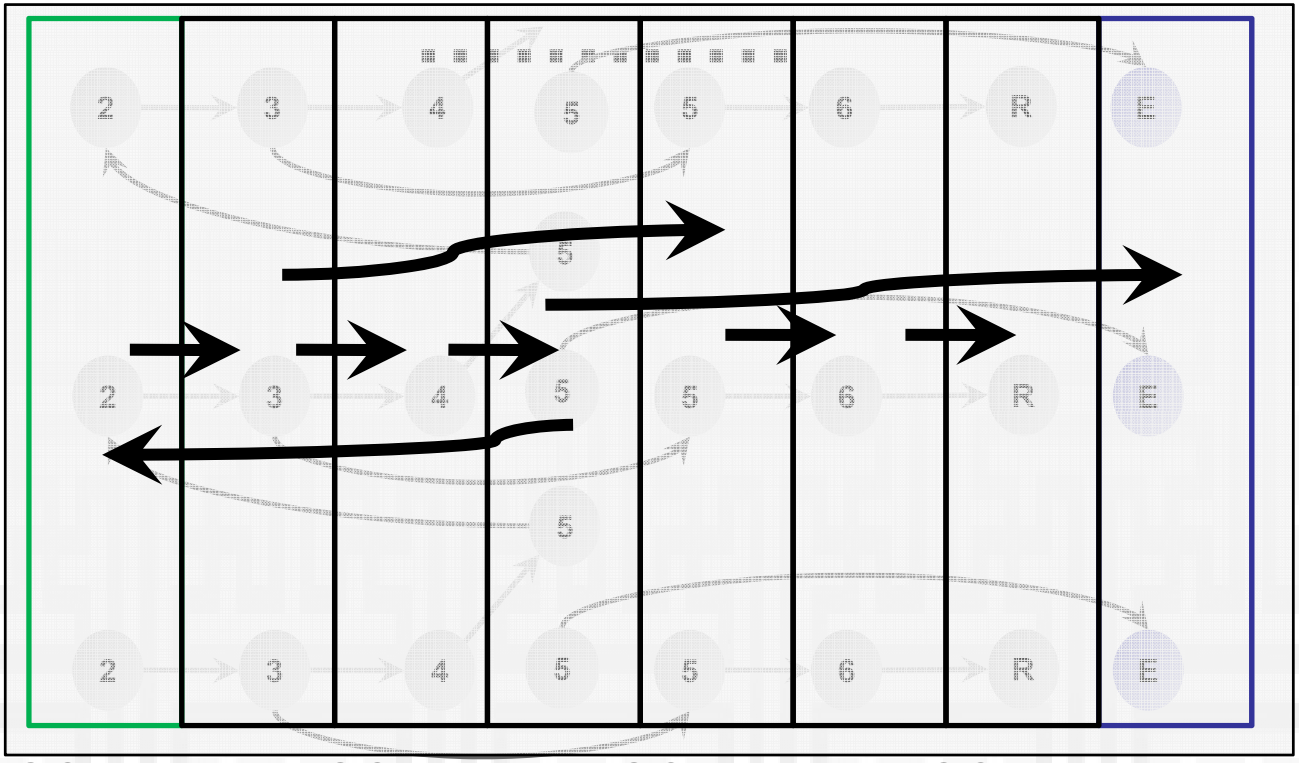
```
lock();
old =
new;
```

pc ↦ 3
$\neg p1$
$p2$

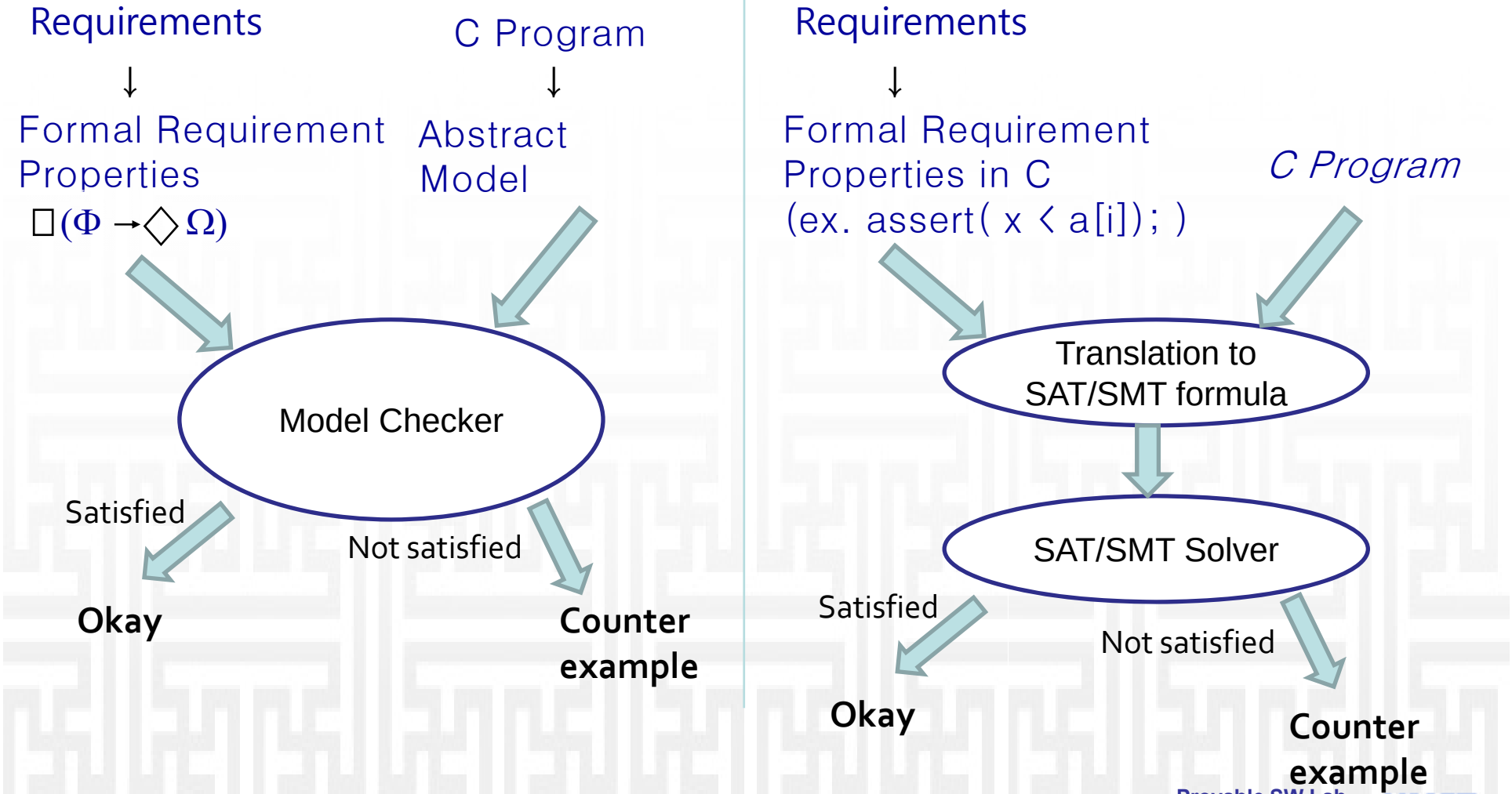
$p1 \triangleq$ (lock = 0)
$p2 \triangleq$ (lock = 1)

Over-approximation

- If there exists a transition between s_1 and s_2, then also there exists a transition between abstract state of s_1 and s_2



Overview of Bounded Model Checking



Model Checking as a SAT/SMT problem (1/4)

- **CBMC (C Bounded Model Checker, In CMU)**
  - Handles function calls using inlining
  - Unwinds the loops a fixed number of times (bounded MC)
    - Thus, a user has to know a upper bound of each loop
      - In practice, it does not cause serious limitation since
        - » Most loops has clear upper bounds
        - » Even when we do not know the upper bound, we can still get analysis result with possibility of false alarms
  - Allows user input to be modeled using non-determinism
    - So that a program can be checked for a set of inputs rather than a single input
  - Allows specification of assertions which are checked using the bounded model checking

Model Checking as a SAT/SMT problem (2/4)

Original code

```
x=0;
while (x < 2) {
  y=y+x;
  x++;
}
```

Unwinding the loop 3 times

```
x=0;
if (x < 2) {
  y=y+x;
  x++;}
if (x < 2) {
  y=y+x;
  x++;}
if (x < 2) {
  y=y+x;
  x++;}
assert (! (x < 2))
```

Original code

```
x=x+y;
if (x!=1)
  x=2;
else
  x++;
assert(x<=3);
```

Convert to SSA

```
x1=x0+y0;
if (x1!=1)
  x2=2;
else
  x3=x1+1;
x4=(x1!=1)?x2:x3;
assert(x4<=3);
```

Generate constraints

$C \equiv x_1=x_0+y_0 \wedge x_2=2 \wedge x_3=x_1+1 \wedge (x_1!=1 \wedge x_4=x_2 \vee x_1=1 \wedge x_4=x_3)$

$P \equiv x_4 <= 3$

Check if $C \wedge \neg P$ is satisfiable,
if it is then the assertion is violated

$C \wedge \neg P$ is converted to Boolean logic using a bit vector representation for the integer variables
$y_0, x_0, x_1, x_2, x_3, x_4$

Model Checking as a SAT/SMT problem (4/4)

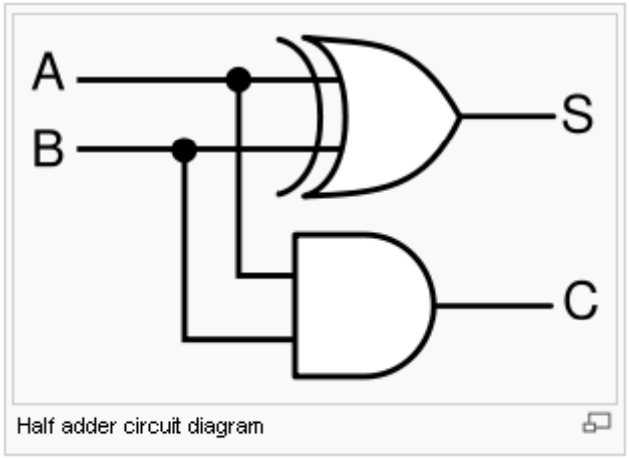
- Example of arithmetic encoding into pure propositional formula

  Assume that x,y,z are three bits positive integers represented by propositions $x_0x_1x_2$, $y_0y_1y_2$, $z_0z_1z_2$

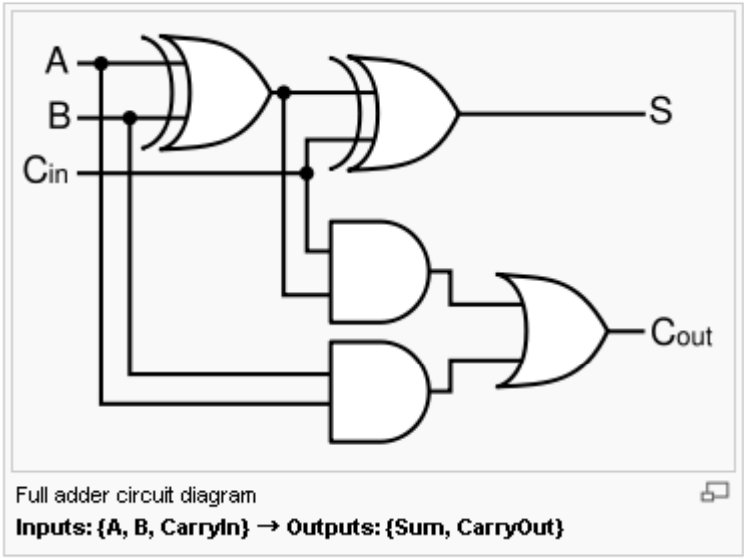
  $C \equiv z=x+y \equiv \; (z_0 \leftrightarrow (x_0 \oplus y_0) \oplus ((x_1 \wedge y_1) \vee (((x_1 \oplus y_1) \wedge (x_2 \wedge y_2)))$

  $\wedge \; (z_1 \leftrightarrow (x_1 \oplus y_1) \oplus (x_2 \wedge y_2))$

  $\wedge \; (z_2 \leftrightarrow (x_2 \oplus y_2))$



Half adder circuit diagram



Full adder circuit diagram

Inputs: {A, B, CarryIn} → Outputs: {Sum, CarryOut}

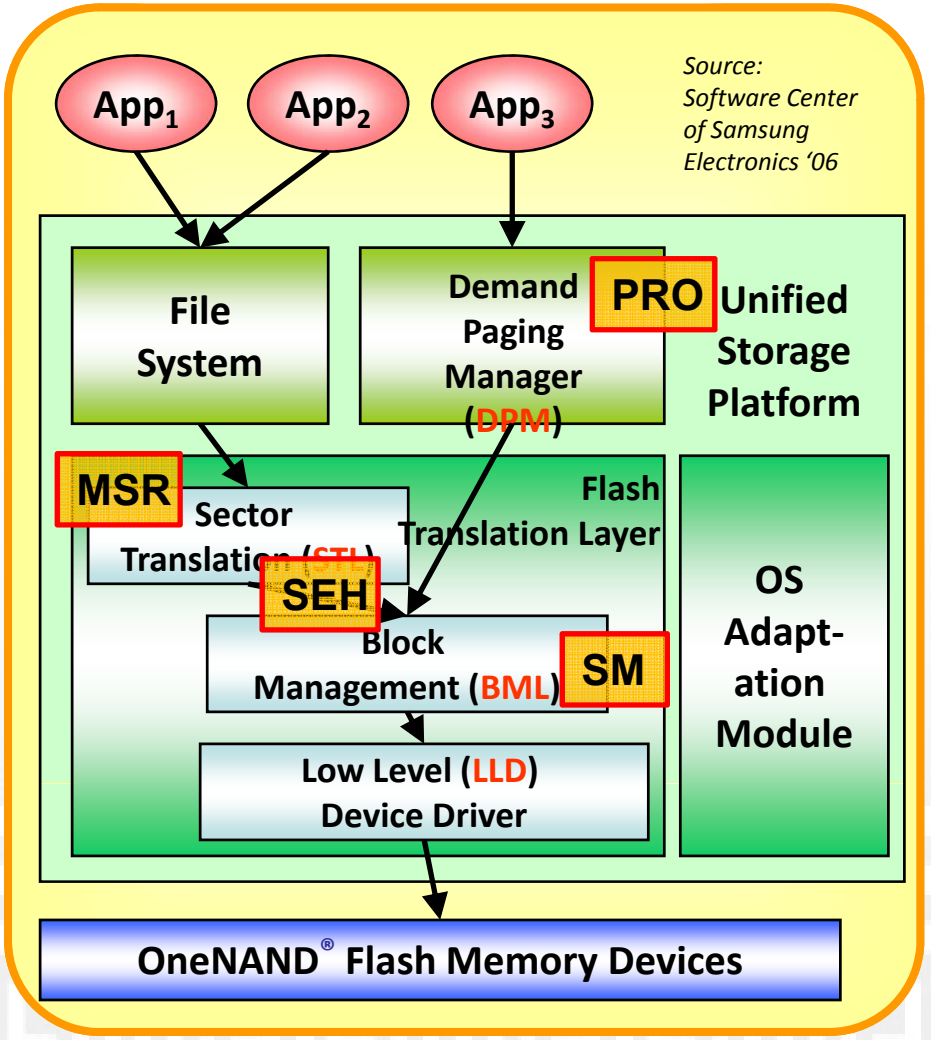
*** Note that a translated program in BitVector Theory is almost equivalent to the SAT formula**

PART II: Verification of Synchronization Behavior

- Synchronization between urgent read operation and generic operations
- Operations on the BML semaphores

Overview of the Unified Storage Platform

- **Characteristics of OneNAND® flash**
  - Each memory cell can be written limited number of times only
    - Logical-to-physical sector mapping
    - Bad block management
    - Wear-leveling
  - XIP by emulating NOR interface through demand-paging scheme
    - Multiple processes access the device concurrently
    - Urgent read operation should have a higher priority
    - Synchronization among processes is crucial
  - Performance enhancement
    - Multi-sector read/write
    - Asynchronous operations
    - Deferred operation result check



Synchronization between Urgent Read and Generic Operations

- Blast found a violation of assert(pstInfo->bNeedToSave||saved)

```
01:...
02:Location: id=5#16 src="LLD.c"; line=406
03: Block(*(pstReg@PriRead ).nInt =0;
04: *(pstReg@PriRead ).nCmd = 176;
05: bEraseCmd@PriRead = 1;)
06:Location: id=5#18 src="LLD.c"; line=410
07: Pred(bEraseCmd@PriRead? != 0)
08:...
09:Location: id=5#46 src="LLD.c"; line=494
10: FunctionCall(__assert_fail(
11: "!(pstInfo->bNeedToSave==1)||saved" ... ))
12:Error found! The system is unsafe :-(
```

- -foci –cref options
- 0.18 s w/ 6.6 MB

- CBMC found a violation of assert(pstInfo->bNeedToSave||saved)

```
01:...
02:State 14 file LLD.c line 408 function PriRead
03: LLD::PriRead::1::bEraseCmd=1
04:State 15 file LLD.c line 412 function PriRead
05: LLD::PriRead::1::1::2::nWaitingTimeOut=...
06:...
07:Violated property:
08: file LLD.c line 494 function PriRead
09: assertion !(_Bool)pstInfo->bNeedToSave ...
10:VERIFICATION FAILED
```

- --slice –no-bounds-check –no-pointer-check –no-div-by-zero-check
- 8 s with 325 MB: MiniSAT
- 0.1 s with 17 MB: Boolector
- Timeout (more than 2 hours) : Yices
- Timeout : Z3

- **PriRead()**
  - **234 lines long, two simple loops whose upper bound are constants, no sub-procedure calls**

- **Experimental setup**
  - 3 Ghz Xeon w/ 32 gigabyte mem
  - Fedora Linux 7, Blast 2.5, CBMC 3.3.2, MiniSAT 1.1.4, Boolector 1.012, Z3 2.0.1, Yices 1.0.19

Operations on the BML Semaphores

- **We analyzed all 14 BML functions that use BML semaphores using Blast and CBMC**
  - Blast and CBMC directly analyze a C file into which a user inserts an assert statement to check a requirement property as Blast does

```
int sm=1;
void BML_Close() { …
  OAM_AcquireSM(pstVol->nSm);
  {sm--; assert(sm==0);}
  …
  OAM_ReleaseSM(pstVol->nSm);
 {sm++; assert(sm==1);}
  …
  assert(sm == 1);
  return;
…}
```

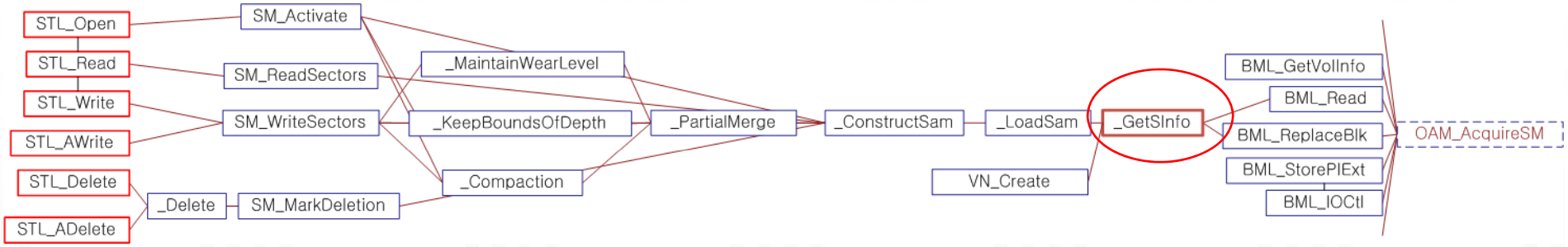
Operations on the BML Semaphores (cont.)

- **No loop inside (4 function):**
  - _close, BML_GetVolInfo, BML_ReplaceBlk, BML_StorePIExt
- **Loops w/ bound <10**
  - BML_EraseBlk, BML_FlushOp, BML_IOCtl, BML_Read, BML_Write
- **Loops w/ no upper bounds (set as 10)**
  - BML_Copy, BML_CopyBack, BML_MEraseBlk, BML_Mread, BML_MWrite

Function	LOC	Blast		CBMC	
		Time (sec)	Mem (MB)	Time (sec)	Mem (MB)
_Close	101	1.8	20	3.9	56
BML_Copy	216	1.3	19	23.2	190
BML_CopyBack	248	1.7	19	25.2	154
BML_EraseBlk	127	1.8	19	19.8	107
BML_FlushOp	82	1.6	19	3.5	56
BML_GetVolInfo	57	1.1	19	3.6	57
BML_IOCtl	174	2.4	20	3.6	56
BML_MEraseBlk	202	1.3	19	88.8	472
BML_MRead	167	1.2	19	6.3	71
BML_MWrite	178	1.2	19	17.4	109
BML_Read	236	1.9	19	6.2	70
BML_ReplaceBlk	66	1.3	19	3.3	56
BML_StorePIExt	49	1.1	19	3.5	56
BML_Write	195	2.0	19	20.8	114
Average	150	1.6	19	16.4	116

**** CBMC w/ MiniSAT only***

Incomplete Exception Handling (1/2)



- We have checked whether STL_[*] always return **`STL_CRITICAL_ERROR`** whenever OAM_AcquireSM raises semaphore exception (**`BML_RELEASE_SM_ERROR`**)

Incomplete Exception Handling (2/2)

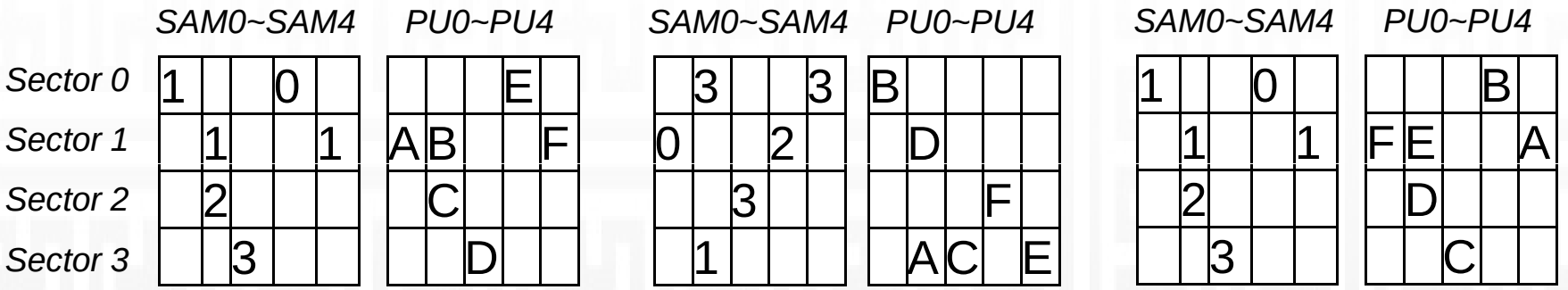
- Initially, Blast analysis raised false alarms
  - Return flags defined with bit-wise operators should be properly modified
  - FOCI interpolant solver could not handle full linear arithmetic formula, but difference logic formula only
- After fixing these problems, Blast detected the bug

Call graph root	LOC (call graph)	Blast Time (sec)	Blast Mem (MB)	CBMC Time (sec)	CBMC Mem (MB)
STL_Read	1654	24.4	19	31.2	1762
STL_Write	2646	T/O	T/O	1404.9	3718
STL_Awrite	2611	T/O	T/O	1014.9	3244
STL_Delete	2489	66.5	23	565.9	2139
STL_Adelete	2497	64.5	22	808.0	2196
STL_Open	2785	T/O	T/O	324.1	5781
Average	2447	51.8	21.5	708.2	3140

- Blast spent more than 1 hr to analyze pointer alias for STL_Write, STL_Awrite, and STL_Open
- CBMC with upper bound 2 could detect the bug

Modeling of SM_ReadSectors()

- **Modeling strategy**
  - Pointers are replaced with array indexes because Spin and NuSMV does not have explicit pointer data type.
  - Model all control structures, but minimized data structures
    - There exist 5~10 VUNs and 2 LUNs containing 5~8 sectors
    - 1 unit contains 4 sectors each of which is 1 byte long
  - All possible sector distributions are explored exhaustively



(a) A distribution of "ABCDEF"

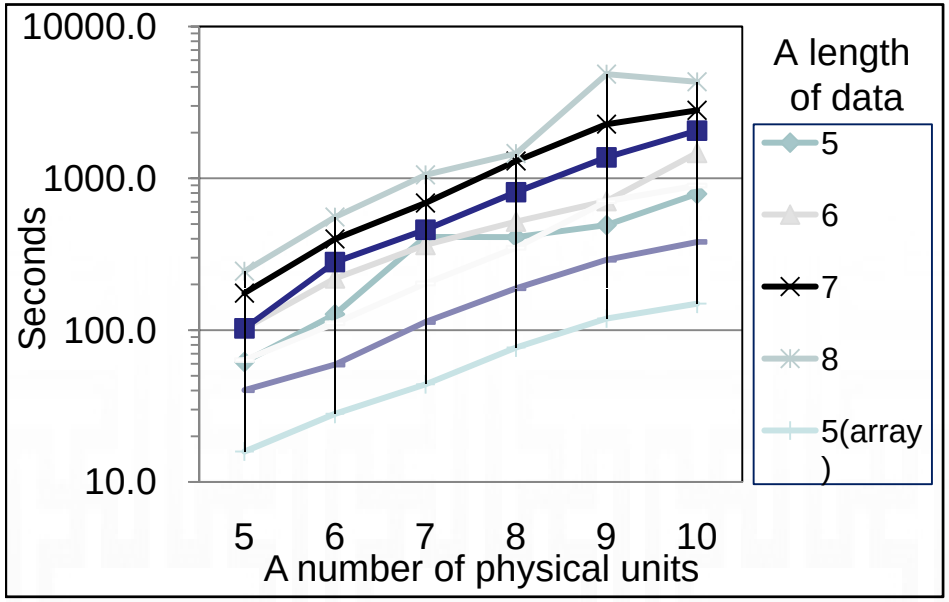
(b) Another distribution of "ABCDEF"

(c) A distribution of "FEDCBA"

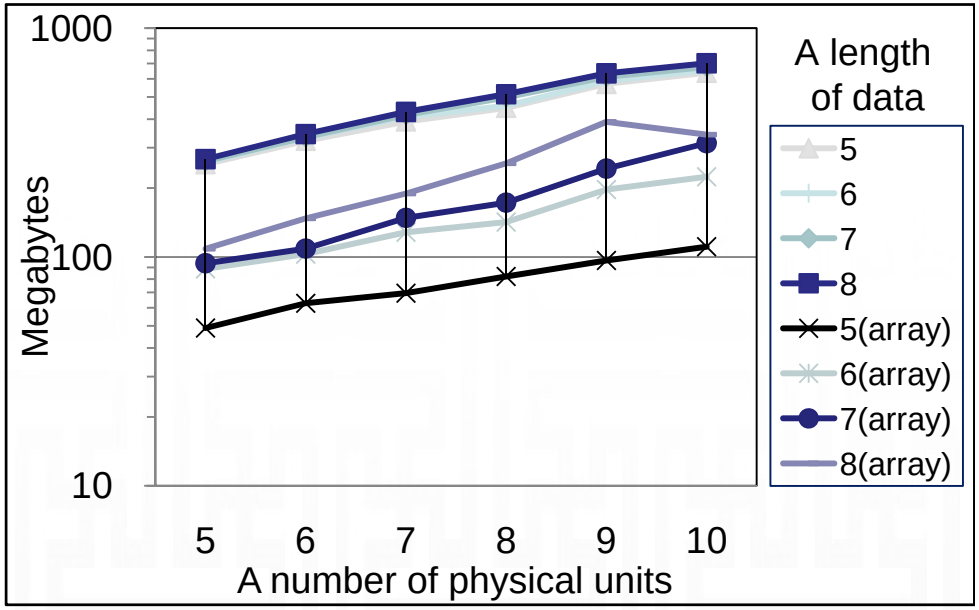
- **Requirement property**

```
buf[0]==logical_sectors[0] && buf[1]==logical_sectors[1] &&
buf[2]==logical_sectors[2] &&  buf[3]==logical_sectors[3] &&
buf[4]==logical_sectors[4] &&  buf[5]==logical_sectors[5])
```

Verification Results of CBMC-SAT



(a) Time consumption



(b) Memory consumption

$\times 10^6$	5		6		7		8		9		10	
	var	clause	var	clause	var	clause	var	clause	var	clause	var	clause
5	1.3	4.2	1.6	5.3	1.9	6.6	2.3	7.9	2.7	9.2	3.0	11
6	1.3	4.3	1.6	5.5	2.0	6.7	2.3	8.0	2.7	9.4	3.1	11
7	1.3	4.4	1.7	5.6	2.0	6.9	2.4	8.3	2.8	9.7	3.2	11
8	1.4	4.5	1.7	5.8	2.1	7.1	2.5	8.4	2.9	9.9	3.3	11

Table 2. The sizes of the SAT CNF instances of MSR with different configurations [b.

KAIST

CBMC-SAT v.s. CBMC-SMT

- **Blast could not analyze MSR due to its limitation on array handling (see lessons learned)**
- **Hypothesis H1**
  - BitVector SMT-solver is faster than SAT-solver
  - Rationale
    - BitVector utilizes high-level structural information before bit-blasting
- **Hypothesis H2**
  - AUFLIA logic can be solved faster than AUFBV logic
  - Rationale
    - Simplex algorithm of SMT solver for linear arithmetic is much faster than bit manipulation of SAT solver

Lessons Learned

- **Effectiveness of Software Model Checkers for Embedded Software**
  - High productivity and reliability compared to manual testing
  - Less binary library
  - Difficulty of physical testing
  - Reasonable performance
- **Limitation of Blast**
  - False positive
    - Complex operation as uninterpreted function
    - Flow-insensitive may-analysis of pointers
    - Imprecise array/struct access
  - False negative
    - Ignore integer overflow/underflow
    - Assume that all pointer operations are safe
    - Assume that only variables of the same type can be aliased

Future Work

- **Theorem Proving for MSR**
  - A challenging problem for theorem proving
  - Collaboration with Jim Woodcock (Univ. of York, UK)
  - NuSMV specification of MSR can be reused as invariants as starting point

- **Concolic testing**
  - More industry friendly cosmetics ☺
    - Test cases are essential assets for industry developers
  - Collaboration with Samsung Advanced Institute of Technology and DMC @ Samsung Electronics