Automated Analysis of Industrial Embedded Software

Moonzoo Kim
Provable Software Lab
KAIST, South Korea

Thanks to Hotae Kim and Yoonkyu Jang
Samsung Electronics, South Korea
Strong IT Industry in South Korea

Automated Analysis of Industrial Embedded Software
## Embedded Software in Two Different Classes

<table>
<thead>
<tr>
<th></th>
<th>Consumer Electronics</th>
<th>Safety Critical Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Examples</strong></td>
<td>Smartphones, flash memory platforms</td>
<td>Nuclear reactors, avionics, cars</td>
</tr>
<tr>
<td><strong>Market competition</strong></td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Life cycle</strong></td>
<td>Short</td>
<td>Long</td>
</tr>
<tr>
<td><strong>Development time</strong></td>
<td>Short</td>
<td>Long</td>
</tr>
<tr>
<td><strong>Model-based development</strong></td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Important value</strong></td>
<td>Time-to-market</td>
<td>Safety</td>
</tr>
</tbody>
</table>

Automated Analysis of Industrial Embedded Software
Personal Research Roadmap

Past: RV (dynamic) && MC (static)
- Runtime Verification: FMSD’04
- Model Checking: Spin’08
- Software Model Checking: ASE ‘08 TSE’11

Current: Extended Concolic Testing
- Concolic Testing: ICST’12a, FACJ’12, FSE’11a
- Distributed Concolic Testing: ICST’12b, FSE’11b, ICTAC’10
- Hybrid Algorithm (i.e., w/ Genetic Alg): ISSRE ’11, FSE’10

Future: Concolic Testing with Intelligence
- Statistic Inference
- Machine Learning
- User Assistance

Better Industrial Application

Moonzoo Kim
Automated Analysis of Industrial Embedded Software
Part I: Experience from SW Model Checking

Target system: Samsung Unified Storage Platform (USP) for OneNAND® flash memory (around 30K lines of C code)

- Characteristics of OneNAND® flash memory
  - Each memory cell can be written a limited number of times only
    - Logical-to-physical sector mapping
    - Bad block management, wear-leveling, etc
  - Concurrent I/O operations
    - Synchronization among processes is crucial
  - XIP by emulating NOR interface through demand-paging scheme
    - Binary execution has a highest priority
  - Performance enhancement
    - Multi-sector read/write
    - Asynchronous operations
    - Deferred operation result check
Results of Unit Analysis through CBMC and BLAST [TSE’11]

- Demand paging manager (234 LOC)
  - Detected a bug of not saving the status of suspended erase operation

- Concurrency handling
  - Confirmed that the BML semaphore was used correctly in all 14 BML functions (150 LOC on average)
  - Detected a bug of ignoring BML semaphore exceptions in a call sequence from STL (2500 LOC on average)

- Multi-sector read operation (MSR) (157 LOC)
  - Provided high assurance on the correctness of MSR
    - no violation was detected even after exhaustive analysis (at least with a small number of physical units (~10))

- In addition, we evaluated and compared pros and cons of CBMC and BLAST empirically
Logical to Physical Sector Mapping

1:N mapping from a LUN to PUNs

Sector mapping

In flash memory, logical data are distributed over physical sectors.

Moonzoo Kim  Automated Analysis of Industrial Embedded Software
Multi-sector Read Operation (MSR)

- MSR reads adjacent multiple physical sectors once in order to improve read speed
  - MSR is 157 lines long, but highly complex due to its 4 level loops
  - 4 parameters to specify logical data to read (from, to, how long, read flag)

- The requirement property is to check
  - after_MSR -> (∀i. logical_sectors[i] == buf[i])

- We built a verification environment model for MSR
Exponential Increase of Distribution Cases

\[
\sum_{i=1}^{n-1} \left( \binom{4}{i} C_4 \times 4! \right) \times \left( \binom{4}{n-i} C_{l-4} \times (l-4)! \right)
\]
Environment Modeling

1. **One PU is mapped to at most one LU**

2. **Valid correspondence between SAMs and PUs:**
   If the \(i\) th LS is written in the \(k\) th sector of the \(j\) th PU, then the \(i\) th offset of the \(j\) th SAM is valid and indicates the \(k\)'th PS,
   
   Ex> 3\(^{rd}\) LS (‘C’) is in the 3\(^{rd}\) sector of the 2\(^{nd}\) PU, then \(\text{SAM1}[2] == 2\)
   
   \[i=2 \quad k=2 \quad j=1\]

3. **For one LS, there exists only one PS that contains the value of the LS:**
   The PS number of the \(i\) th LS must be written in only one of the \((i \mod 4)\) th offsets of the SAM tables for the PUs mapped to the corresponding LU.

\[
\forall i, j, k \ (LS[i] = PU[j].sect[k] \rightarrow (SAM[j].valid[i \mod m] = true \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \qua
Loop Structure of MSR

01: curLU = LU0;
02: while (curLU != NULL) {

Loop1: iterates over LUs

03:     readScts = # of sectors to read in the current LU
04:     while (readScts > 0) {

Loop2: iterates until the current LU is read completely

05:         curPU = LU->firstPU;
06:     while (curPU != NULL) {

Loop3: iterates over PUs linked to the current LU

07:         while (...) {

Loop4: identify consecutive PS’s in the current PU

08:             conScts = # of consecutive PS’s to read in curPU
09:             offset = the starting offset of these consecutive PS’s in curPU
10:         }
11:         BML_READ(curPU, offset, conScts);
12:     readScts = readScts - conScts;
13:     curPU = curPU->next;
14: }
15: }
16: curLU = curLU->next;
17:}
Model Checking Results of MSR [Spin’08, TSE’11]

- Verification of MSR by using NuSMV, Spin, and CBMC
- No violation was detected within $|LS| \leq 8$, $|PU| \leq 10$
  - $10^{10}$ configurations were exhaustively analyzed for $|LS| = 8$, $|PU| = 10$

**Time complexity LS = 6**

**Space complexity LS = 6**

---

Moonzoo Kim  
*Automated Analysis of Industrial Embedded Software*
Feedbacks from Samsung Electronics

Main challenge:
• IT industry is not mature enough to conduct unit testing

1. Current SW development of Samsung is not ready to apply unit testing
   ▸ Tight project deadline does not allow defining detailed asserts and environment models

2. Needs large scalability even at the cost of accuracy
   ▸ Rigorous automated tools for small unit (i.e., SW model checker) is of limited practical value

3. Many embedded SW components have dependency on external libraries
   ▸ Pure analysis methods on source code only are of limited value

4. It is desirable to generate test cases as a result of the analysis.
   ▸ Current SW V&V practice operates on test cases
Background on Concolic Testing

- **Concrete runtime execution guides symbolic** path analysis
  - a.k.a. dynamic symbolic execution (DSE), white-box fuzzing

- **Automated test case (TC) generation technique**
  - Applicable to a large target program (no memory bottleneck)
  - Applicable to testing stages seamlessly
  - External binary library can be handled (partially)

- **Explicit path model checker**
  - All possible execution paths are explored based on the generated TCs
  - Anytime algorithm
    - User can get partial analysis result (i.e., TCs) anytime
  - Analysis of each path is independent from each other
    - Parallelization for linear speed up
    - Ex. Scalable Concolic testing for Reliability (SCORE) framework [ICST’12a]
Hierarchy of SW Coverages

- Complete Value Coverage (CVC)
- Complete Path Coverage (CPC)
- Prime Path Coverage (PPC)
- Complete Round Trip Coverage (CRTC)
- Simple Round Trip Coverage (SRTC)
- Edge Coverage (EC)
- Edge-Pair Coverage (EPC)
- Node Coverage (NC)
- All-defs Coverage (ADC)
- All-uses Coverage (AUC)
- All-DU-Paths Coverage (ADUP)
Concolic Testing Example

// Test input a, b, c
void f(int a, int b, int c) {
    if (a == 1) {
        if (b == 2) {
            if (c == 3*a + b) {
                target();
            }
        }
    }
} }

- Random testing
  - Probability of reaching Error( ) is extremely low
- Concolic testing generates the following 4 test cases
  - (0,0,0): initial random input
    - Obtained symbolic path formula (SPF) φ: a!=1
    - Next SPF ψ generated from φ: !(a!=1)
  - (1,0,0): a solution of ψ (i.e. !(a!=1))
    - SPF φ: a==1 && b!=2
    - Next SPF ψ: a==1 && !(b!=2)
  - (1,2,0)
    - SPF φ: a==1 && (b==2) && (c!=3*a +b)
    - Next SPF ψ: a==1 && (b==2) && !(c!=3*a +b)
  - (1,2,5)
    - Covered all paths and
      reached target()
Part II: Experience from Concolic Testing using CREST

Target system: Samsung Smartphone Platform

Unit-level testing
1. Busybox ls (1100 LOC)
   - 98% of branches covered and 4 bugs detected
2. Samsung security library (2300 LOC)
   - 73% of branches covered and a memory violation bug detected

System level testing
1. Samsung Linux Platform (SLP) file manager
   - detected an infinite loop bug
2. 10 Busybox utilities
   - Covered 80% of the branches with 40,000 TCs in 1 hour
   - A buffer overflow bug in grep was detected
3. Libexif
   - 300,000 TCs in 4 hours
   - 1 out-of-bound memory access bug, 1 null pointer dereferences, and 4 divide-by-0 bugs were detected
LibEXIF (Exchangeable Image File Format)

- libexif contains 238 functions in C (14KLOC)
- An IFD consists of
  - a 2 byte counter to indicate a number of tags in the IFD, tag arrays, 4 byte offset to the next IFD.
- Each tag consists of
  - tag id (2 bytes), type (2 bytes), count (i.e., a number of values) (4 bytes), value (or offset to the value if the value is larger than 4 bytes) (4 bytes).
- Manufacturer note tag is used for manufacturers of EXIF writers to record any desired information
  - Camera manufacturers define a large number of their own maker note tags
  - maker note tags are not specified in the official EXIF specification.
  - Ex. Canon defines more than 400 maker note tags.
Testing Strategies

- Open source oriented approach
  - Focusing on runtime failure bugs only
    - Null-pointer dereference, divide-by-0, out-of-bound memory accesses

- Baseline concolic testing
  - Input EXIF tag size fixed at 244 bytes
  - Full symbolic

- Focus on the maker note tags w/ concrete image files.
  - 5 among 10 largest functions are for maker notes
  - These 5 functions takes 27% of total branches

- Compare two popular Concolic testing tools
  - CREST-BV and KLEE

- Comparison with Coverity Prevent
Testing Result 1

Table I
Statistics on the Baseline Concolic Testing Experiments by Using KLEE

<table>
<thead>
<tr>
<th>Time option (sec)</th>
<th>DFS time (sec)</th>
<th># of TC</th>
<th>Br. cov. (%)</th>
<th>Random path time (sec)</th>
<th># of TC</th>
<th>Br. cov. (%)</th>
<th>Random search time (sec)</th>
<th># of TC</th>
<th>Br. cov. (%)</th>
<th>Covering new time (sec)</th>
<th># of TC</th>
<th>Br. cov. (%)</th>
<th>DFS + covering new time (sec)</th>
<th># of TC</th>
<th>Br. cov. (%)</th>
<th>Total of the 5 search strategies time (sec)</th>
<th># of bugs detected</th>
<th>TC gen. speed (#/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>900</td>
<td>1001</td>
<td>1289</td>
<td>8.1</td>
<td>2577</td>
<td>2280</td>
<td>11.1</td>
<td>2294</td>
<td>3192</td>
<td>11.1</td>
<td>2574</td>
<td>3072</td>
<td>11.1</td>
<td>1954</td>
<td>2022</td>
<td>11.1</td>
<td>10400</td>
<td>11855</td>
<td>11.1</td>
</tr>
<tr>
<td>1800</td>
<td>1903</td>
<td>2450</td>
<td>8.1</td>
<td>5530</td>
<td>4121</td>
<td>11.1</td>
<td>4832</td>
<td>5277</td>
<td>11.1</td>
<td>4944</td>
<td>4083</td>
<td>19.7</td>
<td>4928</td>
<td>3089</td>
<td>19.7</td>
<td>22137</td>
<td>19020</td>
<td>19.7</td>
</tr>
<tr>
<td>3600</td>
<td>3705</td>
<td>4868</td>
<td>8.1</td>
<td>10506</td>
<td>7084</td>
<td>11.1</td>
<td>9406</td>
<td>9945</td>
<td>19.1</td>
<td>12609</td>
<td>4543</td>
<td>20.4</td>
<td>10018</td>
<td>7685</td>
<td>19.7</td>
<td>46244</td>
<td>34125</td>
<td>20.4</td>
</tr>
</tbody>
</table>

Table II
Statistics on the Baseline Concolic Testing Experiments by Using CREST-BV

<table>
<thead>
<tr>
<th>Corresponding KLEE time option (sec)</th>
<th>DFS time (sec)</th>
<th># of TC</th>
<th>Br. cov. (%)</th>
<th>Random path time (sec)</th>
<th># of TC</th>
<th>Br. cov. (%)</th>
<th>Control flow graph (CFG) based time (sec)</th>
<th># of TC</th>
<th>Br. cov. (%)</th>
<th>Total of the 3 search strategies time (sec)</th>
<th># of bugs detected</th>
<th>TC gen. speed (#/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>900</td>
<td>1001</td>
<td>12671</td>
<td>20.2</td>
<td>2577</td>
<td>100934</td>
<td>9.3</td>
<td>900</td>
<td>25191</td>
<td>9.3</td>
<td>900</td>
<td>25191</td>
<td>21.8</td>
</tr>
<tr>
<td>1800</td>
<td>1903</td>
<td>22317</td>
<td>20.2</td>
<td>5530</td>
<td>171531</td>
<td>9.3</td>
<td>1800</td>
<td>25752</td>
<td>9.3</td>
<td>1800</td>
<td>25752</td>
<td>21.8</td>
</tr>
<tr>
<td>3600</td>
<td>3705</td>
<td>42499</td>
<td>20.3</td>
<td>10506</td>
<td>259625</td>
<td>10.3</td>
<td>3600</td>
<td>65644</td>
<td>10.3</td>
<td>3600</td>
<td>65644</td>
<td>21.8</td>
</tr>
</tbody>
</table>

Out-of-bound memory access bug detected
- exif_data_load_data () of exif-data.c as follows (line 2):
  1: if (offset + 6 + 2 > ds) { return; }
  2: n = exif_get_short(d+6+offset, ...)
Testing Result 2

Table III

<table>
<thead>
<tr>
<th>Time option (sec)</th>
<th>DFS (Sum on the 6 files)</th>
<th>Random path (Sum on the 6 files)</th>
<th>Random search (Sum on the 6 files)</th>
<th>Covering new (Sum on the 6 files)</th>
<th>DFS+covering new (Sum on the 6 files)</th>
<th>Total of the 5 strategies on the 6 files each</th>
<th># of bugs detected</th>
<th>TC gen. speed (#/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (sec)</td>
<td># of TC</td>
<td>Br.cov. (%)</td>
<td>time (sec)</td>
<td># of TC</td>
<td>Br.cov. (%)</td>
<td>time (sec)</td>
<td># of TC</td>
</tr>
<tr>
<td>900</td>
<td>5424</td>
<td>15804</td>
<td>44.5</td>
<td>5526</td>
<td>4800</td>
<td>44.3</td>
<td>5592</td>
<td>6684</td>
</tr>
<tr>
<td>1800</td>
<td>10830</td>
<td>24936</td>
<td>44.7</td>
<td>11010</td>
<td>8172</td>
<td>44.7</td>
<td>11154</td>
<td>10758</td>
</tr>
<tr>
<td>3600</td>
<td>21642</td>
<td>39270</td>
<td>44.7</td>
<td>21996</td>
<td>11342</td>
<td>44.7</td>
<td>22416</td>
<td>15378</td>
</tr>
</tbody>
</table>

Table IV

<table>
<thead>
<tr>
<th>Corresponding KLEE time option (sec)</th>
<th>DFS (Sum on the 6 files)</th>
<th>Random path (Sum on the 6 files)</th>
<th>CFG based (Sum on the 6 files)</th>
<th>Total of the 3 strategies on the 6 files each</th>
<th># of bugs detected</th>
<th>TC gen. speed (#/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (sec)</td>
<td># of TC</td>
<td>Br.cov. (%)</td>
<td>time (sec)</td>
<td># of TC</td>
<td>Br.cov. (%)</td>
</tr>
<tr>
<td>900</td>
<td>5424</td>
<td>93645</td>
<td>48.7</td>
<td>5526</td>
<td>130387</td>
<td>58.7</td>
</tr>
<tr>
<td>1800</td>
<td>10830</td>
<td>174173</td>
<td>48.7</td>
<td>11010</td>
<td>245873</td>
<td>59.3</td>
</tr>
<tr>
<td>3600</td>
<td>21642</td>
<td>309931</td>
<td>48.7</td>
<td>21996</td>
<td>433570</td>
<td>59.6</td>
</tr>
</tbody>
</table>

- KLEE detected 1 null-pointer-dereference
- CREST-BV detected 4 divide-by-0 bugs in addition
Null-pointer-dereference bug

```c
for(i=0;i<sizeof(table)/sizeof(table[0]);i++)
//t is a maker note tag read from an image
if (table[i].tag==t) {
    //Null-pointer dereference occurs!!!
    if(!*table[i].description)
        return "";
```

Divide-by-0 bug

```c
vr=exif_get_rational(...);
//Added for concolic testing
assert(vr.denominator!=0);
a = vr.numerator / vr.denominator
```
Testing Result 3

- **Comparison with Coverity Prevent**
  - Prevent detected the following null-pointer dereference bug, which KLEE/CREST-BV did not detect
    - because test-mnoot.c does not call the buggy function.
  ```c
  1: if(!loader||(loader->data_format ...) {
  2: exif_log(loader->log, ...);
  ```

- However, no bugs detected by concolic testing was detected by Prevent
  - Not surprising
    - (Prevent spent only 5 minutes to analyze libexif)
Lessons Learned from Real-world Application

- Practicality of Concolic testing
  - 1 null-pointer dereference, 1 out-of-bound memory access, and 4 divide-by-0 in reasonable time
- Note that
  - libexif is very popular OSS tampered by millions of users
  - we did not have background on LIBEXIF!!

- Importance of Testing Methodology/Strategy
- Still state space explosion is a big obstacle
  - Average length of symbolic path formula = 300
    => In theory, there exist $2^{300}$ test cases to test
Conclusion and Future Work

- Formal verification techniques really work in IT industry!
  - Model checking and concolic testing detected hidden bugs in industrial embedded software

- To alleviate the limitations of concolic testing
  - External function summaries through dynamic invariance generation
  - Develop a new search strategy for fast branch coverage

- Data mining on a huge set of runtime execution information
  - (semi) Automated oracle generation through dynamic invariant generation
  - Automated debugging

- Technical papers can be downloaded at [http://pswlab.kaist.ac.kr](http://pswlab.kaist.ac.kr)