Verification of Industry Code: Challenges

R Venkatesh
r.venky@tcs.com
Overview

- **Focus of talk**
  - Scalability problems in industry code
  - Ideas we are exploring

- **Formal verification @ TRDDC**
  - Apply academic ideas to address quality related problems
    - Experiments and tools
    - Adapt as required
      - Scale up
      - Specific solutions

- Based on experiences with embedded software
Context

- Finding bugs early in software

- Model based development
  - Matlab Simulink
  - Statecharts
  - Code
    - Generated and hand written

- Analysis and Testing
  - Most bugs can be found
Code Analysis

- Standard + other properties
  - Zero division
  - Correct use of semaphores

- Dataflow analysis + model checking
  - Variable ranges from static analysis

- Precision is the key challenge
  - Model checking does not scale up
## Code Characteristics

<table>
<thead>
<tr>
<th>Application</th>
<th>Size</th>
<th>Key Characteristics</th>
<th>FPS(ZD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infotainment</td>
<td>2MLOC (1 task)</td>
<td>Large, large arrays(512), loops(unknown bounds)</td>
<td>77</td>
</tr>
<tr>
<td>Smart card component</td>
<td>7K</td>
<td>Loops with large bounds and unknown bounds</td>
<td>55</td>
</tr>
<tr>
<td>Several</td>
<td>Upto 36K</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

```java
ej = nondet() * 2;
for ( ; j < 512; j += 2) {
    assert( j + 1 < 512);
}

t = nondet_long();
while((t / sec_366) > 0) {
    if( y % 4)  t -= sec_365;
    else t -= sec_366;
    y++ ;
}
... assert (m < 12 );```
Test Generation

- Code coverage
  - Modified Decision Condition Coverage

- Very similar to property checking
  - Most states will be reachable
  - High coverage needed
  - False positives not an issue

- Scaling up is the key challenge

Diagram:

1. C Code
2. AutoGen
3. Model checker
4. Test Cases
Code Characteristics

- Driver assist + odometer cluster
- Generated code
- Recursive code
- Nested loops
- Counters + floating operations

```c
while ( j++ <= 31 && !l)
    for (i = 0; i <= 31; i++)
        if (*)
            f(a[i]);
            l = i;

while c(a[l], a[l + 1] )
    l++;
```

```c
while ( *)
    recursion
    <counters>++;
    assert (counter < k );
```
Current Ideas being Explored
Loop Abstraction

- Replace loops by small bounded loops

- One execution of body
  - Each distinct path
  - Distinct output variable

- Recurrence relations
  - Linear

- Naïve refinement

while (*)

\[ o_n = f(<io>) \]

for ( i in 1..n )

\[ k = \ast; \]
\[ <io> = \ast/recur(k); \]
\[ o_n = f(<io>) \]
Guessing Invariants : Daikon

- Generate random traces
- Guess invariants
  - Daikon
  - Template based
- Replace complex code by invariants
- Works well in practice
Statecharts Analysis

- Size
  - Per statechart
    - ~ 5 states, ~ 6-7 transitions
    - Translates to ~200 lines of C code
  - ~ 500 statecharts, composed in parallel

- Real valued clock variables, ~ 1-2 per statechart

- Very long paths to reach some states
  - A fourth of the states did not reach in depth 50

- loops in each statechart
Summary

- **Success**
  - Scales up well to several thousand lines
  - Found several bugs
    - Production code
    - Medical, smart card, auto ...

- **Limitations**
  - Scalability
    - ECUs of millions of lines of code
    - Financial software much bigger
  - Distributed systems
    - Multiple ECUs

- **Need order of magnitude scale up**
  - Compositional, heuristics