Reducing Time and Efforts When Verifying Large Software Systems with Klever

Ilia Zakharov
Ivannikov Institute For System Programming of RAS
ilja.zakharov@ispras.ru
Klever Verification Framework

Intended for finding bugs in large software systems using existing software verification tools

- C programs
  - GNU C, Microsoft C
  - x86, ARM
  - > 200 KLOC
- Requirements supported by software verification tools used as backends
  - Memory safety
  - Data races
  - API usage
- Deliver both results and a program-adapted tool
  - Coverage reports, error traces and results of bugs triage
  - A tool to check regressions or reproduce results
Workflow

Development
- Implement program-specific parts in the framework
- Prepare models and specifications

Verification
- Find bug
- Deliver the tool and results
Klever Verification Framework

1. Source Code
2. Program Decomposition
3. Environment Modeling
4. Requirements Modelling
5. Verification
6. Results Assessment
7. Bugs or Proofs
Source Code Preparation
Goals

• Limit scope to specific components, source code versions, architectures
• Prepare requirements
• Plan development stages
• Find bugs or prove correctness
• Deliver results or a tool
Prepare Build Base

- Source Code ➔ Build System ➔ Executables
- Clade and CIF ➔ Build Base
Program Decomposition
Program Decomposition

- Reduce environment modelling efforts
- Prevent timeouts
- Get rid of unsupported code
Extract Components as Fragments

1. Determine logical components
   • With unique interface
   • With common interface

2. Separate libraries

3. Remove auxiliary or irrelevant parts
   • Debug
   • Tests
Manual or Automatic Decomposition

• Define fragments explicitly in advance
• Develop algorithms to decompose the program using its build base
Provide Environment Models
Kinds of Models

<table>
<thead>
<tr>
<th>Kind of a Model</th>
<th>External Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Model</td>
<td>Undefined functions that init/uninit resources and influence the control flow</td>
</tr>
<tr>
<td>Requirement Model</td>
<td>Specific API that should be properly used by the fragment</td>
</tr>
<tr>
<td>Environment Model</td>
<td>Functions that call entry points and influence these calls</td>
</tr>
</tbody>
</table>

Program Fragment

Requirement Model

Common Model

Environment Model
Environment Model

Component

Lib

Program
Interaction Scenarios

1. Component ➔ Lib
2. Program ➔ Component
3. Component ➔ Program
4. Program ➔ Component
5. Component ➔ Lib
Intermediate Environment Model

- `tty_alloc_driver (failed)`
- `tty_alloc_driver (success)`
- `tty_set_operations`
- `tty_register_driver (failed)`
- `tty_register_driver (success)`
- `put_tty_driver`
- `tty_unregister_driver`
- `moxa_open (failed)`
- `moxa_open (success)`
- `moxa_close`
- `moxa_write`

Activate/deactivate
Environment Model Generator

- Environment Model Specifications
- Program Fragment
- Scenario Model Builders
- Intermediate Environment Model
- Translator
- C Code
Environment Model Generator

- Environment Model Specifications
- Program Fragment

Scenario Model Builders
- Provide specifications as templates
- Select auxiliary function models
- Adjust completeness of the environment model

Intermediate Environment Model
- Manually develop environment model specifications

Translator

C Code
Provide Requirement Specifications
Requirement Specification

```c
int cnt = 0;

int try_module_get(struct module * m) {
    ret = ldv_random_neg_int();
    if (!ret)
        cnt++;
    return ret;
}

void module_put(struct module * m) {
    cnt--;
}

void ldv_check_final_state(void) {
    ldv_assert(cnt==0);
}
```
Requirement Specifications Development

1. Support an empty requirement to measure the coverage
2. Support memory safety and data race safety requirement specifications
3. Implement other requirement specifications
4. Develop tests for requirement models
Analyse Results
Verification Results

• Error traces (witnesses)
• Coverage reports
• Logs
• Resource consumption statistics
Use cases

Development
- Uncovered entry points
- Complicated code
- Classify fails

Refinement
- Marks and tags for errors and false positives
- Regression tests

Verification
- Find bugs
- Prepare final marks and tags
Evaluation
Manual Effort at Verification of Linux Device Drivers and Subsystems

<table>
<thead>
<tr>
<th>Stage</th>
<th>Serial device drivers (20KLOC)</th>
<th>All device drivers (4MLOC)</th>
<th>Subsystems (1MLOC)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development of decomposition</td>
<td>0,25 man-months (100 LOC Python)</td>
<td>-</td>
<td>0,25 man-months (100 LOC Python)</td>
<td>0,5 man-months (200 LOC Python)</td>
</tr>
<tr>
<td>algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development of environment model</td>
<td>3 man-months (3 KLOC Python)</td>
<td>-</td>
<td>0,5 man-months (500 LOC Python)</td>
<td>3,5 man-months (3,5 KLOC Python)</td>
</tr>
<tr>
<td>builders</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development of environment model</td>
<td>4,5 man-months (7 KLOC DSL)</td>
<td>5,5 man-months (10 KLOC DSL)</td>
<td>-</td>
<td>10 man-months (17 KLOC DSL)</td>
</tr>
<tr>
<td>specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development of requirement</td>
<td>6 man-months (550 LOC DSL)</td>
<td>9 man-months (950 LOC DSL)</td>
<td>0,25 man-months (200 LOC DSL)</td>
<td>15,25 man-months (1500 LOC DSL)</td>
</tr>
<tr>
<td>specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>13,75 man-months</td>
<td>14,5 man-months</td>
<td>1 man-month</td>
<td>29,25 man-months</td>
</tr>
</tbody>
</table>
## BusyBox Applets Verification

<table>
<thead>
<tr>
<th>Stage</th>
<th>Efforts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development of decomposition algorithms</td>
<td>0,25 man-months (100 LOC Python)</td>
</tr>
<tr>
<td>Development of environment model builders</td>
<td>0 man-months</td>
</tr>
<tr>
<td>Development of environment model specifications</td>
<td>0,25 man-months (200 LOC DSL)</td>
</tr>
<tr>
<td>Development of requirement specifications</td>
<td>0,5 man-months (300 LOC DSL)</td>
</tr>
<tr>
<td>Total</td>
<td>1 man-month</td>
</tr>
</tbody>
</table>
Why do you care

• Another point of view
• Train your verification tool
• Get new verification tasks
Thank You!

https://github.com/ldv-klever/klever - Mirror
https://forge.isprats.ru/projects/klever - Issue tracker
https://github.com/17451k/cif - CIF
https://github.com/17451k/clade - Clade
http://linuxtesting.org/kernel - Other links and verification projects
Build Base

• Various information about the program
  • Source code
  • Build command graph
  • File dependencies graph
  • Callgraph
  • ...

• Easy to access
  • Movable archive with all sources and data
  • Python API to access the data
## Summary

<table>
<thead>
<tr>
<th>Development</th>
<th>Decomposition</th>
<th>Environment Modeling</th>
<th>Requirement Specifications Development</th>
<th>Analyzing results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Manually prepared decomposition specification</td>
<td>Coarse model without restrictions</td>
<td>Empty requirement</td>
<td>Code coverage</td>
</tr>
<tr>
<td></td>
<td>Decomposition algorithms</td>
<td>Scenario model builders</td>
<td>Requirement specifications, common model</td>
<td>Code coverage, marks, tags</td>
</tr>
<tr>
<td></td>
<td>Algorithms to verify libraries</td>
<td>Environment model specifications, tests</td>
<td>Requirement specifications, common model, tests</td>
<td>Code coverage, marks, tags</td>
</tr>
<tr>
<td>Verification</td>
<td>-</td>
<td>-</td>
<td>Common model</td>
<td>Code coverage, marks, tags</td>
</tr>
</tbody>
</table>
## Verification Time

<table>
<thead>
<tr>
<th>Verification Job</th>
<th>2 physical cores</th>
<th>4 physical cores</th>
<th>30 * 4 physical cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial device drivers (30KLOC)</td>
<td>5h</td>
<td>2.7h</td>
<td>0.5h</td>
</tr>
<tr>
<td>All device drivers (3MLOC)</td>
<td>600h</td>
<td>195h</td>
<td>11h</td>
</tr>
</tbody>
</table>