Combining static & dynamic analysis for software verification

PETIOT Guillaume
PhD student

KOSMATOV Nikolaï
GIORGETTI Alain
Advisors

JULLIAND Jacques
Supervisor
PLAN

– Subject
– Context
– Problem, Motivations, Solutions
– Overview
– Example
– Conclusion, Perspectives
Combining static & dynamic analysis for software verification

- Static analysis: abstract interpretation, deductive verification, etc.
  - complete
  - imprecise

- Dynamic analysis: test, runtime verification
  - precise
  - incomplete
## CONTEXT

**SOFTWARE VERIFICATION AT CEA/LSL**

<table>
<thead>
<tr>
<th>Framework</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frama-C</strong></td>
<td>framework of modular analysis of C</td>
</tr>
<tr>
<td><strong>Kernel</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ACSL</strong></td>
<td>specification language</td>
</tr>
</tbody>
</table>

**Plug-ins**

- **Value**: Value analysis
- **WP**: Deductive verification
- **PathCrawler**: Tests generation, all-paths coverage
- **E-ACSL**: Translation from ACSL to C
- **SANTE**: Collaboration Value + slicing + PathCrawler
- **Etc.**:
PROBLEM

– How to verify generic properties like runtime errors not handled by SANTE?
– And properties specified by the user?

MOTIVATIONS

– Automation
– Reliable alarm classification: low «false positives» rate
– Overcome the drawbacks of each method

SOLUTIONS

– Combining slicing, static analysis and testing to overcome drawbacks of each method
```c
int x2 (int i)
{  int k = 2 * i ;
   /*@ assert k > 0 ; */
   return k ; }
```

```c
int x2 (int i)
{  int k = 2 * i ;
   e_acsl_assert(k > 0) ;
   return k ; }
```

```c
void main()
{  int i = -35 ;
    x2 (i) ; }
```
CONCLUSION, PERSPECTIVES

CONCLUSION

– Handling ACSL annotations by PathCrawler
  • Safety assertions on overflows, pointers, etc.
– Counter-example generation
  • Updating property status in Frama-C
  • Reusable in other plug-ins

PERSPECTIVES

– Handling ACSL pre- and post-condition by PathCrawler
– Application, experiments
Thank you