Trickle: Automated Infeasible Path Detection Using All Minimal Unsatisfiable Subsets

Bernard Blackham†, Mark Liffiton‡, Gernot Heiser†

† School of Computer Science & Engineering, UNSW
Software Systems Research Group, NICTA
Sydney, Australia

‡ Illinois Wesleyan University
Motivation

• The desire for a *trustworthy* kernel to build reliable *mixed-criticality real-time* systems
Motivation

• The desire for a trustworthy kernel to build reliable mixed-criticality real-time systems

• Using seL4 to guarantee:
  – functional correctness through formal proof
    (Klein et al., SOSP 2009)
  – timing constraints through sound WCET analysis
    (Blackham et al., RTSS 2011)
Motivation

• The accuracy of WCET estimates directly affects the hardware provisioning of such systems (e.g. CPU speed) → cost

• Inaccurate WCET estimates by static analysis can be caused by **infeasible paths**

• By excluding culprit infeasible paths
  → less provisioning required
  → reduced hardware cost
Infeasible paths

```c
int f(int a)
{
    if ((a & 4) == 0)
        ...
    else
        ...
    ...
    if ((a & 4) == 0)
        ...
    else
        ...
}
```
Infeasible paths

```c
int f(int a)
{
    if ((a & 4)==0)
        ...
    else
        ...
    ...
    ...
    if ((a & 4)==0)
        ...
    else
        ...
}
```
Infeasible paths

```c
int f(int a)
{
    if ((a & 4) == 0)
        ...
    else
        ...
    ...
    if ((a & 4) == 0)
        ...
    else
        ...
}
```
Infeasible paths

conflicts with

X

Y
Background

- **Chronos** used to compute WCET via IPET

- **Sequoll** can validate manually provided infeasible path information using a model checker
  (Blackham & Heiser, RTAS 2013)

- Reduces risk of human error when specifying infeasible path information

- **Can we find infeasible paths automatically?**
Finding infeasible path constraints

All-at-once

Find all infeasible path criteria over whole program

Compute WCET

Done!

Directed Iterative Refinement

Compute WCET

Reconstruct path

Path valid?

Y

Done!

N

Add infeasible path criteria
seL4 is large

- Small by microkernel standards
- Large by WCET standards

<table>
<thead>
<tr>
<th>C source</th>
<th>Binary (ARM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>~8,700 lines</td>
<td>~10,000 instructions</td>
</tr>
<tr>
<td>316 functions</td>
<td>228 functions</td>
</tr>
<tr>
<td>76 loops</td>
<td>56 loops</td>
</tr>
<tr>
<td></td>
<td>2,384 basic blocks</td>
</tr>
<tr>
<td></td>
<td>~400,000 basic blocks when inlined</td>
</tr>
</tbody>
</table>
Trickle: Directed Iterative Refinement

- Compute WCET
- Reconstruct path
- Path valid?
- Add infeasible path criteria
- Done!
Types of infeasible path criteria

• Infeasible path criteria may arise because of:

local constraints

```c
y = count;
if (y > 15)
    y = 15;
while (y > 0)
{
    ...
    y = y - 1;
}
```

global program invariants

```c
// count guaranteed to be <= 15
y = count;
while
{
    ...
    y = y - 1;
}
```
Types of infeasible path criteria

- Infeasible path criteria may arise because of:

  - **Local constraints**
    
    ```
    y = count;
    if
      y =
    while
    {
      y = y -
    }
    ```

  - **Global program invariants**
    
    ```
    // count guaranteed to be <= 15
    y = count;
    
    while (y > 0)
    {
      ...
      y = y - 1;
    }
    ```
Types of infeasible path criteria

- Infeasible path criteria may arise because of:

  **Local constraints**

  ```
  y = count;
  if (y > 15)
    y = 15;
  while (y > 0)
  {
    ...
    y = y - 1;
  }
  ```

  **Global program invariants**

  ```
  // count guaranteed to be <= 15
  y = count;

  while (y > 0)
  {
    ...
    y = y - 1;
  }
  ```
Detecting Infeasible Paths

- **Sequoll** computes a **branch condition** for every conditional branch or conditional instruction, in terms of SSA variables.

- Once a worst-case path is identified, **Trickle** collects all branch conditions required to execute it, as **SMT expressions**.

- All branch conditions are given to an **SMT solver** to find a satisfying assignment.
Detecting Infeasible Paths

• If SMT solver finds a satisfying assignment, path is declared **feasible**\*  
  
• If SMT solver shows that the constraints are unsatisfiable, the path is **infeasible**  
  ➡ An **unsatisfiable subset** is returned

\* up to the limit of reasoning ability of the SMT solver and Trickle
Detecting Infeasible Paths

A: $a = 0$
B: $a = b$
C: $b \neq 1$

Satisfiable! 👍

Assignment: $a = 0$, $b = 0$
Detecting Infeasible Paths

A: $a = 0$
B: $a = b$
C: $b = 1$

Unsatisfiable 😞

Minimal Unsatisfiable Subset: \{A, B, C\}

$E_A + E_B + E_C < 3$
• A path may contain 1000s of instructions
• And 100s of branch constraints
• And may contain several MUSes
• SMT solvers typically only find one
• Can we reduce the number of refinement iterations?
Trickle: Enter CAMUS

**Sequoll** – framework for analysis of compiled ARM binaries

**CAMUS** – Compute All Minimal Unsatisfiable Subsets

**Yices** – SMT solver
• Developed by Liffiton & Sakallah (JAR 2008)

• Finds all minimal unsatisfiable subsets of a given set of constraints

→ i.e. finds all infeasible path constraints along a given path
CAMUS

• The **worst-case run time** of the CAMUS algorithm is **exponential** in the number of MUSes (+ SMT solver time)

How can we prevent this?
• Try with a smaller subset of constraints first
• Choose constraints close together on path as they are more likely to conflict
• Increase size of window and repeat if no constraints found
Results

Estimated worst-case execution time of seL4

- Baseline (no infeasible path information)
- Trickle applied to baseline
- Trickle + human efforts

000’s of cycles
Results

Number of iterations to find WCET

- SMT unsat core only
- CAMUS

© NICTA 2014
Results

Difference in runtime to compute WCET*

* Implementation-specific compilation overheads subtracted
Limitations

• What about loops?
  – Analysis is limited to loop-free regions of code
  – Limitation of SMT solvers
  – Limitations of IPET method

• Run time of analysis is long
  – Full analysis takes > 2 hours
Research directions

• Integrate Trickle with proof invariants
• Find infeasible paths across loop iterations
• Improved CAMUS algorithm to avoid exponential behaviour (no need for sliding window)
• Improve memory aliasing analysis
Summary

Trickle is able to:

• automatically compute infeasible path information on compiled ARM binaries
• improve WCET estimates of an IPET analysis
• reason about more interesting constraints than integer intervals (e.g. bit arithmetic)

→ reduce scope for errors in WCET analysis!

Download it!

Bernard.Blackham@nicta.com.au