Using SMT Solving for the Lookup of Infeasible Paths in Binary Programs

by Jordy Ruiz and Hugues Cassé
1. Context
   Introducing thoughts
2. Analysing the semantics of a binary program
   The foundations of our work
3. Finding infeasible paths
   Explaining the mechanics of this analysis
4. Conclusion
   Some closing thoughts and talk about future works
Context
Improving the WCET estimation

- Find a safe, *tight* timing bound
- Infeasible paths are the main source of overestimation in WCET computation

- Identifying infeasible paths refines the WCET estimation
Working on binary programs

Working directly on binaries is **harder**

- low expressivity of machine instructions
- larger size of program
- loosely typed registers
- obscure structure of data in memory
- ...

...
Working directly on binaries is **harder** but is **more adapted**:

- low expressivity of machine instructions
- larger size of program
- loosely typed registers
- obscure structure of data in memory
- ...

- not mapping properties from source to binaries
- independent of compiler
- available source libraries are not required
- easy injection in WCET computation
Working directly on binaries is harder but is more adapted:

- low expressivity of machine instructions
- larger size of program
- loosely typed registers
- obscure structure of data in memory
- ...

- short-circuit condition evaluation
  ```
  if (x && a)
      /* ... */
  if (x && b)
      /* ... */
  ```
Working on binary programs

Working directly on binaries is **harder** but is **more adapted**:

- low expressivity of machine instructions
- larger size of program
- loosely typed registers
- obscure structure of data in memory
- ...

- short-circuit condition evaluation
  ```
  if (x)
      if (a)
          /* ... */
  if (x)
      if (b)
          /* ... */
  ```
Analysing the semantics of a binary program
Semantic instructions

Architecture dependent?

ARM machine instructions

STM DB
sp!, {r4, lr}

OTAWA
semantic instructions

seti t1, 1
add r3, r3, t1

seti t2, 4
seti t1, 8
sub t1, r13, t1
set t3, t1
store r4, t1, int64
add t1, t1, t2
store r14, t1, int64
add t1, t1, t2
set r13, t3
Semantic instructions

Architectures dependent? No!

**ARM machine instructions**

```
ADD r3, r3, #1
```

**OTAWA semantic instructions**

```
seti t1, 1
add r3, r3, t1
```
Semantic instructions

Architecture dependent? No!

**ARM machine instructions**

**ADD** r3, r3, #1

**STMDB** sp!, {r4, lr}

**OTAWA semantic instructions**

seti t1, 1
add r3, r3, t1

seti t2, 4
seti t1, 8
sub t1, r13, t1
set t3, t1
store r4, t1, int64
add t1, t1, t2
store r14, t1, int64
add t1, t1, t2
set r13, t3
Abstract interpretation

- Maintain an abstract program state for each path
- Top-to-bottom analysis
- Inline calls
- Program states are represented by a conjunction of predicates

\[ \gamma(\bigwedge \phi_i) = \{ x \in S \mid \bigwedge \phi_i(s) \} \]
Updating the abstract program state

- $r_{13} = SP_0 + 0$
- $r_3 = r_1$

(initial state)

$SP_0$ is the initial value of the stack pointer
Updating the abstract program state

\[
\text{ADD} \ r_3, r_3, \ #1
\]

\[ r_{13} = SP_0 + 0 \]

\[ r_3 = r_1 \]
ADD r3, r3, #1
seti t1, 1

- $r_{13} = SP_0 + 0$
- $r_3 = r_1$
- $t_1 = 1$
Updating the abstract program state

\[
\begin{align*}
\text{ADD } & r3, r3, \#1 \\
& \text{seti } t1, 1 \\
& \text{add } r3, r3, t1 \\
& r_{13} = SP_0 + 0 \\
& r_3 - t_1 = r_1 \\
& t_1 = 1
\end{align*}
\]
**Updating the abstract program state**

\[
\begin{align*}
\text{ADD} & \quad r3, r3, \#1 \\
\text{seti} & \quad t1, 1 \\
\text{add} & \quad r3, r3, t1 \\
\end{align*}
\]

- \( r_{13} = SP_0 + 0 \)
- \( r_3 - 1 = r_1 \)
- \( t_1 = 1 \)
Upgrading the abstract program state

**ADD** r3, r3, #1

seti t1, 1

add r3, r3, t1

**STMDB** sp!, {r4, lr}

- \( r_{13} = SP_0 + 0 \)
- \( r_3 - 1 = r_1 \)
- \( t_1 = 1 \)
Updating the abstract program state

**ADD**  
r3, r3, #1  
seti t1, 1  
add r3, r3, t1

**STMDB**  
sp!, {r4, lr}  
seti t2, 4  
seti t1, 8

- $r_{13} = SP_0 + 0$
- $r_3 - 1 = r_1$
- $t_2 = 4$
- $t_1 = 8$
Updating the abstract program state

**ADD** r3, r3, #1
- seti t1, 1
- add r3, r3, t1

**STMDB** sp!, {r4, lr}
- seti t2, 4
- seti t1, 8
- sub t1, r13, t1

- $r_{13} = SP_0 + 0$
- $r_3 - 1 = r_1$
- $t_2 = 4$
- $t_1 = SP_0 - 8$
Adding the abstract program state

**ADD**  r3, r3, #1
    seti t1, 1
    add r3, r3, t1

**STMDB**  sp!, {r4, lr}
    seti t2, 4
    seti t1, 8
    sub t1, r13, t1
    set t3, t1

- \( r_{13} = SP_0 + 0 \)
- \( r_3 - 1 = r_1 \)
- \( t_2 = 4 \)
- \( t_1 = SP_0 - 8 \)
- \( t_3 = SP_0 - 8 \)
### Updating the abstract program state

**ADD** r3, r3, #1
- seti t1, 1
- add r3, r3, t1

**STMDB** sp!, {r4, lr}
- seti t2, 4
- seti t1, 8
- sub t1, r13, t1
- set t3, t1
- store r4, t1, *int64*

- \( r_{13} = SP_0 + 0 \)
- \( r_3 - 1 = r_1 \)
- \( t_2 = 4 \)
- \( t_1 = SP_0 - 8 \)
- \( t_3 = SP_0 - 8 \)
- \( [SP_0 - 8] = r_4 \)
Updating the abstract program state

**ADD** r3, r3, #1  
seti t1, 1  
add r3, r3, t1

**STMDB** sp!, {r4, lr}  
seti t2, 4  
seti t1, 8  
sub t1, r13, t1  
set t3, t1  
store r4, t1, int64  
add t1, t1, t2  
store r14, t1, int64  
set r13, t3

- $r_{13} = SP_0 - 8$
- $r_3 - 1 = r_1$
- $t_2 = 4$
- $t_1 = SP_0 - 4$
- $t_3 = SP_0 - 8$
- $[SP_0 - 8] = r_4$
- $[SP_0 - 4] = r_{14}$
Finding infeasible paths
Example of infeasible path

Accounting for all 4 paths, WCET = 23 cycles + ...

But:
$$\neg (x < 10) \land (x < 0) \models \bot$$

Without the infeasible path, WCET = 21 cycles + ...

Finding infeasible paths
Example of a simple abstract program state:

- $r_0 > 8$
Example of a simple abstract program state:

- \( r_0 > 8 \)
- \([0x8008] = 0\)
Example of a simple abstract program state:

- $r_0 > 8$
- $[0x8008] = 0$
- $r_{13} = SP_0 - 24$  
  \(SP_0\) remains constant throughout the program
Example of a simple abstract program state:

- $r_0 > 8$
- $[0x8008] = 0$
- $r_{13} = SP_0 - 24$
- $r_0 = r_1$

$SP_0$ remains constant throughout the program.
Example of a simple abstract program state:

- \( r_0 > 8 \)
- \( [0x8008] = 0 \)
- \( r_{13} = SP_0 - 24 \)  \( SP_0 \) remains constant throughout the program
- \( r_0 = r_1 \)
- \( r_1 = 0 \)
Finding infeasible paths

Example of a simple abstract program state:

- $r_0 > 8$
- $[0x8008] = 0$
- $r_{13} = SP_0 - 24$
- $r_0 = r_1$
- $r_1 = 0$

This program state is unsatisfiable! ("UNSAT")

$\Rightarrow$ The current path is infeasible

Example:

$0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9$
Labelled predicates

Label predicates by the basic block(s) that generated them:

- \(r_0 > 8^{(1,5)}\)
- \([0x8008] = 0^{(3)}\)
- \(r_{13} = SP_0 - 24^{(4)}\)
- \(r_0 = [SP_0 - 16]^{(9)}\)
- \([SP_0 - 16] = 0^{(9)}\)

- Full infeasible path:
  \(0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9\)
Labelled predicates

Label predicates by the basic block(s) that generated them:

- \( r_0 > 8^{(1,5)} \)
- \([0\times8008] = 0^{(3)}\)
- \( r_{13} = SP_0 - 24^{(4)} \)
- \( r_0 = [SP_0 - 16]^{(9)} \)
- \([SP_0 - 16] = 0^{(9)}\)

- Full infeasible path: 
  \(0 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 8 \rightarrow 9\)
- Minimized infeasible path: 
  \(1 \rightarrow 5 \rightarrow 9\)
Satisfiability Modulo Theories solver:

- a SAT solver enhanced with multiple theories:
  - Rational/Integer/Booleans
  - Arrays
  - BitVectors
  - ...

⇒ We use Quantifier-Free Linear Integer Arithmetic
SMT solving

Satisfiability Modulo Theories solver:

- a SAT solver enhanced with multiple theories:
  - Rational/Integer/Booleans
  - Arrays
  - BitVectors
  - ...

→ We use Quantifier-Free Linear Integer Arithmetic

- receives a list of assertions then seeks a model (satisfiability check)
Some SMT solvers feature UNSAT cores:

- Triggered when a system is proven unsatisfiable
- Gives a minimal set of assertions that preserves unsatisfiability

Example:

\[
\begin{align*}
0 &> 8 \\
[0, x] &\subseteq 0 \\
r_0 &> = 0 \\
r_1 &< 2 \\
r_0 &< [2, x] \\
[r_0 &< [2, x] &\subseteq [2, x] = 0)
\end{align*}
\]
Some SMT solvers feature UNSAT cores:

- Triggered when a system is proven unsatisfiable
- Gives a minimal set of assertions that preserves unsatisfiability
- Example:
  
  - $r_0 > 8$
  - $[0x8008] = 0$
  - $r_{13} = SP_0 - 24$
  - $r_0 = [SP_0 - 16]$
  - $[SP_0 - 16] = 0$
Macro analysis

- Working List algorithm: “only process a Basic Block if all paths leading to it have been processed”
Macro analysis

- Working List algorithm: “only process a Basic Block if all paths leading to it have been processed”

Finding infeasible paths
Macro analysis

Working List algorithm: “only process a Basic Block if all paths leading to it have been processed”
Macro analysis

- Working List algorithm: “only process a Basic Block if all paths leading to it have been processed”
Macro analysis

- Working List algorithm: “only process a Basic Block if all paths leading to it have been processed”
Working List algorithm: “only process a Basic Block if all paths leading to it have been processed”
Macro analysis

- Working List algorithm: “only process a Basic Block if all paths leading to it have been processed”
Macro analysis

- Working List algorithm: “only process a Basic Block if all paths leading to it have been processed”

- Loops:
  - Iterate and merge with previous state until fixpoint is reached
  - When a fixpoint is reached, enable SMT checks to find infeasible paths valid at every iteration
Processing loops

- To start things off, merge all incoming states
Processing loops

- To start things off, **merge all incoming states**
- Parse the loop body normally
To start things off, merge all incoming states

Parse the loop body normally

Then merge with the previous state
Processing loops

- To start things off, merge all incoming states
- Parse the loop body normally
- Then merge with the previous state
- Reapeat until fixpoint is reached
Processing loops

- To start things off, **merge all incoming states**
- Parse the loop body normally
- Then **merge with the previous state**
- Reapeat until fixpoint is reached
To start things off, merge all incoming states

Parse the loop body normally

Then merge with the previous state

Repeat until fixpoint is reached

Do SMT checks
To start things off, **merge all incoming states**

Parse the loop body normally

Then **merge with the previous state**

Repeat until fixpoint is reached

**Do SMT checks**

Exit the loop
Merging

A very rough merging algorithm: predicate set intersection

- $r_{13} = SP - 4$
- $r_0 = [SP - 8]$
- $r_1 = 0$
- $r_2 = 0$
- $[0x8008] = 16$

becomes:

- $r_{13} = SP - 4$
- $r_0 = [SP - 8]$
- $r_1 = 1$
- $r_2 > 0$
Merging

A very rough merging algorithm: predicate set intersection

- $r_{13} = SP - 4$
- $r_0 = [SP - 8]$
- $r_1 = 0$
- $r_2 = 0$
- $[0x8008] = 16$

becomes:

- $r_{13} = SP - 4$
- $r_0 = [SP - 8]$
## Mälardalen benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>BB (#)</th>
<th>Time (s)</th>
<th>IPs found with minimization</th>
<th>without minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SMALL BENCHMARKS (NO MERGING REQUIRED)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ndes</td>
<td>57</td>
<td>0.267</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>expint</td>
<td>70</td>
<td>0.748</td>
<td>14</td>
<td>34</td>
</tr>
<tr>
<td>edn</td>
<td>75</td>
<td>0.537</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>prime</td>
<td>118</td>
<td>4.368</td>
<td>22</td>
<td>43</td>
</tr>
<tr>
<td>compress</td>
<td>122</td>
<td>1.801</td>
<td>10</td>
<td>19</td>
</tr>
<tr>
<td>select</td>
<td>136</td>
<td>45.598</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>qsortexam</td>
<td>155</td>
<td>28.201</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>adpcm</td>
<td>323</td>
<td>0.074</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td><strong>LARGE BENCHMARKS (MERGING REQUIRED)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ud</td>
<td>153</td>
<td>17.477</td>
<td>6</td>
<td>23</td>
</tr>
<tr>
<td>minver</td>
<td>449</td>
<td>188.339</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>state mate</td>
<td>453</td>
<td>193.849</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>ludcmp</td>
<td>632</td>
<td>143.088</td>
<td>11</td>
<td>510</td>
</tr>
<tr>
<td>nsichneu</td>
<td>754</td>
<td>250.385</td>
<td>4586</td>
<td>8620</td>
</tr>
<tr>
<td>qurt</td>
<td>2777</td>
<td>773.805</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>lms</td>
<td>3098</td>
<td>915.434</td>
<td>259</td>
<td>2376</td>
</tr>
<tr>
<td>fft1</td>
<td>6123</td>
<td>2223.125</td>
<td>25</td>
<td>815</td>
</tr>
</tbody>
</table>
Closing Thoughts

+ Working directly on the binaries
Closing Thoughts

- Working directly on the binaries
- Outputting short infeasible paths
  - Reduces complexity of analyses that exploit infeasible paths
Closing Thoughts

+ Working directly on the binaries
+ Outputting short infeasible paths
  - Reduces complexity of analyses that exploit infeasible paths
  - Brutal state merging
Closing Thoughts

- Working directly on the binaries
- Outputting short infeasible paths
  - Reduces complexity of analyses that exploit infeasible paths
  - Brutal state merging
  - Time calculation explosion
Future works

- Program slicing

Looking for loop invariants

Experiment with other SMT theories than QF-LIA

Experimentation to estimate the impact on the WCET
Future works

- Program slicing
- Looking for loop invariants
Future works

- Program slicing
- Looking for loop invariants
- Experiment with other SMT theories than QF-LIA
Future works

- Program slicing
- Looking for loop invariants
- Experiment with other SMT theories than QF-LIA
- Experimentations to estimate the impact on the WCET
Questions?
### Mälardalen benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>BB (#)</th>
<th>Time (s)</th>
<th>Inf. paths found with minimization</th>
<th>w/o minimization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 edge</td>
<td>Minimized</td>
</tr>
<tr>
<td><strong>SMALL BENCHMARKS (NO MERGING REQUIRED)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ndes</td>
<td>57</td>
<td>0.267</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>expint</td>
<td>70</td>
<td>0.748</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>edn</td>
<td>75</td>
<td>0.537</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>prime</td>
<td>118</td>
<td>4.368</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>compress</td>
<td>122</td>
<td>1.801</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>select</td>
<td>136</td>
<td>45.598</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>qsortexam</td>
<td>155</td>
<td>28.201</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>adpcm</td>
<td>323</td>
<td>0.074</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td><strong>LARGE BENCHMARKS (MERGING REQUIRED)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ud</td>
<td>153</td>
<td>17.477</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>minver</td>
<td>449</td>
<td>188.339</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>statemate</td>
<td>453</td>
<td>193.849</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>ludcmp</td>
<td>632</td>
<td>143.088</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>nsichneu</td>
<td>754</td>
<td>250.385</td>
<td>0</td>
<td>1352</td>
</tr>
<tr>
<td>qurt</td>
<td>2777</td>
<td>773.805</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>lms</td>
<td>3098</td>
<td>915.434</td>
<td>26</td>
<td>22</td>
</tr>
<tr>
<td>fft1</td>
<td>6123</td>
<td>2223.125</td>
<td>0</td>
<td>25</td>
</tr>
</tbody>
</table>