Formal Verification of a WCET Estimation Tool

Sandrine Blazy\textsuperscript{1}, André Maroneze\textsuperscript{1}, David Pichardie\textsuperscript{2}, Isabelle Puaut\textsuperscript{1}

\textsuperscript{1} University of Rennes 1 – France
\textsuperscript{2} ENS Rennes, France

08/07/2014
Formal methods in industry

Formal methods increasingly applied in industry

Formal verification

- Machine-checked proofs
- Specifications → executable code

Useful for industrial-size applications

- Examples: seL4 (NICTA), CompCert (Inria)
Interactive proof assistants (e.g. ACL2, Coq, Isabelle)
- Logic specification language → properties & theorems
- Functional programming language → algorithms
- Interactive (step-by-step) proof construction
- Executable code generation
Coq general scheme

Specifications

Theorems

Proof scripts

Executable code

Proof verification

Coq Kernel

Trusted Computing Base
Current tools perform very sophisticated analyses
  Formal verification helps to better understand them
    (e.g. implicit assumptions, corner cases)
Idea: formally verify an existing WCET estimation method
  Integration within a compiler (CompCert)
Products of the verification
  Correctness theorem for WCET estimation
  Verified tool (+ experimental evaluation)
1. Architecture of our formalized tool
2. Formalization approach
3. Experimental evaluation
4. Conclusion and future work
WCET estimation tool architecture [Wilhelm08]
Architecture of the formalized WCET estimation tool

- Program
- Control flow analysis
- Hardware Features
- Estimate Computation
- WCET Estimation

WCET estimation tool
Architecture of the formalized WCET estimation tool

Program → Control flow analysis → Loop bounds → Estimate Computation → WCET Estimation

WCET estimation tool
Overview

Architecture of the formalized WCET estimation tool

- C Program
- CompCert C Compiler
- Compilation
- Control flow analysis
- Loop bounds
- Estimate Computation
- WCET estimation tool
- WCET Estimation
CompCert

- Moderately optimizing, formally verified C compiler
  - Several intermediate languages
    - E.g. RTL → data-flow analyses/optimizations
CompCert

- Moderately optimizing, formally verified C compiler
  - Several intermediate languages
    - E.g. RTL → data-flow analyses/optimizations
- Semantic preservation theorem
  - Proof that *compilation preserves program behavior*
Architecture of the formalized WCET estimation tool
Architecture of the formalized WCET estimation tool

C → RTL → ASM

Control flow analysis → Loop bounds

Loop bound estimation → Estimate Computation → WCET Estimation
Overview

Architecture of the formalized WCET estimation tool

- Based on one of the methods used by SWEET
- Combination of reusable techniques
Loop bound estimation

- **Program slicing**
  - Simplifies the program while preserving loop iterations
  - Improves the *precision* of the estimation
- **Value analysis by abstract interpretation**
  - Safe over-approximation of variable values
  - Intervals of machine integers (32-bit)
- **Bound computation**
  - Variable values → local bounds → nested loops → global bounds
Architecture of the formalized WCET estimation tool

1. C → RTL
2. RTL → ASM
3. Control flow analysis → Loop bounds
4. Loop bounds → Estimate Computation
5. Estimate Computation → WCET Estimation

Loop bound estimation:
- Program slicing
- Value analysis
- Bound computation

Implicit Path Enumeration Technique (IPET)
Outline

1. Architecture of our formalized tool
2. Formalization approach
3. Experimental evaluation
4. Conclusion and future work
Formal verification approach

- Specify a formal semantics

- Define correctness theorems
  - Using the formal semantics

- Perform the proof

\[ \sigma = \langle \ell, E, cs \rangle \]
\[ \sigma \rightarrow \sigma' \]

**Theorem (Bound correctness)**

Let \( P \) be a program s.t. \(...\)

\( \ldots \text{then } cs(\ell) \leq \text{bound}(\ell). \)

**Lemma correct_bounds:**

\[
\text{forall } P, \sigma, \quad (\text{reaches } P, \sigma) \rightarrow \ldots
\]

**Proof.**

- intros \( P, \sigma \).
- induction reaches.

\[
\ldots
\]

Qed.
Formal RTL semantics (simplified)

⇒ Defined in CompCert
⇒ Standard small-step semantics

CompCert’s RTL semantics

- Program state: \( \sigma = < \ell, E > \)
  - \( \ell \): program point (node in the CFG)
  - \( E \): environment (maps variables to values)
- Execution step relation
  \( P \vdash < \ell, E > \rightarrow < \ell', E' > \)
- Reachable state
  \( \sigma \in \text{reach}(P) \iff \sigma_0 \rightarrow^* \sigma \)
- Execution trace: \( tr = [\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma] \)
  List of reachable states
Adapting the RTL semantics

Addition of execution counters

<table>
<thead>
<tr>
<th>Modified RTL semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program state: $\sigma = &lt;\ell, E, cs&gt;$</td>
</tr>
<tr>
<td>$cs$ : counters (node $\mapsto \mathbb{N}$)</td>
</tr>
<tr>
<td>Execution step relation</td>
</tr>
<tr>
<td>$P \vdash &lt;\ell, E, cs&gt; \rightarrow &lt;\ell', E', cs'&gt;$</td>
</tr>
<tr>
<td>Counters incremented at each step</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{<a,E,\{#a$\mapsto$0,#b$\mapsto$0}\>} & \rightarrow \text{<b,E',\{#a$\mapsto$1,#b$\mapsto$0\>}
\end{align*}
\]
Theorem (Bound correctness)

Let $P$ be a program such that $P$’s execution terminates with counters $cs$, and let $\ell$ be a program point in $P$. Then $cs(\ell) \leq \text{bound}(P)(\ell)$.

$\text{bound}$: function computing the loop bound estimation

E.g. \( \text{bound}(P)(\ell) = \text{bounds}(\text{value}(\text{slice}(P, \ell))) \)
Let $P$ be a program such that $P$’s execution terminates with counters $cs$, and let $\ell$ be a program point in $P$. Then $cs(\ell) \leq \text{bound}(P)(\ell)$.

**bound**: function computing the loop bound estimation

E.g. $\text{bound}(P)(\ell) = \text{bounds}(\text{value}((\text{slice}(P, \ell)))$)

Informally:
For every terminating execution of program $P$,
Theorem (Bound correctness)

Let $P$ be a program such that $P$'s execution terminates with counters $cs$, and let $\ell$ be a program point in $P$. Then $cs(\ell) \leq \text{bound}(P)(\ell)$.

$\text{bound}$: function computing the loop bound estimation
E.g. $\text{bound}(P)(\ell) = \text{bounds}(\text{value}(\text{slice}(P, \ell)))$

Informally:
For every terminating execution of program $P$, the actual execution counters of any program point $\ell$
Theorem (Bound correctness)

Let $P$ be a program such that $P$'s execution terminates with counters $cs$, and let $\ell$ be a program point in $P$. Then $cs(\ell) \leq \text{bound}(P)(\ell)$.

**bound**: function computing the loop bound estimation

**E.g.** $\text{bound}(P)(\ell) = \text{bounds}(\text{value}(\text{slice}(P, \ell)))$

**Informally:**

For every terminating execution of program $P$, the *actual* execution counters of any program point $\ell$ are overestimated by the result of the loop bound estimation.
Theorem (Bound correctness)

Let $P$ be a program such that $P$'s execution terminates with counters $cs$, and let $\ell$ be a program point in $P$. Then $cs(\ell) \leq \text{bound}(P)(\ell)$.

-bound: function computing the loop bound estimation
E.g. $\text{bound}(P)(\ell) = \text{bounds}(\text{value}(\text{slice}(P, \ell)))$

Informally:
For every terminating execution of program $P$, the actual execution counters of any program point $\ell$ are overestimated by the result of the loop bound estimation.
**Correctness theorem**

RTL bounds → ASM bounds

**Start-to-end-correctness** theorem

- Uses CompCert’s annotations + semantic preservation theorem

```c
int i = 0;
while (i < 5) {
    _annot("loop");
    i++;
}
```

```asm
1: x1 = 0
2: if (x1 >= 5) goto 6
3: x2 = builtin annot "loop"
4: x1 = x1 + 1
5: goto 2
6:
```

```assembly
.L100:
    cmpwi 0, 4, 5
    bf 0, .L101
    # annotation: loop
    addi 4, 4, 1
    b .L100
.L101:
```

C ➔ RTL ➔ Assembly
Overview of the formalized WCET estimation tool

Control flow analysis

Loop bound estimation

- Program slicing
- Value analysis
- Bound computation

Loop bounds

Estimate Computation

WCET Estimation
Proof techniques

Complementary techniques

- Direct proof
  - Specify and formalize the algorithm
  - A posteriori, verified validation

- Correctness ensured for a single input (runtime cost)
Proving program slicing correctness

- Efficient program slicing $\rightarrow$ imperative data structures
  - E.g. program dependency graph
  - $\Rightarrow$ Complex proof

- Validation $\rightarrow$ decouples algorithm and proof

Proof strategy
- Define and prove relation between original and sliced programs
- Code an efficient validator which checks it
Loop bound estimation

Repeat steps for remaining components, then compose the proofs
Estimate computation

- C → RTL → ASM
- Control flow analysis → Loop bounds → Estimate Computation → WCET Estimation
- Loop bound estimation
  - Program slicing
  - Value analysis
  - Bound computation
  - Implicit Path Enumeration Technique (IPET)
Implicit Path Enumeration Technique [Malik95]

Control flow → linear programming (LP) system

- Represent execution counters for CFG nodes and edges with variables $x_i$ and $e_{i,j}$

Entry/exit constraints
$$x_{entry} = 1 \quad x_{exit} = 1$$

Flow constraints ($\sum e_{in} = x_i = \sum e_{out}$)
$$e_{entry,1} + e_{5,1} = x_1 = e_{1,exit} + e_{1,2}$$

Loop constraints (derived from loop bounds)
$$x_1 \leq 6 \quad \leftarrow \text{loop bound estimation theorem}$$

WCET estimate: \( \max (\sum x_i.t_i) = 21 \text{ instructions} \)

→ Here, $t_i = 1$ (hardware cost coefficient)
IPET correctness and proof

Approach similar to RTL: ASM semantics + counters
  - $X(i) \rightarrow$ nodes  \hspace{1cm} $E(i,j) \rightarrow$ edges

Correctness: (actual WCET) $\leq$ (WCET estimate)

Algorithm + proof

1. LP generation $\rightarrow$ direct proof
2. External (non-verified) LP solver
3. LP validation $\rightarrow$ based on *Farkas certificates*
Outline

1. Architecture of our formalized tool
2. Formalization approach
3. Experimental evaluation
4. Conclusion and future work
Why to evaluate?

- Proof $\rightarrow$ ensure *correctness*
- Evaluation $\rightarrow$ measure *precision*

$\Rightarrow$ Objective: check whether results are practically useful

- Evaluated on the Mälardalen benchmarks
  - Loop bound estimation
  - Value analysis
  - WCET estimation

$\Rightarrow$ Compiler integration $\rightarrow$ transformations for improved precision
Results of the WCET estimation

Comparison: WCET estimate vs. exact WCET
→ ASM emulator + known worst-case input
Experimental evaluation

WCET estimation

Results of the WCET estimation

Comparison: WCET estimate vs. exact WCET

→ ASM emulator + known worst-case input

<table>
<thead>
<tr>
<th>Program</th>
<th>Standard Overestimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnt</td>
<td>18.3% ✔</td>
</tr>
<tr>
<td>cover</td>
<td>10.9% ✔</td>
</tr>
<tr>
<td>crc</td>
<td>100.2%</td>
</tr>
<tr>
<td>edn</td>
<td>141.5%</td>
</tr>
<tr>
<td>expint</td>
<td>2601.6%</td>
</tr>
<tr>
<td>fdct</td>
<td>0.0% ✔</td>
</tr>
<tr>
<td>fibcall</td>
<td>0.9% ✔</td>
</tr>
<tr>
<td>jfdctint</td>
<td>0.0% ✔</td>
</tr>
<tr>
<td>lcdnum</td>
<td>50.9%</td>
</tr>
<tr>
<td>matmult</td>
<td>11.5% ✔</td>
</tr>
<tr>
<td>ndes</td>
<td>12.2% ✔</td>
</tr>
<tr>
<td>ns</td>
<td>88.3%</td>
</tr>
<tr>
<td>nsichneu</td>
<td>106.1%</td>
</tr>
<tr>
<td>qurt</td>
<td>168.2%</td>
</tr>
<tr>
<td>ud</td>
<td>225.1%</td>
</tr>
</tbody>
</table>

✔: overestimation < 10%  ✔: 10% ≤ overestimation < 20%
Experimental evaluation

WCET estimation

Results of the WCET estimation

CompCert integration → useful transformations for WCET estimation

- **Loop inversion** (for / while → do-while)

<table>
<thead>
<tr>
<th>Program</th>
<th>Standard Overestimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnt</td>
<td>18.3% ✓</td>
</tr>
<tr>
<td>cover</td>
<td>10.9% ✓</td>
</tr>
<tr>
<td>crc</td>
<td>100.2%</td>
</tr>
<tr>
<td>edn</td>
<td>141.5%</td>
</tr>
<tr>
<td>expint</td>
<td>2601.6%</td>
</tr>
<tr>
<td>fdct</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>fibcall</td>
<td>0.9% ✓</td>
</tr>
<tr>
<td>jfdctint</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>lcdnum</td>
<td>50.9%</td>
</tr>
<tr>
<td>matmult</td>
<td>11.5% ✓</td>
</tr>
<tr>
<td>ndes</td>
<td>12.2% ✓</td>
</tr>
<tr>
<td>ns</td>
<td>88.3%</td>
</tr>
<tr>
<td>nsichneu</td>
<td>106.1%</td>
</tr>
<tr>
<td>qurt</td>
<td>168.2%</td>
</tr>
<tr>
<td>ud</td>
<td>225.1%</td>
</tr>
</tbody>
</table>

✓: overestimation < 10%  ✔: 10% ≤ overestimation < 20%
CompCert integration → useful transformations for WCET estimation

- **Loop inversion** (\(for/while \rightarrow do-while\))

<table>
<thead>
<tr>
<th>Program</th>
<th>Standard Overestimation</th>
<th>Loop Inversion Overestimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnt</td>
<td>18.3% ✓</td>
<td>2.8% ✓</td>
</tr>
<tr>
<td>cover</td>
<td>10.9% ✓</td>
<td>11.5% ✓</td>
</tr>
<tr>
<td>crc</td>
<td>100.2%</td>
<td>99.5%</td>
</tr>
<tr>
<td>edn</td>
<td>141.5%</td>
<td>110.4%</td>
</tr>
<tr>
<td>expint</td>
<td>2601.6%</td>
<td>2419.7%</td>
</tr>
<tr>
<td>fdct</td>
<td>0.0% ✓</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>fibcall</td>
<td>0.9% ✓</td>
<td>1.1% ✓</td>
</tr>
<tr>
<td>jfdctint</td>
<td>0.0% ✓</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>lcdnum</td>
<td>50.9%</td>
<td>55.2%</td>
</tr>
<tr>
<td>matmult</td>
<td>11.5% ✓</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>ndes</td>
<td>12.2% ✓</td>
<td>3.6% ✓</td>
</tr>
<tr>
<td>ns</td>
<td>88.3%</td>
<td>0.2% ✓</td>
</tr>
<tr>
<td>nsichneu</td>
<td>106.1%</td>
<td>106.1%</td>
</tr>
<tr>
<td>qurt</td>
<td>168.2%</td>
<td>165.7%</td>
</tr>
<tr>
<td>ud</td>
<td>225.1%</td>
<td>217.3%</td>
</tr>
</tbody>
</table>

✓: overestimation \(< 10\%\)          ✓: 10\% \(\leq\) overestimation \(< 20\%\)
## Results of the WCET estimation

### CompCert integration → useful transformations for WCET estimation

#### Loop unrolling

<table>
<thead>
<tr>
<th>Program</th>
<th>Standard Overestimation</th>
<th>Loop Inversion Overestimation</th>
<th>Inversion+Unrolling Overestimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnt</td>
<td>18.3% ✓</td>
<td>2.8% ✓</td>
<td>3.3% ✓</td>
</tr>
<tr>
<td>cover</td>
<td>10.9% ✓</td>
<td>11.5% ✓</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>crc</td>
<td>100.2%</td>
<td>99.5%</td>
<td>99.2%</td>
</tr>
<tr>
<td>edn</td>
<td>141.5%</td>
<td>110.4%</td>
<td>110.4%</td>
</tr>
<tr>
<td>expint</td>
<td>2601.6%</td>
<td>2419.7%</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>fdct</td>
<td>0.0% ✓</td>
<td>0.0% ✓</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>fibcall</td>
<td>0.9% ✓</td>
<td>1.1% ✓</td>
<td>1.1% ✓</td>
</tr>
<tr>
<td>jfdctint</td>
<td>0.0% ✓</td>
<td>0.0% ✓</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>lcdnum</td>
<td>50.9%</td>
<td>55.2%</td>
<td>11.9% ✓</td>
</tr>
<tr>
<td>matmult</td>
<td>11.5% ✓</td>
<td>0.0% ✓</td>
<td>0.0% ✓</td>
</tr>
<tr>
<td>ndes</td>
<td>12.2% ✓</td>
<td>3.6% ✓</td>
<td>3.6% ✓</td>
</tr>
<tr>
<td>ns</td>
<td>88.3%</td>
<td>0.2% ✓</td>
<td>0.2% ✓</td>
</tr>
<tr>
<td>nsichneu</td>
<td>106.1%</td>
<td>106.1%</td>
<td>106.3%</td>
</tr>
<tr>
<td>qurt</td>
<td>168.2%</td>
<td>165.7%</td>
<td>215.2%</td>
</tr>
<tr>
<td>ud</td>
<td>225.1%</td>
<td>217.3%</td>
<td>265.2%</td>
</tr>
</tbody>
</table>

✓: overestimation < 10%  
✓: 10% ≤ overestimation < 20%
Experimental evaluation

WCET estimation

Results of the WCET estimation

**CompCert integration → useful transformations for WCET estimation**

- ✓ Precision improvements with little proof overhead

<table>
<thead>
<tr>
<th>Program</th>
<th>Standard Overestimation</th>
<th>Loop Inversion Overestimation</th>
<th>Inversion+Unrolling Overestimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnt</td>
<td>18.3% ✔</td>
<td>2.8% ✔</td>
<td>3.3% ✔</td>
</tr>
<tr>
<td>cover</td>
<td>10.9% ✔</td>
<td>11.5% ✔</td>
<td>0.0% ✔</td>
</tr>
<tr>
<td>crc</td>
<td>100.2%</td>
<td>99.5%</td>
<td>99.2%</td>
</tr>
<tr>
<td>edn</td>
<td>141.5%</td>
<td>110.4%</td>
<td>110.4%</td>
</tr>
<tr>
<td>expint</td>
<td>2601.6%</td>
<td>2419.7%</td>
<td>0.0% ✔</td>
</tr>
<tr>
<td>fdct</td>
<td>0.0% ✔</td>
<td>0.0% ✔</td>
<td>0.0% ✔</td>
</tr>
<tr>
<td>fibcall</td>
<td>0.9% ✔</td>
<td>1.1% ✔</td>
<td>1.1% ✔</td>
</tr>
<tr>
<td>jfdctint</td>
<td>0.0% ✔</td>
<td>0.0% ✔</td>
<td>0.0% ✔</td>
</tr>
<tr>
<td>lcdnum</td>
<td>50.9%</td>
<td>55.2%</td>
<td>11.9% ✔</td>
</tr>
<tr>
<td>matmult</td>
<td>11.5% ✔</td>
<td>0.0% ✔</td>
<td>0.0% ✔</td>
</tr>
<tr>
<td>ndes</td>
<td>12.2% ✔</td>
<td>3.6% ✔</td>
<td>3.6% ✔</td>
</tr>
<tr>
<td>ns</td>
<td>88.3%</td>
<td>0.2% ✔</td>
<td>0.2% ✔</td>
</tr>
<tr>
<td>nsichneu</td>
<td>106.1%</td>
<td>106.1%</td>
<td>106.3%</td>
</tr>
<tr>
<td>qurt</td>
<td>168.2%</td>
<td>165.7%</td>
<td>215.2%</td>
</tr>
<tr>
<td>ud</td>
<td>225.1%</td>
<td>217.3%</td>
<td>265.2%</td>
</tr>
</tbody>
</table>

 ✓: overestimation < 10%  ✔: 10% ≤ overestimation < 20%
Outline

1. Architecture of our formalized tool
2. Formalization approach
3. Experimental evaluation
4. Conclusion and future work
Conclusion and future work

Formal verification of WCET estimation is feasible
- Decomposition into several steps, composition of proofs
- Reuse of formal frameworks and semantics
- Direct proof + validation
Conclusion and future work

Formal verification of WCET estimation is feasible
- Decomposition into several steps, composition of proofs
- Reuse of formal frameworks and semantics
- Direct proof + validation

Formal guarantees combined with CompCert’s
- Absence of compilation errors + WCET estimation
Conclusion and future work

Formal verification of WCET estimation is feasible
- Decomposition into several steps, composition of proofs
- Reuse of formal frameworks and semantics
- Direct proof + validation

Formal guarantees combined with CompCert’s
- Absence of compilation errors + WCET estimation

Future work
- Formal hardware models with timing information
  - A more realistic WCET estimation
- Other WCET-related techniques (e.g. parametric WCET, WCC-style optimizations)
Final Trusted Computing Base

- CompCert C semantics
- ASM Semantics
- Hardware Model?

C Program

Verified Compiler + WCET Estimation Tool

- Program slicer
- Value analyzer
- LP Solver

ASM Program

WCET Estimation

Trusted Computing Base