Formal Verification of a WCET Estimation Tool

Sandrine Blazy[1], André Maroneze[1], David Pichardie[2], Isabelle Puaut[1]

[1] University of Rennes 1 – France

[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

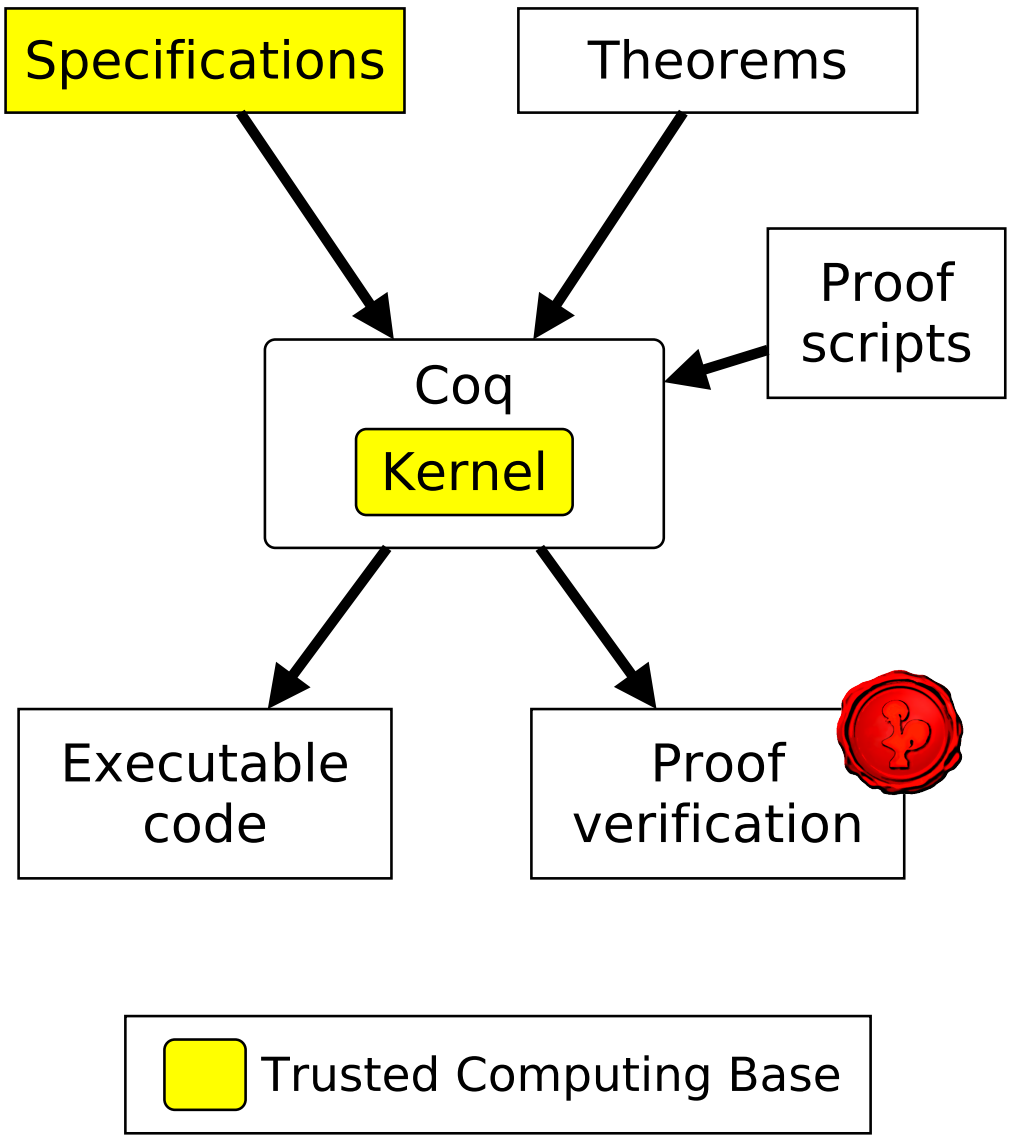
- Examples: seL4 (NICTA), CompCert (Inria)

Formal verification with proof assistants

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



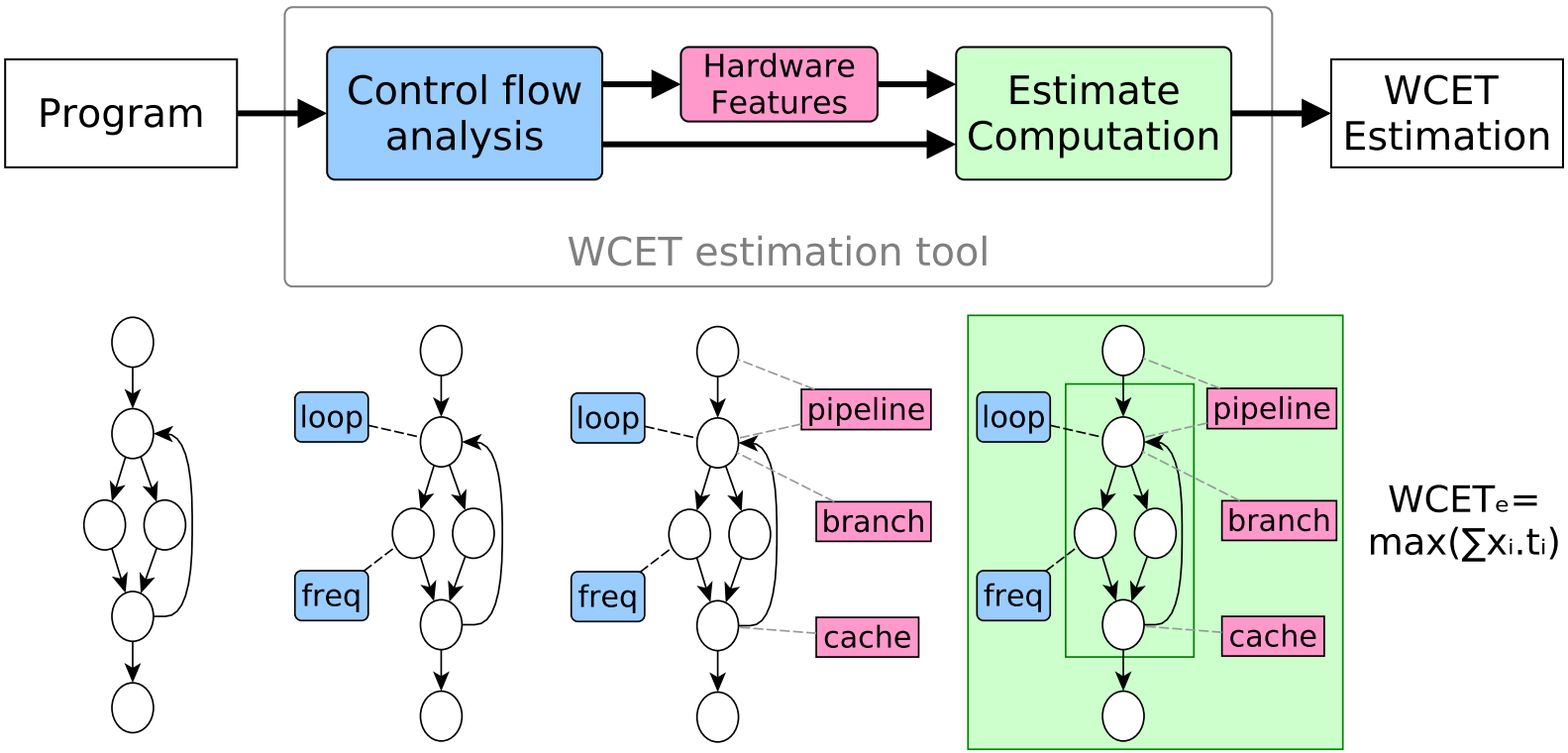
Formal verification for WCET estimation

- 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)

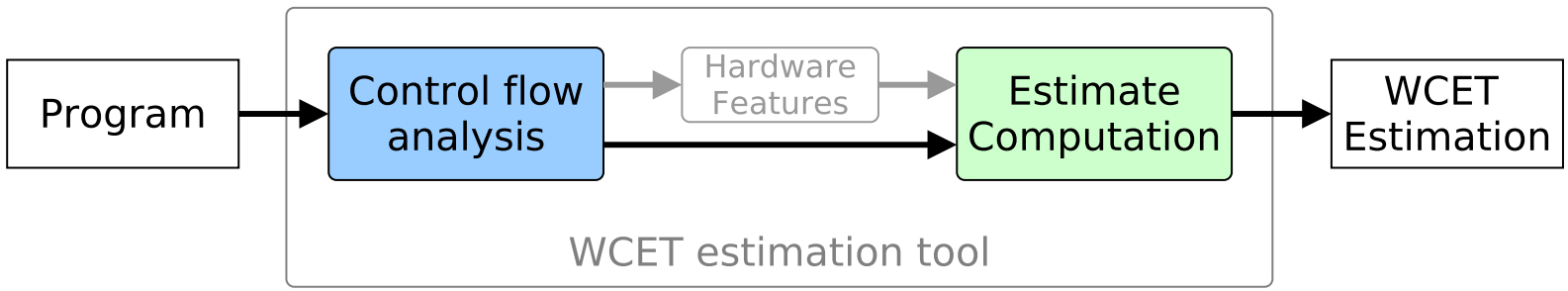
Outline

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

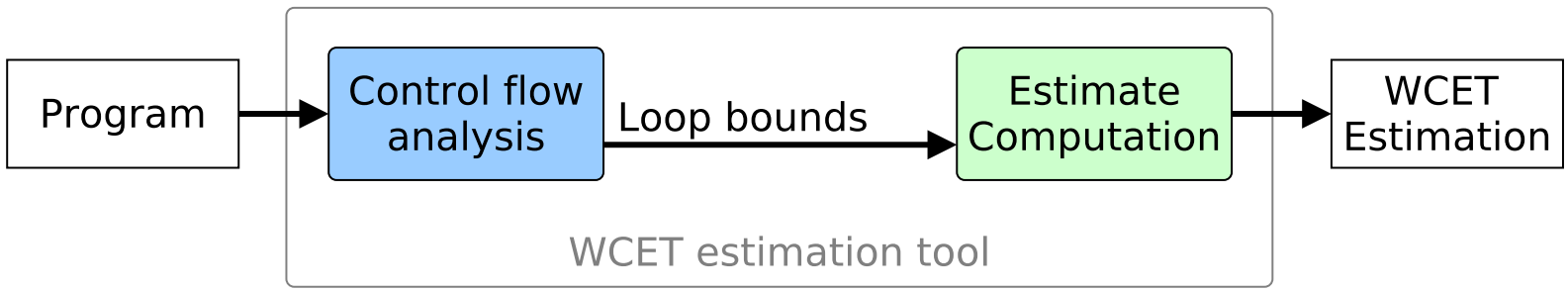
WCET estimation tool architecture [Wilhelm08]



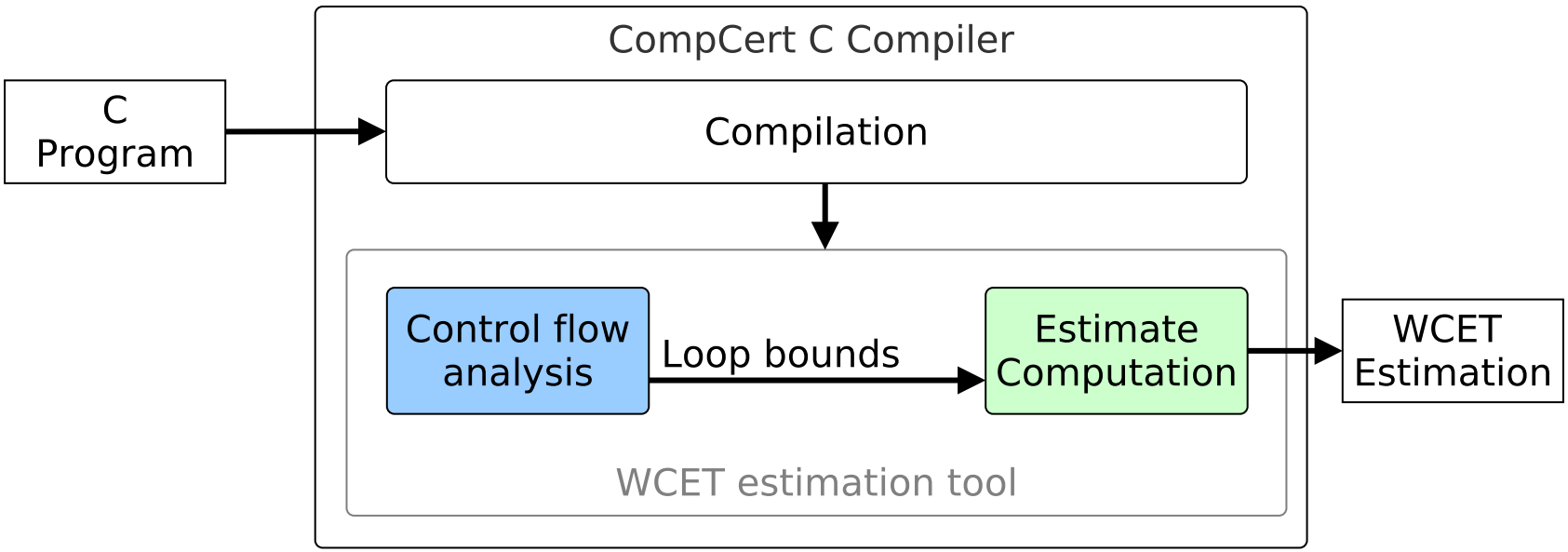
Architecture of the formalized WCET estimation tool



Architecture of the formalized WCET estimation tool



Architecture of the formalized WCET estimation tool

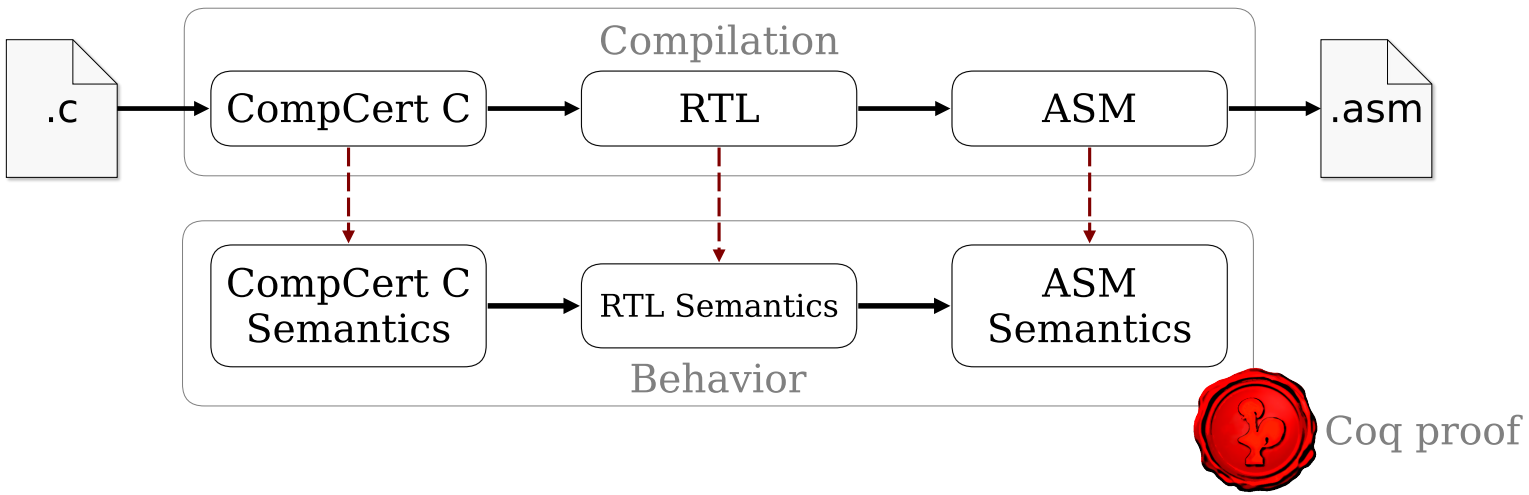


CompCert

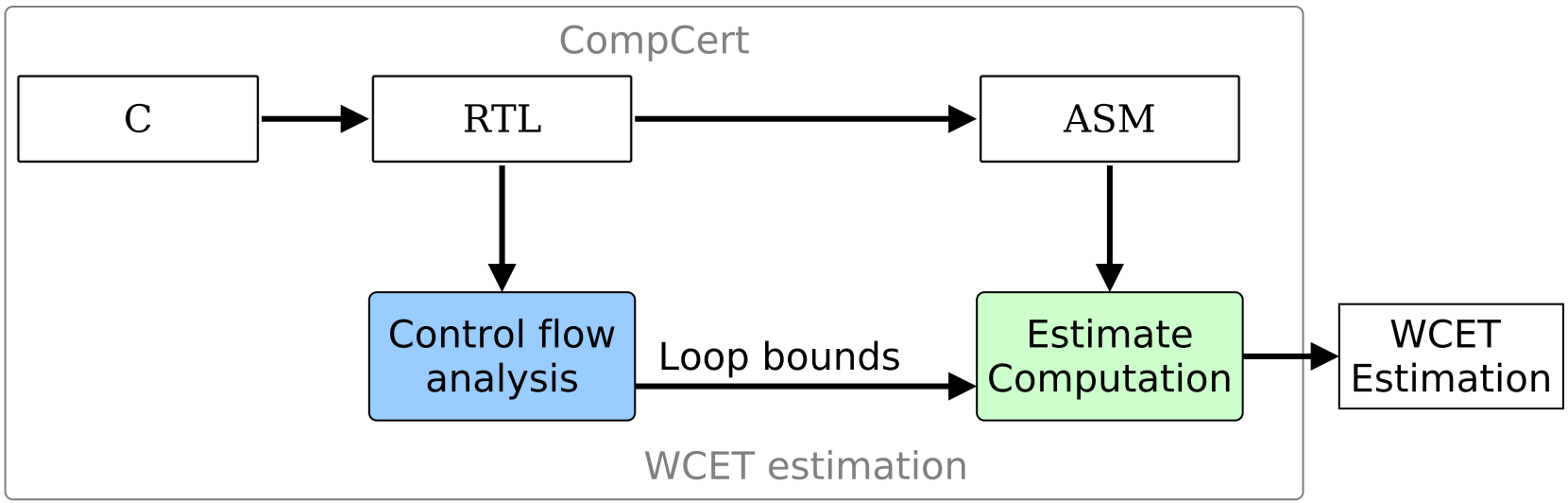
- Moderately optimizing, formally verified C compiler
  - Several intermediate languages
    - E.g. RTL → data-flow analyses/optimizations

CompCert

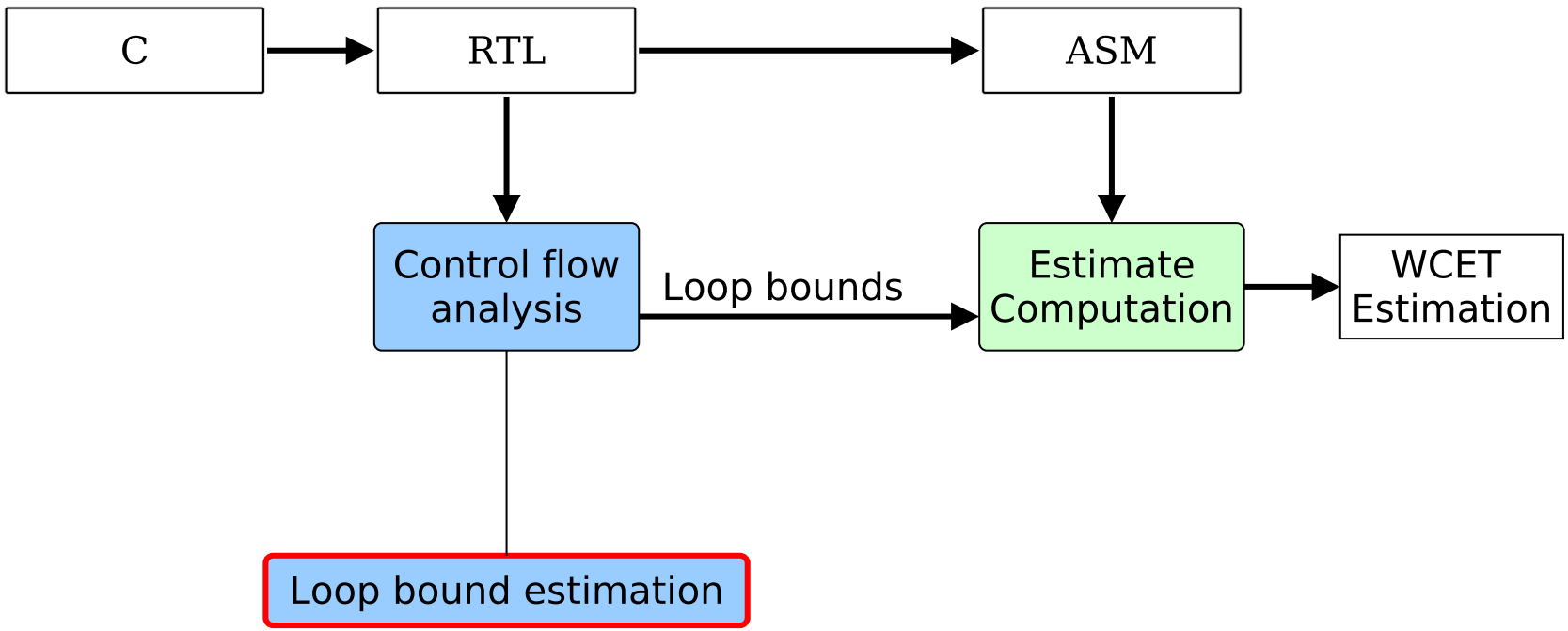
- Moderately optimizing, formally verified C compiler
  - Several intermediate languages
    - E.g. RTL → data-flow analyses/optimizations
- Semantic preservation theorem
  - Proof that *compilation preserves program behavior*



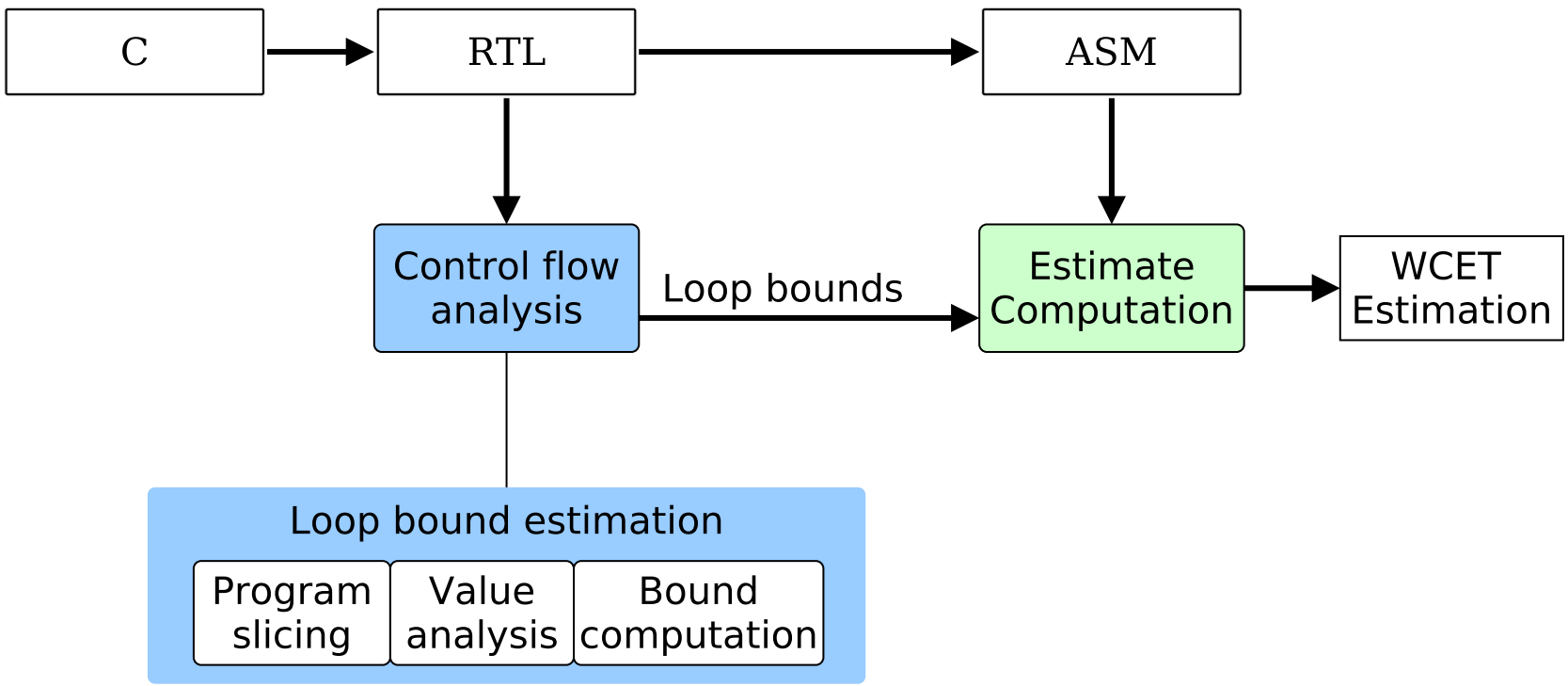
Architecture of the formalized WCET estimation tool



Architecture of the formalized WCET estimation tool

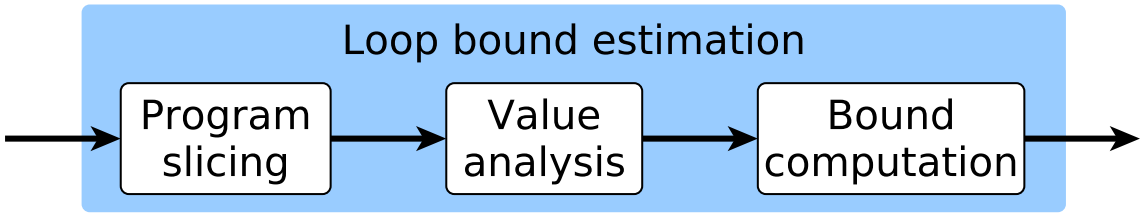


Architecture of the formalized WCET estimation tool



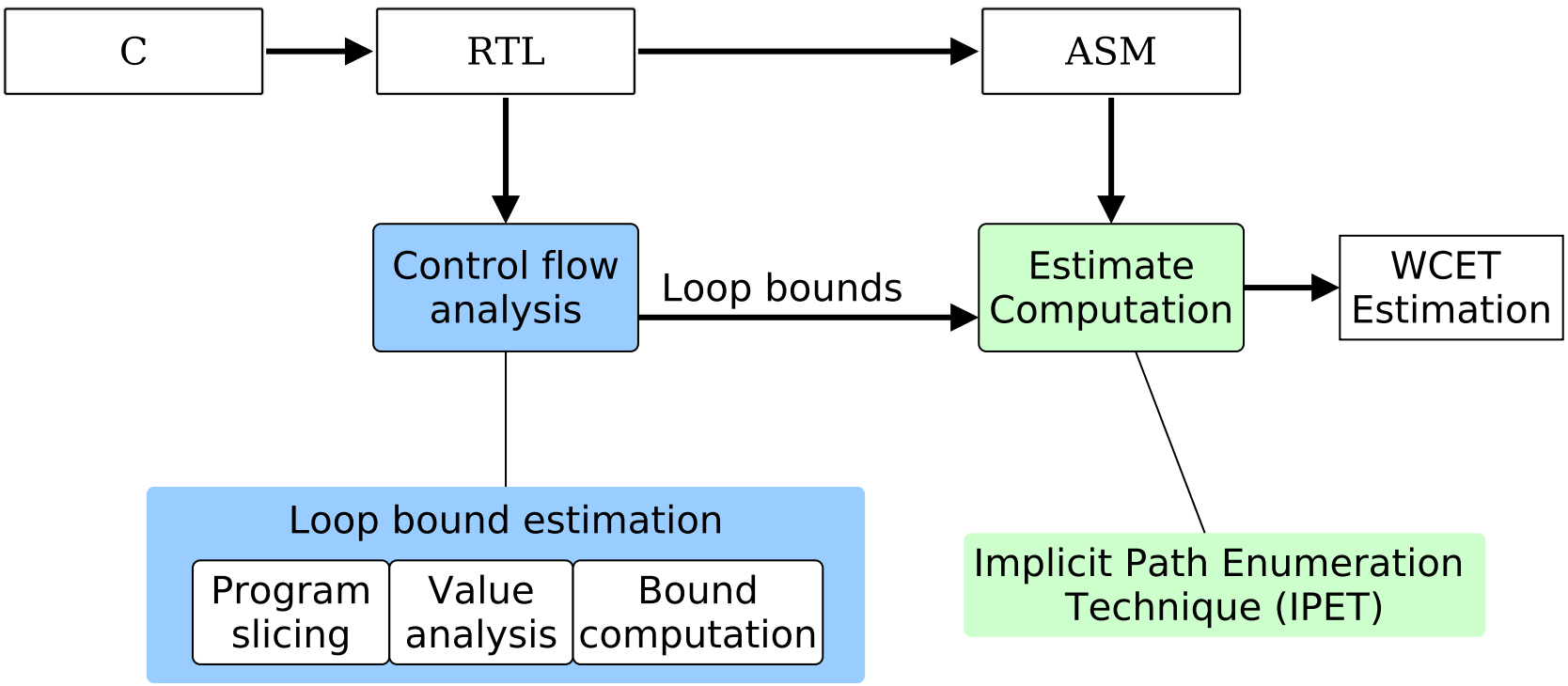
- Based on one of the methods used by SWEET
- Combination of reusable techniques

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



Outline

Formal verification approach

- Specify a formal semantics

$$\sigma =< \ell, E, cs >$$
$$\sigma \rightarrow \sigma'$$
$$\ldots$$

- Define correctness theorems
  - Using the formal semantics

Theorem (Bound correctness)

Let P be a program s.t. ...
... then $cs(\ell) \leq \texttt{bound}(\ell)$.

- Perform the proof

```
Lemma correct_bounds:
  forall P σ,
    (reaches P σ) → ...
Proof.
  intros P σ.
  induction reaches.
  ...
Qed.
```

Formal RTL semantics (simplified)

⇒ Defined in CompCert

⇒ Standard small-step semantics

CompCert's RTL semantics

- Program state: $\sigma = <\ell, E>$

  ℓ: program point (node in the CFG)

  E: environment (maps variables to values)

- Execution step relation

  $P \vdash <\ell, E> \longrightarrow <\ell', E'>$

- Reachable state

  $\sigma \in \mathit{reach}(P) \iff \sigma_0 \longrightarrow^* \sigma$

- Execution trace: $tr = [\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma]$

  List of reachable states

Adapting the RTL semantics

Addition of execution counters

Modified RTL semantics

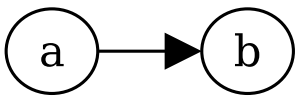
- Program state: $\sigma = <\ell, E, \mathbf{cs}>$

  cs : counters (node $\mapsto \mathbb{N}$)

- Execution step relation

  $P \vdash <\ell, E, cs> \longrightarrow <\ell', E', cs'>$

  - Counters incremented at each step



$<a,E,\{\#a\mapsto 0,\#b\mapsto 0\}> \longrightarrow <b,E',\{\#a\mapsto 1,\#b\mapsto 0\}>$

Correctness theorem

Theorem (Bound correctness)

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

`bound`: function computing the loop bound estimation
E.g. `bound(P)(`ℓ`) = bounds(value(slice(P,`ℓ`)))`

Correctness theorem

Theorem (Bound correctness)

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

`bound`: function computing the loop bound estimation
E.g. `bound(P)(`ℓ`) = bounds(value(slice(P,`ℓ`)))`

Informally:
For every terminating execution of program P,

Correctness theorem

Theorem (Bound correctness)

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

`bound`: function computing the loop bound estimation
E.g. `bound(P)(`ℓ`) = bounds(value(slice(P,`ℓ`)))`

Informally:
For every terminating execution of program P,
the *actual* execution counters of any program point ℓ

Correctness theorem

Theorem (Bound correctness)

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

`bound`: function computing the loop bound estimation
E.g. `bound(P)(`ℓ`) = bounds(value(slice(P,`ℓ`)))`

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

Correctness theorem

Theorem (Bound correctness)

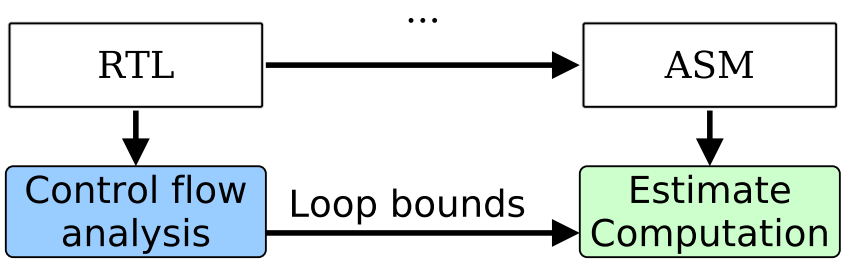
Let P be a program such that P's execution terminates with counters cs, and let ℓ be a program point in P.
Then $cs(\ell) \leq$ `bound`$(P)(\ell)$.

`bound`: function computing the loop bound estimation
E.g. `bound(P)(`ℓ`) = bounds(value(slice(P,`ℓ`)))`

Informally:
For every terminating execution of program P,
the *actual* execution counters of any program point ℓ
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

```
int i = 0;

while (i < 5) {

  __annot("loop");
  i++;
}
```

C

→

```
1: x1 = 0

2: if (x1 >=s 5) goto 6

3: x2 = builtin annot "loop"
4: x1 = x1 + 1
5: goto 2
6:
```

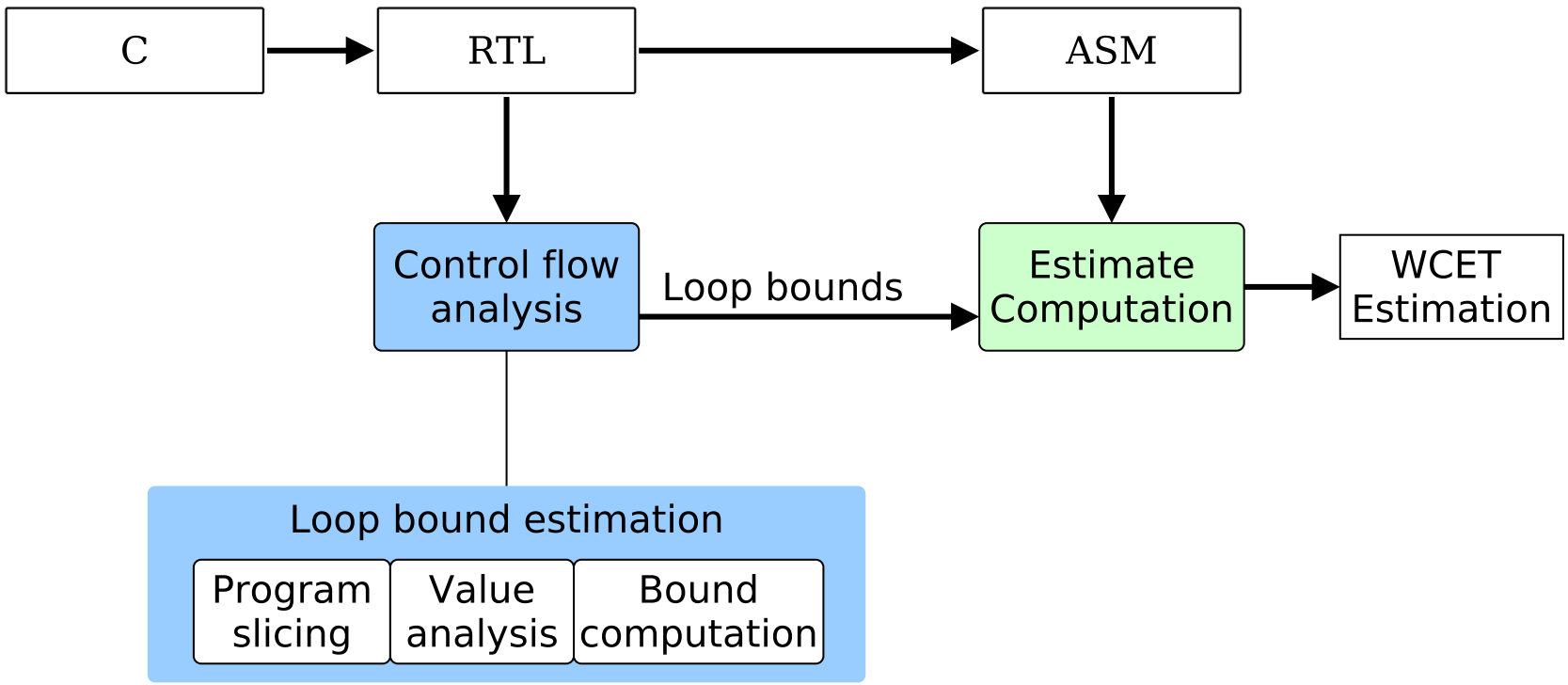
RTL

→

```
    stw 0, 8(1)
.L100:
    cmpwi 0, 4, 5
    bf 0, .L101
# annotation:  loop
    addi 4, 4, 1
    b .L100
.L101:
```

Assembly

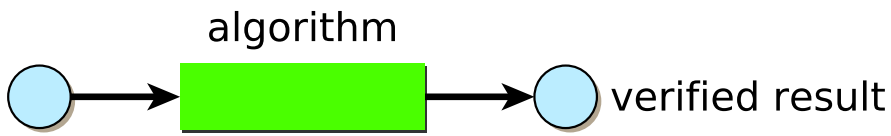
Overview of the formalized WCET estimation tool



Proof techniques

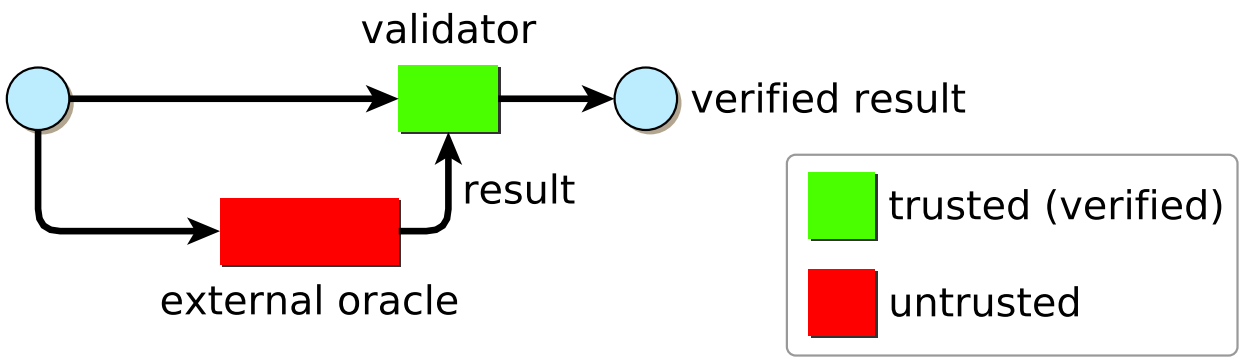
Complementary techniques

- Direct proof



  - Specify and formalize the algorithm

- *A posteriori*, verified validation

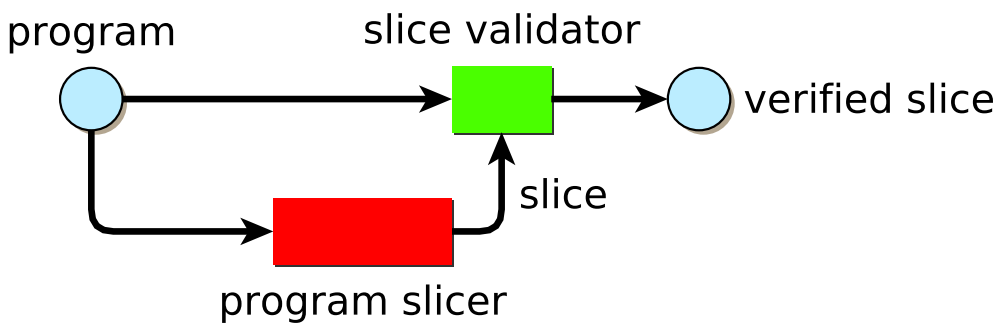


  - Correctness ensured for a single input (runtime cost)

Proving program slicing correctness

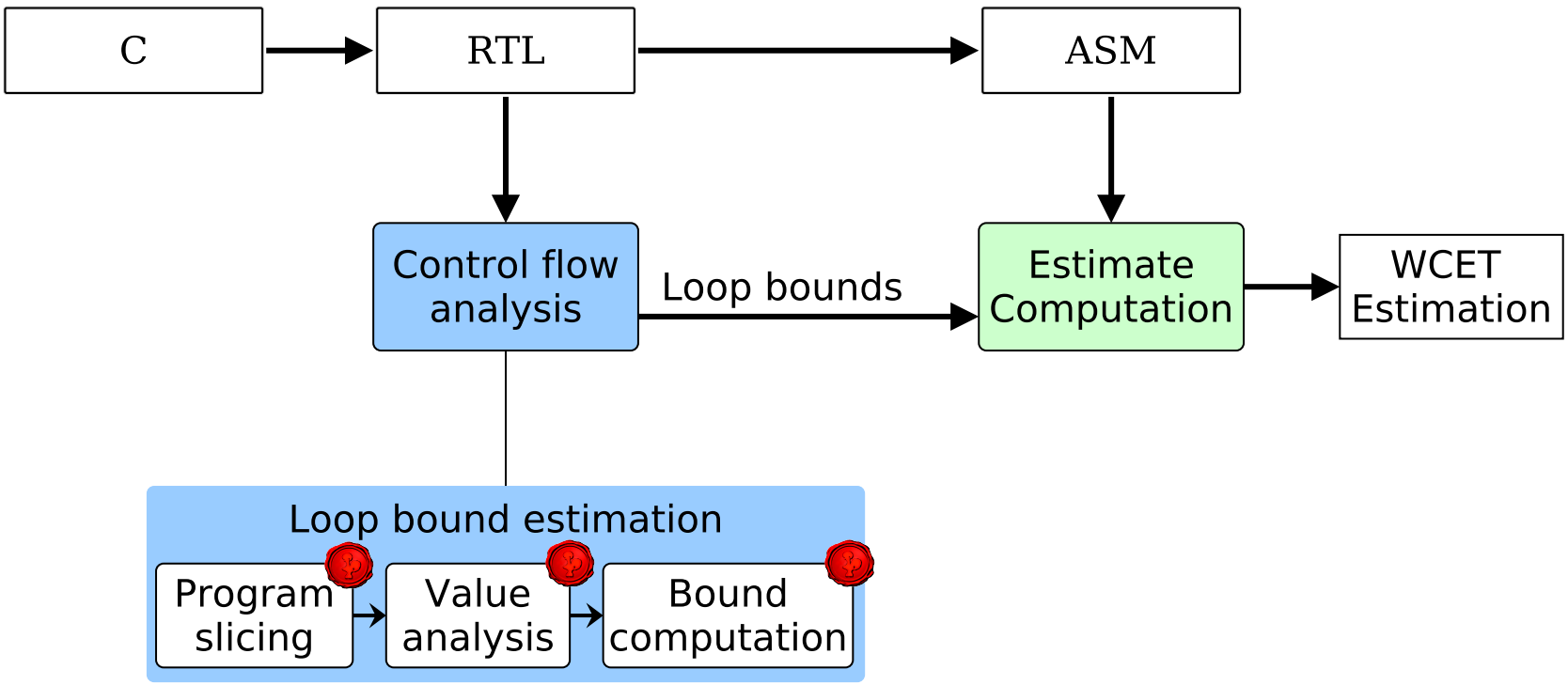
- Efficient program slicing → imperative data structures
  - E.g. program dependency graph

  ⇒ Complex proof
- Validation → decouples algorithm and proof



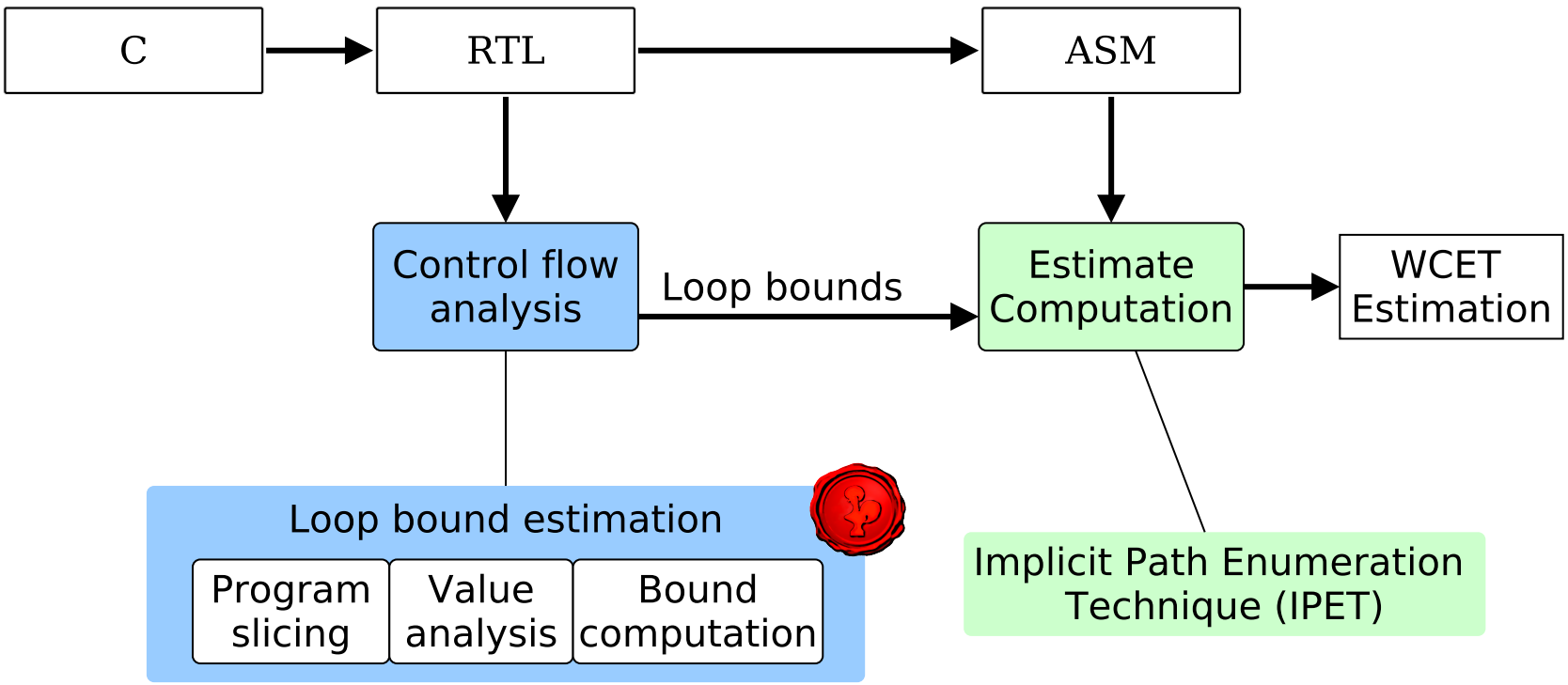
- Proof strategy
  - Define and prove relation between original and sliced programs
  - Code an efficient validator which checks it

Loop bound estimation



Repeat steps for remaining components, then compose the proofs

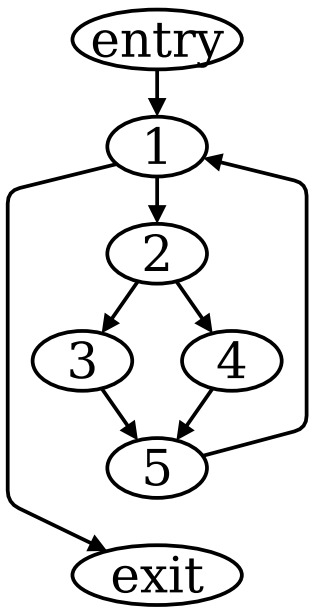
Estimate computation



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

$x_{entry} = 1 \qquad x_{exit} = 1$

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)

$x_1 \leq 6$ ← loop bound estimation theorem

WCET estimate: $\max(\sum x_i.t_i) = 21$ instructions

→ Here, $t_i = 1$ (hardware cost coefficient)

IPET correctness and proof

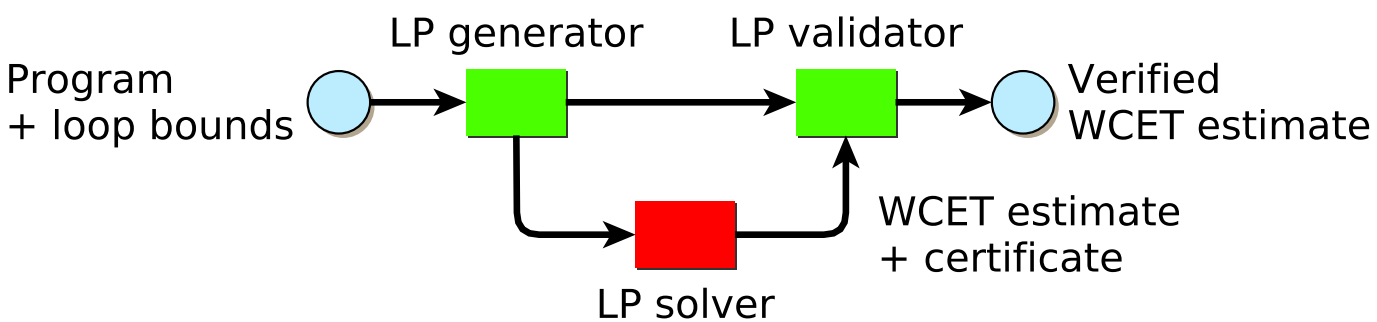
Approach similar to RTL: ASM semantics + counters

- $X(i) \rightarrow$ nodes $\quad E(i,j) \rightarrow$ 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

Why to evaluate?

- Proof → ensure *correctness*
- Evaluation → measure *precision*

⇒ Objective: check whether results are practically useful

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

⇒ Compiler integration → transformations for improved precision

Results of the WCET estimation

Comparison: WCET estimate vs. *exact* WCET

→ ASM emulator + known worst-case input

Results of the WCET estimation

Comparison: WCET estimate vs. *exact* WCET

→ ASM emulator + known worst-case input

	Standard	
Program	Overestimation	
cnt	18.3%	✓
cover	10.9%	✓
crc	100.2%	
edn	141.5%	
expint	2601.6%	
fdct	0.0%	✔
fibcall	0.9%	✔
jfdctint	0.0%	✔
lcdnum	50.9%	
matmult	11.5%	✓
ndes	12.2%	✓
ns	88.3%	
nsichneu	106.1%	
qurt	168.2%	
ud	225.1%	

✔: overestimation $< 10\%$ ✓: $10\% \leq$ overestimation $< 20\%$

Results of the WCET estimation

CompCert integration → useful transformations for WCET estimation

- Loop inversion (`for` / `while` ⟶ `do-while`)

	Standard
Program	Overestimation
cnt	18.3% ✓
cover	10.9% ✓
crc	100.2%
edn	141.5%
expint	2601.6%
fdct	0.0% ✔
fibcall	0.9% ✔
jfdctint	0.0% ✔
lcdnum	50.9%
matmult	11.5% ✓
ndes	12.2% ✓
ns	88.3%
nsichneu	106.1%
qurt	168.2%
ud	225.1%

✔: overestimation $< 10\%$ ✓: $10\% \leq$ overestimation $< 20\%$

Results of the WCET estimation

CompCert integration → useful transformations for WCET estimation

- Loop inversion (`for` / `while` ⟶ `do-while`)

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

✔: overestimation $< 10\%$ ✓: $10\% \leq$ overestimation $< 20\%$

Results of the WCET estimation

CompCert integration → useful transformations for WCET estimation

- Loop unrolling

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

✔: overestimation $< 10\%$ ✓: $10\% \leq$ overestimation $< 20\%$

Results of the WCET estimation

CompCert integration → useful transformations for WCET estimation

✓ Precision improvements with little proof overhead

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

✔: overestimation $< 10\%$ ✓: $10\% \leq$ overestimation $< 20\%$

Outline

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