Precise WCET using SMT

Julien Henry Mihail Asavoae David Monniaux Claire Maïza

Partially funded by ERC STATOR and ANR W-SEPT projects

LCTES, 12^{th} June, 2014

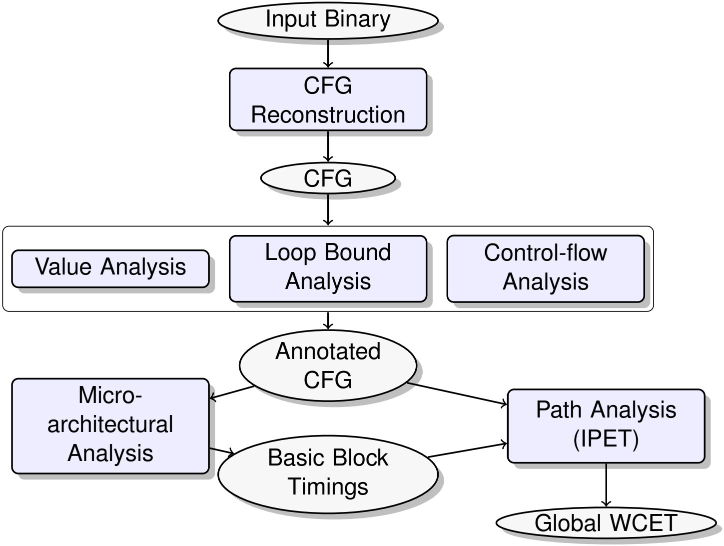
Summary

1. Standard Approach : imprecise
2. Precise BUT Inefficient
3. Precise AND Efficient

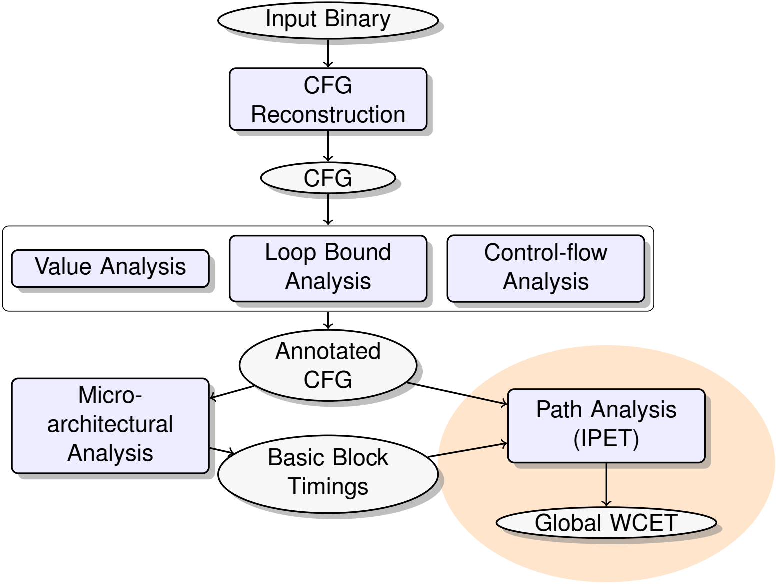
Summary

1. Standard Approach : imprecise
2. Precise BUT Inefficient
3. Precise AND Efficient

WCET: Standard Approach



WCET: Standard Approach



WCET: Standard Approach using ILP

Input:

- CFG of the program
- Basic Blocks timing upper bounds

Output: WCET for the entire CFG

$\rightarrow$ Integer Linear Programming (ILP) problem.

ILP constraints encode:

- control structure
- possibly some infeasible paths :
  “if transition T1 is taken then T2 is not”

WCET: Standard Approach using ILP

Input:

- CFG of the program
- Basic Blocks timing upper bounds

Output: WCET for the entire CFG

→ Integer Linear Programming (ILP) problem.

ILP constraints encode:

- control structure
- possibly some infeasible paths :
  "if transition T1 is taken then T2 is not"

Problem

Imprecise: Worst-case path may be infeasible

Reactive Control Systems

```
void rate_limiter_step() {
  assume (x_old <= 10000);
  assume (x_old >= -10000);
  x = input(-10000,10000);
  if (x > x_old+10)
    x = x_old+10;
  if (x < x_old-10)
    x = x_old-10;
  x_old = x;
}

void main() {
  while (1)
    rate_limiter_step();
}
```

1 “big” infinite loop

$\sim$ Loop-free body

Goal: WCET for 1 loop iteration $<$ some bound

Summary

1 Standard Approach : imprecise

2 Precise BUT Inefficient

3 Precise AND Efficient

Our Method

Replace ILP by **Satisfiability Modulo Theory**

Why? Expressivity : detects every semantically infeasible paths

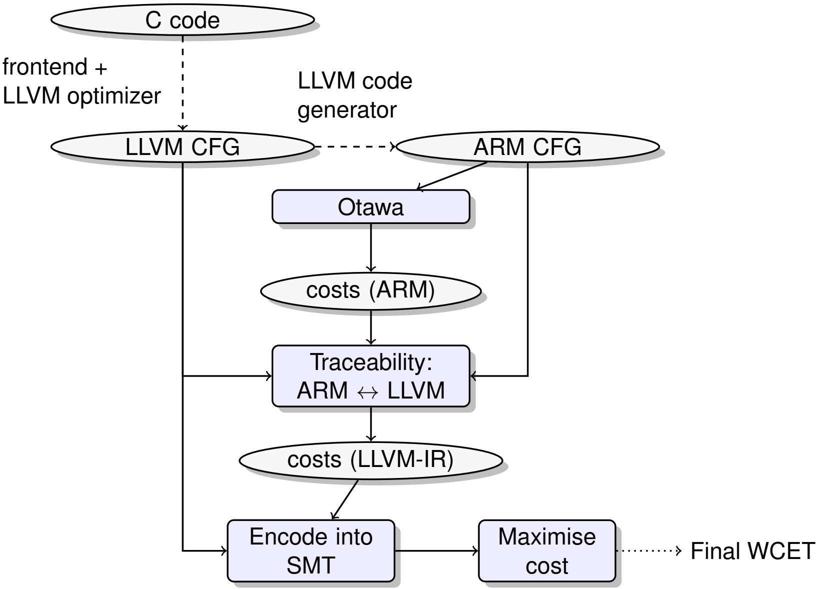
Input:

- **Loop-free** CFG of the program
- Basic Blocks timings (e.g. given by OTAWA)

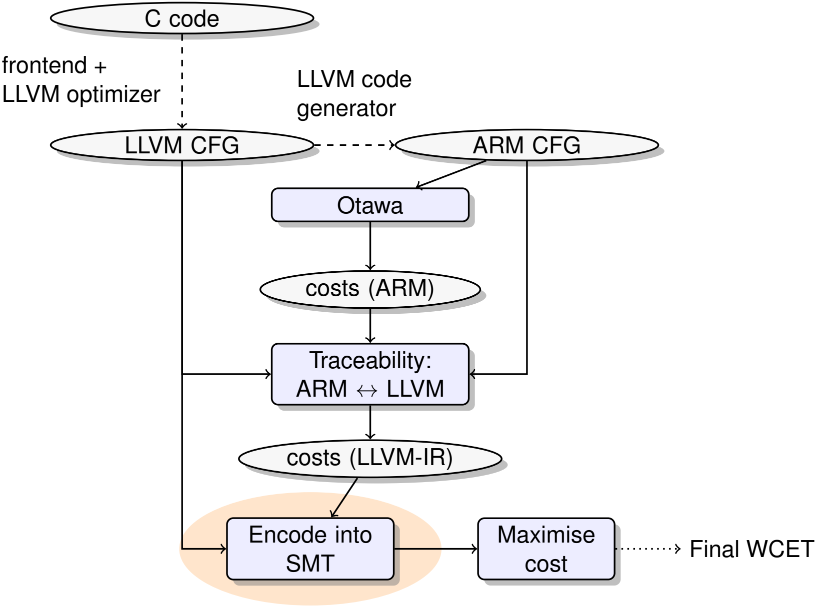
Output:

- WCET for the entire CFG + Worst Case path

Our Workflow



Our Workflow



Satisfiability Modulo Theory

Boolean Satisfiability (SAT):

$$b_1 \wedge ((b_2 \wedge b_3) \vee (b_4))$$

Satisfiability Modulo Theory

Boolean Satisfiability (SAT):

$$b_1 \wedge ((b_2 \wedge b_3) \vee (b_4))$$

$$x \geq 0 \wedge ((y \geq x + 10 \wedge y \leq 0) \vee (x + 1 \geq 0))$$

Modulo Theory: Atoms are elements from a given decidable theory

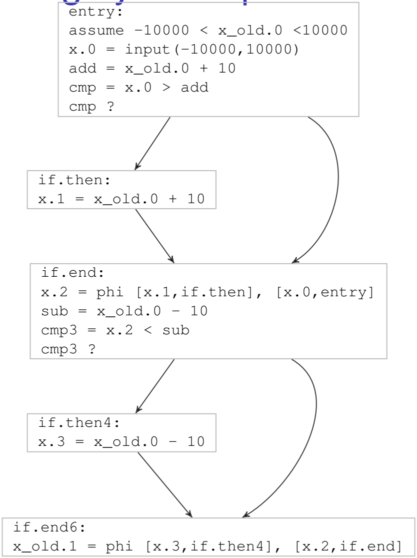
e.g. Linear Integer Arithmetic (LIA)

SMT solvers typically combine a SAT solver + Theory solver

SMT Encoding by Example

```
void rate_limiter_step() {
  assume (x_old <= 10000);
  assume (x_old >= -10000);
  x = input(-10000,10000);
  if (x > x_old+10)
    x = x_old+10;
  if (x < x_old-10)
    x = x_old-10;
  x_old = x;
}

void main() {
  while (1)
    rate_limiter_step();
}
```



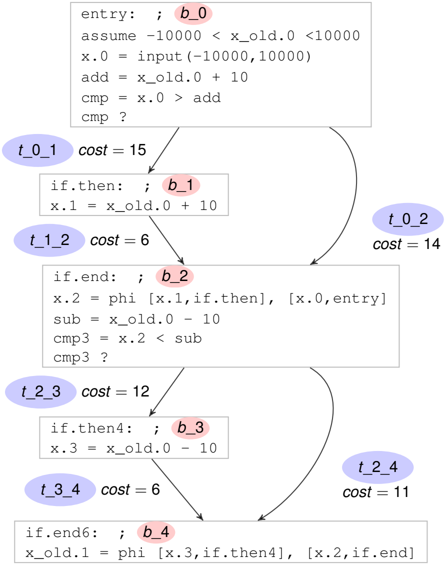
LLVM Control Flow Graph

The SMT formula encodes the feasible program traces:

- 1 Boolean per block
- 1 Boolean per transition

b_i *true* $\leftrightarrow$ trace goes through b_i

Cost for the trace: $\sum b_i * cost_i$

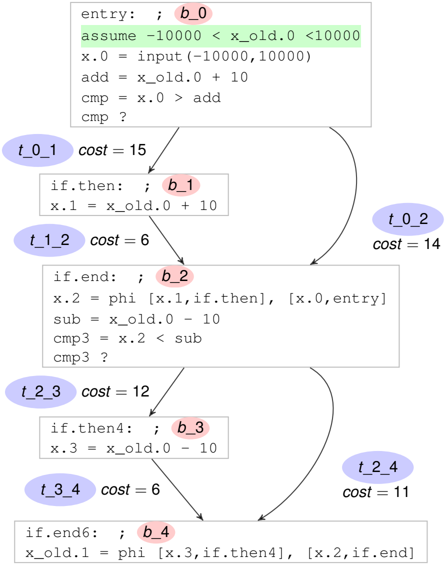


Step 1: encode instructions (Linear Integer Arithmetic)

Static Single Assignment form:

1 SMT variable $\leftrightarrow$ 1 SSA variable

$$
\begin{aligned}
& -10000 \leq x\_old.0 \leq 10000 \\
\wedge\ & -10000 \leq x.0 \leq 10000 \\
\wedge\ & add = (x\_old.0 + 10) \\
\wedge\ & x.1 = (x\_old.0 + 10) \\
\wedge\ & sub = (x\_old.0 - 10) \\
\wedge\ & x.3 = (x\_old.0 - 10) \\
\wedge\ & b\_2 \Rightarrow (x.2 = ite(t\_1\_2, x.1, x.0)) \\
\wedge\ & b\_4 \Rightarrow (x.1 = ite(t\_3\_4, x.3, x.2))
\end{aligned}
$$

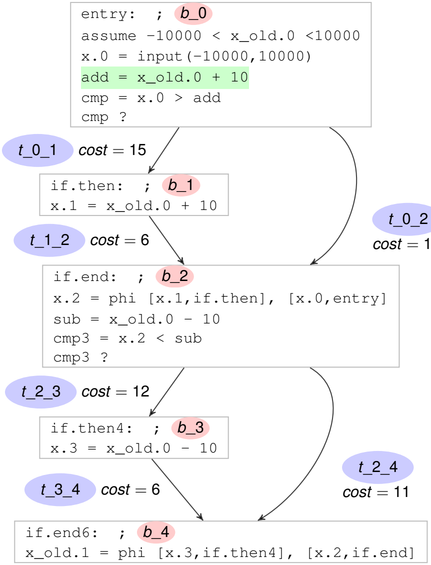


Step 1: encode instructions (Linear Integer Arithmetic)

Static Single Assignment form:

1 SMT variable $\leftrightarrow$ 1 SSA variable

$$
\begin{aligned}
& -10000 \leq x\_old.0 \leq 10000 \\
\wedge \quad & -10000 \leq x.0 \leq 10000 \\
\wedge \quad & add = (x\_old.0 + 10) \\
\wedge \quad & x.1 = (x\_old.0 + 10) \\
\wedge \quad & sub = (x\_old.0 - 10) \\
\wedge \quad & x.3 = (x\_old.0 - 10) \\
\wedge \quad & b\_2 \Rightarrow (x.2 = ite(t\_1\_2, x.1, x.0)) \\
\wedge \quad & b\_4 \Rightarrow (x.1 = ite(t\_3\_4, x.3, x.2))
\end{aligned}
$$

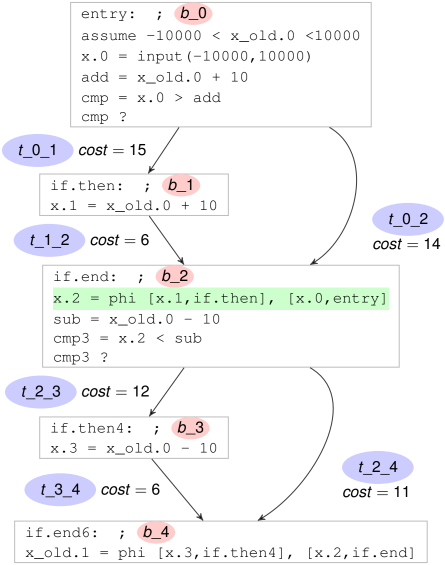


Step 1: encode instructions (Linear Integer Arithmetic)

Static Single Assignment form:

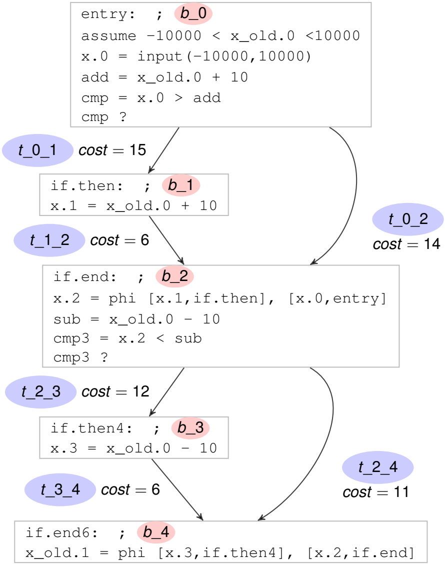
1 SMT variable $\leftrightarrow$ 1 SSA variable

$$
\begin{aligned}
& -10000 \leq x\_old.0 \leq 10000 \\
\wedge \quad & -10000 \leq x.0 \leq 10000 \\
\wedge \quad & add = (x\_old.0 + 10) \\
\wedge \quad & x.1 = (x\_old.0 + 10) \\
\wedge \quad & sub = (x\_old.0 - 10) \\
\wedge \quad & x.3 = (x\_old.0 - 10) \\
\wedge \quad & b\_2 \Rightarrow (x.2 = ite(t\_1\_2, x.1, x.0)) \\
\wedge \quad & b\_4 \Rightarrow (x.1 = ite(t\_3\_4, x.3, x.2))
\end{aligned}
$$



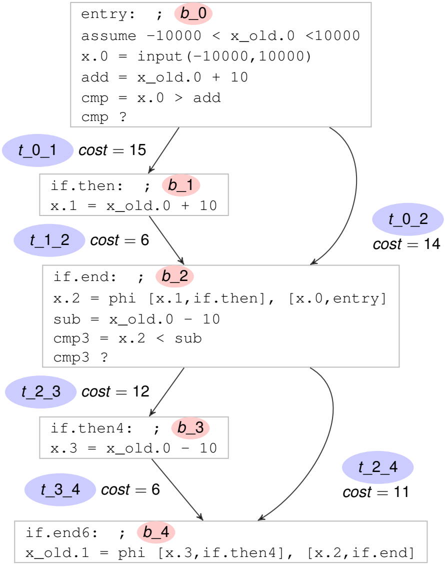
Step 2: encode control flow (Very similar to ILP)

$$
\begin{array}{ll}
\wedge & b\_0 = b\_4 = \mathit{true} \\
\wedge & b\_1 = t\_0\_1 \\
\wedge & b\_2 = (t\_0\_2 \vee t\_1\_2) \\
\wedge & \vdots \\
\wedge & \vdots \\
\wedge & t\_0\_1 = (b\_0 \wedge (x.0 > \mathit{add})) \\
\wedge & \vdots \\
\wedge & \vdots
\end{array}
$$



Step 3: encode timings

$$
\begin{aligned}
\wedge \quad & c\_0\_1 = (if(t\_0\_1) \text{ then } 15 \text{ else } 0) \\
\wedge \quad & c\_0\_2 = (if(t\_0\_2) \text{ then } 14 \text{ else } 0) \\
\wedge \quad & \vdots \\
\wedge \quad & \vdots \\
\wedge \quad & \vdots \\
\wedge \quad & cost = (c\_0\_1 + c\_0\_2 + c\_1\_2 \\
& \qquad + c\_2\_3 + c\_2\_4 + c\_3\_4)
\end{aligned}
$$



1 satisfying assignment $\leftrightarrow$ 1 program trace:

$b\_0 = b\_1 = b\_2 = b\_4 = \text{true}$

$b\_3 = \text{false}$

$t\_0\_1 = t\_1\_2 = t\_2\_4 = \text{true}$

$t\_0\_2 = t\_2\_3 = t\_3\_4 = \text{false}$

$\text{x\_old.0} = 50$

$\text{x.0} = 61$

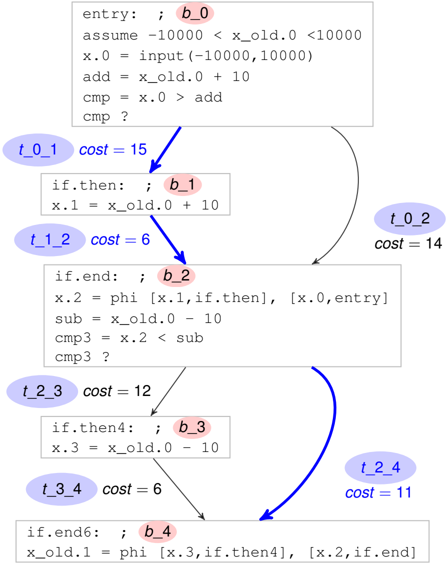
$\text{add} = 60$

$\text{x.1} = 60$

$\text{x.2} = 60$

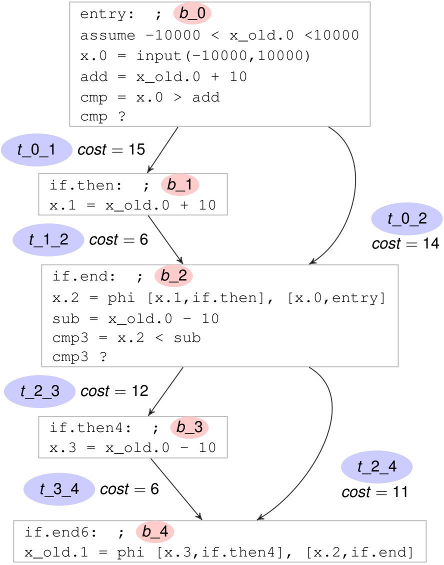
$\text{sub} = 40$

cost = 32



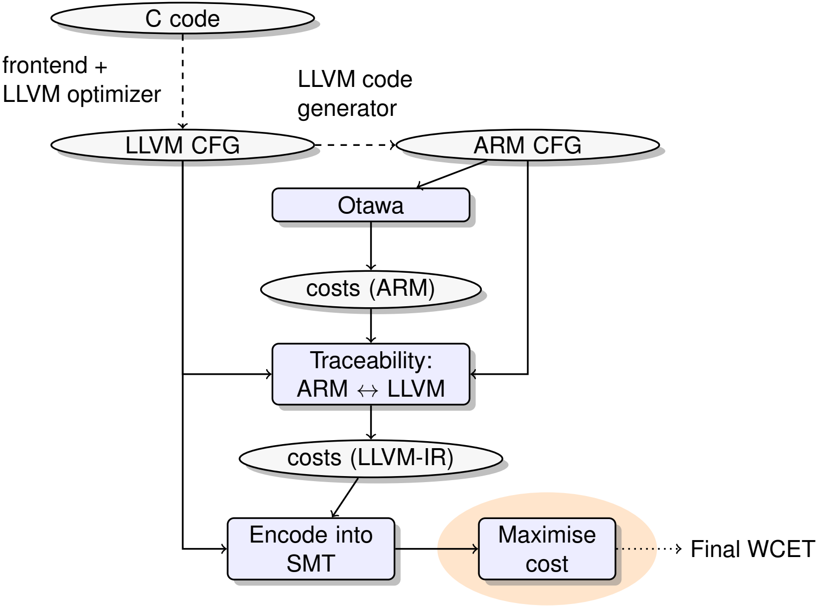
1 satisfying assignment $\leftrightarrow$ 1 program trace:

$b\_0 = b\_1 = b\_2 = b\_4 = \text{true}$
$b\_3 = \text{false}$
$t\_0\_1 = t\_1\_2 = t\_2\_4 = \text{true}$
$t\_0\_2 = t\_2\_3 = t\_3\_4 = \text{false}$
$\text{x\_old.0} = 50$
$\text{x.0} = 61$
$\text{add} = 60$
$\text{x.1} = 60$
$\text{x.2} = 60$
$\text{sub} = 40$
cost = 32



We want the trace with the highest cost

Our Workflow



Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable *cost*.

Using any off-the-shelf SMT solver

Dichotomy strategy (with incremental solving):
Maintain an interval containing the WCET

- Initial interval $[0, 100]$

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable *cost*.

Using any off-the-shelf SMT solver

Dichotomy strategy (with incremental solving):
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable *cost*.

Using any off-the-shelf SMT solver

Dichotomy strategy (with incremental solving):
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70
- new interval $[70, 100]$

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable *cost*.

Using any off-the-shelf SMT solver

Dichotomy strategy (with incremental solving):
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70
- new interval $[70, 100]$
- Is there a trace where *cost* > 85? No

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable *cost*.

Using any off-the-shelf SMT solver

Dichotomy strategy (with incremental solving):
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70
- new interval $[70, 100]$
- Is there a trace where *cost* > 85? No
- new interval $[70, 85]$

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable *cost*.

Using any off-the-shelf SMT solver

Dichotomy strategy (with incremental solving):
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where *cost* > 50? Yes, 70
- new interval $[70, 100]$
- Is there a trace where *cost* > 85? No
- new interval $[70, 85]$
- ...

Computing the WCET

Optimization modulo Theory:
We search for the trace maximizing the variable *cost*.

Using any off-the-shelf SMT solver

Dichotomy strategy (with incremental solving):
Maintain an interval containing the WCET

- Initial interval $[0, 100]$
- Is there a trace where $cost > 50$? Yes, 70
- new interval $[70, 100]$
- Is there a trace where $cost > 85$? No
- new interval $[70, 85]$
- …

A Really Simple Example

$b_1, \ldots, b_n$ unconstrained Booleans

```
if (b1) { /* timing = 2 */ } else { /* timing = 3*/ }
if (b1) { /* timing = 3 */ } else { /* timing = 2*/ }
...
if (bn) { /* timing = 2 */ } else { /* timing = 3*/ }
if (bn) { /* timing = 3 */ } else { /* timing = 2*/ }
```

Basic IPET would find WCET <= (3+3)n = 6n

“Obviously” all traces take time $(3+2)n = 5n$.

A Really Simple Example

$b_1, \ldots, b_n$ unconstrained Booleans

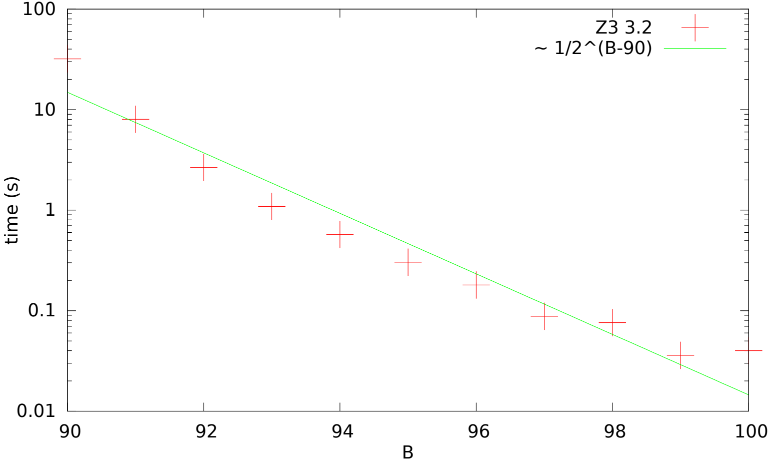
```
if (b1) { /* timing = 2 */ } else { /* timing = 3*/ }
if (b1) { /* timing = 3 */ } else { /* timing = 2*/ }
...
if (bn) { /* timing = 2 */ } else { /* timing = 3*/ }
if (bn) { /* timing = 3 */ } else { /* timing = 2*/ }
```

Basic IPET would find WCET <= (3+3)n = 6n

"Obviously" all traces take time $(3 + 2)n = 5n$.

SMT approach will find 5n, but in a few months...

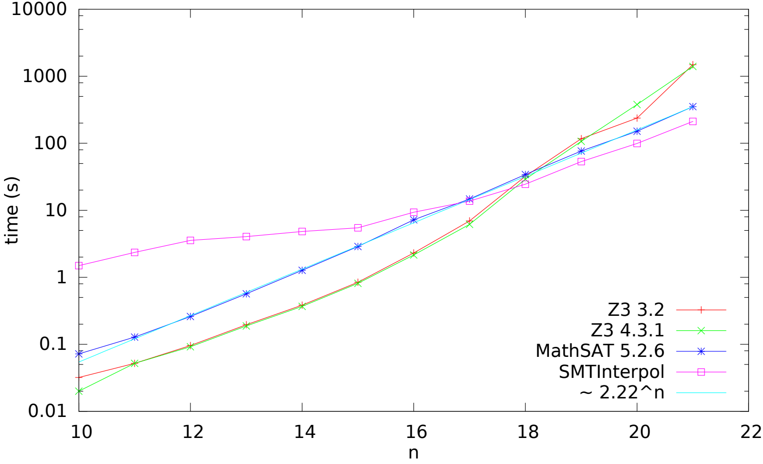
Binary search, $n = 18$, $WCET = 90$



Cost grows exponentially close to the optimum 90.

Proving optimality is costly

Proving that there is no trace longer than $5n$



Cost **exponential** in n (2^n paths)

Why such high cost?

Formula we try to solve:

$$(b_1 \Rightarrow x_1 = 2) \wedge (\neg b_1 \Rightarrow x_1 = 3) \wedge (b_1 \Rightarrow y_1 = 3) \wedge (\neg b_1 \Rightarrow y_1 = 2) \wedge$$
$$\cdots \wedge$$
$$(b_n \Rightarrow x_n = 2) \wedge (\neg b_n \Rightarrow x_n = 3) \wedge (b_n \Rightarrow y_n = 3) \wedge (\neg b_n \Rightarrow y_n = 2) \wedge$$
$$x_1 + y_1 + \cdots + x_n + y_n > 5n$$

All production grade SMT-solver are based on “DPLL($\mathcal{T}$)”:

- enumerate a Boolean choice tree over $b_1, \ldots, b_n$
- cut branches when encountering **inconsistent numerical constraints** (**blocking clauses**).

Diamond formulas

SMT encoding of WCET problems leads to **diamond formulas.**

For every state-of-the-art DPLL($\mathcal{T}$)-based SMT solver:

- Impossible to get **sufficiently general** blocking clauses

Diamond formulas

SMT encoding of WCET problems leads to **diamond formulas.**

For every state-of-the-art DPLL($\mathcal{T}$)-based SMT solver:

- Impossible to get **sufficiently general** blocking clauses
- The solver will **exhaustively enumerate every paths** ($\Rightarrow$ exponential)

Diamond formulas

SMT encoding of WCET problems leads to **diamond formulas.**

For every state-of-the-art DPLL($\mathcal{T}$)-based SMT solver:

- Impossible to get **sufficiently general** blocking clauses
- The solver will **exhaustively enumerate every paths** ($\Rightarrow$ exponential)
- Fully detailed in the paper...

Diamond formulas

SMT encoding of WCET problems leads to **diamond formulas.**

For every state-of-the-art DPLL($\mathcal{T}$)-based SMT solver:

- Impossible to get **sufficiently general** blocking clauses
- The solver will **exhaustively enumerate every paths** ($\Rightarrow$ exponential)
- Fully detailed in the paper…

But we can fix that !

Summary

1 Standard Approach : imprecise

2 Precise BUT Inefficient

3 Precise AND Efficient

SMT solvers miss "obvious" properties

```
...
if (bi) { /* timing 2 */ } else { /* timing 3*/ }
if (bi) { /* timing 3 */ } else { /* timing 2*/ }
...
```

Human remark: "**obviously**, $x_i + y_i \leq 5$ for any i"

$x_i + y_i \leq 5$ is implied by the original formula

"Normal" SMT solvers **do not invent new atomic predicates**: they can't learn it...

SMT solvers miss "obvious" properties

```
...
if (b_i) { /* timing 2 */ } else { /* timing 3*/ }
if (b_i) { /* timing 3 */ } else { /* timing 2*/ }
...
```

Human remark: "**obviously**, $x_i + y_i \leq 5$ for any i"

$x_i + y_i \leq 5$ is implied by the original formula

"Normal" SMT solvers **do not invent new atomic predicates**: they can't learn it...

Predicates have to be syntactically present!

If we conjoin these constraints to the SMT formula, the problem is solved instantaneously. . .

Our Solution

- Distinguish "portions" in the program.
- Compute upper bound B_i on WCET for each portion i (recursive call or rougher bound)
- Conjoin these constraints to the previous SMT formula $c_1 + \cdots + c_5 \leq B_1$, $c_6 + \cdots + c_{10} \leq B_2$, etc.
- Do the binary search as before

Solving time from "nonterminating after one night" to "a few seconds".

Cuts

The new constraints

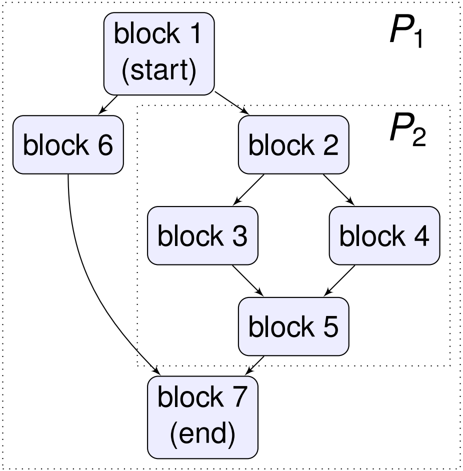
- are implied by the original problem (formulas are **equivalent**)
- but not syntactically present in it
- allow the Theory solver to find much more general blocking clauses $\rightarrow$ prune much larger sets of traces at once

We call them **cuts**, as in Operational Research

How to choose the portions/cuts ?

Syntactic criterion

Simply choose **if-then-else** structures



For P_2:
$c2 + c3 + c4 + c5 \leq c2 + max(c3, c4) + c5$

$\rightarrow$ was sufficient for our industrial benchmarks

How to choose the portions/cuts ?

Semantic criterion

```
...
if (b_i) { /* timing 2 */ } else { /* timing 3*/ }
...
/* not contiguous... */
...
if (b_i) { /* timing 3 */ } else { /* timing 2*/ }
...
```

- “Slice” the program w.r.t b_i
- Recursively call our WCET procedure
- The obtained WCET gives the upper bound for the portion

Experiments with ARMv7

OTAWA for Basic Block timings

PAGAI for SMT, see `pagai.forge.imag.fr`, uses Z3 SMT solver

Cuts : only syntactic criterion

Benchmark name	WCET bounds (#cycles) ILP IPET	SMT	diff	Analysis time (seconds) with cuts	no cuts	#cuts
statemate	3297	3211	2.6%	943.5	$+\infty$	143
nsichneu (1 iteration)	17242	13298	22.7%	6hours	$+\infty$	378
cruise-control	881	873	0.9%	0.1	0.2	13
digital-stopwatch	1012	954	5.7%	0.6	2104.2	53
autopilot	12663	5734	54.7%	1808.8	$+\infty$	498
fly-by-wire	6361	5848	8.0%	10.8	$+\infty$	163
miniflight	17980	14752	18.0%	40.9	$+\infty$	251
tdf	5789	5727	1.0%	13.0	$+\infty$	254

- Malardalen WCET Benchmarks
- Scade designs
- Industrial Code

Conclusion

- Compute the WCET by replacing ILP by SMT
- WCET estimation is notably improved
- Fully automatic
- Scalability issues addressed using “cuts”

Thank you !

Advertisement: **PAGAI Static Analyzer** for C/LLVM

- Used by our WCET computation engine
- Detects array out-of-bounds & integer overflows
- Proves assert statements over numerical variables
- Instrument LLVM bitcode with invariants

Visit `http://pagai.forge.imag.fr`

Extra slides

Related Work

Use of Symbolic Execution. Differences are:

- Craig Interpolants / blocking clauses + cuts
- SMT solvers select litterals out of the program execution order

Other works also make use of SMT, but they do not encode functional semantics

A Possible Boolean Assignment

Take the satisfying assignment where all the b_i's are set to true (the x_i's $= 2$ and y_i's $= 3$)

```
if (b1) { /* timing 2 */ } else { /* timing 3*/ }
if (b1) { /* timing 3 */ } else { /* timing 2*/ }
...
if (bn) { /* timing 2 */ } else { /* timing 3*/ }
if (bn) { /* timing 3 */ } else { /* timing 2*/ }
```

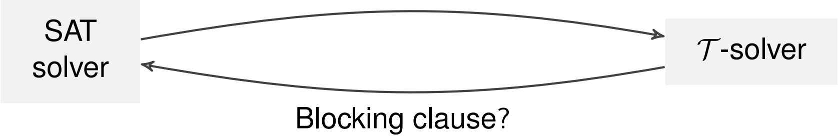
Theory atoms

$$x_1 \leq 2\ ,\quad x_1 \leq 3,\quad \neg(y_1 \leq 2),\quad y_1 \leq 3$$

$$\vdots$$

$$x_n \leq 2\ ,\quad x_n \leq 3,\quad \neg(y_n \leq 2),\quad y_n \leq 3$$

$$x_1 + y_1 + \cdots + x_n + y_n > 5n$$



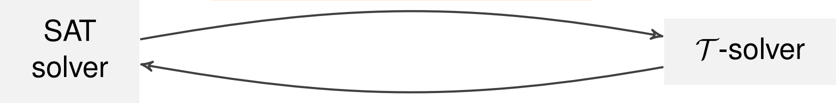
Theory atoms

$$x_1 \leq 2, \quad x_1 \leq 3, \quad \neg(y_1 \leq 2), \quad y_1 \leq 3$$

$$\vdots$$

$$x_n \leq 2, \quad x_n \leq 3, \quad \neg(y_n \leq 2), \quad y_n \leq 3$$

$$x_1 + y_1 + \cdots + x_n + y_n > 5n$$



Blocking clause

$$x_1 \leq 2, \quad y_1 \leq 3$$

$$\vdots$$

$$x_n \leq 2, \quad y_n \leq 3$$

$$x_1 + y_1 + \cdots + x_n + y_n > 5n$$

Blocking clauses only cut **one single trace**. . .

2^n of them.

The solver has to prove them inconsistent **one by one**.

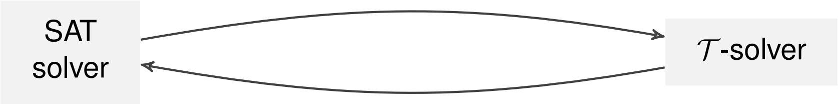
Theory atoms

$$x_1 \leq 2, \quad x_1 \leq 3, \quad \neg(y_1 \leq 2), \quad y_1 \leq 3, \quad x_1 + y_1 \leq 5$$

$$\vdots$$

$$x_n \leq 2, \quad x_n \leq 3, \quad \neg(y_n \leq 2), \quad y_n \leq 3, \quad x_n + y_n \leq 5$$

$$x_1 + y_1 + \cdots + x_n + y_n > 5n$$



Blocking clause?

Cuts all the 2^n traces at once.

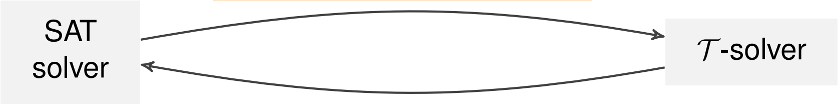
Theory atoms

$$x_1 \leq 2, \quad x_1 \leq 3, \quad \neg(y_1 \leq 2), \quad y_1 \leq 3, \quad x_1 + y_1 \leq 5$$

$$\vdots$$

$$x_n \leq 2, \quad x_n \leq 3, \quad \neg(y_n \leq 2), \quad y_n \leq 3, \quad x_n + y_n \leq 5$$

$$x_1 + y_1 + \cdots + x_n + y_n > 5n$$



Blocking clause

$$x_1 + y_1 \leq 5$$

$$\vdots$$

$$x_n + y_n \leq 5$$

$$x_1 + y_1 + \cdots + x_n + y_n > 5n$$

Cuts all the 2^n traces at once.