# Parametric Timing Analysis for Complex Systems

Sebastian Altmeyer[1], Christian Hümbert[2], Björn Lisper[3], Reinhard Wilhelm[1]

[1]Saarland University

[2]AbsInt GmbH

[3]Mälardalen Hogskola

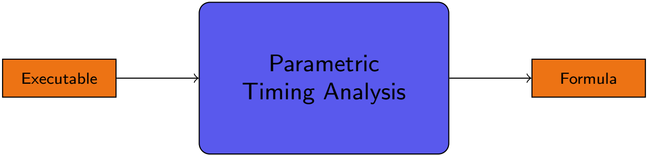
RTCSA 2008, Kaohsiung

# Motivation

- timing analysis essential for hard real-time systems
- many systems depend on input parameters (operating system schedulers, etc.)
- only two possible solutions:
  1. assume upper bounds on the unknown parameters
     $\Rightarrow$ highly overapproximated WCET
  2. restart the analysis for all parameter assignments
     $\Rightarrow$ very high analysis time
- parametric timing analysis delivers timing formula instead of a numeric value

# Outline

# Structure

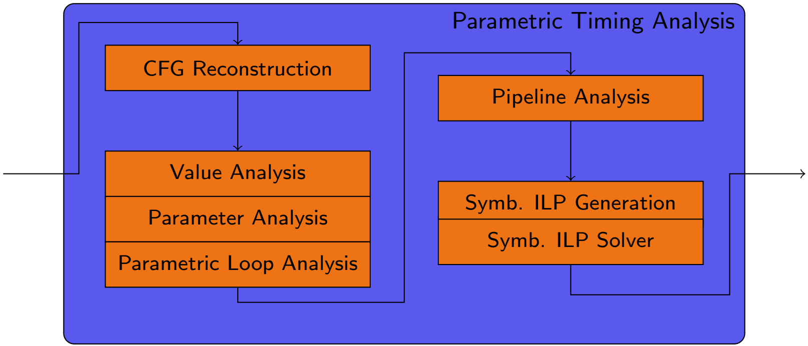



**Input** binary executable (no high-level code)

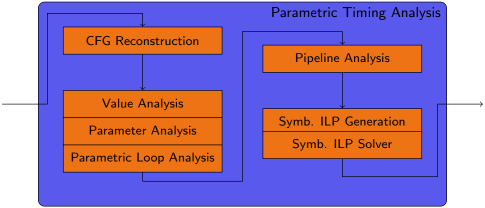
**Output** parametric timing formula

Parametric timing analysis is based on
AbsInt's *aiT Timing Analyzer*

# Internal Structure



# Internal Structure



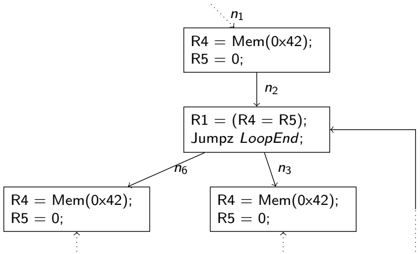

CFG Reconstruction extracts the control flow graph from the executable

# CFG Reconstruction

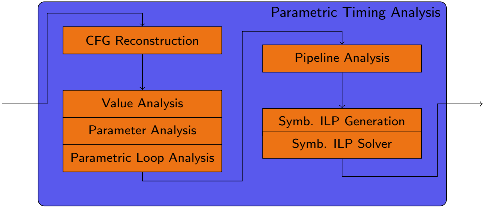
builds the control flow graph.

A control flow graph consists of

- basic blocks (list of instructions always entered at the first and left at the last)
- edges representing possible control flow



# Internal Structure




Value Analysis determines values for registers and memory accesses

Parameter Analysis determines the parameters of the analyzed program

Parametric Loop Analysis determines loop bounds and parametric loop bound expressions

# Value Analysis

determines values of registers and memory cells.

For a register x,
value analysis derives interval $[b_l, b_u]$, s.t. $b_l \leq x \leq b_u$.

Intervals are needed to

- determine addresses of memory accesses (for cache analysis),
- exclude infeasible paths,
- derive loop bounds.

# Parameter Analysis

- on source-code level, parameters and variables clearly separated
- on executable level, there are only registers and memory cells

⇒ a parameter is a register or a memory cell, the program reads from before it writes to.

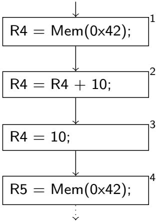
## Parameter Analysis derives

1. set of parameters,
2. parametric dependencies for all program points.

# Parameter Analysis

1. collects parametric dependencies such as: $R_x = R_y + [l_b, u_b]$
2. intervals are taken from value analysis
3. implemented as a data flow analysis

Example:



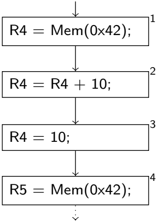

(assume memory cell Mem(0x42) has not been accessed before)

# Parameter Analysis

1. collects parametric dependencies such as: $R_x = R_y + [l_b, u_b]$
2. intervals are taken from value analysis
3. implemented as a data flow analysis

Example:

$\{Mem(0x42)\},\{\}$

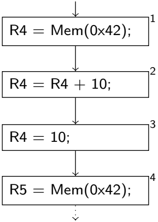


# Parameter Analysis

1. collects parametric dependencies such as: $R_x = R_y + [l_b, u_b]$
2. intervals are taken from value analysis
3. implemented as a data flow analysis

Example:

$\{Mem(0x42)\},\{\}$

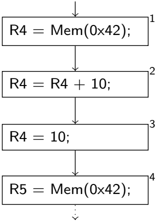



$\{Mem(0x42)\},\{(R4 = Mem(0x42) + [0,0])\}$

# Parameter Analysis

1. collects parametric dependencies such as: $R_x = R_y + [l_b, u_b]$
2. intervals are taken from value analysis
3. implemented as a data flow analysis

Example:




$\{Mem(0x42)\},\{\}$

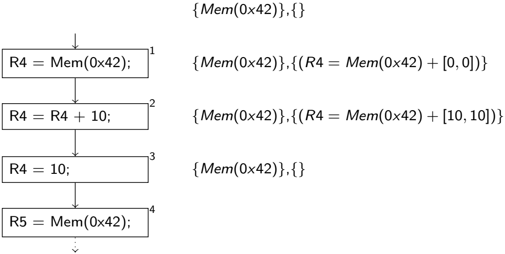
$\{Mem(0x42)\},\{(R4 = Mem(0x42) + [0, 0])\}$

$\{Mem(0x42)\},\{(R4 = Mem(0x42) + [10, 10])\}$

# Parameter Analysis

1. collects parametric dependencies such as: $R_x = R_y + [l_b, u_b]$
2. intervals are taken from value analysis
3. implemented as a data flow analysis

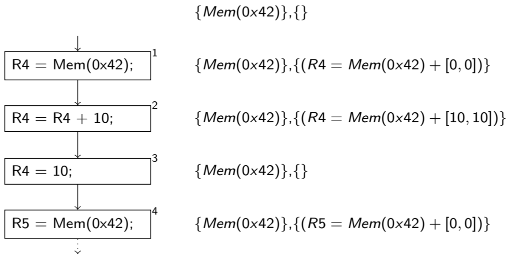
Example:



# Parameter Analysis

1. collects parametric dependencies such as: $R_x = R_y + [l_b, u_b]$
2. intervals are taken from value analysis
3. implemented as a data flow analysis

Example:



# Parametric Loop Analysis

derives symbolic loop bound expressions.

Such as a loop $L$ is bounded by $b_L$, where

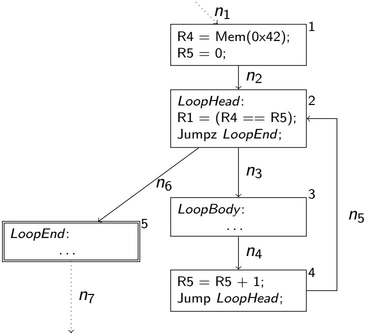
$$b_L = \text{if } Mem(0x42) < 0 \text{ then } \infty \text{ else } Mem(0x42)$$

Analysis

- acquires interval from value analysis
- acquires set of parameters from parameter analysis

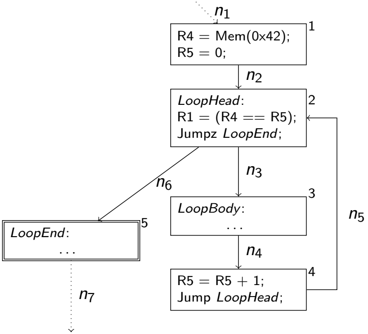
# Parametric Loop Analysis

1. Collection of (potential) *loop counters*
2. Derivation of *loop invariant*
3. Evaluation of loop exits
4. Construction of loop bounds



# Parametric Loop Analysis

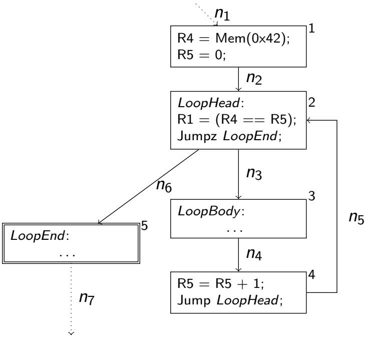
1. Collection of (potential) *loop counters*
2. Derivation of *loop invariant*
3. Evaluation of loop exits
4. Construction of loop bounds




1. potential loop counters
= all register accessed within loop body
(here: $R5$)

# Parametric Loop Analysis

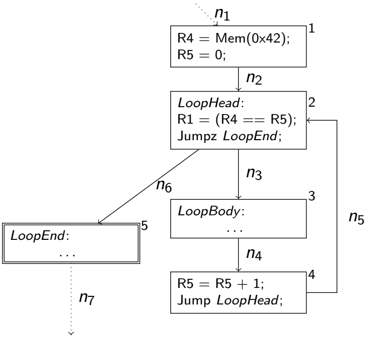
1. Collection of (potential) *loop counters*
2. Derivation of *loop invariant*
3. Evaluation of loop exits
4. Construction of loop bounds




2. loop invariant
= how loop counter changes per iteration
(here: $R5 = R5 + 1$)

# Parametric Loop Analysis

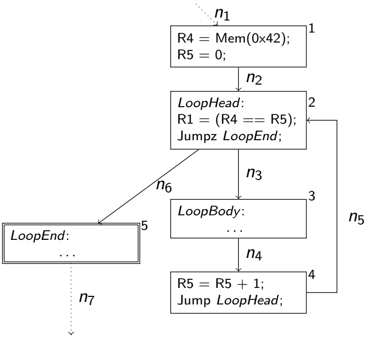
1. Collection of (potential) *loop counters*
2. Derivation of *loop invariant*
3. Evaluation of loop exits
4. Construction of loop bounds




3. evaluation of loop exits (here: $R1 = (R4 == R5)$)
initial values/parameter dependencies:
$R4 = \text{Mem}(0x42), R5 = 0$

# Parametric Loop Analysis

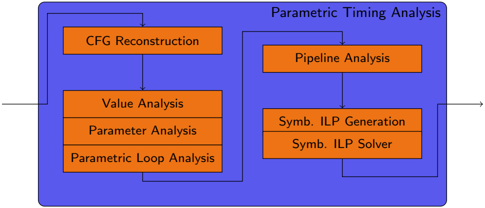
1. Collection of (potential) *loop counters*
2. Derivation of *loop invariant*
3. Evaluation of loop exits
4. Construction of loop bounds




4. construction of loop bounds:

$$b_L = \text{if } Mem(0x42) < 0 \text{ then } \infty \text{ else } Mem(0x42)$$

# Internal Structure




**Pipeline Analysis** simulates the low-level behavior of the target architecture

**Symbolic ILP Generator** generates a symbolic ILP which represent the search for the longest program path

**Symbolic ILP Solver** solves the symbolic analysis ILP and returns the symbolic timing formula

# Pipeline Analysis

In modern processors, execution time depends on
caches, branch prediction, speculative execution, ...

Thus, pipeline analysis

- simulates processors low level behavior,
- determines WCETs for all basic blocks.

# Parametric Path Analysis

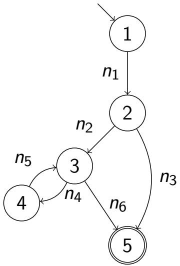
computes WCET by searching longest path.

Analysis consists of two steps:

1. creating ILP with parameters
2. solving symbolic ILP to produce symbolic timing formula

# Parametric Path Analysis - IPET

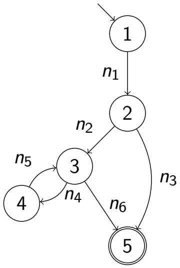
Searching for path with highest execution time by using
Implicit Path Enumeration Technique (IPET)




variables $n_i$ denote how often edge $i$ is traversed

# Parametric Path Analysis - IPET

Searching for path with highest execution time by using Implicit Path Enumeration Technique (IPET)

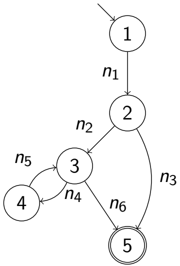



$n_1 = 1;$

first node is entered exactly once

# Parametric Path Analysis - IPET

Searching for path with highest execution time by using Implicit Path Enumeration Technique (IPET)

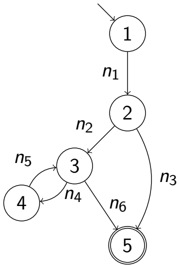



$$n_1 = 1;$$
$$n_1 = n_2 + n_3;$$
$$n_2 + n_5 = n_4 + n_6;$$
$$n_4 = n_5;$$

sum of successors traversals equals sum of predecessor traversals

# Parametric Path Analysis - IPET

Searching for path with highest execution time by using Implicit Path Enumeration Technique (IPET)

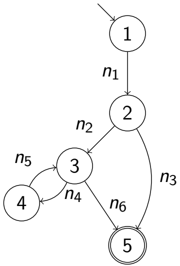



$$
\begin{aligned}
n_1 &= 1; \\
n_1 &= n_2 + n_3; \\
n_2 + n_5 &= n_4 + n_6; \\
n_4 &= n_5; \\
n_4 &<= b_L n_2;
\end{aligned}
$$

loop $L$ is executed $b_L$ times as often as it is entered
($b_L$ represents loop bound expression)

# Parametric Path Analysis - IPET

Searching for path with highest execution time by using Implicit Path Enumeration Technique (IPET)

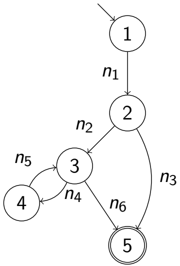



$$
\begin{aligned}
n_1 &= 1; \\
n_1 &= n_2 + n_3; \\
n_2 + n_5 &= n_4 + n_6; \\
n_4 &= n_5; \\
n_4 &<= b_L n_2; \\
n_3 + n_6 &= 1;
\end{aligned}
$$

last node is entered exactly once

# Parametric Path Analysis - IPET

Searching for path with highest execution time by using Implicit Path Enumeration Technique (IPET)




$$n_1 = 1;$$
$$n_1 = n_2 + n_3;$$
$$n_2 + n_5 = n_4 + n_6;$$
$$n_4 = n_5;$$
$$n_4 <= b_L n_2;$$
$$n_3 + n_6 = 1;$$

$$\max : \sum_i \left( \sum_{\forall j : n_j \text{ enters } B_i} c_i n_j \right)$$

objective function: maximize execution time by maximizing $c_i n_j$
($c_i$ = WCET of basic block $n_J$ enters)

# Parametric Path Analysis - Non-linear IPET constraints

relative loop constraint

$$n_4 <= b_L n_2;$$

is non-linear and has to be replaced by absolute constraint

$$n_4 <= c \cdot b_L;$$

by bounding variable $n_2$ by a constant $c$.

Note: all variables $n_i$ can be bounded if all loop bounds for all loops exist

# Parametric Path Analysis - Symbolic ILP

Symbolic ILP with parameters representing loop bound expressions.

Using PIP[1] to solve symbolic ILPs

PIP uses

- symbolic simplex, and
- symbolic cutting plane algorithm.

Note:

- PIP is usually bottleneck of the analysis
- number of parameters only bounded by performance of PIP

[1] http://www.piplib.org

# Short Note on the Timing Formulae

PIP outputs QUAST-file which is hardly human-readable

⇒ one additional step needed to:

1. pretty-print timing formula
2. evaluate symbolic loop bound expressions, and
3. evaluate timing formula

# Example of a Timing Formulae

```
(if #[ 0 1 -1]
 (list                          (list
  #[ 1 -376 -274]                #[ 1 -428 -222]
  #[ 0 0 1]                      #[ 0 1 0]
     ...                            ...
  #[ 0 0 0]                      #[ 0 0 0]
  #[ 0 0 0]                      #[ 0 0 0]
     ...                            ...
  #[ 0 0 1]                      #[ 0 1 0]
 )                              )
)
```

# Example of a Timing Formulae

first row only artificial – can be neglected

last row denotes a constant

$$\begin{bmatrix} 0 & 7 & 3 \end{bmatrix}$$

middle row(s) denotes the parameter(s)

$$= 7p + 3$$

# Example of a Timing Formulae

```
(if #[ 0 1 -1]
 (list
  #[ 1 -376 -274]
  #[ 0 0 1]
     ...
  #[ 0 0 0]
  #[ 0 0 0]
     ...
  #[ 0 0 1]
 )
 (list
  #[ 1 -428 -222]
  #[ 0 1 0]
     ...
  #[ 0 0 0]
  #[ 0 0 0]
     ...
  #[ 0 1 0]
 )
)
```

# Example of a Timing Formulae

if $(p - 1 \geq 0)$

```
(if #[ 0 1 -1]
```

then $(376p + 274)$

```
#[ 1 -376 -274]
#[ 0 0 1]
    ...
#[ 0 0 0]
#[ 0 0 0]
    ...
#[ 0 0 1]
)
```

else $(428p + 222)$

```
#[ 1 -428 -222]
#[ 0 1 0]
    ...
#[ 0 0 0]
#[ 0 0 0]
    ...
#[ 0 1 0]
)
)
```

# Precision

Parametric analysis (PA) has less information than numeric analysis (NA)
$\Rightarrow$ PA at most as precise as NA

Two main sources for a worse precision:

1. absolute loop bounds, and
2. path-exclusion.

# Precision - Absolute Loop Bounds

1. absolute loop bounds:

   absolute loop bounds ($n_2 <= b_L n_1;$) are less precise than relative ones ($n_2 <= c \cdot b_L;$)

   $c$ is only a bound, and thus can overapproximate true loop entry count

2. path-exclusion:

   numeric analysis knows the actuals values during analysis

   $\Rightarrow$ numeric analysis can exclude some paths which the parametric can not

# Testsetting

- Parametric timing analysis for PowerPC 565 and 755
- examples from Mälardalen WCET benchmark suite
  - Insertion sort
  - Matrix Multiplication
  - Square Root Computation by Taylor Series
  - Cyclic Redundancy Check (2 parameters)
- performed on: Intel Core Duo 1,66 Mhz, 1024 MB RAM
- compiled with: gcc-cross-compiler

# Testcases

We compared parametric analysis (PA) to numeric analysis (NA)

- **PA** compute timing formula once and instantiate it
- **NA** compute one WCET bound for each value-assignment

# Insertion Sort

- one normal parametric loop (initializing an array)
- one nested parametric loop that sorts the values (according to insertion-sort)
- parametric in the size of the array

| n | NA | PA | Diff. (%) |
|---|---|---|---|
| 0 | 1 494 | 1 798 | 20.3 |
| 1 | 1 910 | 2 086 | 9.1 |
| 10 | 118 411 | 121 579 | 2.7 |
| 100 | 10 788 631 | 10 791 799 | $<0.1$ |

$$Time(n) = \begin{cases} 1798 & \text{if } n < 1 \\ 2086 & \text{if } n = 1 \\ 1067n^2 + 1188n + 2999 & \text{otherwise} \end{cases}$$

# Matrix Multiplication

- one nested parametric loop initializing the two matrices
- nested parametric loops of depth 3 to multiply matrices (naive approach)
- parametric in the dimension of the matrices ($n \times n$)

| n | NA | PA | Diff. (%) |
|---|---|---|---|
| 0 | 2 915 | 3 046 | 4.5 |
| 1 | 5 244 | 8 371 | 59.6 |
| 10 | 745 156 | 890 884 | 19.6 |
| 100 | 669 888 316 | 683 246 374 | 2.0 |

$$Time(n) = \begin{cases} 3046 & \text{if } n < 1 \\ 2431n^3 + 2003n^2 \\ \quad +663n + 3274 & \text{otherwise} \end{cases}$$

# Square Root Computation

- computes square root using Taylor series
- one parametric loop
- parameter adjusts iteration depth/precision

| n | NA | PA | Diff. (%) |
|---|---|---|---|
| 0 | 208 | 208 | 0 |
| 1 | 208 | 208 | 0 |
| 10 | 3331 | 3331 | 0 |
| 100 | 34561 | 34561 | 0 |

$$Time(n) = \begin{cases} 208 & \text{if } n < 1 \\ 347(n-1) + 208 & \text{otherwise} \end{cases}$$

# Cyclic Redundancy Check

- 8bit cyclic redundancy check
- two parametric loops (not nested)
- two different parameters:
  - $n =$ the length of the input stream
  - $a$ restricts size of input alphabet

# Results - Cyclic Redundancy Check

| n | a | NA | PA | Diff. (%) |
|---|---|---|---|---|
| 1 | 64 | 50545 | 50545 | 0 |
| 1 | 256 | 203377 | 203377 | 0 |
| 10 | 64 | 51904 | 51904 | 0 |
| 10 | 256 | 204736 | 204736 | 0 |
| 100 | 64 | 65494 | 65494 | 0 |
| 100 | 256 | 218326 | 218326 | 0 |

$$Time(n,a) = \begin{cases} 397 & \text{if } n < 1 \wedge a < 1 \\ 796(a-1) + 397 & \text{if } n < 1 \wedge a \geq 1 \\ 151(n-1) + 397 & \text{if } n \geq 1 \wedge a < 1 \\ 151(n-1) + 796(a-1) + 397 & \\ & \text{if } n \geq 1 \wedge a \geq 1 \end{cases}$$

# Conclusions

Parametric timing analysis

- derives symbolic WCET formula automatically
  - identifies the parameters/computes parametric dependencies
  - derives symbolic loop bound expressions
  - creates and solves symbolic ILP to determine WCET formula
- is implemented as a prototype for PowerPC 565/755

Results of the parametric analysis:

- are usually as precise as numeric results for non-nested loops,
- converge to the numeric results as the values increase for nested loops

Usually, symbolic ILP solver is bottleneck of the analysis.

**Thanks for your attention!**