Design and Analysis of Time-Critical Systems

WCET Analysis: A Primer

Jan Reineke @ Saarland University Computer Science

ACACES Summer School 2017
Fiuggi, Italy
What does the execution time of a program depend on, on a single-core machine?

**Input-dependent control flow**

**Microarchitectural State**

Pipeline, Memory Hierarchy, Interconnect
The Single-core WCET Analysis Problem

Consider all possible program inputs

Consider all possible initial states of the hardware

\[ WCET_H(P) := \max_{i \in \text{Inputs}} \max_{h \in \text{States}(H)} ET_H(P, i, h) \]

Measuring or simulating the execution time for all inputs and all hardware states is not feasible in practice:
- There are too many.
- We cannot control the initial hardware states.
  ➞ Need for approximation!
Requirements for WCET Analysis

1. Upper bounds must be safe, i.e. not underestimated.
2. Upper bounds should be tight, i.e. not far away from real execution times.
3. Analysis effort must be tolerable.
Standard WCET Analysis Approach Today: Separation of Concerns + Abstraction

- **Value Analysis:**
  Determines invariants for the values in registers and in memory

- **Separation:**
  1. Bound possible microarchitectural executions using abstractions.
  2. Determine constraints on control flow (e.g. loop bounds) through program by abstractions.

- **Combination:** combine 1 and 2 to bound execution time of the whole program.

**Depends on hardware**

**Depends on program semantics**
Structure of WCET Analyzers

- **Input Executable**
  - Reconstructs a control-flow graph from the binary.
  - Determines invariants for the values in registers and in memory.

- **CFG Reconstruction**
  - Determines invariants on the control flow, by
    - Determining loop bounds,
    - Identifying infeasible paths.
  - Determines possible microarchitectural executions.

- **Value Analysis**
  - Determines constraints on the control flow, by
    - Determining loop bounds,
    - Identifying infeasible paths.

- **Control Flow Analysis**
- **Micro-architectural Analysis**

- **Global Bound Analysis**
  - Determines a worst-case path and an upper bound on the WCET.

- **WCET Bound**
Structure of WCET Analyzers
Employed Techniques

- **Timing Analysis Framework**
  - **Input Executable**
    - CFG Reconstruction
    - Value Analysis
    - Control Flow Analysis
    - Global Bound Analysis
    - WCET Bound

  - **Employed Techniques**
    - **Abstract Interpretation of the Program**
      - Abstract Interpretation of the Program + Hardware Model
      - Integer Linear Programming

  - **Micro-architectural Analysis**
    - Determines bounds on execution times of basic blocks.
    - Based on an abstract model of the microarchitecture, including detailed models of the pipeline and the memory hierarchy.

  - **Determines bound on execution times of program fragments.**

  - **Abstract Interpretation of Program + Hardware Model**

  - **WHENS Bound**

- **Jan Reineke**
  - Timing Analysis and Timing Predictability
  - 2. April 2012
Running Example

```c
int main(int x, int[] a) {
    int x = x % 5;
    int y = 42;
    while (x < y) {
        if (a[x] < a[x+1])
            x++
        else
            x += 2;
    }
    return x;
}
```
Structure of WCET Analyzers

Reconstructs a control-flow graph from the binary.

Determines invariants on the control flow, by
- Determining loop bounds,
- Identifying infeasible paths.

Determines invariants for the values in registers and in memory.

Determines possible microarchitectural executions.

Determines a worst-case path and an upper bound on the WCET.

Input
Executable

CFG
Reconstruction

Value
Analysis

Control
Flow
Analysis

Micro-
architectural
Analysis

Global
Bound
Analysis

WCET Bound
Value Analysis

Determines **invariants on values of registers** at different program points. Invariants are often in the form of **enclosing intervals** of all possible values.

Where is this information used?
- Microarchitectural Analysis
  - Pipeline Analysis
  - Cache Analysis
- Control-Flow Analysis
  - Detect infeasible paths
  - Derive loop bounds

```
R1 = R1 % 5
R2 = 42
R1 = R1 + 1
R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?
return R1
R1 < R2?
R1 = R1 + 2
R1 = R1 + 1
```
Value Analysis
Intuition of Interval Analysis

Can be formalized as Abstract Interpretation. ➜ Yields soundness and termination guarantees.
Structure of WCET Analyzers

**Input Executable**
- **Reconstructs a control-flow graph from the binary.**
- **Determines invariants for the values in registers and in memory.**

**CFG Reconstruction**
- Determines invariants on the control flow, by
  - Determining loop bounds,
  - Identifying infeasible paths.

**Value Analysis**
- Determines possible microarchitectural executions.

**Control Flow Analysis**
- Determines a worst-case path and an upper bound on the WCET.

**Micro-architectural Analysis**
- Determines a worst-case path and an upper bound on the WCET.

**Global Bound Analysis**
- Determines a worst-case path and an upper bound on the WCET.

**WCET Bound**
Control-Flow Analysis

R1 = R1 % 5
R2 = 42
R1 < R2?
R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?
R1 = R1 + 2
R1 = R1 + 1
return R1

R1 = [0, 41]
R2 = [42, 42]
R1 increases by at least 1 in every iteration
⇒ Can enter loop at most 42 times

There are multiple approaches to control-flow analysis. Not the focus of this course.

Can we also come up with a lower bound?
Structure of WCET Analyzers

Reconstructs a control-flow graph from the binary.

Determines invariants for the values in registers and in memory.

Determines invariants on the control flow, by
- Determining loop bounds,
- Identifying infeasible paths.

Determines possible microarchitectural executions.

Determines a worst-case path and an upper bound on the WCET.
Microarchitectural Analysis

Ideal 1970s world: one instruction = one cycle

*Today:*
- Pipelining
- Branch prediction + speculative execution
- Caches
- DRAM-based main memory

- Execution time of individual instruction highly variable and dependent on state of microarchitecture
- Need to analyze in which states the microarchitecture may be in when executing an instruction
Pipelining

- Instruction execution is split into several stages
- Several instructions can be executed in an overlapped fashion
- Some processors can start more than one instruction per cycle: VLIW, Superscalar
- Some processors can execute instructions out-of-order
Hardware Features: Pipelines

*Ideal Case*: One Instruction per Cycle, but there are Hazards!
Pipeline Hazards

- **Data Hazards**: Operands not yet available (Data Dependences)
- **Resource Hazards**: Consecutive instructions use same resource
- **Control Hazards**: Conditional branch
- **Instruction-Cache Hazards**: Instruction fetch causes cache miss
- **Data-Cache Hazards**: Load causes cache miss

Assuming worst case everywhere is not an option; it would be too pessimistic!

➔ Have to statically analyze the possible microarchitectural behaviors.
Basis of Microarchitectural Analysis: View of Processor as a State Machine

- Processor (pipeline, cache, registers, memory) viewed as a big state machine, performing transitions every clock cycle.
- Starting in an initial state for an instruction, transitions are performed, until a final state is reached:
  - final state: instruction has left the pipeline
  - # transitions: execution time of instruction
- Transitions may be annotated with events indicating e.g. a bus access, or a cache miss.
View of Processor as a Big State Machine

Initial states

Final states

WCET = 9

Can associate microarchitectural states with instructions in program.

R1 = R1 % 5
R2 = 42
R1 < R2 ?
R3 = MEM[a+R1]
R4 = MEM[a+R1+4]
R3 < R4?
R1 = R1 + 2
R1 = R1 + 1

return R1

WCET = 9

Can associate microarchitectural states with instructions in program.
View of Processor as a Big State Machine

State space of machine is **too large** to explore explicitly.

→ Need for **sound** and **compact** approximation.

Can associate microarchitectural states with instructions in program.
Abstracted State Machine

State space is product of:
- “microarchitectural state”, i.e. pipeline and cache state, and
- “program state”, i.e., register and memory contents including the program inputs

**First Abstraction:**
Discard program state (which is dealt with in control-flow analysis)

**Second Abstraction:**
Find abstract domains that compactly represent large sets of concrete microarchitectural states
How to Achieve “Sound Approximation”? Abstract Interpretation in a Nutshell

1. Every abstract state $s^\#$ represents a set of $\text{conc}(s^\#)$ concrete states:
How to Achieve “Sound Approximation”? Abstract Interpretation in a Nutshell

2. Local Consistency:
The successors of the concretization of an abstract state $s^\#$ are represented by $s^\#'$s successors:
How to Achieve “Sound Approximation”? Abstract Interpretation in a Nutshell

Concrete State Machine

Abstracted State Machine

Local Consistency

Local Consistency

sound approximation
Consequences of Abstraction: Nondeterminism

Nondeterminism:
In contrast to the concrete model, in the abstract model, one state can have several successor states.

Each abstract state represents a set of concrete states, which may have different successor states.
E.g. one may result in a cache hit, the other in a cache miss.

Consequences:
→ The abstract execution graph includes spurious executions, which leads to overapproximation of the WCET
→ There is a tradeoff between analysis cost and precision
Consequences of Abstraction: Cycles

*Cyclicity*: The abstract model may have cycles.
This is due to abstraction from the “program state”. E.g. abstract states do not capture the value of variables in a loop.

**Consequences:**
- The abstract execution graph alone cannot be used to derive any WCET bound
- Need to combine information with control-flow analysis results
Structure of WCET Analyzers

- **Input Executable**

- **CFG Reconstruction**
  - Reconstructs a control-flow graph from the binary.

- **Value Analysis**
  - Determines invariants for the values in registers and in memory.

- **Control Flow Analysis**
  - Determines invariants on the control flow, by
    - Determining loop bounds,
    - Identifying infeasible paths.

- **Micro-architectural Analysis**
  - Determines possible microarchitectural executions.

- **Global Bound Analysis**
  - Determines a worst-case path and an upper bound on the WCET.

- **WCET Bound**
Global Bound Analysis aka Path Analysis aka Implicit Path Enumeration

- Determines a worst-case path and an upper bound on the WCET.
- Formulated as integer linear program (ILP).
Integer linear programming

Linear programming (LP)

Objective function → maximize $c^T x$

Linear constraints → subject to $Ax \leq b$

and $x \geq 0$

... + Restriction to integers = ILP.

LP is in polynomial time, yet, ILP is NP hard, but often efficiently solvable in practice.

Solvers (e.g. CPLEX) determine the maximal value of the objective function + corresponding valuation of variables.
Global Bound Analysis
aka Path Analysis aka Implicit Path Enumeration

Determines a worst-case path through the abstract execution graph and an upper bound on the WCET:

- Introduce a variable for each edge in abstract execution graph to capture how often this edge is taken
- Encode structure of graph via linear constraints
- Encode loop bounds and other infeasible path information via linear constraints

\[
\text{max } x_a + x_b + x_c + \ldots
\]
\[
\text{s.t. } \text{Structural Constraints} \\
\text{Infeasible Path Constraints} \\
\text{Loop Bound Constraints}
\]
Global Bound Analysis: Small Example

\[ \begin{align*}
\text{max } x_a + x_b + x_c + x_d + x_e + x_f \\
\text{s.t. } x_a &= 1 \\
   x_a &= x_b \\
   x_b + x_f &= x_c \\
   x_c &= x_d \\
   x_d &= x_e + x_f \\
   x_c &\leq 5
\end{align*} \]

Loop Bound Constraint

Solution:
\[ \begin{align*}
   x_a &= x_b = x_e = 1 \\
   x_c &= x_d = 5 \\
   x_f &= 4
\end{align*} \]

\[ x_a + x_b + x_c + x_d + x_e + x_f = 17 \]
Summary and Outlook

- Separate Analysis into SW and HW aspects:
  - SW: Control-flow Analysis
  - HW: Microarchitectural Analysis
  - Combine results using Integer Linear Program

- Abstraction:
  - Employ sound abstractions to solve undecidable problems approximately
  → will see such an abstraction for caches next
Literature (very incomplete)

WCET Analysis:

Loop Bounds: