Design and Analysis of Time-Critical Systems

WCET Analysis: A Primer

Jan Reineke @ SAARLAND UNIVERSITY

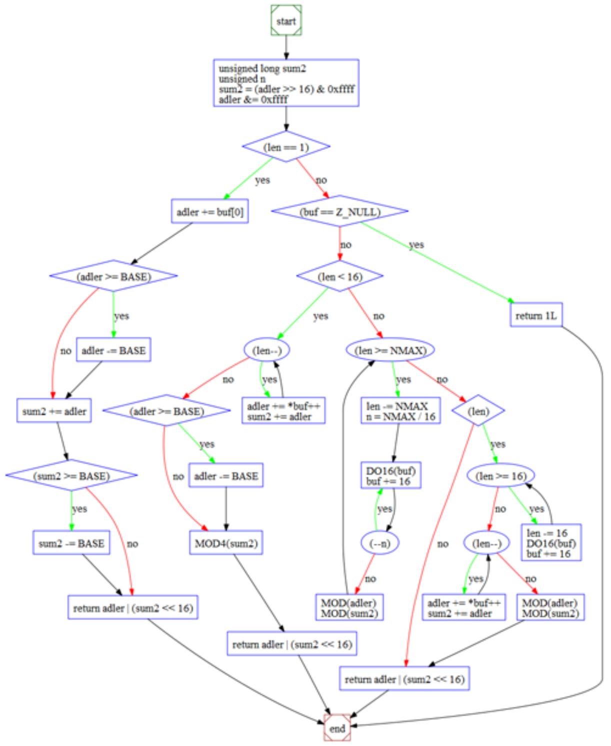
COMPUTER SCIENCE

ACACES Summer School 2017

Fiuggi, Italy

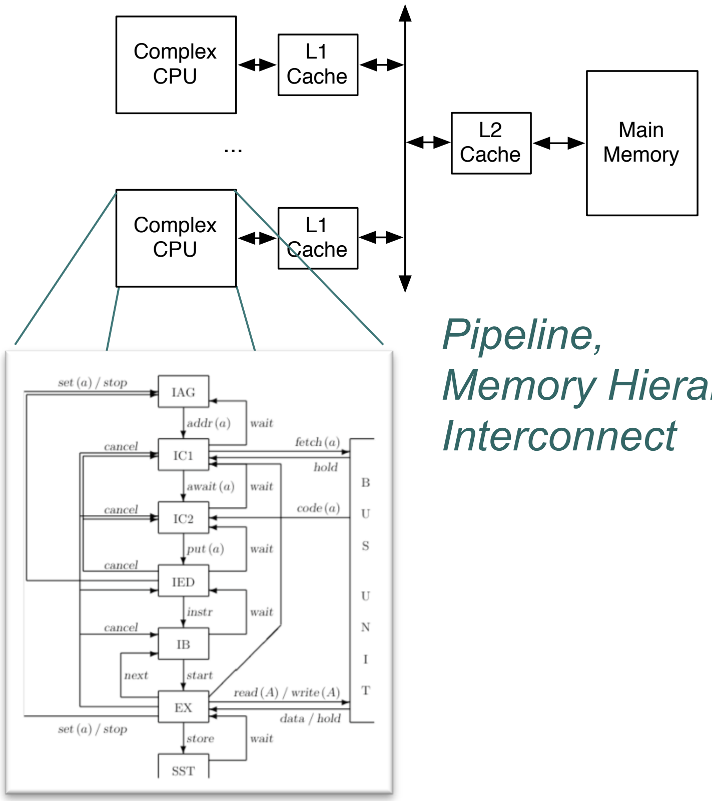
What does the execution time of a program depend on, on a single-core machine?

Input-dependent control flow



+

Microarchitectural State



Pipeline, Memory Hierarchy, Interconnect

The Single-core WCET Analysis Problem

$$WCET_H(P) := \underset{i \in Inputs}{\overset{\substack{\text{Consider all possible}\\ \text{program inputs}\\ \downarrow}}{\max}} \ \underset{h \in States(H)}{\overset{\substack{\text{Consider all possible initial}\\ \text{states of the hardware}\\ \downarrow}}{\max}} \ ET_H(P, i, h)$$

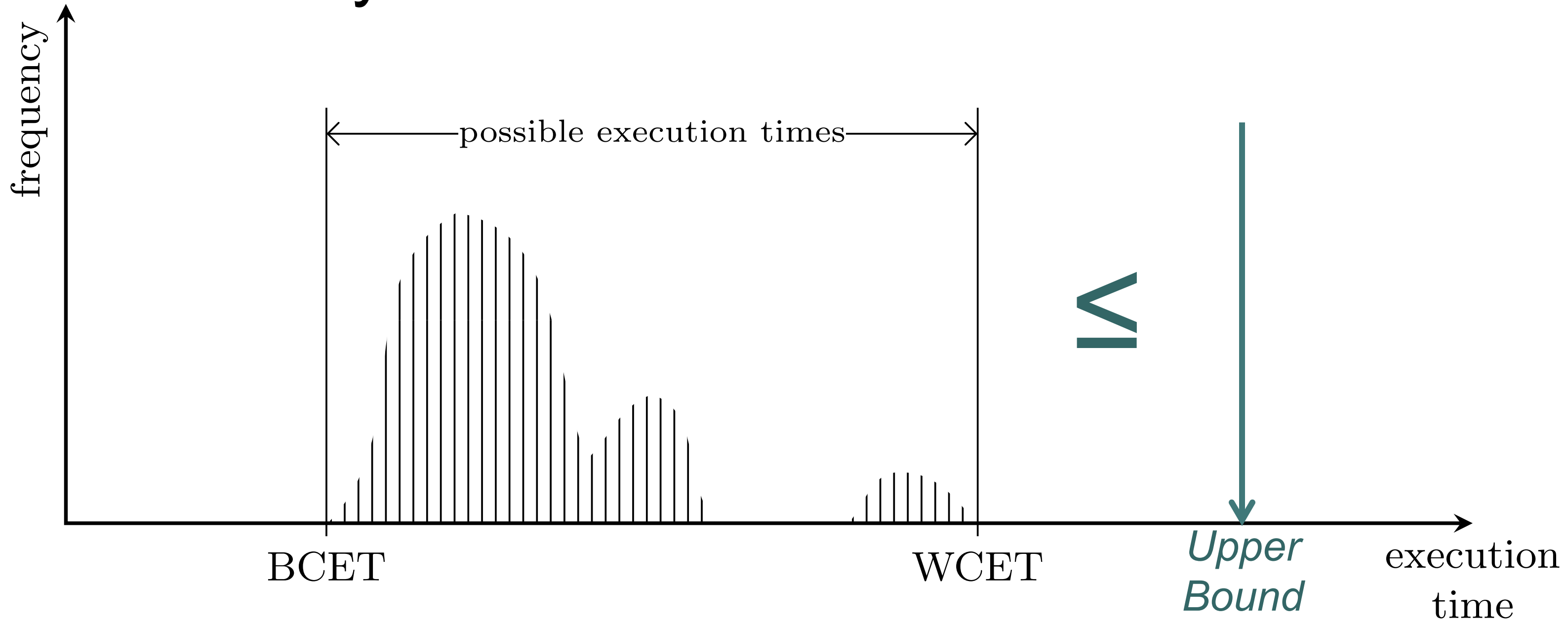
Measuring or simulating the execution time for all inputs and all hardware states is not feasible in practice:

- *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:

Depends on hardware

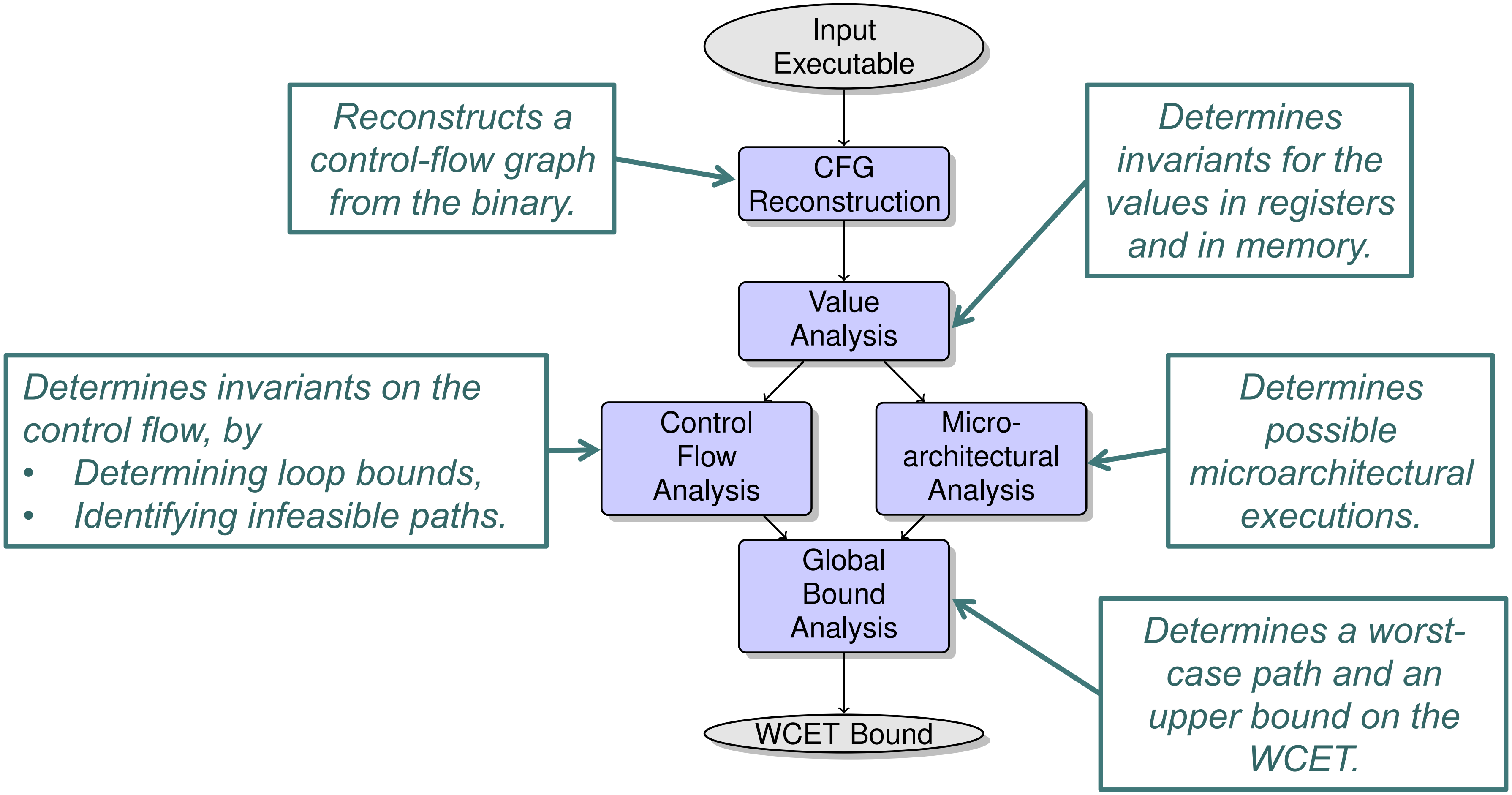
1. Bound possible microarchitectural executions using abstractions.

Depends on program semantics

2. Determine constraints on control flow (e.g. loop bounds) through program by abstractions.

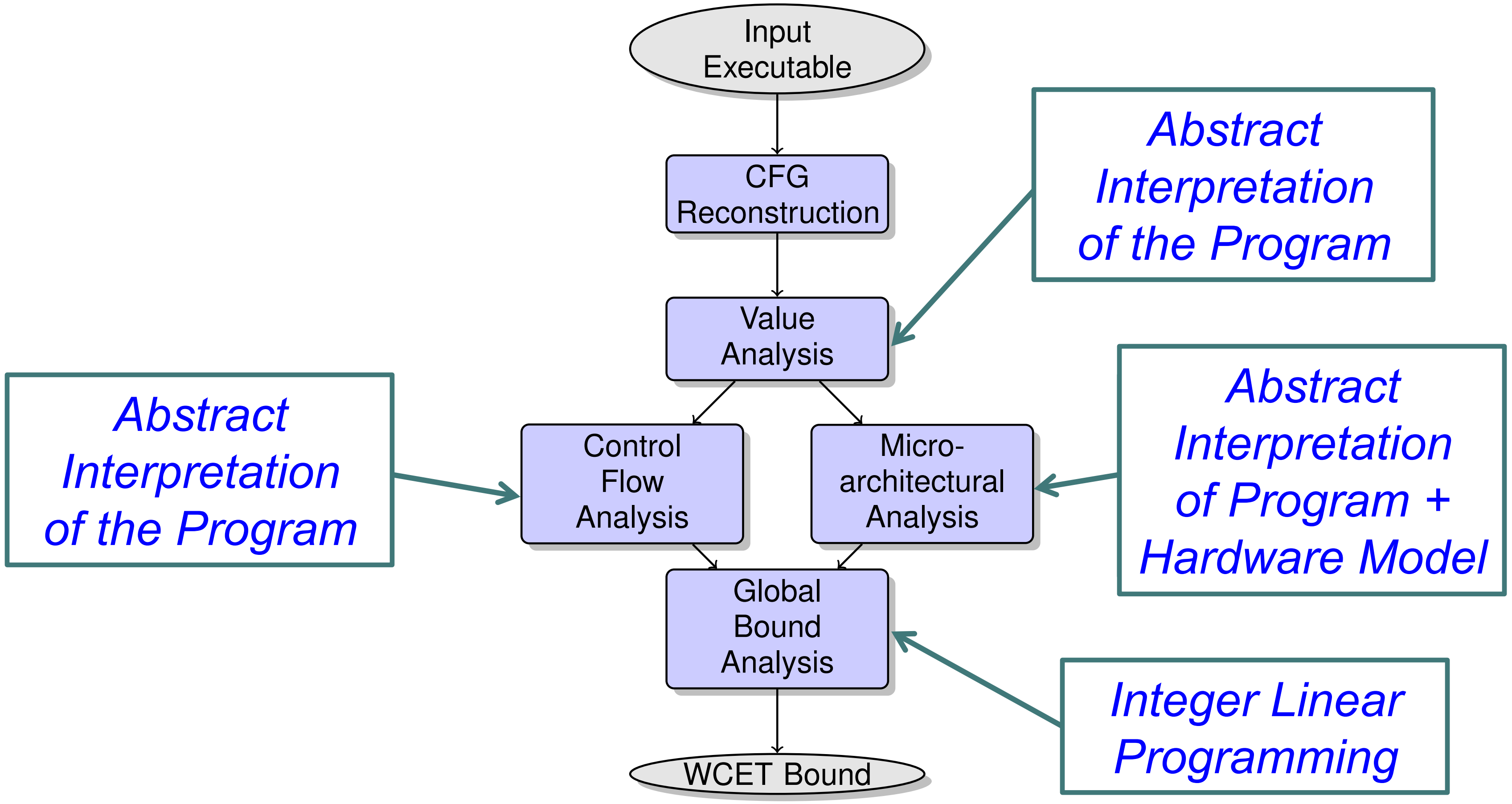
- Combination: combine 1 and 2 to bound execution time of the whole program.

Structure of WCET Analyzers



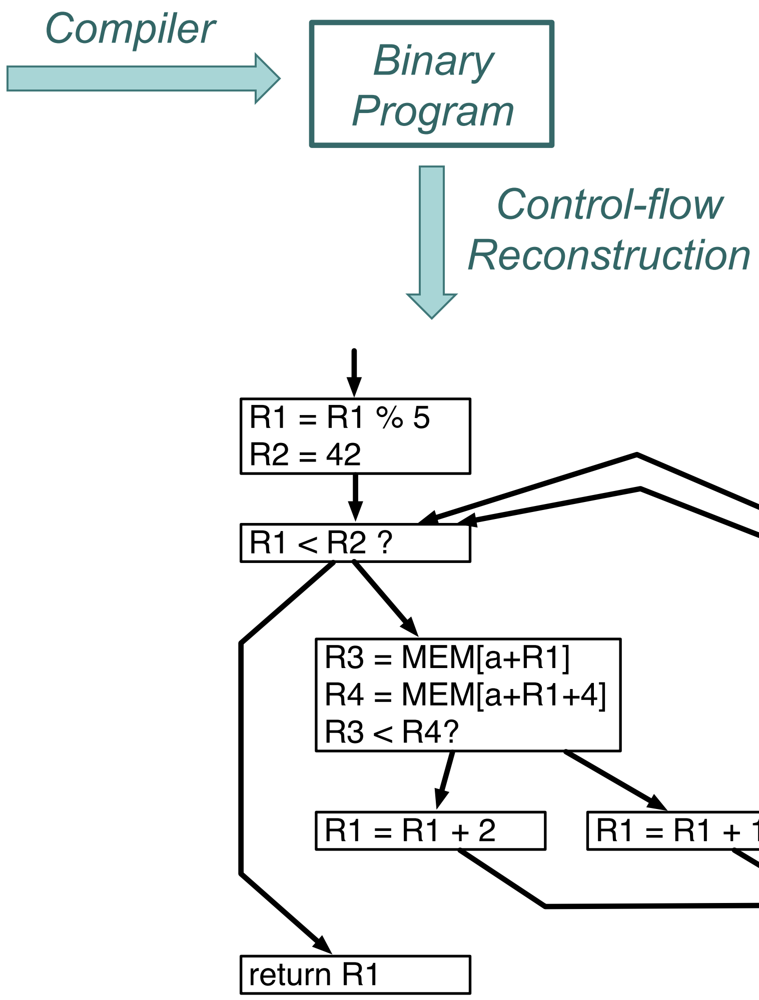
Input Executable
CFG Reconstruction
Value Analysis
Control Flow Analysis
Micro-architectural Analysis
Global Bound Analysis
WCET Bound
Reconstructs a control-flow graph from the binary.
Determines invariants for the values in registers and in memory.
Determines invariants on the control flow, by
• Determining loop bounds,
• Identifying infeasible paths.
Determines possible microarchitectural executions.
Determines a worst-case path and an upper bound on the WCET.

Structure of WCET Analyzers
Employed Techniques

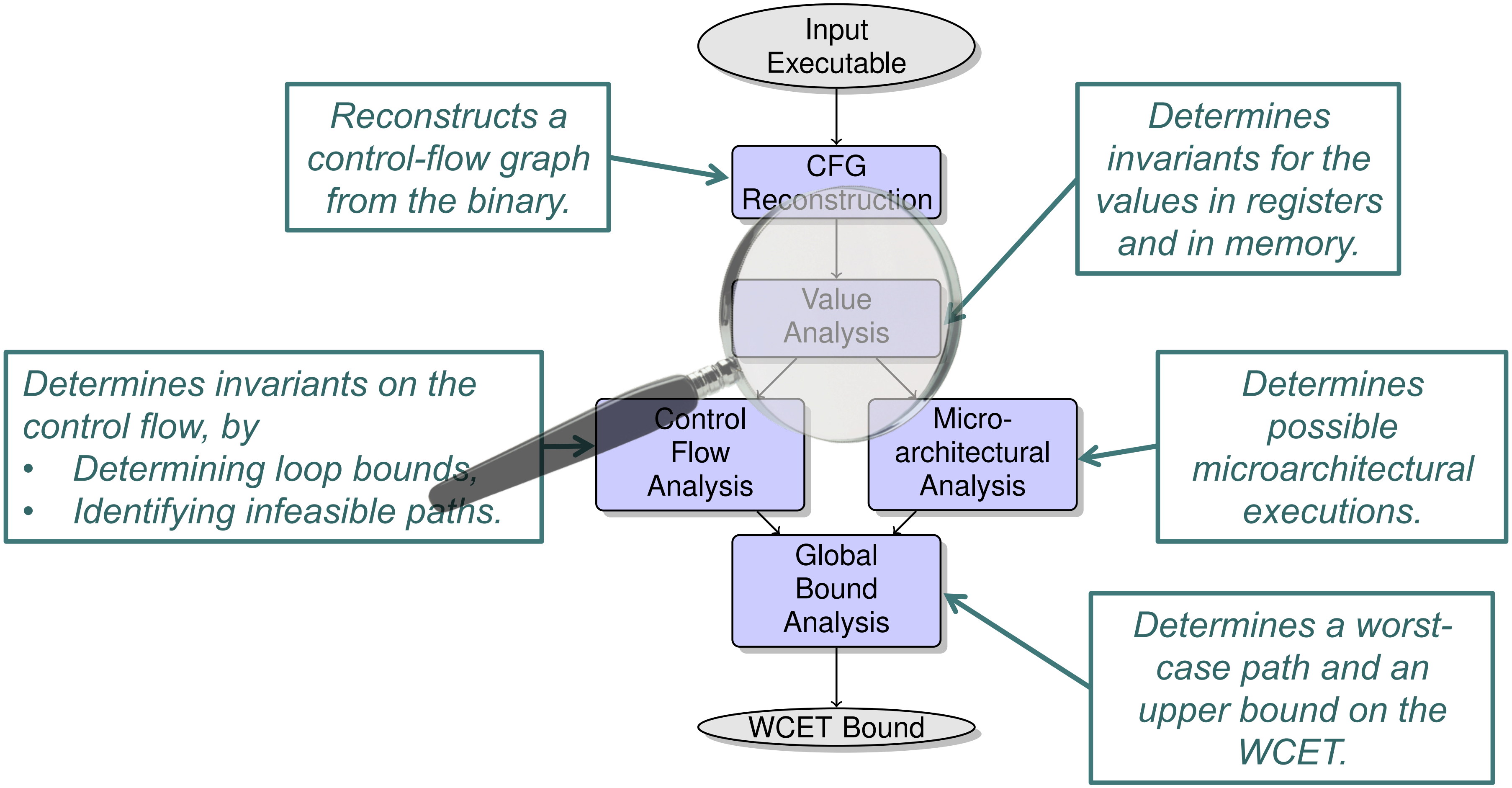

Input Executable
CFG Reconstruction
Value Analysis
Control Flow Analysis
Micro-architectural Analysis
Global Bound Analysis
WCET Bound
Abstract Interpretation of the Program
Abstract Interpretation of the Program
Abstract Interpretation of Program + Hardware Model
Integer Linear Programming

Running Example

```
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 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.

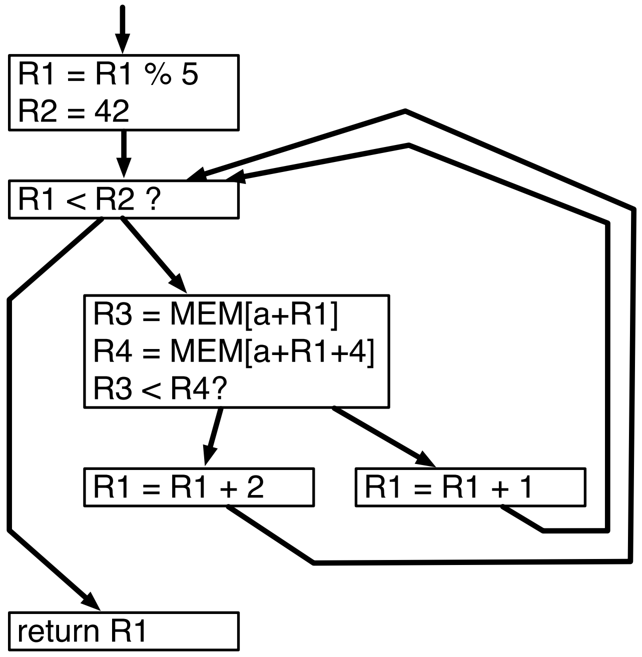
Determines a worst-case path and an upper bound on the WCET.

Value Analysis

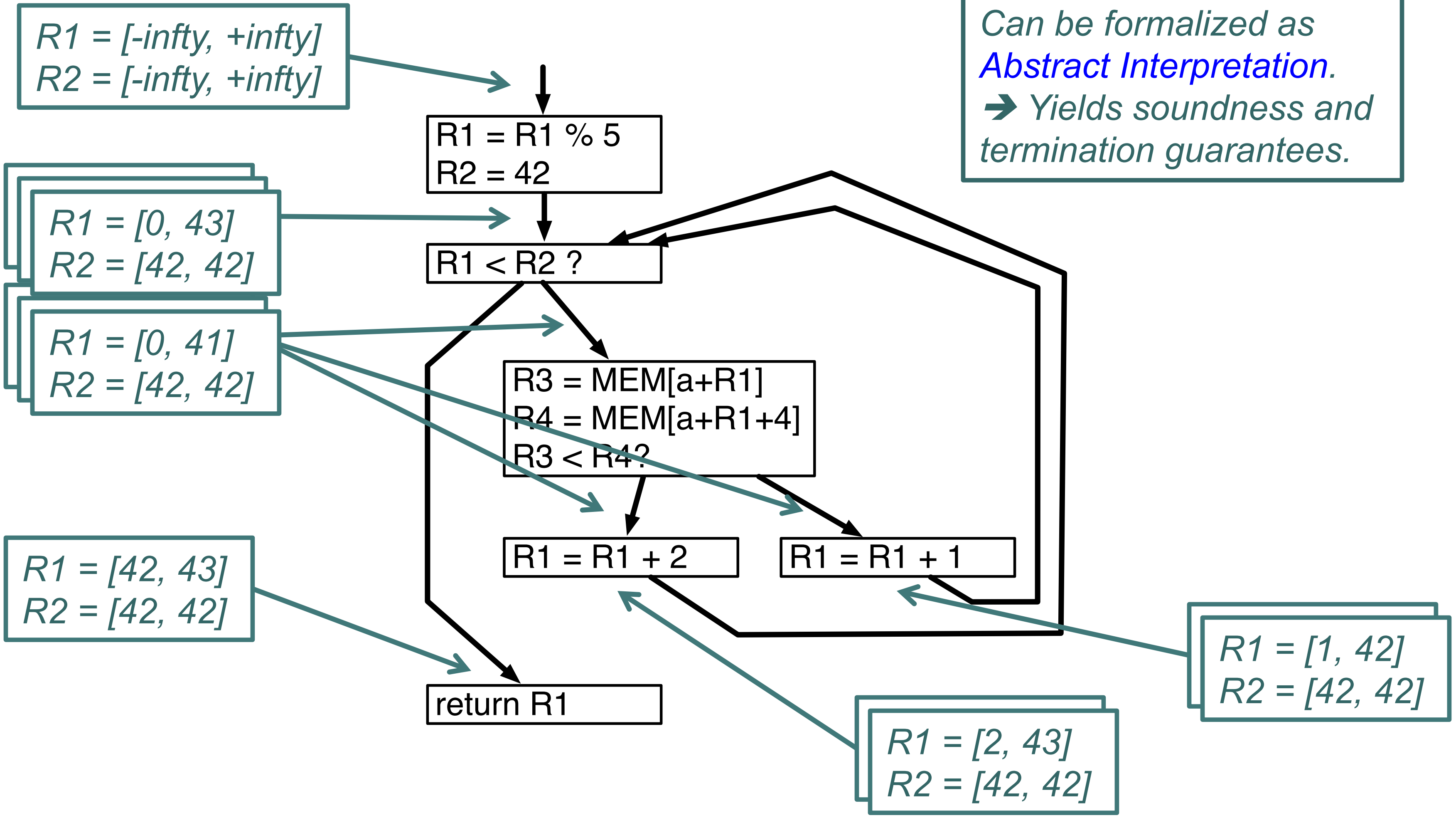
Determines invariants on values of registers at different program points. Invariants are often in the form of enclosing intervals of all possible values.

Where is this information used?

- Microarchitectural Analysis
  - Pipeline Analysis
  - Cache Analysis
- Control-Flow Analysis
  - Detect infeasible paths
  - Derive loop bounds

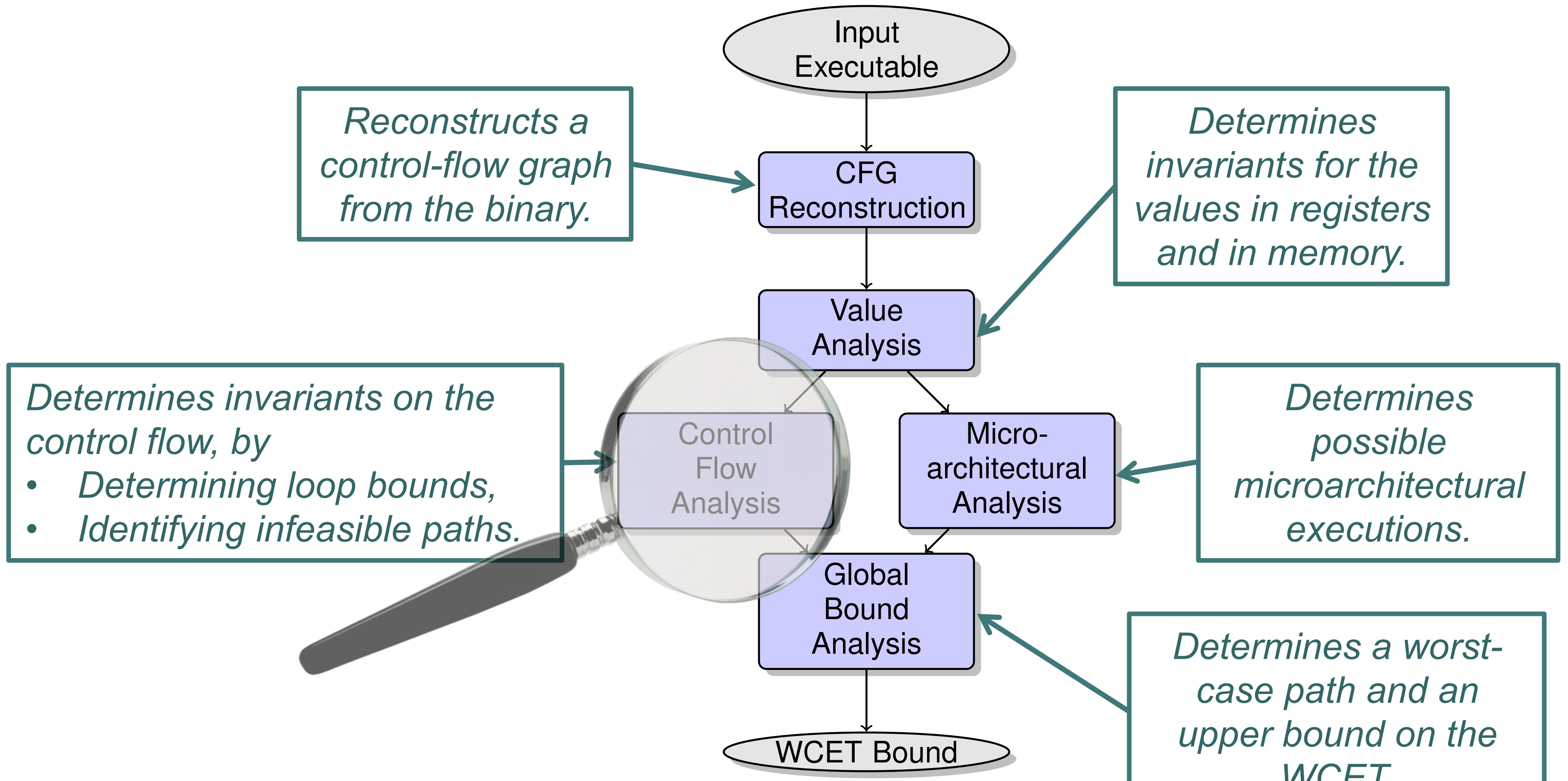


Value Analysis
Intuition of Interval Analysis



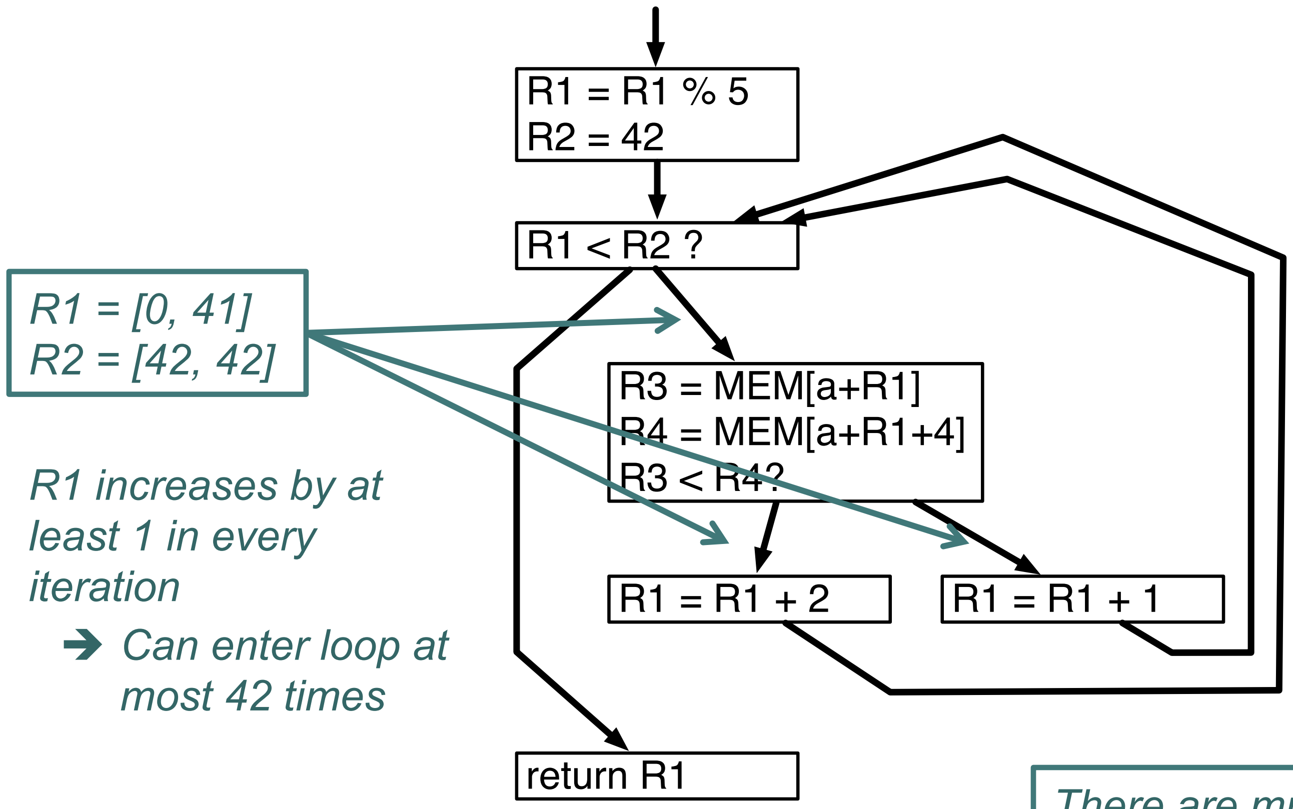
Can be formalized as Abstract Interpretation.
➔ Yields soundness and termination guarantees.

Structure of WCET Analyzers



Input Executable
CFG Reconstruction
Value Analysis
Control Flow Analysis
Micro-architectural Analysis
Global Bound Analysis
WCET Bound
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.

Control-Flow Analysis



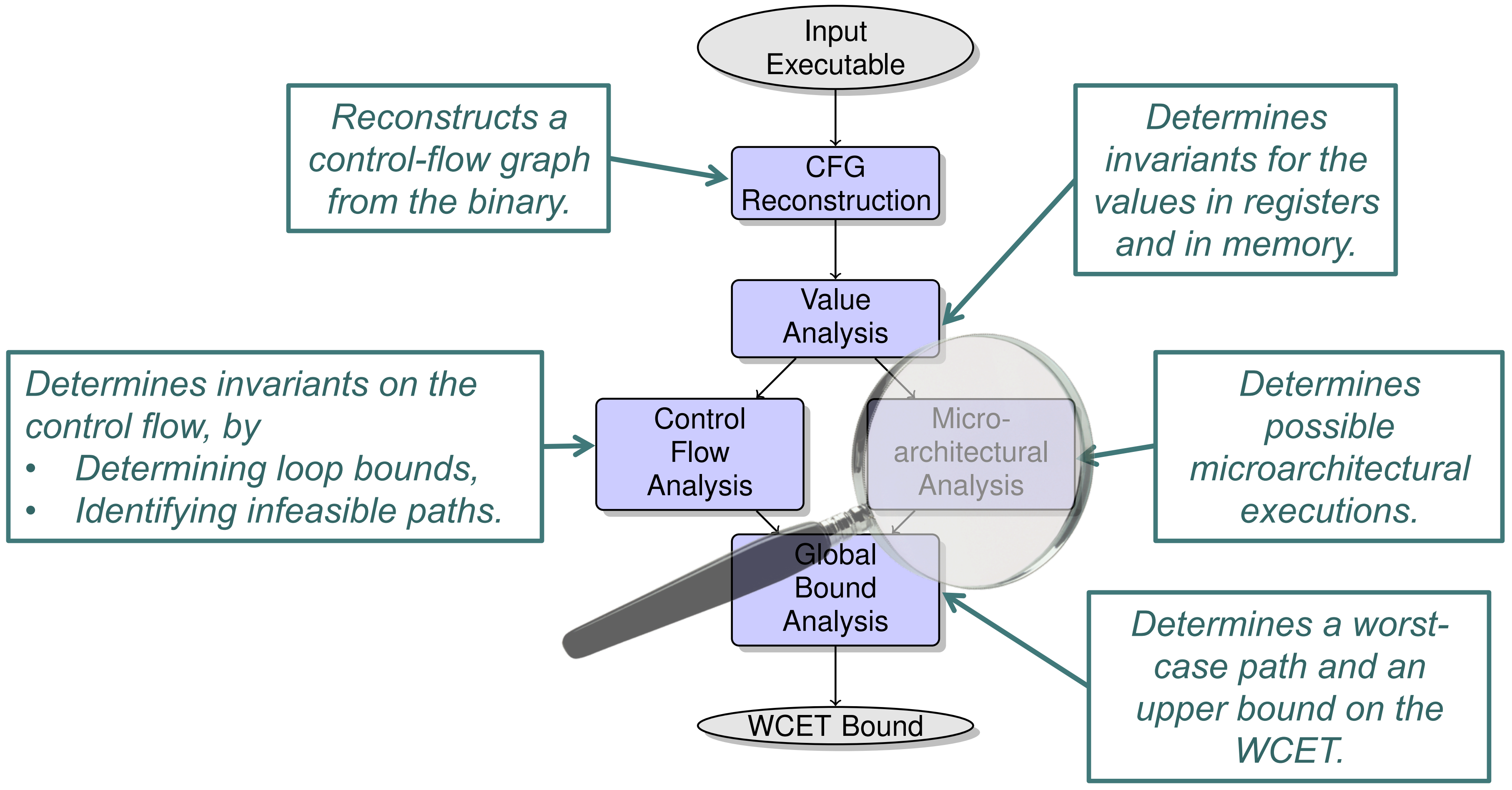
R1 increases by at least 1 in every iteration

➔ *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

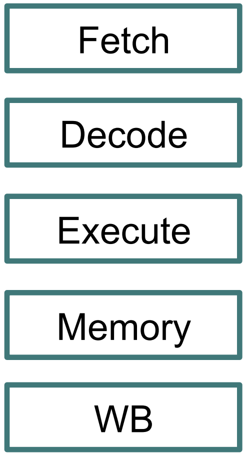
→ Execution time of individual instruction highly variable and dependent on state of microarchitecture

→ Need to analyze in which states the microarchitecture may be in when executing an instruction

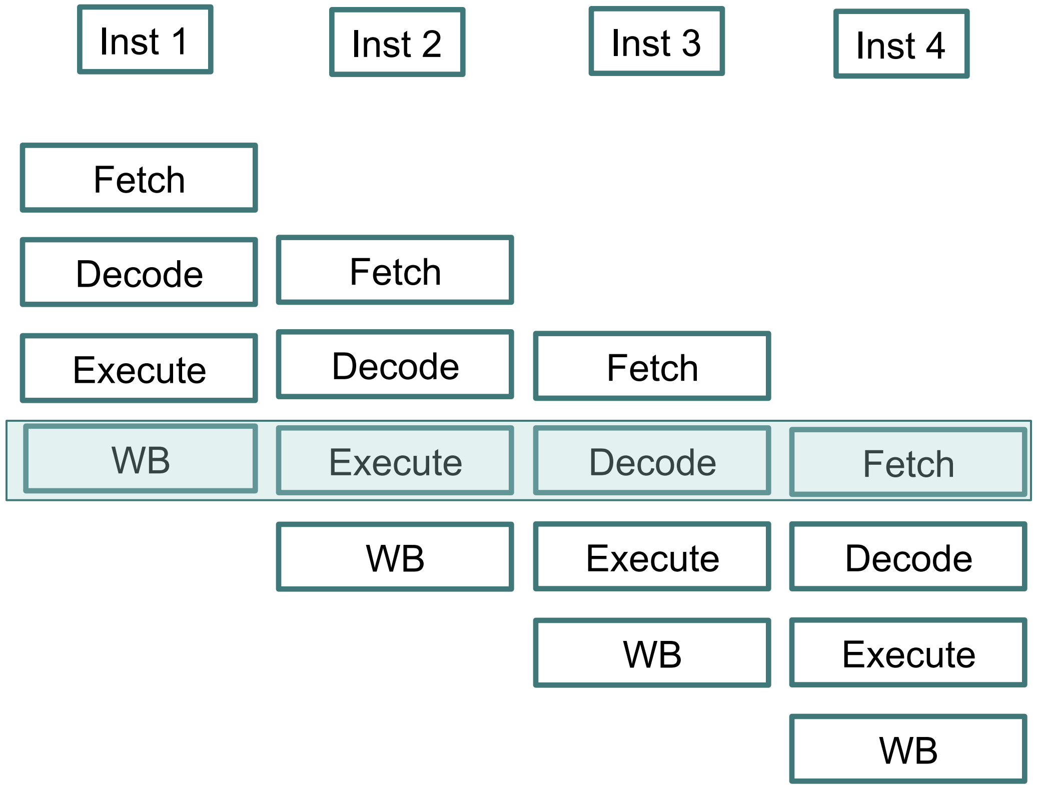
Pipelining

- 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

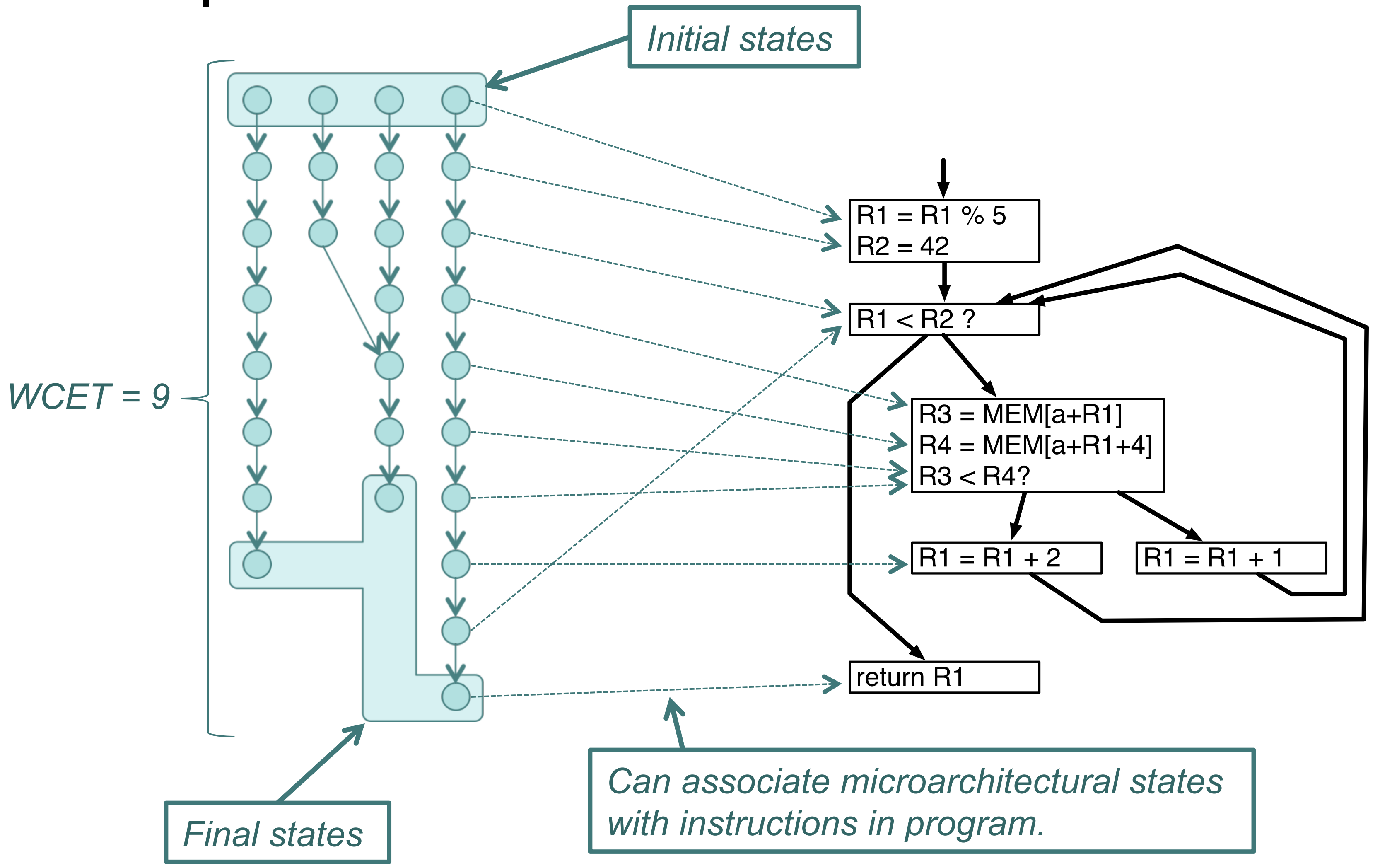
Assuming worst case everywhere is not an option; it would be too pessimistic!

→ *Have to statically analyze the possible microarchitectural behaviors.*

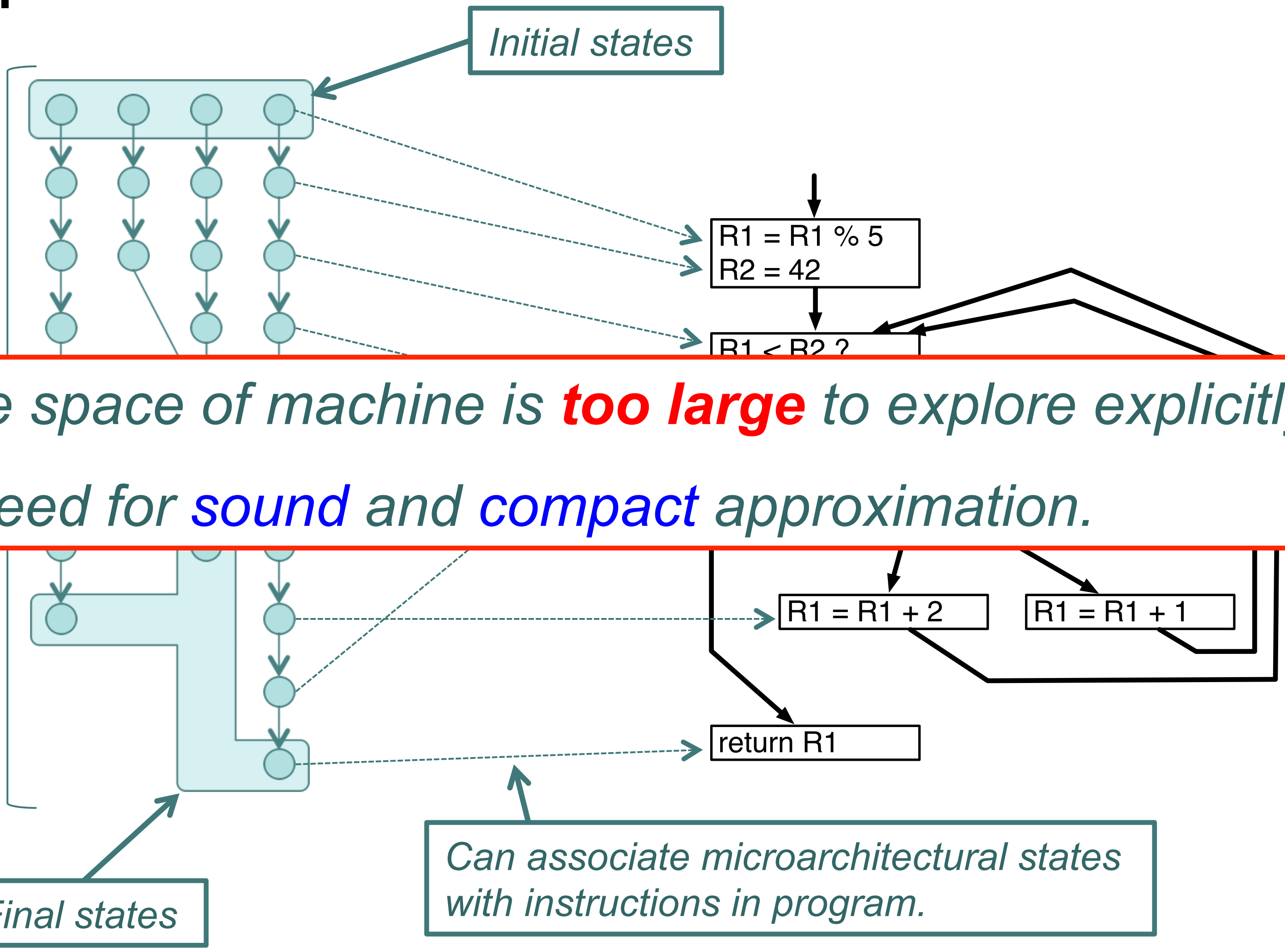
Basis of Microarchitectural Analysis: View of Processor as a State Machine

- Processor ($\overbrace{\text{pipeline, cache,}}^{\textit{“μarchitectural state”}}$ $\overbrace{\text{registers, memory}}^{\textit{“program state”}}$) 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



View of Processor as a Big State Machine



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

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

Abstracted State Machine

State space is product of

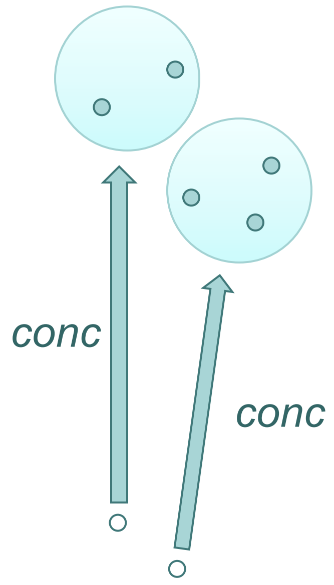
- "microarchitectural state", i.e. pipeline and cache state, and
- "program state", i.e., register and memory contents including the program inputs

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

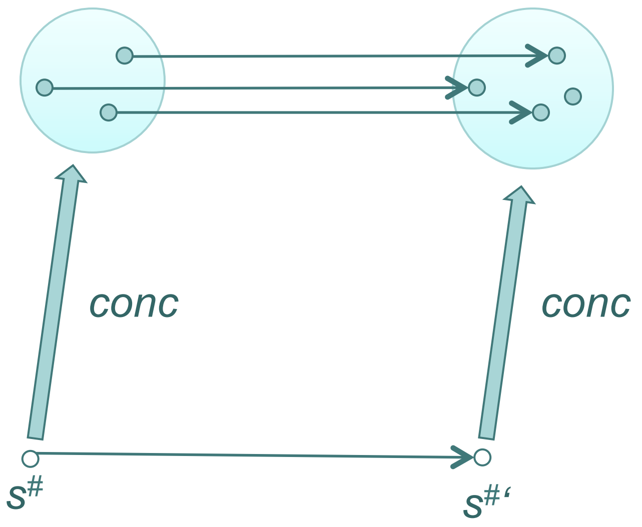
1. Every abstract state $s^\#$ represents a set of $conc(s^\#)$ concrete states:



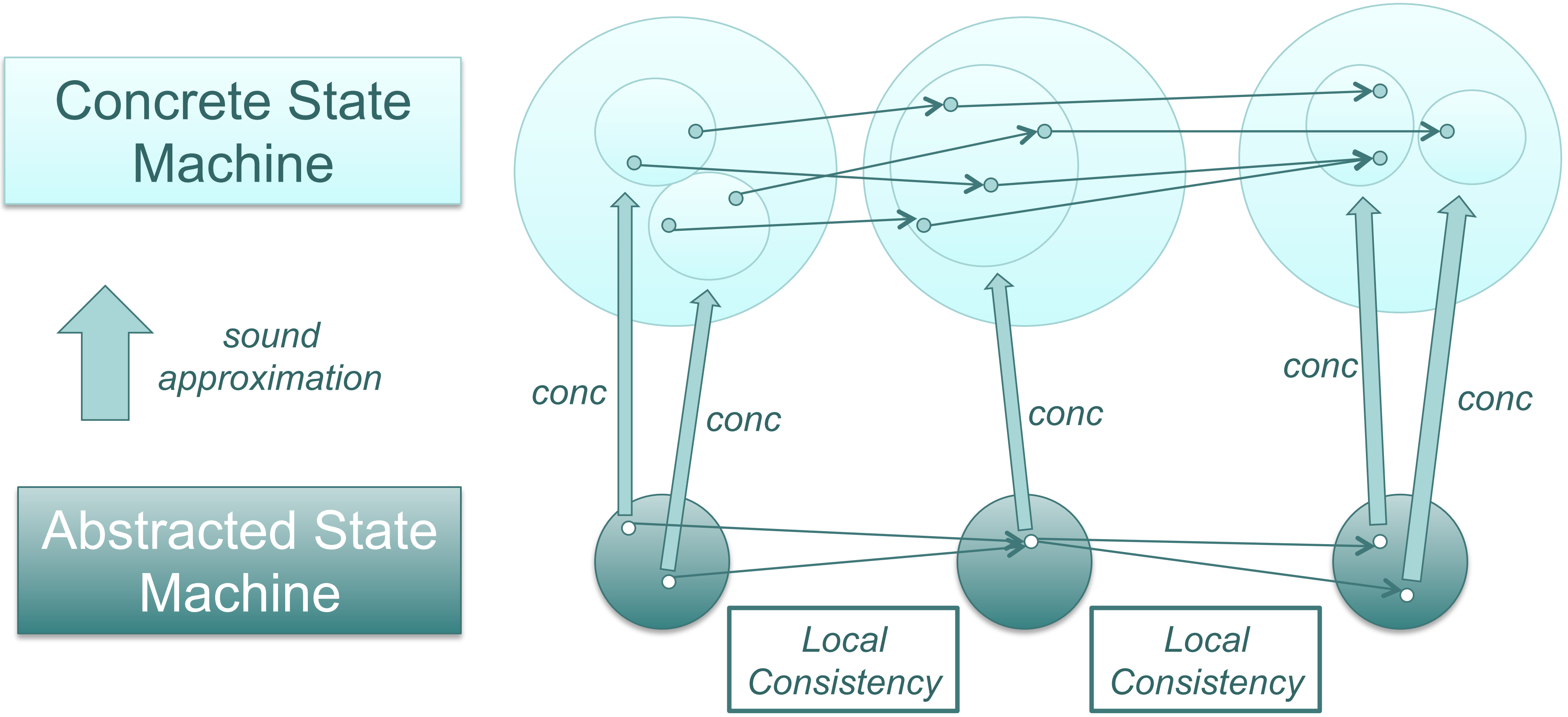
How to Achieve "Sound Approximation"? Abstract Interpretation in a Nutshell

2. ***Local Consistency***:

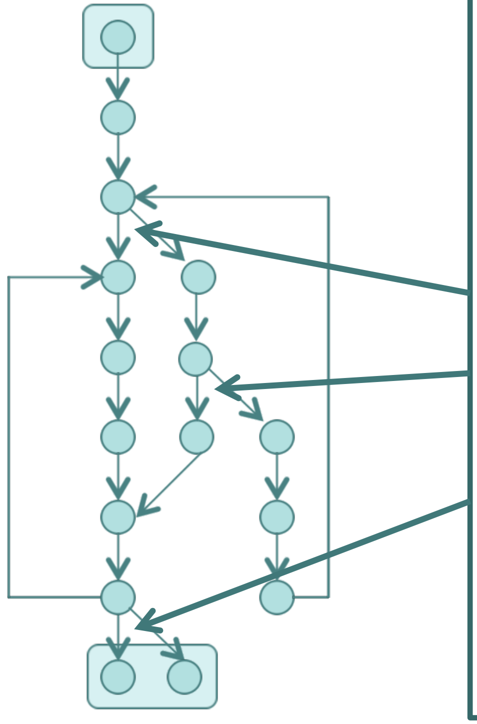
The successors of the concretization of an abstract state $s^{\#}$ are represented by $s^{\#}$'s successors:



How to Achieve "Sound Approximation"? Abstract Interpretation in a Nutshell



Consequences of Abstraction: Nondeterminism

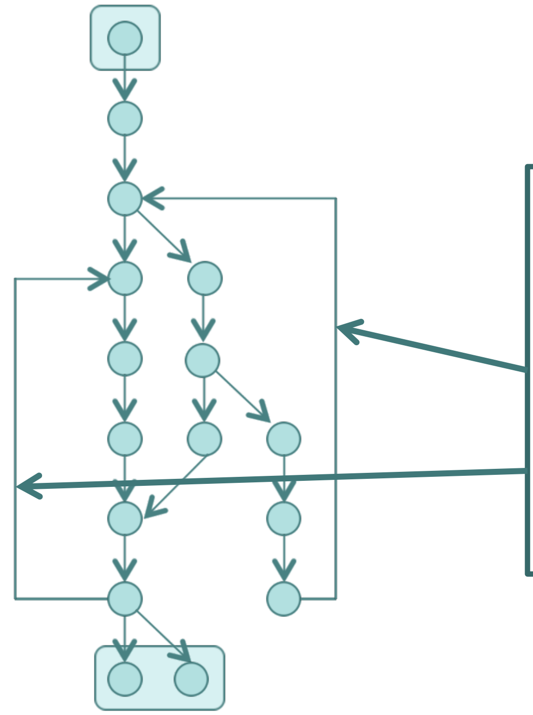


Nondeterminism:
In contrast to the concrete model, in the abstract model, one state can have several successor states.

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

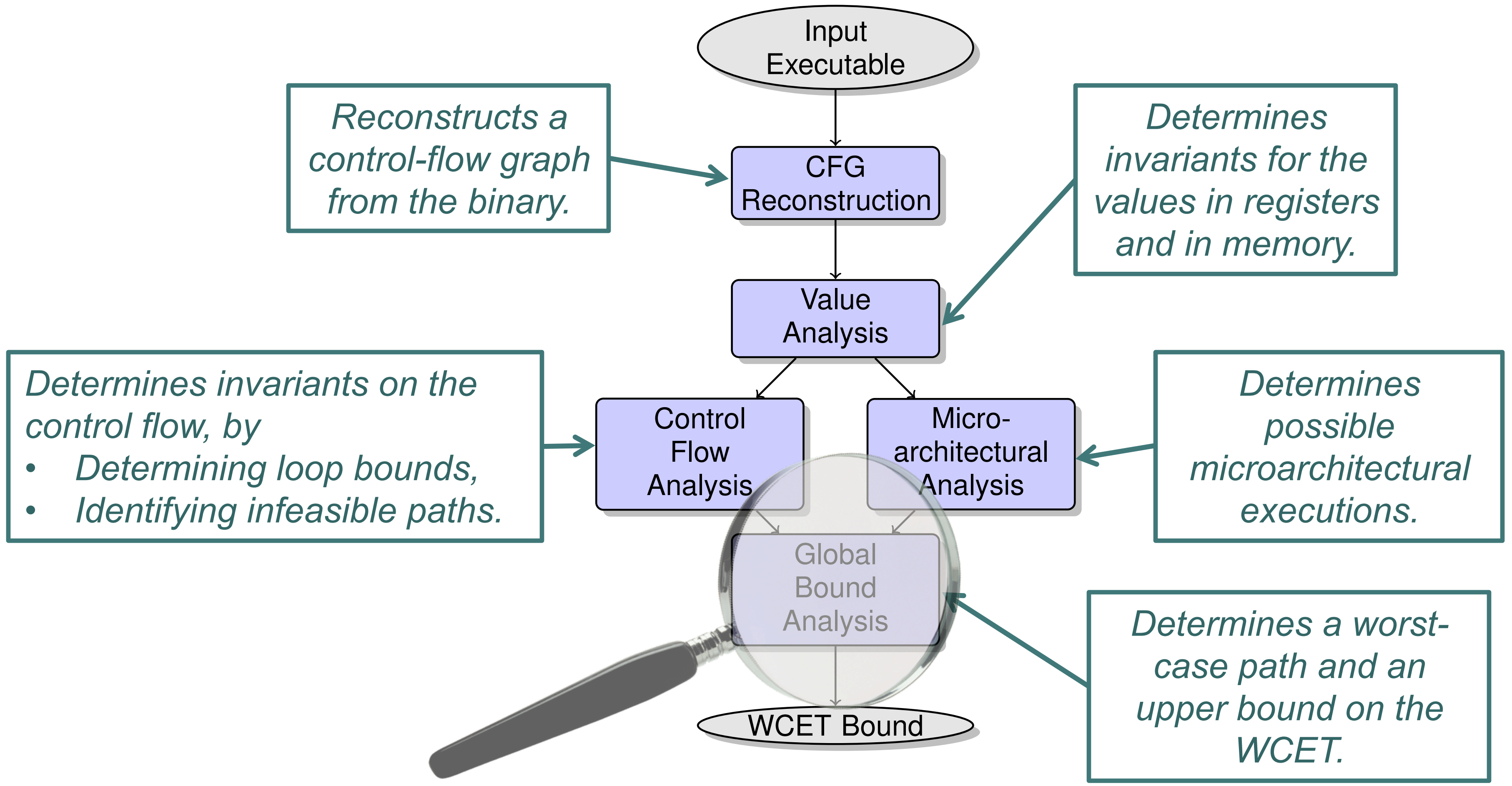


Cyclicity:
The abstract model may have cycles.

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

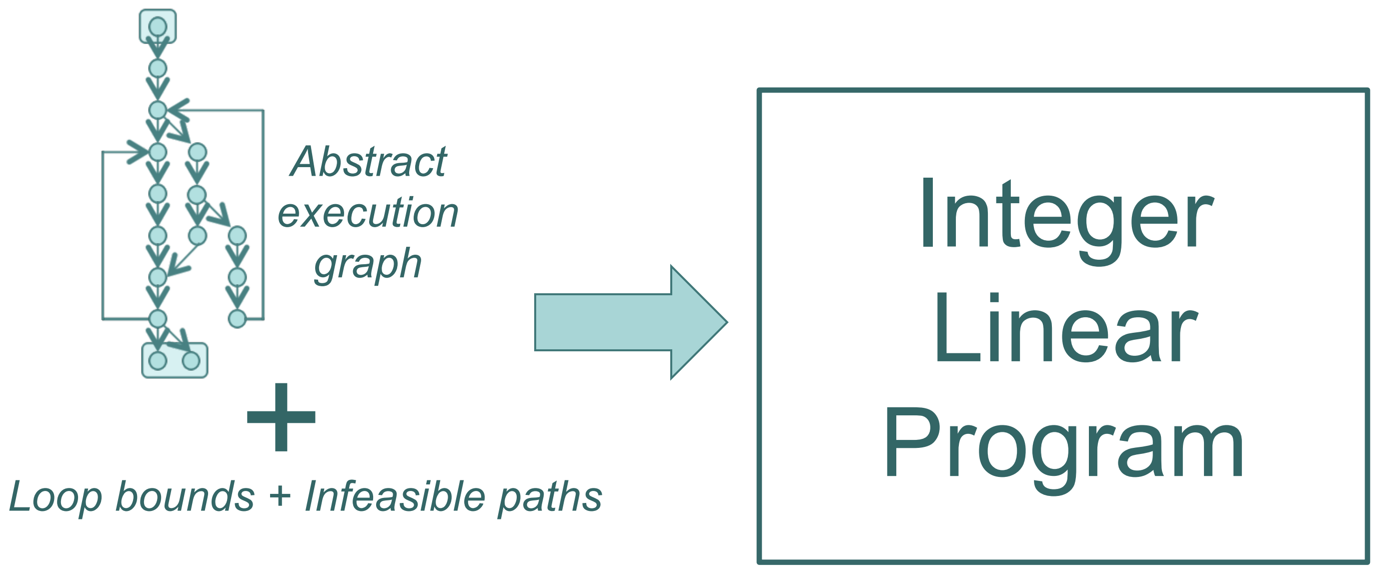
Structure of WCET Analyzers



Input Executable
CFG Reconstruction
Value Analysis
Control Flow Analysis
Micro-architectural Analysis
Global Bound Analysis
WCET Bound
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.

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 $\longrightarrow$ $\mathit{maximize}\ \ c^T x$

Linear constraints $\longrightarrow$ $\mathit{subject\ to}\ \ Ax \le b$

$\mathit{and}\ \ x \ge 0$

… + Restriction to integers = ILP.

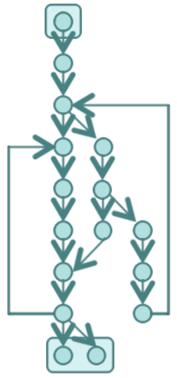
LP is in polynomial time, yet, ILP is NP hard,
but often efficiently solvable in practice.

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



Abstract execution graph

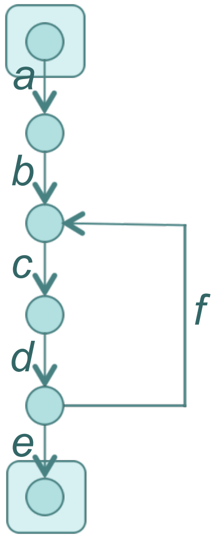
+

Loop bounds + Infeasible paths

Integer linear program:

$\max x_a + x_b + x_c + \ldots$
s.t. *Structural Constraints*
Infeasible Path Constraints
Loop Bound Constraints

Global Bound Analysis: Small Example



+

Loop Bound = 5

$$\max x_a + x_b + x_c + x_d + x_e + x_f$$

$$\text{s.t.}\quad \left.\begin{aligned} x_a &= 1 \\ x_a &= x_b \\ x_b + x_f &= x_c \\ x_c &= x_d \\ x_d &= x_e + x_f \end{aligned}\right\} \text{Structural Constraints}$$

$$x_c <= 5 \quad \text{Loop Bound Constraint}$$

Solution:

$x_a = x_b = x_e = 1$

$x_c = x_d = 5$

$x_f = 4$

$\rightarrow x_a + x_b + x_c + x_d + x_e + x_f = 17$

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:

- Li, Malik: Performance analysis of embedded software using implicit path enumeration, In: Proceedings LCTRTS, 1995
- Ferdinand et al.: Reliable and Precise WCET Determination for a Real-Life Processor, In: Proceedings EMSOFT, 2001
- Stephan Thesing. Safe and Precise WCET Determination by Abstract Interpretation of Pipeline Models. PhD thesis, Saarland University, 2004
- Ingmar Jendrik Stein. ILP-based Path Analysis on Abstract Pipeline State Graphs. PhD thesis, Saarland University, 2010

Loop Bounds:

- Cullmann, Martin: Data-flow based detection of loop bounds, In: Proceedings WCET, 2007
- Ermedahl, Sandberg, Gustafsson, Bygde, Lisper: Loop bound analysis based on a combination of program slicing, abstract interpretation, and invariant analysis, In: Proceedings WCET 2007
- De Michiel, Bonenfant, Casse, Sainrat: Static Loop Bound Analysis of C Programs Based on Flow Analysis and Abstract Interpretation, In: Proceedings RTCSA, 2008