Verification of Real-Time Systems
Global Bound Analysis aka Path Analysis

Jan Reineke

Advanced Lecture, Summer 2015
Structure of WCET Analyzers

- **Input Executable**
  - **CFG Reconstruction**
    - Reconstructs a control-flow graph from the binary.
  - **Value Analysis**
    - Determines invariants on the control flow, by
      - Determining loop bounds,
      - Identifying infeasible paths.
  - **Control Flow Analysis**
  - **Micro-architectural Analysis**
  - **Global Bound Analysis**

- **Determines bounds on execution times of program fragments.**
- **Determines a worst-case path and an upper bound on the WCET.**

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

**Jan Reineke**


April 2012

7 / 54
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 bounds on execution times of program fragments.
- 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
Global Bound Analysis aka Path Analysis

- Combines results of **control-flow analysis** and **microarchitectural analysis** to characterize all possible executions of a program on a given microarchitecture.

- Searches for **longest execution** among those deemed possible.
Result of Microarchitectural Analysis: Abstract Collecting Trace Semantics

Basic Block Execution Times (in cycles):
- BB0: 2 or 3
- BB1: 2 or 3
- BB2: 2 or 3
- BB3: 2
- BB4: 4
- BB5: 3
Result of Control-Flow Analysis

- Loop bounds: how often can the loop body be executed for each execution of the loop?
- Sometimes: infeasible paths, as e.g. in
  ```c
  if (a > 0) then
      fast();
  else
      slow(); // does not modify a
  if (a > 1) then
      slow();
  ```
“Traditional” Path Analysis

- Encode problem as (Integer) Linear Program
  - Introduce one variable $x_e$ for each edge $e$ in the control-flow graph that captures the execution frequency of that edge
  - Structural constraints: “Kirchhoff’s law”: inflow = outflow at every program point
  - Loop bounds and knowledge about infeasible paths as additional constraints
  - Objective function:
    \[
    \max_{e} \sum_{e} c_e x_e \\
    \text{s.t. structural constraints + loop bounds, etc. hold}
    \]
Traditional Path Analysis: Example Structural Constraints

Basic Block Execution Times (in cycles):

- BB0: 2 or 3
- BB1: 2 or 3
- BB2: 2 or 3
- BB3: 2
- BB4: 4
- BB5: 3

\[
x_0 = x_5 = 1 \\
x_0 + x_4 = x_1 + x_5 \\
x_1 = x_2 \\
x_2 = x_3 \\
x_3 = x_4
\]
Traditional Path Analysis: Example Loop Bounds

Basic Block Execution Times (in cycles):
- BB0: 2 or 3
- BB1: 2 or 3
- BB2: 2 or 3
- BB3: 2
- BB4: 4
- BB5: 3

\[ x_1 \leq n \times x_0, \text{ where } n \text{ is the loop bound} \]
Traditional Path Analysis: Example

Objective Function

Basic Block Execution Times (in cycles):
- BB0: 2 or 3
- BB1: 2 or 3
- BB2: 2 or 3
- BB3: 2
- BB4: 4
- BB5: 3

This can be pessimistic. Why?

\[
\max_{e} c_e x_e
\]

\[
= \max c_0 x_0 + c_1 x_1 + c_2 x_2 + c_3 x_3 + c_4 x_4 + c_5 x_5
\]

\[
= \max 3 x_0 + 3 x_1 + 3 x_2 + 2 x_3 + 4 x_4 + 3 x_5
\]
State-Sensitive Path Analysis aka “Prediction-File” based Path Analysis

Idea: Distinguish different microarchitectural paths if they exhibit different timing
→ Excludes impossible combinations of worst-case timings of different basic blocks

Approach:
- Microarchitectural states at the beginning of each basic block take the role of program points in the traditional analysis
- Introduce “frequency variable” for each non-dominated path from one such state to another.
State-Sensitive Path Analysis: Example Structural Constraints

What are the weights in the objective function?

Basic Block Execution Times (in cycles):
- BB0: 2 or 3
- BB1: 2 or 3
- BB2: 2 or 3
- BB3: 2
- BB4: 4
- BB5: 3

\[ \begin{align*}
X_{0,1}\rightarrow1,0 + X_{0,2}\rightarrow1,0 &= \\
X_{1,0}\rightarrow2,0 + X_{1,0}\rightarrow2,1 &= \\
X_{2,0}\rightarrow3,0 + X_{2,1}\rightarrow3,0 &= \\
X_{3,0}\rightarrow4,0 + X_{3,0}\rightarrow4,1 &=
\end{align*} \]
How to take into account cumulative information such as cache persistence?

Prohibits certain micro-architectural paths:

- If block $b$ is persistent, then at most one edge may be taken that corresponds to a miss to block $b$.
- Need to expose the information that an edge corresponds to a particular event, such as a cache miss to block $b$. 
Taking into account cumulative information: Cache Persistence Example

Introduce a variable $x_{b,\text{miss}}$ that counts the number of misses to $b$.

Add persistence constraints for $b$:

$$x_{b,\text{miss}} \leq 1 \text{ or } x_{b,\text{miss}} \leq x_{\text{scope}}$$

where $x_{\text{scope}}$ is the number of times the scope is entered in which $b$ is persistent.

Frequency of edges $e$ that correspond to misses to $b$ should not exceed $x_{b,\text{miss}}$:

$$\sum_{e} x_{e} \leq x_{b,\text{miss}}$$
Conclusions

High-level ideas of state-of-the-art path analysis:

- Encode all program paths implicitly by set of linear constraints.
- Objective function corresponds to cost of a particular path.
- Take into account microarchitectural states for higher precision → “State-sensitive path analysis”
- Expose events that can be bounded cumulatively, like cache misses.