HW/SW Codesign
AS 2014

WCET Analysis

3. December 2014

Mirela Botezatu
bmirela@student.ethz.ch
Outline

• First hour: discuss WCET analysis, revisit slides, ask doubts
• Second hour: solve exercise problem, discuss solution
Exercise Topics

- Why?
- Basic blocks
- Computing WCET
- Value Analysis
- Cache Analysis
WHY WCET ANALYSIS?

Keyword: Abstraction
BASIC BLOCKS
Basic Block

- **Definition:** A basic block is a sequence of instructions where the control flow enters at the beginning and exits at the end, without stopping in-between or branching (except at the end).

\[
t1 := c - d \\
t2 := e \times t1 \\
t3 := b \times t1 \\
t4 := t2 + t3 \\
if \ t4 < 10 \ goto \ L
\]
Basic Blocks

- *Determine basic blocks of a program:*
  1. *Determine the block beginnings:*
     the first instruction
     targets of un/conditional jumps
     instructions that follow un/conditional jumps
  2. *Determine the basic blocks:*
     there is a basic block for each block beginning
     the basic block consists of the block beginning and
     runs until the next block beginning (exclusive) or
     until the program ends
Example

/* k >= 0 */
s = k;
WHILE (k < 10) {
    IF (ok)
        j++;
    ELSE {
        j = 0;
        ok = true;
    }
    k ++;
}
r = j;
Exercise

• From the sheet. Problem (a)
CALCULATING WCET
Calculation of the WCET

- **Definition:** A program consists of $N$ basic blocks, where each basic block $B_i$ has a worst-case execution time $c_i$ and is executed for exactly $x_i$ times. Then, the WCET is given by

$$WCET = \sum_{i=1}^{N} c_i \cdot x_i$$

- the $c_i$ values are determined using the static analysis.
- how to determine $x_i$?
  - structural constraints given by the program structure
  - additional constraints provided by the programmer (bounds for loop counters, etc.; based on knowledge of the program context)
**Structural Constraints**

\[
s = k;  \\
\text{WHILE} \ (k<10)  \\
\text{if} \ (\text{ok})  \\
\text{j++;}  \\
\text{j = 0; \ ok = true;}  \\
\text{k++;}  \\
\text{r = j;}  \\
d10 \downarrow
\]

**Flow equations:**
\[
\begin{align*}
\text{d1} &= \text{d2} = x_1 \\
\text{d2} + \text{d8} &= \text{d3} + \text{d9} = x_2 \\
\text{d3} &= \text{d4} + \text{d5} = x_3 \\
\text{d4} &= \text{d6} = x_4 \\
\text{d5} &= \text{d7} = x_5 \\
\text{d6} + \text{d7} &= \text{d8} = x_6
\end{align*}
\]
Additional Constraints

s = k;

WHILE (k<10)

if (ok)

j++;
ok = true;

j = 0;
k++;

r = j;

loop is executed for at most 10 times:

x_3 <= 10 \cdot x_1

B5 is executed for at most one time:

x_5 <= 1 \cdot x_1
WCET - ILP

- **ILP with structural and additional constraints:**

program is executed once

\[
WCET = \max \left\{ \sum_{i=1}^{N} c_i \cdot x_i \mid d_1 = 1 \land \sum_{j \in \text{in}(B_i)} d_j = \sum_{k \in \text{out}(B_i)} d_k = x_i, i = 1 \ldots N \land \text{additional constraint } s \right\}
\]
Exercise

• Comment on the following programming styles
  – Statically defined variables
  – Recursive functions
  – Goto
VALUE ANALYSIS
Value Analysis

• **Motivation**:
  – Provide access information to data-cache/pipeline analysis
  – Detect infeasible paths
  – Derive loop bounds

• **Method**: calculate intervals at all program points, i.e. lower and upper bounds for the set of possible values occurring in the machine program (addresses, register contents, local and global variables).
Value Analysis – Abstract Interpretation

• **abstract domain** – related to concrete domain by abstraction and concretization functions, e.g. $L \rightarrow \text{Intervals}$ where Intervals = $LB \times UB$, $LB = UB = \text{Int} \cup \{-\infty, \infty\}$

• **abstract transfer functions** for each statement type – abstract versions of their semantics e.g. $+: \text{Intervals} \times \text{Intervals} \rightarrow \text{Intervals}$ where $[a,b] + [c,d] = [a+c, b+d]$ with $+$ extended to $-\infty, \infty$

• **a join function** combining abstract values from different control-flow paths e.g. $t: \text{Interval} \times \text{Interval} \rightarrow \text{Interval}$ where $[a,b] t [c,d] = [\min(a,c), \max(b,d)]$
Value Analysis

D1:[-4,4], A0:[0x1000,0x1000]

- Intervals are computed along the CFG edges
- At joins, intervals are "unioned"

move #4, D0

D0:[4,4], D1:[-4,4], A0:[0x1000,0x1000]

add D1, D0

D0:[0,8], D1:[-4,4], A0:[0x1000,0x1000]

Move [A0], D1

Which address is accessed here?

access [0x1000,0x1008]
Exercise

• Is the abstraction LB x UB always sufficient?

• Consider \( f(x) = |x - 2| \), and \( x = [1, 5] \). What is the abstraction of \( f(x) \)
CACHE ANALYSIS
Cache Analysis

• How to statically precompute cache contents:

  – **Must Analysis:**
    For each program point (and calling context), find out which blocks are in the cache. Determines safe information about cache hits. Each predicted cache hit reduces WCET.

  – **May Analysis:**
    For each program point (and calling context), find out which blocks may be in the cache. Complement says what is not in the cache. Determines safe information about cache misses. Each predicted cache miss increases BCET.
Must Analysis

Abstraction

\[ \alpha \]

\[
\{ \}
\]

\[
\{\}
\]

\[
\{z,x\}
\]

\[
\{s\}
\]
Must Analysis

Concretization

\[ z, x \in \left\{ s \in \{z, x\}, s \right\} \]
LRU: Transfer for Must

concrete
[ access s ]

abstract
[ access s ]

"young"
“old”

Age
Join for Must

Join (must)

```
{ a }
{   }
{ c, f }
{ d }

{ c }
{ e }
{ a }
{ d }
```

"intersection + maximal age"

**Interpretation:**
memory block \(a\) is definitively in the (concrete) cache => always hit
Contexts

- Cache contents depends on the context, i.e. calls and loops

- First iteration loads the cache:
  - Intersection loses most of the information.

- Distinguish as many contexts as useful:
  - 1 unrolling for caches
  - 1 unrolling for branch prediction (pipeline)
Exercise

• Comment on the following different cache policies
  – Evict a random cache line in case of miss
  – Evict a specific cache line in case of miss