TU WIEN

Classifications of Code Annotations and Discussion of Compiler-Support for Worst-Case Execution Time Analysis

Raimund Kirner
Peter Puschner

Real-Time Systems Group
Vienna University of Technology

SW Development and WCET analysis for real-time applications

- SW is mostly written in imperative programming languages
  - Trend: graphical interfaces like state charts
  - Also domain-specific languages like data-flow languages (transformed to imperative implementations.)
- Today, only very small fractions of performance-critical code is written in assembly code.
- Precise and accurate WCET analysis wanted?
  → analysis of object code mandatory!!

Why is it hard to analyse code semantics for WCET analysis?

- Platform-specific behavior (memory layout, ROM-areas, memory-mapped IO, …)
- Control-flow is well hidden within low-level constructs of object code (indirect jumps, …)
- Formal limitations of computability (loop bounds, correlation of code predicates, type-inference in object-oriented languages, …)
- etc.

Work around: Guide the analysis by means of code annotations (1)

- Annotations to find jump-targets [aiT]:

  instruction <addr> calls <target-list>;
  instruction <addr> branches to <target-list>;
- Labeling assembly-instructions with high-level meaning [aiT]:

  instruction <addr> is a return;

Work around: Guide the analysis by means of code annotations (2)

- Describing possible values of variables [aiT]:

  condition <addr> is always <bool value>;
  snippedt <addr> is newer executed;
  instruction <addr> is entered with <state>;
- Describe boundaries of memory access [aiT]:

  instruction <addr> accesses <addr-range>;

Work around: Guide the analysis by means of code annotations (3)

- Describe iteration bounds of loops [Bound-T]:

```
subprogram "<fn-name>"
   loop that uses <var-name>
         repeats <= 7 times;
   end loop;
end "<fn-name>";
```

(written in external code annotation file)

Reflection on code annotations

- Code annotations may be erroneous
- Code annotations often have to be done at object code level ⇨ update/check necessary when changing and re-compiling the code
- Manually annotating the code is labor-intensive (especially at the object-code level !!!)
- Questions:
  Are there code annotations that can be avoided?
  Can the process of annotation be simplified?

Classification of Code Annotations (1)

- Platform Property Annotations (PPA)
- CFG Reconstruction Annotations (CRA)
- Program Semantics Annotations (PSA)
- Auxiliary Annotations (AA)

Classification of Code Annotations (2)

Platform Property Annotations (PPA)

- Description of special semantics behind the access of memory-mapped IO (local annotations).
- Description of platform properties that influence the execution time (mostly global annotations).
- Examples:
  description of memory layout, properties of target hardware, …

Classification of Code Annotations (3)

CFG Reconstruction Annotations (CRA)

- Help to build basic structures of code analysis:
  - control-flow graph (CFG) and call-graph
- Allow to reverse-engineer the high-level code structure from the low-level assembly statements.
- Examples:
  list of targets for branch instr, type labeling of branch instr, …

Classification of Code Annotations (4)

Program Semantics Annotations (PSA)

- Local description of program behavior
- Support for
  - path analysis
  - value analysis
  - etc.
- Examples:
  flow annotations like loop bounds, description of variable values, …

Compiler-Support for WCET Analysis (1)

Information accessible by the compiler:

- Program representation at source and object code level.
- Performed code transformations to obtain object code

What a compiler can help:

- Describe CFG of object code (to avoid the need of CRA)
- Provide mapping of source-code annotations to object code (simplification by writing annotations at source code level instead of object-code level) [PPA,PSA,AA]

Compiler-Support for WCET Analysis (2)

What a compiler can help (2):

- Emit properties of a program's execution behavior (reduce amount of code annotations) [PSA]
  - in general, some PSA may remain mandatory
  - but in practice almost no PSA is mandatory!
- Improve predictability of code
  - imitation of "WCET-oriented programming"
  - e.g., single-path conversion can reduce the execution-time jitter of real-time programs
  - support of predictable HW mechanisms (e.g., prefetching, scratchpad memory, cache locking),

Final remark on code annotations

Not all code annotations are artificial:

- application-context (operation mode) may be specified to refine the analysis [PSA] (e.g. range on initial variable values, CFG restriction)
- mapping of target-specific properties to code may be mandatory [PPA] (e.g. memory-mapped IO)