Towards Semantics-Based WCET Analysis

Mihail Asavoae

Faculty of Computer Science,
Alexandru Ioan Cuza University, Iasi, Romania

Joint Work with Dorel Lucanu (UAIC) and Grigore Roșu (UIUC)
Outline

1 Preliminaries

2 Introduction in K

3 The Language Definition

4 Framework for Timing Analysis

5 Conclusions
Outline

1 Preliminaries
2 Introduction in K
3 The Language Definition
4 Framework for Timing Analysis
5 Conclusions
Worst-Case Execution Time

... settings

- **WCET analysis:** the longest execution time of a program running on a particular architecture
- **Program:**
  consider all executions - path analysis
- **Architecture:**
  consider the impact over the executed program - processor behavior analysis
Typical WCET Analyzer

... issues and solutions

Program:

Architecture:

DISCLAIMER: Image taken from Reinhard Wilhem & all
“The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools”, in TECS 2008
Typical WCET Analyzer

... issues and solutions

Program: CFG extraction,

Architecture:

DISCLAIMER: Image taken from Reinhard Wilhem & all
"The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools", in TECS 2008
Typical WCET Analyzer

... issues and solutions

Program: CFG extraction, loop bounds calculation,

Architecture:

DISCLAIMER: Image taken from Reinhard Wilhem & all
"The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools", in TECS '08

Mihail Asavoae: Towards Semantics-Based WCET Analysis — WCET 2011
Typical WCET Analyzer
... issues and solutions

Program: CFG extraction, loop bounds calculation, infeasible paths detection

Architecture:

DISCLAIMER: Image taken from Reinhard Wilhem & all
"The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools", in TECS 2008
Typical WCET Analyzer
... issues and solutions

Program: CFG extraction, loop bounds calculation, infeasible paths detection

Architecture: processor modeling

DISCLAIMER: Image taken from Reinhard Wilhem & all
"The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools", in TECS 2008
Typical WCET Analyzer

... issues and solutions

Program: CFG extraction, loop bounds calculation, infeasible paths detection

Architecture: processor modeling, timing predictability

DISCLAIMER: Image taken from Reinhard Wilhem & all
“The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools”, in TECS 2008

Mihail Asavoae: Towards Semantics-Based WCET Analysis — WCET2011
Typical WCET Analyzer

... issues and solutions

EXECUTABLE SEMANTICS?

DISCLAIMER: Image taken from Reinhard Wilhem & all
“The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools”, in TECS 2008
**Formal Executable Semantics**

- **Execute** programs
  - as executed by the real language
- **Detect/eliminate** undefined programs
- **Build** verification tools
  - a language definition has the necessary information to help building abstractions out of it
Knowledge of *infeasible paths* leads to improvement on the result of the analysis.

Infeasibility has two causes:
- semantic dependecies in the program
- restrictions on the set of input data values

But there is another kind of infeasible paths we should know about ...
Knowledge of infeasible paths leads to improvement on the result of the analysis.

Infeasibility has two causes:
- semantic dependencies in the program
- restrictions on the set of input data values

But there is another kind of infeasible paths we should know about ... erroneous paths.
Erroneous Paths

... existing approaches

- A priori usage of dedicated static analyzers to detect such runtime errors
- Code annotation with error testing predicates
- Usage of convenient assumptions over the input data
Erroneous Paths

... existing approaches

- A priori usage of dedicated static analyzers to detect such runtime errors
- Code annotation with error testing predicates
- Usage of convenient assumptions over the input data
- Information about error checks is available in the language definition!
Erroneous Paths
... existing approaches

- A **DIV**: Divide signed.
  - Opcode: 0x48
  - Format: DIV rs,rt
  - Semantics:
    - DIV0(GPR(RT))
    - SET_LO(GPR(RS) / GPR(RT))
    - SET_HI(GPR(RS) % GPR(RT))

- Information about error checks is available in the language definition!

The WCET Problem
... meets a semantic-based solution
The WCET Problem
... meets a semantic-based solution
The WCET Problem
... meets a semantic-based solution
The WCET Problem
... meets a semantic-based solution


The **K Framework**

... emerged from the rewrite logic semantics project

- **Rewrite-based executable** framework specialized for design and analysis of programming languages

- Distinctive characteristics:
  - a *highly concurrent* rewrite abstract machine
  - an elegant definitional technique
  - a concise specialized notation

- Features along the presentation!
The K Framework

... emerged from the rewrite logic semantics project

- **K-Maude:**
  prototype providing interpreter and state space exploration

- Executable semantics for various programming paradigms, with the declared goal of fully implementing real-life languages:
  - imperative (C in 800+ rules)
  - hardware (Verilog)

- Support for analysis/verification tools:
  type inference/type checking, explicit state model checking, Hoare-style program verification with Matching Logic

Outline

1 Preliminaries

2 Introduction in $K$

3 The Language Definition

4 Framework for Timing Analysis

5 Conclusions
### SSRISC Language

The SSRISC Language is a subset of the Simplescalar PISA assembly language, with ALU-, load/store, branch/jump and break instructions. The language definition is given in K, a specification language, as follows:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td><code>add Reg, Reg, Reg;</code></td>
<td>[strict (2 3)]</td>
</tr>
<tr>
<td>addi</td>
<td><code>addi Reg, Reg, Imm;</code></td>
<td>[strict (2)]</td>
</tr>
<tr>
<td>mult</td>
<td><code>mult Reg, Reg;</code></td>
<td>[strict]</td>
</tr>
<tr>
<td>div</td>
<td><code>div Reg, Reg;</code></td>
<td>[strict]</td>
</tr>
<tr>
<td>j</td>
<td><code>j Addr;</code></td>
<td>[strict]</td>
</tr>
<tr>
<td>jr</td>
<td><code>jr Reg;</code></td>
<td>[strict]</td>
</tr>
<tr>
<td>beq</td>
<td><code>beq Reg, Reg, Addr;</code></td>
<td>[strict (1 2)]</td>
</tr>
<tr>
<td>bne</td>
<td><code>bne Reg, Reg, Addr;</code></td>
<td>[strict (1 2)]</td>
</tr>
<tr>
<td>lw</td>
<td><code>lw Reg, Off(Reg);</code></td>
<td>[strict (3)]</td>
</tr>
<tr>
<td>sw</td>
<td><code>sw Reg, Off(Reg);</code></td>
<td>[strict (3)]</td>
</tr>
<tr>
<td>break</td>
<td><code>break;</code></td>
<td></td>
</tr>
</tbody>
</table>
SSRISC Language

... formal semantics in $\mathbb{K}$

- A language definition in the $\mathbb{K}$ framework has the following components:
  1. configurations
  2. computations
  3. rewrite rules
Configurations represent the necessary entities to capture the program semantics:

\[
\begin{align*}
&\langle K\rangle_k \quad \langle \text{Int32} \rangle_{\text{pc}} \quad \langle \text{Int32} \rangle_{\text{lo}} \quad \langle \text{Int32} \rangle_{\text{hi}} \\
&\langle \text{Map} [ \text{Regs} \mapsto \text{Int32} ] \rangle_{\text{regs}} \quad \langle \text{Int32} \rangle_{\text{break}}
\end{align*}
\]
Configurations represent the necessary entities to capture the program semantics:

\[
\langle K \rangle_k \langle \text{Int32} \rangle_{pc} \langle \text{Int32} \rangle_{lo} \langle \text{Int32} \rangle_{hi} \\
\langle \text{Map} [ \text{Regs} \mapsto \text{Int32} ] \rangle_{\text{regs}} \langle \text{Int32} \rangle_{\text{break}}
\]
**Configurations** represent the necessary entities to capture the program semantics:

\[ \langle K \rangle_k \langle \text{Int32} \rangle_{pc} \langle \text{Int32} \rangle_{lo} \langle \text{Int32} \rangle_{hi} \]

\[ \langle \text{Map}[\text{Regs} \mapsto \text{Int32}] \rangle_{\text{regs}} \langle \text{Int32} \rangle_{\text{break}} \]

Where is the memory cell?
Memory Modeling
... in the K module system

TIME MODEL
<k>

COMM-INTERFACE
getPC, getd, putd, ...

LANGUAGE
DEFINITION
<k> <pc>
<k> <hi> <lo>
<regs> <break>

I-CACHE
<k>
<ic> <param>
<replace>
<profile>

MEMORY
<k>
<cmem>
<dmem>

MAIN MODULE

Mihail Asavoae: Towards Semantics-Based WCET Analysis — WCET2011
SSRISC Language

... formal semantics in $K$

- The rules enable transitions in the program execution
  - There are two kinds of rules:
    1. structural
    2. computational
**SSRISC Language**

... formal semantics in \( K \)

- **The rules** enable transitions in the program execution. There are two kinds of rules:
  1. *structural*
  2. *computational*

1. Define the "states" of an abstract rewrite transitional system, e.g.:

\[
\langle \text{lw } Rd, \text{ Off } (V_1) ; \text{updateReg(getd(Off +Int32 V_1), Rd)} \rangle_k
\]
The rules enable transitions in the program execution.

There are two kinds of rules:

1. **structural**
2. **computational**

Define the "transitions" of an abstract rewrite transition system, e.g.:

\[
\langle \text{updateReg}(I, Rd) \rangle_k \langle \cdots Rd \mapsto \_ I \cdots \rangle_{\text{regs}}
\]
Outline

1. Preliminaries
2. Introduction in K
3. The Language Definition
4. Framework for Timing Analysis
5. Conclusions
The Language Definition Framework for Timing Analysis

Conclusions

Semantics-Based Framework for WCET
... centered around the language definition

K-Specification

Syntax

Semantics

Abstract semantics 1
(symb propagation)

Abstract semantics 2
(refined symb propagation)

Abstract semantics n

K-Specification
for Microarchitecture

WCET bound

WCET bound

Implicit state exploration

Explicit state exploration

Mihail Asavoae:
Towards Semantics-Based WCET Analysis — WCET 2011
Target: keep the language semantics definition unchanged!
- **Target**: keep the language semantics definition unchanged!

- \( \mathcal{K} \) enables the possibility to work with the language semantics definition in two ways:
  - directly, e.g. Abstract semantics 1
  - at the meta level, e.g. Abstract semantics 2
Semantics-Based Framework for WCET

... methodology

- **Target**: keep the language semantics definition unchanged!

- **K** enables the possibility to work with the language semantics definition in two ways:
  - directly, e.g. Abstract semantics 1
  - at the meta level, e.g. Abstract semantics 2

- Getting back to the erroneous path detection ...
Error Path Detection

... example - via Abstraction 1

Mihail Asavoae: Towards Semantics-Based WCET Analysis — WCET 2011
Error Path Detection

... example - via Abstraction 1
Error Path Detection
... example - via Abstraction 2

1. r2 ← 1
2. r1 ≠ 0
   - r1 = 0
3. r3 ← r2
4. r4 ← r1 + r2
5. r4 ← r1 + r2
   - r4 ≠ r3
6. r3 ← r3 - r2
7. r3 ← r3 - r2
   - r2 / r3
8. r3 ← r1 - r4
9. r2 / r3
10. X
Error Path Detection
... example - via Abstraction 2
Background information
... on the technology we use

- $K$ specifications are compiled into Maude rewrite theories
- the `search` command performs reachability analysis on the
  abstract rewrite transitional system produced by the $K$
  specification of the language
Background information
... on the technology we use

- K specifications are compiled into Maude rewrite theories
  - the search command performs reachability analysis on the
    abstract rewrite transitional system produced by the K
    specification of the language

- To guarantee a safe WCET bound, we explore all program
  executions, using reachability for the state that has in the
  $\langle - \rangle_k$ cell a special token, called last
  - normal termination: the instruction for last was executed
  - error termination: the final executed instruction is a break
Background information
... on the technology we use

- **K** specifications are compiled into Maude rewrite theories
  - the `search` command performs reachability analysis on the abstract rewrite transitional system produced by the **K** specification of the language

- To guarantee a safe WCET bound, we explore all program executions, using reachability for the state that has in the $\langle \cdot \rangle_k$ cell a special token, called `last`
  - normal termination: the instruction for `last` was executed
  - error termination: the final executed instruction is a `break`

- **Site:**
  
  http://code.google.com/p/k-framework/
  source/browse/ → inProgress → ssrisc
Outline

1 Preliminaries

2 Introduction in K

3 The Language Definition

4 Framework for Timing Analysis

5 Conclusions
Towards Semantics-Based WCET Analysis

... a different perspective on the problem

- We propose a framework for WCET analysis based on the formal executable semantics of a RISC assembly language.

- We use the concrete semantics of a program running on a particular architecture to obtain the abstract semantics for computing time bounds.

- The focus is on the error path detection at the low level, a subproblem of the infeasible path detection.