COST-ONLINE_STSM-IC1202-21876 Extraction of Semantic Properties for the WCET analysis === REPORT ===

Mihail Asăvoae

VERIMAG/UJF, France
{mihail.asavoae}@imag.fr

1 Background

It is desirable to use model-based designs to build hard real-time systems; for example, control engineering applications are often generated from synchronous designs. Any execution of a hard real-time system is subject to certain timing constraints, making the knowledge of a Worst-Case Execution Time (WCET) bound necessary. A typical workflow for the WCET analysis employs static analysis to derive semantic properties on the program and the underlying architecture. From the point of view of the program, deriving properties like path infeasibility (i.e. an execution path is infeasible when it could not be exercised for any input data) is necessary to improve on the accuracy of the analysis.

2 STSM - Setting, Workplan and Results

My work is part of the project W-SEPT (wsept.inria.fr) which, quoting from the site is a collaborative research project focusing on worst-case execution time guarantees. The project aims to improve the precision and the traceability of semantics information through the compilation flow, from high-level description (such as the Lustre synchronous language) down to C and binary levels. In other words, the project focuses on how to extract, express and exploit the program semantics (through what we call semantic properties) in order to improve the precision of the WCET analysis. I worked or I am currently working in the following three directions which I planned to investigate during my visit in Saarbrücken, at Universität des Saarlandes and AbsInt:

- *how to extract semantic properties from Scade designs*
- *how to improve on architecture support, for an SMT-based path analysis*
- *how to extract semantic properties using counters*

This document is organized according to the STSM proposal. Each of three directions of work is presented in a dedicated subsection which contains: a work summary and a report on how each particular point was addressed during my visit. The document ends with a conclusion section.

2.1 Extraction of semantic properties from Scade designs

Scade specification language is a mixed synchronous language based on the data-flow style of Lustre and control-flow style of Esterel. In the project W-SEPT, I worked in [4] how to discover, on Scade designs, properties about path infeasibility to improve the precision of a WCET analysis. The aiT timing analyzer developed by AbsInt [1] is already integrated into Scade Suite platform, as described in [2]. I intended to learn more about this aiT-Scade integration (e.g. how it is implemented, if it includes extraction of semantic properties from the Scade design etc) to be able to increase the precision of the WCET analysis with flow information derived at the design level.

This point was practically addressed in a discussion with Dr. Florian Martin from AbsInt. The integration works as follows: the C code generated from a Scade module is compiled and the aiT analyzer performs timing analysis on the corresponding executable. This timing analysis is supported by an additional file containing flow facts which are expressed in an annotation language developed at AbsInt (i.e. with extension `.ais`). This file is automatically generated for each Scade module and it contains only the necessary loop bounds (i.e. the C code contains only loops as for-statements). The `.ais` file does not include other path infeasibility information (though it could be manually added) and the aiT analyzer does not employ any specialized analysis, except the existing ones in the tool, to extract such information. The Scade-aiT integration was designed to provide development support for Scade platform and additional Scade semantic properties could be extracted by the Scade compiler `kcg`.

The current approach is limited, because the semantic properties in the .ais file are overly conservative. For example, the Scade tool could generate two types of for-statements - one which iterates through all the elements and another which iterates until a certain condition is satisfied (e.g. when the iteration passes through the first three elements out of a total of ten elements).

2.2 Enriching architecture support for an SMT-based approach

Since the SMT-based technology (theories, solvers) improved tremendously in the last several years, we decided to test how SMT solving could help to compute tighter WCET bounds. As we use an SMT encoding of an intermediate language (LLVM-IR), it is necessary to integrate architecture information over this encoding. I worked, in [3], on how to lift the results of architecture analyses from the binary level to the intermediate-level representation. Our SMT-based approach suffers from a number of shortcomings, the most prominent one being that our analysis is imprecise due to limited architecture analyses. We are currently using the architecture analyses provided by an academic timing analyzer. I intend to explore how our SMT-based approach performs with architectural support for the state of the art timing analyzer - aiT (e.g. what kind of analyses are available, what kind of file format is used to keep the result of an architecture analysis etc.).

This point was practically addressed in discussions with Dr. Gernot Gebhard and Dr. Christoph Cullmann from AbsInt. The aiT tool supports several complex processor behavior analyses (i.e. cache and pipeline analyses for several processors, a complete listing being presented on the company website). The results of these analyses are encoded in a particular format - the so-called prediction file. This file contains accurate architectural information as abstract pipeline state graphs (i.e. a condensed representation of all possible pipeline states) and it is integrated in a modified LP-based path analysis. Currently, this file is part of the internals of aiT timing analyzer and cannot be directly accessed. Therefore, we could not integrate the results of architecture analysis of aiT in our SMT-based approach, in the current setting, however several solutions are possible (e.g. measurement-based results)

2.3 Extraction of semantic properties using counters

In order to complement our high-level approach to extract semantic properties, we have explored the possibility of using abstract interpretation on the program instrumented with special variables called counters. I work on strategies for automated program instrumentation using counter and then how to extract useful semantic properties (i.e. loop bounds and infeasible paths). I intended to investigate how such properties are handled by the techniques developed on-site (e.g. what kind of patterns are used in the loop bound detection, what kind of infeasibility relations could be extracted automatically etc).

This point was practically addressed in discussions with Mr Simon Wegener from AbsInt. Our discussions were mostly on the capabilities of the aiT tool to extract and express semantic properties (in the context of an ILP-based path analysis). The aiT timing analyzer uses a combination of pattern matching and interval analysis to extract loop bounds and infeasible information. All the loops are represented as functions and the loop iterator variable is a parameter of this function. As such, pattern matching could be applied either at the parameter place in the function signature (i.e. an architecture-dependent pattern) or on how the parameter is modified in the function/loop body (i.e. code structure-based pattern). There are around 10-15 patterns covering both single and nested loops.

Pattern matching is used with a simple interval analysis, which computes intervals for the program variables. Based on the interval analysis results, certain infeasibility relations could be detected. For example, if a program condition checks if a variable is smaller than, say 10 and the interval analysis yields an interval from 0 to 9, then one of the branches is never executed. The counter-based approach that we are currently developing in Verimag is more powerful, though expensive and with a reasonable tradeoff, it produces better results.

3 Conclusions

My work in the host institution also included individual interactions/meetings with Prof. Reinhard Wilhelm and Prof. Jan Reineke on various aspects of the WCET analysis (e.g. ILP formulation, abstract interpretation), with Mohamed Abdel Maksoud, Michael Jacobs and Sebastian Hahn on architecture and tool support (e.g. intermediate-code support) and with Prof. Sebastian Hack and Klaas Boesche on the SMT-based formulation for the path analysis. My visit has also included a talk for the groups of Profs. Reinhard Wilhelm, Jan Reineke and Sebastian Hack as well as the company AbsInt.

In summary, my plan consisted of three lines of work, regarding both the techniques and tools developed at Universität des Saarlandes and AbsInt. The main focus is to improve on the extraction of semantic properties (i.e. better loop bounds and infeasibility relations) to compute tight WCET bounds. The research on using counters to extract semantic properties using counters stands, out of the three directions of investigation, as the most promising one for an active research collaboration.

Acknowledgements I would like express my gratitude to the following institutions and people, all kindly supported my scientific mission - to the project Timing Analysis on Code-Level (TACLe) supported by ICT COST Action IC1202, coordinated by Prof. Bjorn Lisper and to the coordinator for the STSMs, Prof. Isabelle Puautl, from the home institution - Verimag, to my supervisors Prof. Claire Maiza and Dr. Pascal Raymond and from the host institutions Universität des Saarlandes and AbsInt to Prof. Reinhard Wilhelm and everybody mentioned in this document.

References

1. AbsInt Angewandte Informatik: aiT Worst-Case Execution Time Analyzer.
2. C. Ferdinand, R. Heckmann, T. Sergent, D. Lopes, B. Martin, X. Fornari, and F. Martin. Combining a high-level design tool for safety-critical systems with a tool forwcet analysis on executables. In *ERTS2*, 2008.
3. J. Henry, M. Asavoae, D. Monniaux, and C. Maiza. How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics. In *LCTES 2014*, pages 43–52, 2014.
4. P. Raymond, C. Maiza, C. Parent-Vigouroux, F. Carrier, and M. Asavoae. Timing analysis enhancement for synchronous program. In *RTS*, 2014. to appear.