ACADEMIC REPORT

Purpose of the STSM

Set of Software Quality Static Analyzers (SSQSA) [1] is a set of software tools for static analysis that are incorporated in the framework developed to target the common aim – consistent software quality analysis. The main characteristic of all integrated tools is the independency of the input computer language. This characteristic gives the tools more generality comparing to the other similar static analyzers. Each of incorporated analyzers can be uniformly applied to any software systems that are written in different programming languages. Furthermore, integration of the analyzers enables consistent analysis of different aspects of software product.

Language independency is achieved by enriched Concrete Syntax Tree (eCST) [2] that is used as an intermediate representation of the source code. In SSQSA framework each input program is internally represented on two levels. Each compilation unit is represented by specific, language independent eCST representation in XML format. As each of these trees are independent, software networks are used to reflect connections between compilation units. Generation of software networks is based on eCST representation of the source code and fully relies on information contained in it.

eCST is the concrete syntax tree enriched with so-called universal nodes that describe elements of language semantics. Here we involve the concept of imaginary nodes used in Abstract Syntax Tree building. Universal nodes can be described as specific imaginary nodes common for all input languages.

The purpose of STSM was to initiate research in direction of introducing support for timing analysis and Worst Case Execution Time (WCET) calculation [3] in SSQSA framework (collecting literature, preliminary research, specifying the main directions of the research, exchanging ideas with colleagues fro the host institution and early correction of the research idea based on of the results of the grant period, etc).

Description Of The Work Carried Out During The STSM

The aim of current research is to provide prerequisites for introducing the support for timing analysis and Worst Case Execution Time (WCET) calculation [3] in SSQSA framework. First step is to develop all needed models and develop techniques supporting timing analysis such is control-flow analysis [4]. This task is currently in the main research goal under the SSQSA project.

Let us first consider intermediate representation of the programs to be analysed provided by SSQSA.

The main characteristic of SSQSA framework is it's independency of the input computer language based on language independent tree representation of the source code so-called enriched Concrete Syntax Tree (eCST). Basic concept used here is to use universal nodes to enrich syntax tree so that they annotate semantic of the construct in the sub-tree of it [2]. Universal nodes are divided in three levels:

- High-level eCST universal nodes mark entities declaration on the architectural level. Interface-level declarations of packages, classes, modules and methods, procedures, functions, etc. , as well as explicitly stated high-level relations between them (such as inheritance, instantiation, implementation, etc.).
- Middle-level eCST universal nodes are those used at the level of entity definition. They appear in the body of the entities and mark individual statements, groups of statements or parts of statements with appropriate concept expressed by them (jump statement, loop statement, branch statement, condition, import statement, etc.).
- Low-level eCST universal nodes are universal nodes that mark individual tokens with appropriate lexical category (keywords, separators, identifiers, etc.).

Based on eCST representation of the source code we can independently of input language generate other source code representations.

Generation of eCFG (enriched Control Flow Graf) is one of the first tasks to be completed toward the static timing analysis in the SSQSA framework. This is to be done based on middle and low level universal nodes. Work on this task is still in progress, but it is already in the final phase.

Furthermore, based on mainly high-level universal nodes we can generate different kind of software networks completely independently of input language [3]. For timing analysis, the most important network is one corresponding to call graph. Based on eCFG and this network we can create interprocedural CFG [4]

Based on these program representations we can implement any of widely used approach to determine upper bound of execution time (tree based, path based, implicit path enumeration, etc.) and supporting analysis [4, 5, 6]

In the focus of contemporary research is appropriate support for control flow analysis. Even if we do not take care about input language, special attention is paid to languages and development approaches that strongly rely on recursion [7]. So far, this was a weak point of SSQSA framework. During the STSM period another activity was dedicated to consideration of behaviour of SSQSA framework when takes functional code from input. In this direction we have to introduce support for some functional languages (e.g. Erlang, Scheme, etc). Work on this task is also still in progress.

When we will have all accomplished all preparation stages, when intermediate representations are created and the background analyses are implemented, we can move forward to real timing analysis and WCET calculation. First, we have to determine the set of universal nodes needed for planned analyses

Based on current observations and similar approach [8] we can conclude that we have all needed universal nodes needed for timing analysis. All needed universal nodes existed in the catalogue or have been added in the process of transformation of eCST to eCFG. Furthermore, based on [8] we have to enrich these universal nodes by adding timing information. This will be done in the postprocessor of eCST by adding of attributes to XML elements representing universal nodes.

After all initiated task are completed we can move to another stage. We can implement WCET algorithm. Furthermore one of the next steps is introducing support for some domain specific languages to SSQSA framework (e.g. ALF [9]) [10].

The main activities during the grant period were dedicate to determination of the set of universal nodes needed to timing analysis, and their configuration. This was followed by activities on control flow representation generation and different aspects of control flow analysis.

Description Of The Main Results Obtained

The main result determined set of universal nodes needed for static timing analysis, and attributes for each node to be additionally enriched by timing information. Additional results are in the domain of internal source code representation and their improvements so that they enable WCET calculation. Finally, aspects of static analysis, more precisely control-flow analysis and static timing analysis related to functional languages were considered.

Future Collaboration With Host Institution (If Applicable)

There are several possible directions for cooperation with the host institution in the following period, but with the connecting points to static timing analysis we can select the most important ones.

- better support for functional languages by SSQSA framework
- improvement of control-flow analysis
- timing analysis for functional languages

Foreseen Publications/Articles Resulting Or To Result From The STSM (If Applicable)

It is expected to publish at least one paper presenting preliminary idea for initiated tasks based on this introductory research and to submit to some workshop oriented to timing analysis or static analysis in general. If reviewers approve the idea we can expect to continue the research and to have more serious publications.

References

[1] **Rakić G**., Budimac Z., and Savić M.. 2013. Language independent framework for static code analysis, *In Proceedings of the 6th Balkan Conference in Informatics (BCI '13).,* Thessaloniki, Greece, September 19-21, 2013, ACM, New York, NY, USA, 236-243.

[2] **Rakić G.**, Budimac Z., Introducing Enriched Concrete Syntax Trees, *In Proc. of the 14th International Multiconference on Information Society (IS), Collaboration, Software And Services In Information Society (CSS),* October 10-14, 2011, Ljubljana, Slovenia, Volume A, pp. 211-214,

[3] Savić M., **Rakić G**. , Budimac Z., Ivanović Z., A language-independent approach to the extraction of dependencies between source code entities, Information and Software Technology (2014), doi: http://dx.doi.org/10.1016/j.infsof.2014.04.011

[4] Lokuciejewski, P., & Marwedel, P. (2010). Worst-case execution time aware compilation techniques for real-time systems. Springer.

[5] Wilhelm R., Engblom J, Ermedahl A., Holsti N., Thesing S, Whalley D., Bernat G., Ferdinand C., Heckmann R., Mitra T., Mueller F., Puaut I, Puschner P., Staschulat J., and Per Stenstrom., The worst-case execution-time problem\—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.* 7, 3, Article 36 (May 2008), 53 pages.

[6] Lokuciejewski P, Marwedel P.,. Combining Worst-Case Timing Models, Loop Unrolling, and Static Loop Analysis for WCET Minimization. In *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems* (ECRTS '09). IEEE Computer Society, Washington, DC, USA,pp. 35-44.

[7] **Rakić G**., Budimac Z. and Bothe K., Introducing Recursive Complexity, *In (AIP Conference) Proc. Of International Conference of Numerical Analysis and Applied Mathematics ICNAAM2013, 3nd Symposium on Computer Languages, Implementations and Tools (SCLIT),* September 21-27, 2013, Rhodes, Greece, AIP Conference Proceedings, 357-361

[8] Parsa S., and Mehdi S.. "A XML-Based Representation of Timing Information for WCET Analysis.", Journal of mathematics and computer science, Vol 8, Issue3, 2014, pp. 205-214

[9] Gustafsson, J., Ermedahl, A., Lisper, B., Sandberg, C., & Källberg, L. (2009, June). ALF–a language for WCET flow analysis. In *Proc. 9th International Workshop on Worst-Case Execution Time Analysis (WCET'2009)*, Dublin, Ireland, pp. 1-11

[10] Kolek J, **Rakić G,** Savić M., Two-dimensional Extensibility of SSQSA Framework, *In Proceedings of the 2nd Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications,* Novi Sad, Serbia, September 15-17, 2013., pp. 35-43