Toward Language Independent Worst-Case Execution Time Calculation

GORDANA RAKIĆ and ZORAN BUDIMAC, Faculty of Science, University of Novi Sad

Set of Software Quality Static Analyzers (SSQSA) is a set of software tools for static analysis that is 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. Language independency is achieved by enriched Concrete Syntax Tree (eCST) that is used as an intermediate representation of the source code. This characteristic gives the tools more generality comparing to the other similar static analyzers. The aim of this paper is to describe an early idea for introducing support for static timing analysis and Worst Case Execution Time (WCET) calculation at code level in SSQSA framework.

Categories and Subject Descriptors: **D.2.8 [Software engineering]**: Metrics - *Performance measures*

General Terms: Languages, Experimentation, Measurement

Additional Key Words and Phrases: Worst case execution time, language independency

1. INTRODUCTION

The quality of each product, and therefore also the quality of the software product, can be described as the degree to which a given product meets the needs and requirements of users. Software quality model defined by standard ISO 9126-1[1] distinguishes six attributes of software quality: functionality, usability, reliability, efficiency, portability, and maintainability.

The mentioned attributes of software quality can be monitored, evaluated, and controlled at early stages of software development by examining the source code and other static artefacts, or during the execution and testing process. Assessment of software quality attributes that is made on the source code or any of its internal representations without executing the program is called static analysis, while analysis of the program during execution time is called dynamic analysis. In the modern approach of software development, a great importance is given to monitoring and quality control in the early stages of development. Therefore static analysis becomes more important.

One of the important quality attributes of real-time systems is the execution time. It is highly important for these systems to provide required services on time. One of the parameters to be measured in order to guarantee this attribute in real-time system quality monitoring is Worst Case Execution Time (WCET) [Wilhelm et al. 2008; Lokuciejewski and Marwedel 2009; Lokuciejewski and Marwedel 2011]. It is measured as a part of timing analysis and provides value of the longest execution time of a program that can ever occur. It can also be predicted as a part of static analysis as well as measured as a part of dynamic analysis.

This paper provides early research toward support of static timing analysis and WCET calculation in SSQSA framework. In doing that, we shall take ALF as a domain specific language for WCET [Gustafsson et al. 2009] as our starting point and import the basic timing data into a SSQSA framework. The main difference between SQQSA set of static analyzers and ALF is in their level of abstraction: while ALF is intermediate *language*, enriched Concrete Syntax Tree - eCST (on which SSQSA framework is based) is a universal intermediate *data structure*. Furthermore, there are only several translators to ALF (C, C++,

This work was partially supported by the Serbian Ministry of Education, Science and Technological Development through project "Intelligent Techniques and Their Integration into Wide-Spectrum Decision Support", no. OI174023 and by EU ICT COST action IC1202 "Timing Analysis on Code-Level (TACLe)".

Author's address: G. Rakić, Z. Budimac, Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Trg Dositeja Obradovića 4, 21000 Novi Sad, Serbia, email: {goca, zjb}@dmi.uns.ac.rs.

In: Z. Budimac, T. Galinac Grbac (eds.): Proceedings of the 3rd Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA), Lovran, Croatia, 19.-22.9.2014, published at http://ceur-ws.org.

[1] http://www.sqa.net/iso9126.html

some assembler languages)[2] [Gustafsson et al. 2009] while SSQSA supports many more high-level languages (Modula-2, Delphi, Java, C#,...). By including ALF into a SSQSA environment, we hope that we can introduce the WCET analysis to a much broader class of languages.

In the rest of the paper we describe the planned approach. Background for a described idea is provided in the subsequent section, while section number 3 illustrates the idea. Related work is provided by section number 4. Conclusion with plans for future work is given in the last section.

2. BACKGROUND

The basic idea for integrating static timing analysis into SSQSA framework is to introduce support for domain specific languages (e.g. ALF [Gustafsson et al. 2009]) and to use the eCST Generator [Rakic and Budimac 2013, Kolek et al. 2013] to generate eCST containing all needed information. Exporting of timing information is to be done by attaching timing attributes to universal nodes representing specific program constructs as proposed by [Parsa and Mehdi 2014].

2.1 SSQSA internal representations of a software product

The main characteristic of SSQSA framework is its independency of the input computer language based on language independent tree representation of the source code - enriched Concrete Syntax Tree (eCST). Basic concept used here is to use universal nodes to enrich syntax tree so that they annotate semantics of the construct in its sub-tree. Universal nodes are constructed 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 the eCST representation of the source code we can (independently of an input language) generate other source code representations.

Generation of eCFG (enriched Control Flow Graph) 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 a middle and low level universal nodes. Work on this task has been recently finished.

Furthermore, based on mainly high-level universal nodes we can generate different kind of software networks completely independently of an input language [Savic et al. 2014]. For timing analysis, the most important network is one corresponding to a call graph. Based on an eCFG and this network we can create inter-procedural CFG.

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 [Wilhelm et al. 2008, Lokuciejewski and Marwedel 2011]. However, for implementation of the WCET calculation algorithm we need to additionally enrich eCST by timing analysis specific attributes. This is to be introduced as XML attributes added by postprocessor based on ALF representation of the source code.

2.2 ALF language integration

ALF is an intermediate language used to represent input code written in language on high, middle, or low level. It can be also used to represent intermediate codes. Code represented by ALF language is adapted so that the WCET calculation is enabled. In other words, it contains all needed information.

[2] http://www.mrtc.mdh.se/projects/wcet/home.html

Currently, three translators exist: a translator from C/C++ to ALF, translator from proprietary IAR intermediate code to ALF, and translator from binary code to ALF.

By generating eCST for source code represented by ALF code we will be able to run WCET algorithms on it.

As described in [Kolek et al. 2013], we will need a grammar for ALF language in order to produce scanners and parsers to be used in eCSTGenerator. Furthermore, we have to determine which universal nodes are needed to implement algorithm. From this point of view the most important universal nodes already exist in the current catalogue. Furthermore it is possible that we will need to introduce a set of domain specific universal nodes.

3. ILLUSTRATION OF THE IDEA

Let us look at the input source code [Gustafsson et al. 2009]:

```
if (x > y) z = 42;
```

Figure 1 represents corresponding eCST for this segment of source code.

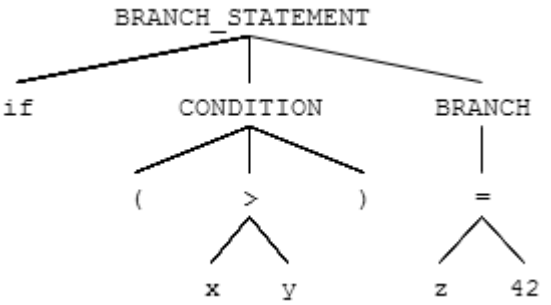

Fig. 1. eCST for if statement in C code

This segment can be translated into the ALF code below:

```
{switch    {s_le 32       {load 32 {addr 32 {fref 32 x} {dec_unsigned 32 0}}}
                          {load 32 {addr 32 {fref 32 y} {dec_unsigned 32 0}}}}
           {target        {dec_unsigned 1 1}
                          {label 32 {lref 32 exit} {dec_unsigned 32 0}}}}
{store     {addr 32 {fref 32 z} {dec_unsigned 32 0}}
           with {dec_signed 32 42}}
           {label 32 {lref 32 exit} {dec_unsigned 32 0}}
```

Generated segment of the ALF source code is much longer than the corresponding C code. Therefore corresponding eCST is also much larger. Figure 2 represents corresponding eCST for generated segment of ALF source code. Here we provide only the part of generated tree in order to demonstrate equivalency of trees for two languages.

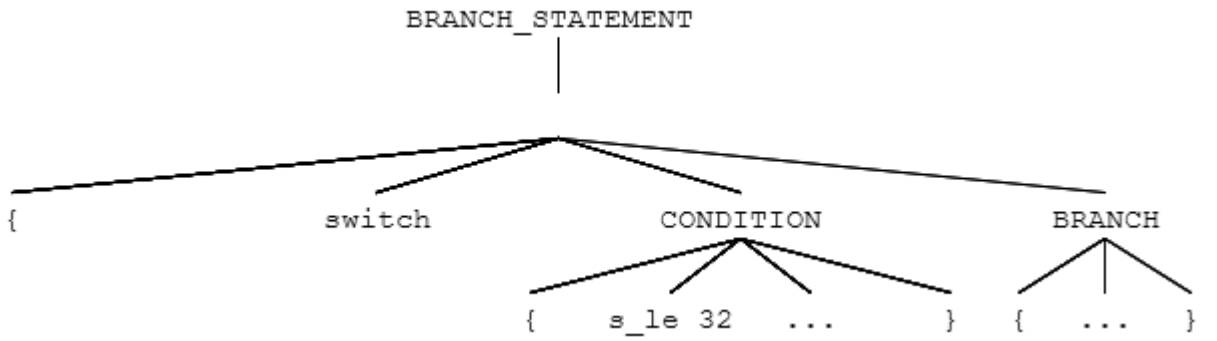

Fig. 2. Part of eCST for switch statement in ALF code

3.1 Exporting information

In this section we will demonstrate idea for extracting XML timing information. For these purporses we will take basic example of branch statement. For all other needed information we can follow similar idea to export needed data.

Based on [Parsa and Mehdi 2014] to store timing information for switch statement we need best (BTime), worst (WTime), and total (TotalTime) execution time attributes, and for each branch we need (execution) Time:

```
<SwitchBlockK BTime=n WTime=n TotalTime=n>
        <CaseBlockK Time=n>
        …
        </CaseBlockK>
        …
</SwitchBlockK>
```

For If statement we need the same attributes.

```
<IfBlockK BTime=n WTime=n TotalTime=n … >
        <ThenBlockK Time=n … >
        …
        </ThenBlockK>
        <ElseBlock Time=n >
                …
        </ElseBlockK>
</IfBlockK>
```

In eCST representation this would be encompassed by the unique BRANCH_STATEMENT node

```
< BRANCH_STATEMENT" BTime=n WTime=n TotalTime=n>
        < BRANCH" Time=n>
                …
        < BRANCH" Time=n>
                …
```

Similarly we can extract all needed information from the source code to our XML representation (loops, conditions, etc.)

This XML representation will enable the flow of timing information between the tools, but also visualization of generated information, which means easier manipulation with timing facts.

4. RELATED WORK

Timing analysis and WCET calculation is a very actual research topic and many tools are currently under development. Our attention is on tools calculating WCET on the level of a source code.

aiT[3] [Lokuciejewski and Marwedel 2011] is a tool for static WCET analysis and is used to compute a safe upper bound of WCET. It accepts binary executable as its input, from which the control-flow graph is reconstructed. This graph is a code representation on which several static analyses are implemented to compute the execution time of each instruction. A global path analysis is used to compute overall WCET bound of tasks.

Bound-T[4] is a WCET static analyzer with similar characteristics. It takes machine code as input and (based on control flow paths) generates WCET bounds and (optionally) stack-usage bounds.

FORTAS (the FORmal Timing Analysis Suite)[5] combines execution time measurements with static program analysis techniques. It estimates WCET of software tasks running on embedded real-time systems based on a hybrid approach following the general principles of measurement-based timing analysis.

SWEET (Swedish WCET Analysis Tool)[6] is a WCET analysis tool consisting of a flow analysis, a low-level analysis, and a WCET estimation. SWEET analyzes the intermediate format ALF [Gustafsson et al. 2009]. Given a code format, SWEET can perform a WCET analysis for it if there is a translator into ALF. Therefore, SWEET currently supports C/C++, IAR, and binaries, as previously mentioned.

By integration WCET static analyzer in SSQSA framework we can expect to cover wider specter of possible inputs which is the main goal of this research.

5. CONCLUSION AND FURTHER WORK

In this paper we propose a possible approach to enable support for WCET calculation into SSQSA framework. An idea based on introducing support for domain specific language is described in order to implement and test calculation algorithm. After this task is completed, exploration of the possibilities for extending the analysis to other supported languages is needed. The aim is to enable uniform application of the same algorithm implementation to all supported languages. This task should have two phases. First phase is to support language independent flow analysis. This will be the straightforward activity as generation of all language independent code representation is already enabled. Second phase is the platform dependent generation of WCET values. This task will require deeper research on extracting specific information to eCST. It may require introducing of the domain specific universal nodes. By these nodes we would annotate domain specific information in eCST. In validation stage, gained results are to be compared with results generated by existing language specific WCET calculation tools. First level validation will be by comparing gained results with the results generated by the SWEET tool.

REFERENCES

J. Gustafsson, A. Ermedahl, B. Lisper, C. Sandberg, L. Källberg. 2009. 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

J. Kolek, G. Rakić, M. Savić. 2013. 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.

P. Lokuciejewsk, P. Marwedel. 2009. 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

P. Lokuciejewski, P., P. Marwedel, P. 2011. Worst-case execution time aware compilation techniques for real-time systems. Springer.

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

G. Rakić, Z. Budimac. 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.

G. Rakić, Z. Budimac**.** 2011., 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] http://www.absint.com/ait/

[4] http://www.bound-t.com/

[5] http://www.fortastic.net/

[6] http://www.mrtc.mdh.se/projects/wcet/home.html

M. Savić., G. Rakić, Z. Budimac, M. Ivanović (2014), 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

R. Wilhelm, J. Engblom, A. Ermedahl., N. Holsti., S. Thesing, D. Whalley., G. Bernat., C. Ferdinand., R. Heckmann., T. Mitra, F. Mueller., I. Puaut, P. Puschner., J. Staschulat., Per Stenstrom. 2008. 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.

.