**Automated Error Diagnosis Tool Clafer Compiler**

ECE750-T24

Matthew Ma, Jesus Jesus Alejandro Padilla

Department of Electrical and Computer Engineering, School of Computer Science

University of Waterloo

Waterloo, Ontario

# **Abstract**

Software bugs have been causing tremendous issues in software reliability and system availabilities. Compilers like all other software, would also be prone to bugs, logical errors which in some case are more critical to performance and accuracy of the generated results.

In this paper, we present a novel compiler bug diagnosis tool that can assist the programmer to localize the error at each stage of the compiler. We apply the principle of regression testing to compare against two versions of the compiler, and determine the changes from different stage of the compiler generation represented by intermediate representations. We apply efficient algorithm to compare the IR, and filter out the legal changes from rule set which the user can specify through user interface or as a template.

We experimented our tool with Clafer compiler from the Generative Software Development Lab at the University of Waterloo, and the results show that our tool can determine most of the compiler bugs that are confined within Clafer language with acceptable performance overhead, memory usage and user experience,

Keywords: software bug; compiler; IR; regression testing; Clafer; XPath; encapsulation

1. **INTRODUCTION**

**1.1 Motivation**

Software reliability is of key concern in both industry and academia research. Although program analysis in detecting and fixing software bugs have been studied widely for the past decades, software bugs still exists, and continue to cause problems in data integrity, result accuracy and system availability to name a few.

Compilers like all other software, would also be prone to bugs, logical errors which in some case are more critical to the accuracy of the generated results.

In Microsoft Visual C++ Team Blog, hundreds of bugs are listed for well developed IDE, Visual Studio 2012[1]. The number is overwhelming considering the fact that Visual Studio has been out on market for years. Furthermore, it is clearly stated that this number are far from exhaustive; they don't contain private Connect bugs or non-Connect bugs (which come from many sources: compiler testers, STL maintainers with last names ending in J, other MS teams like Windows and Office, external customers with support contracts, etc.).  Also, some public Connect bugs aren't represented, in particular, /analyze and /ZW bugs. Thus it is not hard to imagine the scale and resources a commercial company spends in diagnosing, testing and fixing the compiler bugs.

With growing interests in solving compiler related bugs, we decided to study further to develop automated error diagnosis tool for open source compiler. For time constraints and limited resources, it is decided to focus this study on Clafer compiler from the GSD Lab at the University of Waterloo.

**1.2 BACKGROUND**

Clafer is a lightweight modeling language developed at Waterloo’s GSD Lab. This language provides a simple syntax that’s easy to learn and that allows the generation of models that can later be verified for consistency. It can be used in many scenarios. Currently it’s being applied to software product lines, concept modeling, multi-objective optimization. Although Clafer is not yet widely adopted, some researchers have used it in academic teaching and various research projects. Due to its open source initiatives, Clafer is available with no cost to the public, and this modeling language is at its continual development by GSD lab researchers. It’s believed that as the language matures, it’s going to be used more extensively in academia and industry.

The core of Clafer is the compiler which translates the user written model into other formats, such as Alloy, Clafer Sugar, XML and others for reasoning and processing. Once the model is in Alloy, Clafer Instance Generator can be used to generate either valid instances, that prove the model is correct, or no instances if there’s a problem with the model. Clafer compiler is written in functional language, Huskal.

The current methodology and the toolset used to test Clafer are not yet available; there is no testing framework for compiler related bugs, and the user could only manually go through the source code while relying on experience to determine the source of the error. During the compilation process, positive examples are generated and added to a repository to be used in the future references. Through user questionnaire, we determined that the typical tests applied are to compare if the output is exactly the same as the original positive example; the comparison is done using a simple text diff which is inefficient and would introduce inaccuracies as the compiler evolves. Moreover, a simple update in the compiler might cause certain innocuous changes, for instance, different IDs, that will make all the comparison tests to start failing.

Furthermore, when a real bug within the compiler is detected, the current testing approach provides no summary, visual representation or any other form of help whatsoever to track the fault down. It’s up to the developer to retrieve the useful information from all the differences identified and perform the troubleshooting to identify the root of the error.

It’s obvious that to fix the bugs in Clafer compiler, it is required a more sophisticated, efficient and user friendly approach; moreover, this approach should be automated to accommodate the on-going growth of the scale of Clafer compiler. There are several problems, some of which are particular to Clafer, that need to be addressed to be able to provide a more comprehensive solution, and solving these problems is at the core of this research project.

1. **OUR CONTRIBUTIONS**

In this paper, we propose a novel error diagnosis tool for compiler which can help user to localize the error in different stages of the compiler of interests. More specifically, our tool includes the following major key ideas and contributions.

(1)By intercepting the Clafer compiler, we proposed a new approach to produce intermediate representation for each stage of the compilation process within Clafer through the API we create. By doing so, we could apply regression testing idea to find the differences at each stage of the compilation process. We realize that error tends to propagate as differences in the format of IR so our tool always recommend the user to fix the potential bugs from the earliest stage as identified by the earliest “bad” changes.

(2)We modified existing difference comparison tool and output our IR in desired format to increase the efficiency in finding the IR differences from two version of the compiler. To allow selection and filtering legal changes, we introduce the concept of rule set, where users can specify the legal changes that are allowed from the compiler version changes. This interactive approach not only gives users more flexibility but also reduces the time taken to track down to the compiler bugs as demonstrated in later sections.

(3)Last but not least, we apply XPath and XSLT techniques interfaced with Java so that our tool has extensive control on the change information within the IR. This is very novel approach as it means that we can further enhance out tool in the future not only for diagnosis purpose but also for fixing the error within the compiler.

1. **PROPOSED IDEAS AND OVERVIEW**

In higher hierarchical view, our approach is based on comparing two versions of the compilers to determine the bugs within one of them. We use the idea from regression testing, and we assume that the previous version of the compiler is often bug-free after certain time of debugging and maintenance. Although there is chances that the older version of the compiler has more bugs and our assumption might not be correct, by investigating Clafer compiler further and acquiring expert’s suggestion, it is determined that this assumption is reasonable to make for this specific case.

**3.1IR Production**

Since one of the main goal is to reduce the user’s time spent in finding compiler bugs, we propose to give user more information about where does the changes between two compilers come from.

IR is the intermediate representation that a compiler generates during compilation process. We propose to generate IR at all the stages involved in Clafer’s compiling process so that it can narrow done the IR differences in each stage. By doing so, user can find directly to the phase of the compiler code that implement this stage of compilation process. As discussed earlier, error tends to propagate as differences in the format of IR so the user can pinpoint to fix the potential bugs from the earliest stage as identified by the earliest “bad” changes.

Since Clafer compiler is written in Huskal, our first task is to create an API that can ask Clafer compiler to generate the IR for our readily use.

**3.2Difference Finder**

Once IR from each stage of the compilation process is present, we want to find the differences amongst them. Difference finding is of critical importance as this is one of the bottlenecks for performance issues. Since IRs can be big in size, efficient algorithm is required to compare against two IR documents and extract all the changes accurately. In our implementation which will be discussed in detail, we noticed the hierarchical structure in the IR, and we modified the IR structure accordingly to accommodate efficiently with the difference finding algorithm we are employing.

It should be also noted that the IR found at this step should include all the differences amongst the two versions of the compilers.

**3.3 Extracting and Filtering Legal Changes**

As noted above, there are certain changes in the compiler that are perfectly acceptable. For example, when updating the compiler version, the ID number is changed accordingly. Such a change is reflected in the IR, and according to our investigation, these legal changes can be as much as 20%- 30% of the total changes.

Since overwhelming information interferes with user’s effort to determine the compiler errors, we wish to have the capability to operate on each difference found amongst the IRs so that we have the control. In this way, user can explicitly specify the legal changes to be accepted to our tool, interactively eliminating redundant changes.

**3.4User Interface**

At last, we want to have a visual representation for the findings by our tool so that the user can see the differences in the IR from different stage. The user should be also given ability to specify the changes and differences that are acceptable through the UI so that this information is transferred to our tool’s back-end so reflect such a decision dynamically.

We believe UI serves as effective visual aid that serves as an important presentation of our Clafer compiler error diagnosis tool to help the user determine the root error efficiently without overwhelmed by unnecessary information from our tool.

1. **IMPLEMENTATION**

Sdsd

1. **EVALUATION**

It

1. Inverse Measurement Model Map at Single Time Step

##### **References**

1. <http://blogs.msdn.com/b/vcblog/archive/2012/08/10/10338661.aspx>
2. Waslander S., “ME 597: Motion Modelling”, ME 597 Lecture Slides, University of Waterloo, pg. 29, October 1st 2010
3. Waslander S., “ME 597: Mapping I”, ME 597 Lecture Slides, University of Waterloo, pg. 38, October 1st 2010
4. Waslander S., “ME 597: Mapping I”, ME 597 Lecture Slides, University of Waterloo, pg. 30, October 1st 2010
5. Waslander S., “ME 597: Mapping I”, ME 597 Lecture Slides, University of Waterloo, pg. 42, October 1st 2010