Verifying Functional Equivalence Between C and Fortran Programs

Wenhao Wu and Stephen F. Siegel (Advisor)
Verified Software Laboratory, Department of Computer and Information Sciences, University of Delaware
Newark, Delaware, USA
wuwenhao@udel.edu

ABSTRACT
Software verification is a mature research area with many techniques. These verification approaches can be applied to programs written in different programming languages; nevertheless, most verification tools are only designed for programs written in C or Java. As a result, verification tools are inadequate for other languages, such as Fortran. A high level of software safety is mandatory in most of its application scenarios, which makes verification tools for Fortran programs necessary and significant. In this poster, the author illustrates the motivation and objectives of the project with examples. Also, this poster shows an extension (as a Fortran program verifier) of an existing verification platform – CIVL. Additionally, the results of a set of extensive experiments conducted by the author is shown in this poster to indicate that the performance is satisfactory.

KEYWORDS
verification, functional equivalence, Fortran, C

1 INTRODUCTION
Fortran is an imperative programming language strongly suited to intensive numeric computation in scientific areas, which are commonly related with high performance computation (HPC). C is another widely used programming language for developing HPC applications for those fields requiring massive computations. Additionally, both languages have numbers of their own parallel dialects (e.g., MPI[1], OpenMP[3], CUDA[4], OpenACC[7], etc). In spite of the fact that many safety-critical HPC programs involving both languages must be verified to avoid severe accidents caused by defects, few verifiers are designed and developed to verify the functional equivalence between Fortran and C programs while these verification tools can only perform a static analysis in syntactic level (but not in semantic level). The shortage of verifiers (i.e., tools aim to verify the functional equivalence between programs written in Fortran and C) inevitably results in expensive costs on not only maintaining risky software but also unreliable manual verification by inspecting numerous lines of code.

The prototype shown in the poster intends to relieve the shortage. This extended Concurrent Intermediate Verification Language[9] (CIVL) framework should be able to automatically verify the functional equivalence between Fortran and C programs. By parsing and transforming two given set of Fortran and C source code files, the extended CIVL will translate both sets into a CIVL intermediate representation (CIVL-IR). And then, developed verification procedures can be applied on the unified CIVL-IR.

In addition, several sets of experiments are performed on the prototype presented in this extended abstract. Each set is designed for a specific feature of CIVL. According to results from those experiments, the author can conclude that the presented CIVL framework can automatically verify the functional equivalence between basic sequential Fortran and parallel C programs with satisfactory cost on time and memory space.

In the future, the author will continue fulfilling this framework prototype, so that CIVL will be able to verify more complex software with various dialects of both C and Fortran. Also, CIVL primitives for C language will be extended into Fortran to support more powerful verification.

2 MOTIVATION
As the introduction mentioned, the insufficiency of functional equivalence verifiers supporting both C and Fortran will cause various difficulties on maintaining or porting legacy code, mitigating risks of unexpected inequivalent behaviors, and reducing the cost of both human and machine resources.

Currently, a popular alternative approach of verifying functional equivalence is either to convert a program from Fortran to C or vice versa; nevertheless this approach has several considerable defects (e.g., unreliable/inequivalent conversion or outdated tools). Additionally, for those large-scale programs, it is quite difficult and inefficient for human to manually assess the correctness of them.

3 APPROACH
The presented prototype is an extension of CIVL, which is a verification platform for concurrent C programs that use the MPI, OpenMp, CUDA, and PThreads APIs. A Fortran parser has been integrated, which is derived from Open Fortran Parser[8] (OFP) and implements interfaces provided by the CIVL verifier. Furthermore, to adapt the AST structure used by CIVL, an additional procedure of transforming a Fortran parsing tree into CIVL-IR structure has been applied. The CIVL-IRs are represented as sets of Abstract Syntax Trees (AST) constructed from parsing trees generated from source files. CIVL traverses the state space in terms of the required properties (such as deadlock) to search for violations in all reachable states.

3.1 Transforming the parsing tree into CIVL-IR
The transformation from Fortran parsing tree to CIVL-IR is the most challenging and interesting part. The author gives a further illustration about transforming procedure in this sub-section.

The parser can only parse a Fortran parsing tree, but a general CIVL AST must be provided to generate the state space, which is verified by the CIVL verifier. Thus, it is necessary and crucial to build
a transformer converting a Fortran parse tree into an CIVL AST by traversing nodes. And three approaches are used for transforming a Fortran syntactic structure into CIVL-IR structure.

- The first, also the simplest approach, is to interpret a Fortran structure with an existing CIVL AST node representing a completely equivalent structure (in C).
- The second approach is to represent a Fortran structure by using the similar but not same CIVL AST node or structure. (e.g., do-loop structure can be translated as for-loop. Fortran functions can be transformed as C functions with all parameters passed by reference, etc.)
- The last, also the most tricky approach, is to simulate a Fortran units with a totally different CIVL unit or a set of them. (e.g., common blocks in Fortran can be simulated by using global struct in C)

4 EVALUATION

This list shows the environment settings for the evaluation experimentation.

- CPU: 2.5 GHz Intel Core i7
- Memory: 16 GB 1600 MHz DDR3
- Operating System: macOS Sierra ver. 10.12.6
- Dependencies:
  - VSL dependency libraries.
- CIVL: v1.11.1+ of 2017-08-08

Figure 1 shows both information and the result of each major experiment. All experiments perform functional equivalence on a pair of C and Fortran programs

- residual: A pair of simple sequential Fortran and C programs with different implementations.
- harmonic: A pair of sequential Fortran and C programs with exactly same implementations. Some involved math operations are statements in Fortran but functions defined in ‘math.h’ in C
- mmx: A sequential Fortran program and a concurrent C program with similar and equivalent implementations.
- mmx_bad: A sequential Fortran program and a concurrent C program with inequivalent implementations. Thus, an assertion violation should be reported, which indicates both outputs are same.

5 CONCLUSION

According to the evaluation result, the current CIVL framework can automatically verify the functional equivalence between simple sequential Fortran programs and parallel C ones with satisfactory cost on both time and memory space. Additionally, the input Fortran source code can be translated into CIVL-C code (i.e., C source code contains CIVL primitives).

6 FUTURE WORK

In the future work, prospective yet unimplemented functionalities are described, mainly including following 3 aspects:

1. completely supporting on Fortran 2015 Standard[6] (F2015);
2. processing commonly used operations in main Fortran parallel dialects (e.g., OpenMP[3], OpenACC[7], etc);
3. adding CIVL primitives (e.g., $input$, $output$, $assert$, $assume$, etc) for the more powerful verification performed by CIVL.

ACKNOWLEDGMENTS

The author would like to acknowledge Prof. Stephen F. Siegel for all instructions and helps on using and extending CIVL to verify the functional equivalence between Fortran and C programs.

REFERENCES