Creating a Robust AspectMATLAB compiler

Hongji Chen
McGill University
Montreal, QC, Canada
hongji.chen@mail.mcgill.ca

Laurie Hendren (advisor)
McGill University
Montreal, QC, Canada
hendren@cs.mcgill.ca

1. PROBLEM AND MOTIVATION

MATLAB [5] is a dynamic programming language that widely used in scientific computation. Aspect oriented programming (AOP) allows programmers to separate out concerns by defining "pointcuts" which identify certain program events such as writing to a variable, or entering a function, and then associate with those pointcuts advice, or actions, which will execute at those points. This is accomplished by the compiler weaving the advice codes into the appropriate program locations that match the pointcuts.

This project aims to build on previous work to create a robust and semantically well-defined version of AspectMATLAB. In addition, more powerful patterns and matching strategies have been developed and implemented, extending the functionality of AspectMATLAB. Having a robust and well-defined AspectMATLAB will enable scientists to use aspects to add functionality to their base MATLAB programs. Example uses include sparse matrix detection, dynamic type checking, custom profiling, and much more.

2. BACKGROUND AND RELATED WORK

AspectJ [4], an AOP implementation for Java, has had a huge influence in the Java world, and Jastadd [7] framework utilizes the idea of AOP to shorten the development period for compiler by generating AST structure from aspects.

Our research project is based on the success of Toheed Aslam's AspectMATLAB compiler [8] and Andrew Bodzay's AspectMATLAB++ compiler [1]. The existing AspectMATLAB compiler allows programmers to define patterns("pointcuts") and actions("advice"). Along with variable access, function call and execution, which are present in most AOP languages, the existing AspectMATLAB compiler introduces loop patterns to capture loop statement [2, p.15-17], and shape matching and attribute matching, which are both important in scientific computing [3, p.14-15].

3. APPROACH AND UNIQUENESS

The existing compiler demonstrated that the idea of AspectMATLAB was interesting and worthwhile, but it has many weaknesses including: incomplete source code parsing, undefined and unimplemented pattern validation, an incomplete transforming strategy, and limitations on argument parsing. The key motivation of this project was to identify all of the weaknesses and to improve and extend the language definition and the compiler. As a result of this project, the new AspectMATLAB language definition and the compiler is much more complete and robust.

ACM ISBN .
DOI:

4. RESULTS AND CONTRIBUTIONS

The project has resulted in a new implementation of the AspectMATLAB compiler which improves over the previous ones in the following ways:

Use of more Robust front-end: The previous AspectMATLAB implementations used an incomplete front-end, and hence could not parse many MATLAB Programs. This project integrates a much more robust front-end using Samuel Suffos's new MATLAB parser [6].

Clear Grammar: The existing AspectMATLAB compiler provides 11 different types of patterns, but some of the patterns have confusing syntax and/or semantics. We also noticed that some of the patterns are usually used together, such as variable get pattern usually forms a compound pattern with dimension pattern, and type pattern. Thus, we developed a new set of pattern grammar rules to remove the ambiguous and redundant patterns, while also making it backward compatible with the existing grammar.

Semantic Validation: Limited semantic validation is applied on the patterns in the existed AspectMATLAB compiler. Semantic invalidation will make the source code hard to read and even lead to unpredictable results during code transformation. Thus we have developed a simplification and analysis process for patterns. This includes pattern type analysis, pattern modification validation, pattern syntax validation. The current version of AspectMATLAB will raise errors and warnings during compilation.

More Robust Transforming Strategy: We also improved the transforming and weaving strategy for AspectMATLAB. The existing AspectMATLAB uses a simple transformation strategy on cell indexing, variable set and comma separate list handling. The original transformation resulted in improper resolving of assigned values and cannot correctly identify the spanned comma separate list. We have implemented a new transformation framework and a sound transforming strategy, making the compiler more robust.

Extended Argument Matching: In the existing AspectMATLAB, programmers are only allowed to match the function call and execution using the number of arguments. The new AspectMATLAB takes this a step further. We have developed a new matching strategy for argument list which allows the programmer to specify both type and shape information for each argument, as well as using wildcards to make a general match. This general matcher is implemented as a generalized DFA.

This project has identified many weaknesses in the previous AspectMATLAB compilers, and has developed a new version of AspectMatlab, which builds upon a more robust front-end, addresses many shortcomings in the previous AspectMATLAB versions, and provides extensions to the AspectMATLAB language.

The new AspectMATLAB compiler can be used in different kinds of MATLAB optimization, such as sparse matrix tracing, subroutine profiling, loop unrolling and dynamic call graph generation. It can also be used to generate safer MATLAB code, by using aspects for argument type and shape checking. Another important usage is to provide analysis and profiling features using predefined aspects for other MATLAB toolkits, for example, McWeb, a web-based interface for MATLAB analysis, profiling and source-to-source compilation.

Acknowledgements

This work was done as a summer research project, supported by a SURA and NSERC. The authors would like to thank the donors who provide the support for the SURA awards. We would also like to thank Samuel Suffos for his help in integrating his new front-end with AspectMATLAB.

5. REFERENCES

[1] Laurie Hendren Andrew Bodzay. Aspectmatlab++: Annotations, types, and aspects for scientists. *MODULARITY'15*, 2015.

[2] Toheed Aslam. Aspectmatlab: An aspect-oriented scientific programming language. Master's thesis, McGill University, 2010.

[3] Andrew Bodzay. Aspectmatlab++: Developing an aspect-oriented language for scientists. Master's thesis, McGill University, 2014.

[4] Eclipse Foundation. Aspectj. URL: https://eclipse.org/aspectj/.

[5] The MathWorks. Matlab. URL: http://www.mathworks.com/products/matlab/?s_tid=hp_ff_p_matlab.

[6] Samuel Suffos. mclab-parser. URL: https://github.com/Sable/mclab-parser.

[7] JastAdd Team. Jastadd. URL: http://jastadd.org/web/.

[8] Anton Dubrau Toheed Aslam, Jesse Doherty and Laurie Hendren. Aspectmatlab: An aspect-oriented scientific programming language. *AOSD'10*, 2010.