MATLAB to C++ Translator

John Till, University of Tennessee

Although the primary benefit of MATLAB is its ease of use as a high level language, there are many cases were it would be useful to compile MATLAB code. The simplest technical solution to this problem is a translator, that is a program which converts MATLAB code to a language which already has a compiler. Mathworks has developed “MATLAB Coder” [2], and various research groups have created translators [1, 3–6]. However, these translators all require complete static type deduction, whereas dynamic typing is one of the main strengths of MATLAB. The ecosystem around MATLAB would be strengthened by a translator which faithfully implements the MATLAB language, ultimately allowing users to compile MATLAB code without knowing another language. The goal of the translator is to faithfully implement the MATLAB language so that the compiled code is semantically equivalent to interpreted code. Previous MATLAB to C++ translators have not been literal translators in this sense. The translator will be written in MATLAB with the goal of eventually “bootstrapping”, i.e. creating an efficient implementation of the translator by translating its MATLAB source code.

The translator uses the following modules to convert MATLAB code to C++ code:

1. **Scanner**: This phase analyzes the program source text to recognize tokens, which are the smallest chunks to have meaning, e.g. variable names, keywords, and relevant symbols.

2. **Parser**: This phase analyzes the tokens to find the higher level meaning of the program. This is where program statements and expressions are constructed. The output of this stage is an abstract syntax tree as shown in Figure 1.

3. **Symbol Resolution**: The meaning of function and variable names depends on the context of the program. This phase determines what each name means based on MATLAB’s scoping rules.

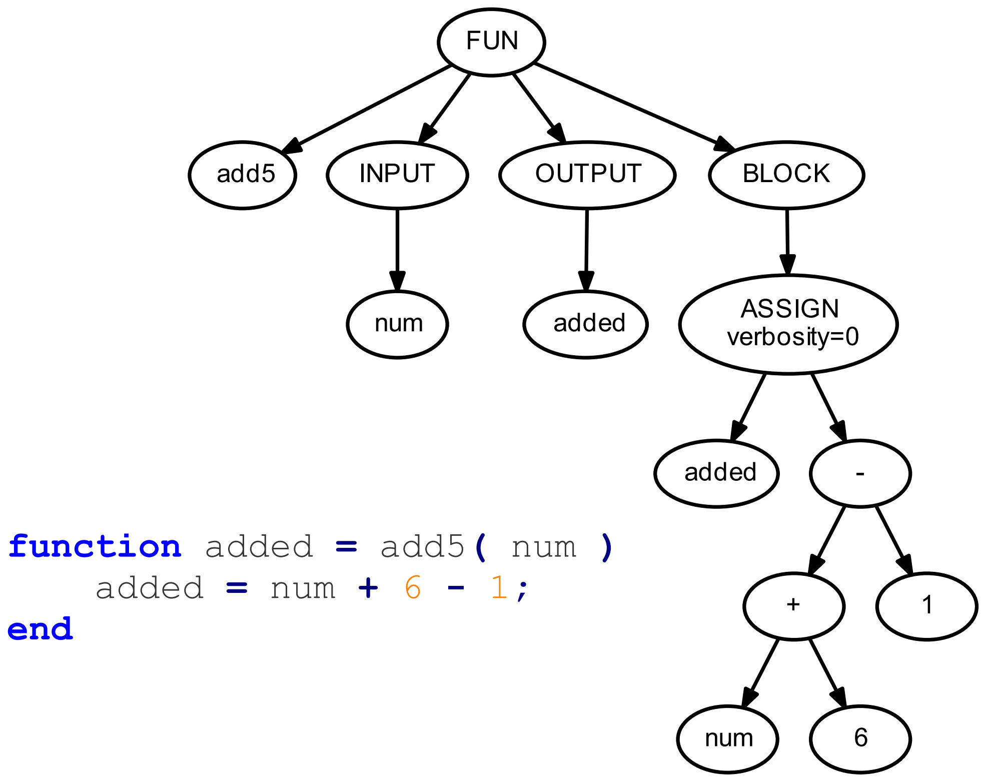
4. **Type Resolution**: MATLAB is dynamically typed and the type of a variable cannot always be statically determined. However, it is beneficial to statically determine types whenever possible as this reduces overhead at runtime.

5. **Size Resolution**: The MATLAB language was constructed to allow easy creation and manipulation of matrix data, so in addition to a type, most variables also have a size. Statically determining the size will improve the performance of compiled code.

6. **Code Generation**: Finally with all the information gathered from previous stages, the C++ code is written. Each programming construct in MATLAB must be mapped to an equivalent construct in C++.

These stages are all basically in place and working, although further refinement is necessary. The PI has implemented the translator with several scalar variable types, but implementing matrices is essential and will likely prove challenging. A previous translator used the Armadillo matrix library because its syntax resembles that of MATLAB [6], but the PI is likely to use Eigen as its header-only format would be an advantage. In addition to supporting dynamic typing, the PI also plans to support dynamic sizing, which is an important feature of the MATLAB interpreter typically not implemented in translators.

MATLAB has complicated scoping rules for nested functions. The functions form a tree of scopes, and any variable used in scopes directly vertical to each other is shared between the scopes.



```
function added = add5( num )
    added = num + 6 - 1;
end
```

Figure 1: The parser output, an unannotated abstract syntax tree, is shown for a simple function. In addition to generating code, the translator can also generate DOT graphs for debugging purposes.

For example, a common identifier of both a parent and child function refers to a shared variable. Two child functions using the same identifier will refer to separate variables, unless the parent also uses that variable. While MATLAB's scoping rules are complicated at a glance, they can be easily implemented in C++ using lambda functions with automatic capture arguments. Without gratuitous detail, many of MATLAB's features such as multi-output functions, ignored output arguments, global variables (which introduce dynamic scoping), and persistent variables can be implemented in C++. By strictly conforming to the MATLAB language, the proposed translator will make compiling MATLAB code as easy as interpreting it.

The proposed Matlab to C++ is operational for a subset of valid MATLAB programs. The scanner, which analyzes the program source text to recognize programming constructs, is complete. The parser, which uses the scanner result and language definition to generate a tree structure of the program, is implemented with an LL(1) recursive descent algorithm and supports the entire MATLAB syntax except for "classdef" and lambda expressions. The symbol resolution phase supports all of MATLAB's scoping rules for a single file, including the use of "global" and "persistent" variables, but calls to functions in external files have yet to be supported. The code generation phase currently supports a limited variety of scalar types.

References Cited

[1] J. Bispo, L. Reis, and J. M. P. Cardoso. Multi-Target C Code Generation from MATLAB. In *Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming - ARRAY'14*, pages 95–100, New York, New York, USA, 2014. ACM Press.

[2] J. Gibson and J. P. Henson. Getting the Most from MATLAB: Ditching Canned Routines and Embracing Coder. *Economics Bulletin*, 36(4):2519–2525, 2016.

[3] L. Hunyadi. MatForce: Supporting Rapid Algorithm Development by Automated Translation of MATLAB Prototypes into C++. In *SE '08 Proceedings of the IASTED International Conference on Software Engineering*, pages 324–329, Innsbruck, Austria, 2008.

[4] H. Kawabata, M. Suzuki, and T. Kitamura. A MATLAB-Based Code Generator for Sparse Matrix Computations. In W.-N. Chin, editor, *Programming Languages and Systems*, pages 280–295. Springer Berlin Heidelberg, 2004.

[5] G. Y. Paulsen, S. Clark, B. Nordmoen, S. Nenakhov, A. Andersson, X. Cai, and H. P. Dahle. Automated Translation of MATLAB Code to C++ with Performance and Traceability. In *The Eleventh International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 50–55, Barcelona, Spain, nov 2017.

[6] G. Y. Paulsen, J. Feinberg, X. Cai, B. Nordmoen, and H. P. Dahle. Matlab2cpp: A Matlab-to-C++ Code Translator. In *2016 11th System of Systems Engineering Conference (SoSE)*, pages 1–5. IEEE, jun 2016.