A new *Aheui* compiler based on Flex, Bison and LLVM

Junghoon Seo
20145071

Summary of the Proposal

Aheui [1] is the first esoteric programming language ever to be designed for the Hangul(Hangeul) . The aim of the language is to reflect the graphical design of Hangul. Aheui has no practicality at all, but it is a language that is often talked about in the Korean developer community because of its visual characteristics. Based on such pure interest, there are many works that have implemented Aheui of various interpreter behaviors. However, no work has been done in compiled form. This project focuses on making Aheui a compilable language on *LLVM*. This work will be done on Aheui's language specification, *Flex*, *Bison*, and *LLVM*. Therefore, the result of this work will be able to use existing functions and advantages of *LLVM* such as machine-independence and common IR optimizer. I just hope that this project will become a kind of another fun story for the Korean developer community.

Background

Aheui is the first esoteric programming language ever to be designed for the Hangul (Hangeul) . The aim of the language is to reflect the graphical design of Hangul. The aim of the language is to reflect the graphical design of Hangul. Aheui is a stack-based programming language that works in a similar fashion to Befunge / Funge-98, but it differs in that it is Hangul, not ASCII. Note that Befunge has *p* and *q* commends to generate self-modifying code because it was originally presented as a "language that is hard to compile" for research purposes, but Aheui has no such commend. The code for outputting the most famous program, "Hello, world!", Looks like this:

```
밤밣따빠밣밟따뿌
빠맣파빨받밤뚜뭏
돋박탕빠맣붏두붇
볻뫃박발뚷투뭏붖
뫃도뫃희멓뭏뭏북
뫃복토범더벌뿌뚜
뽑뽀멓멓더벓뻐뚠
뽀덩벖멓뻐덕더벅
```

Figure 1: A simple "Hello, world!" program in *Aheui*

Due to the interestingness of the language so far, the following tasks related to its implementation have been found.

- Interpreter implementations by other languages or runtime environments [2] such as Ruby, Scala, ActionScript, Rust, Python, PyPy, JavaScript, C, Node.js, and Go. In particular, *Rpaheui*, an interpreter implemented by PyPy, is a JIT interpreter.
- Works for visualization of Aheui code execution : *AVIS* [3], *AheuiChem* [5]
- An alternate implementation of the *Aheui* programming language written in *Aheui*, called *aheui.aheui* [4]

Although *Rpaheui* introduced an asembly-like code generator for code linearization, it was not code that could be used as an ISA. In conclusion, there is no work that essentially transforms the Aheui program for optimization or profiling. On the other hand, in the case of Befunge, which is inevitably more difficult to create a compiler, there are *bef2c* transpilers included in standard Befunge-93.

Goal and Objectives

This project focuses on making Aheui a compilable language on *LLVM*. It is possible to compare the performance of the proposed compiler with the other by comparing some of the references that run *aheui.aheui* on different implementations of *Aheui*. It is also possible to measure the speed of operation for several snippets.

Methods

This work will be done on Aheui's language specification, *Flex* [8], *Bison* [6], and *LLVM* [7]. All backend layers under the optimizer will be replaced by the core structure of *LLVM*, and I would like to use *Lex* and *Bison* in frontend design rather than using *LLVM* internal tools.

- *Flex* is a tool for generating scanners, programs which recognize lexical patterns in text. Lexical analysis is easy to be done with *Flex*.
- *Bison* is a general-purpose parser generator that converts an annotated context-free grammar into a deterministic LR or generalized LR (GLR) parser employing LALR parser tables. At this stage, the AST must be carefully designed.
- *LLVM* is designed for compile-time, link-time, run-time, and idle-time optimization of programs written in arbitrary programming languages. *LLVM* naturally takes the AST and turn it into machine code because it abstracts the actual instructions to something that is similar to an AST. Some opimization can be done in this step.

References

[1] Aheui. `https://aheui.github.io`, 2012 (accessed April 10, 2017).

[2] Aheui working group - git. `https://github.com/aheui`, 2012 (accessed April 10, 2017).

[3] AVIS. `https://aheui.github.io/avis`, 2014 (accessed April 10, 2017).

[4] aheui.aheui. `https://github.com/aheui/aheui.aheui`, 2015 (accessed April 10, 2017).

[5] AheuiChem. `http://yoo2001818.github.io/AheuiChem`, 2015 (accessed April 10, 2017).

[6] Anthony A Aaby. Compiler construction using flex and bison. 2003.

[7] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, page 75. IEEE Computer Society, 2004.

[8] Michael E Lesk and Eric Schmidt. Lex: A lexical analyzer generator, 1975.