Implementing a VSOP Compiler

Roland GREFFE, Florent DE GEETER, Pascal FONTAINE

Part IV

Code Generation

1 Introduction

In this assignment, you will finish your compiler by implementing code generation, according to the semantic rules described in the VSOP manual. You will reuse the lexer, parser and semantic analyzer developed in the previous assignments (you can of course improve them if needed).

You can also implement your own extensions to the VSOP language as described in section 2. This is **optional** and should only be attempted if you already have a working compiler for basic VSOP.

Your program will generate either a native executable, or the textual LLVM code on standard output, depending on arguments passed to your compiler, as described in section 3.

You will also provide a report covering the entire development of your compiler throughout the course with this final assignment. What is expected in your report is detailed in section 4.

Finally, the way you should submit your work for evaluation is described in section 5.

This assignment is due at the latest for the **14th of May** (23:59 CET).

2 Extensions

If you want to experiment a bit with programming language design and/or play with features not present in the basic VSOP language, you can implement your own language extensions. These can range from simple lexical/syntactic extensions (*e.g.* adding a *do-while* construct) to more profound semantic changes (*e.g.* adding multiple inheritance). A few suggestions are present in the manual, but you are free to design your own language extensions.

Your language extension(s) need not be backward compatible with basic VSOP. You can change the language completely. However, keep the short remaining time in mind when designing your extension(s), so that you can realistically implement it (them) before the deadline (if in doubt, ask the teaching assistant whether implementing your extension is realistic or not).

Your compiler should only compile basic VSOP language by default. Extensions should be enabled **only** when adding a `-e` argument before the extended VSOP input file(s).

Of course, any extension you implement should be documented in your report. You should also provide some example input file(s) demonstrating the use of your extension(s).

Note that this part is **optional**, and should only be attempted **if your basic VSOP compiler works!**

3 Output Format

3.1 Native Executable

When neither `-l`, `-p`, `-c` nor `-i` is specified in the arguments, your compiler will output a native executable with the same name as the input file, minus the `.vsop` extension.

3.2 LLVM IR

If called with `-i` as an argument, your compiler will instead output the textual LLVM IR corresponding to the input VSOP source file (or extended VSOP source file) on the standard output.

4 Report

Your report should give an broad overview of your compiler implementation. Which tools have you used? How is it organized? Which module is responsible for what task? Which data structures and algorithms are used? Why have you made the implementation choices you made?

Discuss the potential shift/reduce or reduce/reduce conflicts you had to solve.

If something in your implementation is not obvious from your **documented** source code, also explain it in your report.

If you implemented some VSOP language extensions, describe them in your report.

Discuss the limitations of your current compiler. What would you have done differently, retrospectively? What you would do to improve it, if you had more time?

Finally, we would also appreciate an estimation of the time you passed on the assignment (per student), and what you would suggest to improve the project or the course in general.

Try to be succinct. Your report should not be long, we do not expect a exhaustive report.

5 How to Submit your Work?

As for previous phases, your work should be submitted through the submission platform.

Your folder must also contain your report in PDF format in a `report.pdf` file.

Finally, provide your test VSOP input files in a `vsopcompiler/tests` sub-folder. Any file with `.vsop` extension in that folder will be tested against the reference implementation by the testing script. As the executable generated by your compiler and the one generated by the reference implementation will differ, the comparison will be done by running both executables without any argument. Print out some meaningful results in order to be able to compare the two generated applications. If a file where the `.vsop` extension has been replaced by `.input` is present, it will be used as the standard input of your program.

If you provided some extended VSOP source files, give them another extension (*e.g.* `.vsopext`) so that the testing script does not attempt to compile them with the reference implementation.

Your code will be executed as follows. Your built `vsopc` executable will be called either with the argument `<SOURCE-FILE>`, or with the arguments `-i <SOURCE-FILE>` where `<SOURCE-FILES>` is the path to the input VSOP source code. Your program should then create the native executable or dump the LLVM IR on standard output as described in section 3. If an error is detected, print one or more error messages on standard error instead. If an executable was generated, it will then be executed without argument, possibly with some input on `stdin`, as described above.

Arguments		Action
	`<SOURCE-FILE>`	Generate a native executable.
`-l`	`<SOURCE-FILE>`	Dump tokens on `stdout` and stop.
`-p`	`<SOURCE-FILE>`	Dump *parsed* AST on `stdout` and stop.
`-c`	`<SOURCE-FILE>`	Dump *annotated* AST on `stdout` and stop.
`-i`	`<SOURCE-FILE>`	Dump LLVM IR on `stdout` and stop.
`-e`	`<SOURCE-FILE>`	Treat input source file as an *extended* VSOP file. Generate a native executable.
`-e -i`	`<SOURCE-FILE>`	Treat input source file as an *extended* VSOP file. Dump LLVM IR on stdout and stop.

`-e -l`
`-e -p`
`-e -c`
	`<SOURCE-FILE>`
	`<SOURCE-FILE>`
	`<SOURCE-FILE>`
		Treat input source file as an *extended* VSOP file. Dump tokens or (annotated) AST and stop. All 3 are **optional**, but useful for debugging.

Table 1: Possible `vsopc` arguments.

If you implemented one or more language extensions, your `vsopc` executable can be called with the argument `-e`. We should be able to combine both `-e` and `-i` flags in order to have the LLVM IR corresponding to an extended VSOP input file dumped on standard output.

A summary of possible compiler arguments is given in table 1.

The submission platform will check that your submission is in the right format, and test your compiler. If you want to be able to use that feedback, don't wait until the last minute to submit your compiler (you can submit many times until the deadline, only the last submission will be taken into account).

Also note that **5% of your final grade** will be determined directly by the automated tests for this assignment. Is is thus doubly in your interest to ensure that your code passes all the tests (and early enough).

Plagiarism. Cooperation (even between groups) is allowed, but all cooperation must be clearly referred to in the final report, and we will have **no tolerance at all for plagiarism** (writing together, reusing/sharing code or text). Code available on the web that serves as a source of inspiration must also be properly referred in the report.

Participation. **Each student has to participate** in the project, and do her/his fair share of work. Letting others in the group do all work will be considered as plagiarism.