Implementing a VSOP Compiler

Cyril SOLDANI, Pr. Pierre GEURTS

April 13, 2016

Part IV

Code Generation

1 Introduction

In this assignment, you will finish your compiler by implementing code generation for VSOP programs, 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).

If you want to, you can implement your own extensions to the VSOP language as described in section 2. This is **optional**.

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 Friday the **13th of May** 2016 (23:59 CEST).

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 by default compile only the basic VSOP language. If you implement extensions, these should be enabled **only** when adding a `-ext` 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 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 `-lex`, `-parse`, `-check` nor `-llvm` 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 `-llvm` 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. 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?

If something in your implementation is not obvious from your **commented** 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 the time?

Finally, we would also appreciate an estimation of the time you passed on the assignment, and what you would do to improve the VSOP language or the course in general.

5 How to Submit your Work?

Your work should be submitted through the submission platform.

You should submit an archive named `vsopcompiler.tar.xz`. That archive should contain a `vsopcompiler` folder with your code, and a `Makefile` to build it. A skeleton `vsopcompiler` folder and `Makefile` is provided on the assignments web page. You can actually build your compiler with whatever tool you want (if you need building at all), but still provide a `Makefile` with the targets described here.

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 generated LLVM code and native executables will differ, the comparison will be done by running the generated executables without any argument. Print out some meaningful results in order to be able to compare the two generated applications.

Arguments		Action
	<SOURCE-FILE>	Generate a native executable.
-lex	<SOURCE-FILE>	Dump tokens on stdout and stop.
-parse	<SOURCE-FILE>	Dump *parsed* AST on stdout and stop.
-check	<SOURCE-FILE>	Dump *annotated* AST on stdout and stop.
-llvm	<SOURCE-FILE>	Dump LLVM IR on stdout and stop.
-ext	<SOURCE-FILE>	Treat input source file as an *extended* VSOP file. Generate a native executable.
-ext -llvm	<SOURCE-FILE>	Treat input source file as an *extended* VSOP file. Dump LLVM IR on stdout and stop.

Table 1: Possible vsopc arguments.

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 built as follows. From inside your vsopcompiler folder, make install-tools will be run. This should install any software needed to build your compiler (make and llvm-dev) are already installed. You can use sudo if you need administrator privileges. Then, make vsopc will be issued to build your compiler, which should, obviously, be named vsopc. Appending the letter c (for **C**ompiler) to the name of the language is a common convention for compilers.

Your code will be executed as follows. Your built vsopc executable will be called either with the argument <SOURCE-FILE>, or with the arguments -llvm <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 you implemented one or more language extensions, your vsopc executable will be called with the argument -ext. We should be able to combine both -ext and -llvm 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 reference platform is an amd64 machine, powered by Debian Jessie (a GNU\Linux distribution). You can use the virtual machine provided on the assignments web page. Alternatively, you can use debian/jessie64 with vagrant or debian:jessie with docker, or your own Debian-based machine.

We will try to add test scripts to the submission platform that 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 as many times as you want until the deadline, only the last submission will be taken into account).