Compiling Techniques (CT) Change to 20 credits course

1. Introduction

This document proposes to change the Compiling Techniques (CT) course from 10 to 20 credits. This follows the recent recommendations of the Strategy Committee [Workload and Assessment in Taught Courses, Ian Stark, 2015-10-27] to use more 20-credit courses.

Moving the course to 20 credits would reflect the amount of time the student are actually spending on the coursework. In addition, this would be the opportunity to extend the coursework a little and bring in the experience of Dr. Aaron Smith, a compiler expert from Microsoft Research currently with the status of visiting professor in our department.

2. Current coursework

2.1 Description

The current coursework consists in writing a full compiler for a simple subset of C: small-C.

The coursework is currently subdivided in four parts which are evaluated independently. The implementation language is Java. First, the students have to write a parser manually which distinguish correct programs from incorrect ones. Then they built the AST (Abstract Syntax Tree). In the next phase, the students implement a semantic analyser (name analysis and type checking). Finally, they write a backend that generates bytecode instructions for the Java Virtual Machine (JVM). For this last task we rely on the Java ASM library.

At the end of this coursework the students have developed a full real compiler that can take small-C program as an input, perform semantic anaysis and generate the corresponding Java bytecode program that can be executed within the JVM.

2.2 Assessment

The students are currently assessed based on whether they managed to implement a compiler that produces correct code for some input test programs. The mark is a function of how many test cases are working. The assessment is fully automated with a regression test suite running twice a day giving them instantaneous feedback. In addition, we ask each student to attend a mandatory demo where we ask them questions about their code to ensure they really did the work themselves. Finally, the MOSS (Measure Of Software Similarity) from Standford is in use to detect possible case of cheating.

The assessment system will remain more or less unchanged for next year course.

3. Rational

The rational for moving the course to 20 credits is two-fold. First, the students already spend a significant amount of time developing the compiler. A survey I have conducted (21 responses) show that the median number of hours spent for the coursework alone is 70 hours which is a bit on the high-end for a 10-credit course (considering the last deadline is in week 11).

Secondly, we are delighted to host Dr. Aaron Smith in ICSA, a Royal Academy of Engineering Visiting Professor, who is a compiler expert from Microsoft Research. The intend of his visit is to actually teach some components of the CT course and add some industrial perspective to the coursework. The plan is to extend the coursework and add a 5^{th} part which will deal with implementing a compiler pass in a real compiler infrastructure (LLVM). This would give our students an invaluable experience of using an OpenSource industry-standard compilers written in C++. The survey I conducted showed that students favour Java and C++ equally as an implementation language for the compiler and I think offering them the opportunity to apply their C++ programming skills in this last part would highly reinforce their knowledge of C++ which is highly desirable.

4. Proposed alteration

4.1 Coursework

The first three parts of the coursework will remain unchanged.

The fourth part will now target a real assembly language (MIPS or ARM). This will tie in better with the inf2c and the computer architecture courses. In addition, this will expose the student to a set of important topics such as register allocation which are currently not covered by the coursework.

Finally, a new fifth part, which will consists of coding a compiler pass in LLVM, as discussed earlier, will be introduced. The design of this last part will be done jointly by Dr. Aaron Smith and myself. The deadline for this last assignment would be set in the 14th week, as permitted by the recent document on Workload and Assessment in taught Courses [Ian Stark, 2015-10-27] since the course is coursework only and taught during semester 1. Note that this same document permits a 20-credit coursework-only course to have up to 5 assignments.

1) [25 hours] Parser (including writing test programs + removing ambiguity from grammar)
2) [10 hours] AST builder
3) [10 hours] Semantic analysis (type checking, variable use/def, symbol table construction)
4) [25 hours] Code generation (to ARM or MIPS)
5) [30 hours] Coding a pass in a real compiler infrastructure (LLVM)

In addition to these hours spent on implementation, I expect the students to have to read documentation, for instance in the case of the LLVM compiler infrastructure. There is also the time spent asking and reading answers on the Piazza online discussion forum which is the main source of support for this course. I expect this would amount to about 30 additional hours.

Total : 130 hours

4.2 Lectures

The main problem I encounter with the current coursework is that for the students to be able to finish their coursework, they need to have had all the lectures covering the topics necessary for the coursework. I found that I sometimes had to race through the lectures in order to present all the material before the actual coursework deadline. To mitigate this issue, I would suggest making one of the lecture a “double lecture” as permitted by the University policy [Shared Academic Timetabling, policy and guidance, 3.5].

Having 3 hours of weekly lecture would allow me to present all the necessary material for the coursework well in advance of the actual deadline. It is not my intend to use all the lecture slots and I envision that some of the lectures would not use the double slot available at times (or they might simply be question/answers sessions).

Total : 33 hours (probably a bit less)

4.3 Tutorial / Lab

Finally, the last change I propose is to change the tutorial session to a lab session. In actual fact the current so-called “tutorial” is here to help the student with their coursework; we introduce the coursework, discuss some issues and simply answers questions during these sessions. Therefore, I think these sessions are closer from being labs than tutorials.

Total : 11 hours

6. Course descriptor updates

Course Start

Semester 1

Learning and Teaching activities

Total Hours: 178 (Lecture Hours 33, Seminar/Tutorial Hours 11, Programme Level Learning and Teaching Hours 4, Directed Learning and Independent Learning Hours 130)

Assessment Weightings

Written Exam 0%, Coursework 100%, Practical Exam 0%

Additional Information (Assessment)

- Five practical compiler exercises (Parser, AST, Semantic analysis, Code generation, LLVM pass).

- You should expect to spend approximately 130 hours on the coursework for this course.

Learning outcomes

6.1 - Ability to analyse compilation tasks and to apply standard compilation techniques.

The analysis and application of standard compilation techniques is actually at the heart of the coursework, so this learning outcome is fulfil.

6.2 - Ability to develop, implement and apply modifications to standard compilation techniques and algorithms wherever this is necessary.

The coursework will rely heavily on the students implementing standard compilation techniques and algorithms, so this is fulfilled by the coursework. For instance, they will have to implement recursive descent parsing or emit instructions using tree traversal algorithms. In addition, they will have to adapt these techniques and algorithms to deal with the specifics of the input language and target code. For instance they might have to make the grammar definition unambiguous so that they can effectively implement the parser.

6.3 - Ability to understand and implement design decisions in modern compilers.

The student will be given the main structure of the code to implement their compiler. However, they will have to make design decisions when actually implementing each compiler phase. As such they will acquire a practical knowledge of the impact of design decisions. In addition, the solution to each phase will be presented after submission, allowing them to reflect on their own design choices. The use of LLVM in the last part of the coursework will also expose students to the design of modern compiler infrastructure.