A Compilation Process Visualisation Ecosystem for Efficient Teaching and Learning

Project Proposal

ID: 1466610, October 2017

Introduction

Compiler construction is a well-researched field, and every computer science student should have at least a good overview on the topic. The traditional method in teaching compiler construction is that of following a textbook and for each section incrementally build the compiler. However many aspects of compilation, such as lexing, parsing, abstract machines, automata and many more may need an additional method of learning which bridges the gap between theoretical knowledge and practical implementation.

This project consists of an extensible platform for teaching and learning compiler construction using a subset of Java. The main feature of the platform is for a program to be visualized at various stages of the compilation process. The visualisation techniques will be analysed and developed according to input from early prototypes.

Project Statement

This project will tackle the problems in teaching compilers/compiler construction in University courses. It will firstly identify the the main topics taught in compiler design then focus on the most common topics student find hard to grasp. The main motivation behind this project is the lack of effective visualisation tools when it comes to algorithms related to the compilation process. While the visualisation of traditional algorithms such as sorting, graph searching etc...has been widely used in computer science classes, concepts such as LL, LR, lexical analysis and semantic analysis visualisations (in the context of education) have not been thoroughly investigated.

Therefore the project is an ecosystem of tools that will enable students to learn about the compilation process (at least the frontend) through visualisation and live interaction and the teachers to customise, upload content and prepare assessments.

The hypothesis of the research study is that compiler construction modules, where students learn how to write a compiler and how to generate one using various tools, approach the problem in a traditional setting and a visualisation approach can improve the understanding of generally difficult to grasp concepts. This means that there isn't a bridge between the high level concepts that the students learn from textbooks and the hands-on-practice course. The project will therefore fill the gap by providing the conceptual understanding of

compiler construction through the visualisation and interaction with the compilation process (among other learning styles, described in the 'Methods' section below).

The challenge in this project is to effectively visualize algorithms while maintaining their complexity.

Methods

Research study

In order to test the research study hypothesis the pre-development phase is defined as follows:

- Research the content of multiple compiler courses (American and European Universities compiler construction modules syllabi)
- Design new ways of visualising and interacting with compiler related algorithms and data structures (evaluation done by prototyping)
- Build a targetted questionnaire for UoB CS students that have taken the compiler construction course
- Build a prototype of the overall system and a post-questionnaire to assess the features of the system

Development

The project's ecosystem (CC Tutor) will include (this may vary throughout the development of the system):

- A sized-down compiler for educational purposes called MiniJava
- An API for hooking into the compilation process so that developers can implement their custom visualisations (extension point)
- A student portal with the following features:
  - General information about the compilation process
  - Editor to input code
  - Visualisation of the compilation process with the possibility to choose which layer to perform and which to skip with points of interaction
  - In-depth visualisation for each compiling phase
  - Real-time interaction with the visual components (e.g. how would changing the tokens affect the parsing phase)
  - Exercises that require the student to fill in some stubs in the compilation process (e.g. complete NFA to DFA conversion for lexical analysis)

  - Full compiler (at least frontend) availability to tweak (advanced)

- A lecturer portal with the following features:
  - Upload exercises on specific compilation phases (i.e. defining the stubs)
  - Ability to set custom visualisation (through the above mentioned API)
  - Ability to create and save sessions for the future (e.g. save a successful and an unsuccessful LL parsing, for comparison purposes)
  - Add modules (e.g. advanced sections of the base compiler - if time allows)

The `cctutor` system will be developed in Java including 6 main/top level packages (prefixed by co):

- `cctutor.minijava`: the sized-down compiler for educational purposes
- `cctutor.hooks`: the event driven interface between the compiler and the visualisation framework
- `cctutor.viz.def`: the default visualisation package for the compilation process
- `cctutor.base`: the base portal functionality for both the student and teacher user spaces
- `cctutor.student`: the student portal (uses JavaFX as the graphics library)
- `cctutor.teacher`: the teacher portal (uses JavaFX as the graphics library)

The compilation process is the one defined in Appel, A. and Palsberg, J. (2009).

The visualisation and interactions of the various algorithms will be defined in the prototype. Example of visualisations are real-time NFA to DFA algorithm runs, generation of tokens through a DFA, First and Follow sets construction, parsing table construction etc...
N.B.: The project will at least include the frontend.

Project schedule

Term 1

The output of the first term will be:

- Scientific article
  - Prototype of the system
  - Specification of the system
  - Literature review
  - Data collected from compiler construction modules
- Minijava compiler frontend

Week 3

- Complete paper and articles readings
- Start literature review
- Start sketch of the system
- Consider user requirements

Week 4

- Determine a suitable journal, or conference, format for your article with supervisor
- Background reading
- Visualisations study
- Start literature review
- Collect data about compiler construction modules
- Finish lexer with interface to the hooking package
- Consider user requirements with collected data
- Start working on the overall specification of the software

Week 5

- Background reading
- Analysis of collected data and prepare visualisations
- Visualisations study
- Work on literature review
- Work on prototype
- Work on parser
- Work on specification

Week 6

- Work on prototype (in order to define the visualisations)
- Visualisations study
- Work on parser
- Finish literature review
- Produce overall specification of the software
- Start article writing

Week 7

- Finish prototype of the overall software
- Finish parser with interface to the hooking package
- Work on article composition

Week 8

- Work on semantic analysis
- Specification refinements
- Finish article with all gathered material

Week 9

- Work on semantic analysis

Week 10

- Finish semantic analysis (this ends the frontend of the compiler)

Term 2

The following is a tentative schedule for the second term. Workload will depend on the progress done in the first term. This is an approximate schedule since exact dates have not been released as of writing time.

N.B.: some details have been left out because they're one of the outputs of the scientific article (such as which actual visualisations to show).

Week 1

- Work on lexical analysis visualisation and interaction
- Work on student and teacher platform common features

Week 2

- Work on lexical analysis visualisation and interaction
- Work on student and teacher platform common features

Week 3

- Work on lexical analysis visualisation and interaction
- Work on student and teacher platform common features

Week 4

- Finish lexical analysis visualisation and interaction
- Work on paring visualisation and interaction
- Work on student and teacher platform common features
- Work on individual platform features

Week 5

- Work on paring visualisation and interaction
- Work on semantic analysis visualisation and interaction
- Work on individual platform features
- Finish student and teacher platform common features
- Gather material for report writing

Week 6

- Work on paring visualisation and interaction
- Work on semantic analysis visualisation and interaction
- Work on individual platform features
- Gather material for report writing

Week 7

- Work on paring visualisation and interaction
- Work on semantic analysis visualisation and interaction
- Work on individual platform features
- Report writing

Week 8

- Finish paring visualisation and interaction
- Finish semantic analysis visualisation and interaction
- Finish individual platform features
- Final tweaks and bug fixes to the codebase
- Report writing

Week 9

- Final tweaks and bug fixes to the codebase
- Report writing

Week 10

- Final tweaks and bug fixes to the codebase
- Report writing

Artifacts

The software codebase will be hosted on the School's GitLab server. The documentation will be included in the `docs` directory.

The documentation will include:

- User requirements document
- Document with external links to prototypes, scenarios etc...
- Codebase documentation (classes, methods etc...)
- Software architecture documentation
- Teacher user guide
- Student user guide
- Developer user guide (to extend the visualisation package)

References

[1] Aho, A., Lam, M., Sethi, R. and Ullman, J. (2014). Compilers. Edinburg Gate (Harlow, Essex): Pearson Education Ltd.

[2] Appel, A. and Palsberg, J. (2009). Modern compiler implementation in Java. Cambridge: Univ. Press.

[3] de Oliveira Guimarães, J. (2007). Learning compiler construction by examples. ACM SIGCSE Bulletin, 39(4), p.70.

[4] Fouh, E., Akbar, M. and Shaffer, C. (2012). The Role of Visualization in Computer Science Education. Computers in the Schools, 29(1-2), pp.95-117.

[5] Hundhausen, C., Douglas, S. and Stasko, J. (2002). A Meta-Study of Algorithm Visualization Effectiveness. Journal of Visual Languages & Computing, 13(3), pp.259-290.

[6] Mernik, M. and Zumer, V. (2003). An educational tool for teaching compiler construction. IEEE Transactions on Education, 46(1), pp.61-68.

[7] Mogensen, T. (2010). Basics of compiler design. Copenhagen: University of Copenhagen Department of Computer Science.

[8] Scott Grissom, Myles F. McNally, and Tom Naps. 2003. Algorithm visualization in CS education: comparing levels of student engagement. In Proceedings of the 2003 ACM symposium on Software visualization (SoftVis '03). ACM, New York, NY, USA, 87-94. DOI=http://dx.doi.org/10.1145/774833.774846

[9] Waite, W. (2013). Compiler construction. Springer-Verlag.