Assignment on Ownership and Capabilities for G-INF00 58043 LE13 "Perspective in Informatics 4"

By Alex Potanin (alex@ecs.vuw.ac.nz)

Submission Deadline: 10:30am Friday 31st of January 2020 (via email to Alex)

Introduction

The goal of this assignment is to write a reflective essay comparing two different approaches to ensuring encapsulation and control of access to sensitive parts of programs in modern programming languages. The first approach is known as *ownership types* and is supported by languages such as Rust as well as by prototypes used in research with a number of variations based on ownership annotations or type modifiers. The second approach is known as *object capabilities* and is supported by research languages such as E or Pony as well as some of the more practical implementations such as Caja. Object capabilities can be checked and enforced at runtime or there is research on static checking of capabilities as exemplified by Wyvern programming language discussed in class.

Core (worth 50%)

You will need to choose between an *implementation-based* or *formalisation-based* approach to this assignment. Both will receive equal treatment and would allow you to explore either your programming or mathematical side. To get the *core* of this assignment finished you will need to do one of the following two things:

Implementation-Based Approach

- Learn the basics of the Rust (http://www.rust-lang.org/) programming language using online materials such as tutorials etc.
- Implement a variety of simple examples in the Rust showing off its support for *ownership* (and *immutability*) to demonstrate the advantage it gives for managing access to different components in your program.
- Learn the basics of the E research programming language (http://erights.org/elang/) or Pony research programming languages (https://www.ponylang.io/) – ensure that you understand their support for *object capabilities* (and *reference capabilities* in case of Pony).
- Implement a variety of simple examples in E (or Pony) showing off their support for *object capabilities* to demonstrate the advantage it gives for managing access to different components in your program.
- Write up a summary of what examples you managed to implement in each of the two approaches and why they were good ways to show off the power of either ownership and immutability or object and reference capabilities. Try to do your own research to find other languages and good examples and feel free to discuss your ideas with Alex.
- *Expected size of the summary is around 2000 words. Additionally, I expect the submission of working code examples with instructions of how to compile and run them.*

Formalisation-Based Approach

- Understand the foundations of Featherweight Generic Ownership and Immutability (https://dl.acm.org/citation.cfm?id=1869509 as well as the Technical Report here: https://dspace.mit.edu/handle/1721.1/36850).
- Understand the foundations of both Object and Reference Capabilities (e.g. https://dl.acm.org/citation.cfm?doid=3152284.3133896 or https://dl.acm.org/citation.cfm?id=2398857.2384619 both can be a good start, while these two theses can point at further ideas: http://erights.org/talks/thesis/index.html and https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/GeorgeSteed.pdf).
- Write up a summary of the main features and challenges behind each of the formalisations of the ownership types, immutability, object and reference capabilities, and (if applicable) various type modifiers. Remember to explore further by using Google or asking Alex for more pointers.
- *Expected size of the summary is around 2000 words. Additionally, I expect the submission of the relevant formal details such as syntax and core typing rules for each system as an appendix that can copy the figures (with citations) from the relevant papers or technical reports.*

Completion (worth 30%)

- Write a reflective essay comparing the general approaches of *ownership* and *capabilities* with a clear conclusion of their strengths and weaknesses and *your personal reflection with justification* of which one you think is more promising for the *future programming languages* being designed and developed as we speak. Your essay needs to contain objective justifications with citations and make it clear where you express your subjective options.
- *Expected size of the summary is around 2000 words.*

Challenge (worth 20%)

- Propose your own variation of either a combination of ownership or immutability or type modifiers or object capabilities or refence capabilities and design a simple language model whose syntax supports it.
- Write a description of the semantics of your proposed language combination and what benefits you expect it to bring. You do not need to have a sound formalism or guarantee that your approach would work in practice, but you need to justify your defence of selecting particular features and how they would provide benefit to programmers in terms of encapsulation or access control advantages.
- Come up with some semantic type checking rules and prototype your language extension in Redex: https://docs.racket-lang.org/redex/ .
- Submit both the write up of your variation, including the relevant syntax and type checking rules, and the implementation in Redex that I can run in Dr Racket or similar environment.
- *Expected size of the summary is around 1000 words.*

Summary

- I expect to receive a single PDF file which contains the Core (2000 words), Completion (2000 words), and Challenge (1000 words) sections and appropriate appendixes or additional figures included all in a single file. You can use either LaTeX or Word or any

other tool and you can use format of your choice – I just need to be able to open your single PDF file on my iPad to read and annotate it.

- I expect to also receive a single ZIP file containing in appropriate folders the sample implementations or Racket (Redex) files with “readme.txt” in the root folder explaining the folder structure and how to compile and run everything. I use a Mac but have Windows in a virtual machine and access to Linux box – so if you included relevant instructions, I should be able to run your code. I am also happy for you to place all of the relevant files in a GitHub or similar shared repository and then give me a link to be able to checkout all the files from one place instead of sending me a ZIP file.