LANGUAGES, COMPILERS AND INTERPRETERS: PROJECT ASSIGNMENT

Francesco Gavazzo

November 27, 2023

1 Objective

The goal of this project assignment is the development of an interpreter and a compiler for a simple *higher-order* language. We will follow Leroy's slides on the compilation of functional languages, although you will have to look up some relevant sources by yourself.

2 Structure of the Project

You are required to submit a report (in .pdf) together with accompanying code. The report must contain a detailed explanation of the interpreters and compilers implemented in the code file, as well as the formal specification of the languages considered, their semantics, the formal specification of interpreters and compilers, and statements (or at least an informal discussion) of correctness.

Remark. You are **not** required to prove correctness theorems. What matters is to follow a disciplined methodology, based on definitions, semantics, implementations, and statements/discussions (possibly without formal proofs) relating all these notions. Of course, if you want to give (some) proofs, these are more than welcome (and, if correct, they will give you bonus points); but that is not mandatory.

Ideally, you should follow the same structure used during the laboratory classes:

1. Formal definition of the *abstract syntax* of the language; definitions of legal phrases in the language and of *programs* (i.e. those legal phrases that will be executed).
2. An implementation of such a syntax.
3. Formal definition of small- and big-step semantics for the language. This can be done as we did in class, using the methodology of structural operational semantics. You should also include a definitional interpreter, and a formal link between such an interpreter and small- and big-step semantics. Such a statement is your correctness result for the interpreter of the language.
4. An implementation of the definitional interpreter (see below for technical details).
5. A formal definition of the syntax and semantics of the machine language used for compilation (you will compile to a virtual machine) and of its semantics (as in previous points, and as in class).
6. A compilation procedure from the (source) language to the machine language.
7. An implementation of such a procedure.
8. A discussion about what it means that the compilation is correct, and a formal statement of correctness of compilation (either with respect to semantics or with respect to interpretation).

All formal notions are present in Leroy's slides, so there is nothing new you have to invent. Nevertheless, if you have troubles with some notions, do not hesitate to contact me.

3 The Language

The target language for this project is an extension of the untyped λ-calculus with primitive data (for the sake of the project, feel free to consider natural numbers and Booleans, as we did in class; if you want to make the project more challenging, try to add some richer data structure, such as pairs or the like). This means that the language has variable binding, and anonymous and higher-order functions. To implement the syntax of such a calculus, you will have to learn how to deal with free and bound variables. A way to do so (the one used in Leroy's slides and notes) is to rely on de Bruijn indexes (we will say more about that in class). This is a convenient choice, but it is not mandatory. You can also use standard variables and rename them when necessary at runtime (for that you will need a fresh variable generator). You can find some tutorial explanations in the interpreter section of these notes.

Once syntax is defined and implemented, you move to semantics. Relevant points here are:

- Define a reduction strategy: Leroy does both call-by-value and call-by-name. You are required to do call-by-value, but doing also call-by-name will give you bonus points.
- Understand how to deal with (higher-order) functions. The standard approach is to use *closures* (you will learn more about them in class and, in particular, on Leroy's slides). This way, we implement lexical scope (if you are not familiar with that, have a look at the aforementioned notes) and avoid the use of *capture avoiding substituion*, a difficult-to-implement notion that is key to defining the semantics of the λ-calculus.

At this point, you can define the definitional interpreter quite smoothly.

Next, it is time for compilation. For that, you have to follow Leroy's slides carefully and implement a slight variation of the SECD machine (for call-by-name, Leroy uses Krivine's machine). The semantics of the machine is given in Leroy's slides, but you can find it in almost any textbook on programming language theory (one for all: *Programming Languages: Principles and Paradigms*, by Maurizio Gabrielli and Simone Martini).

4 About the Implementation

Concerning the implementation, you can use the language you prefer, although I would prefer a functional one, such as Haskell or OCaml (even Scheme or Racket are fine). Something in between functional and OO languages, such as Scala, may be an interesting option, too.[1] You can even use a proof assistant (Coq, Adga, Lean, etc.) and try to write correctness statements with it (bonus points if you can give some proof).

5 How to submit

Send the report and the code file to francesco.gavazzo@gmail.com (do **not** use the unipi email address, since from next week I will not be affiliated with the University of Pisa anymore).

6 Recommendations

The project is individual, meaning that each of you is expected to submit her own report and code. Nevertheless, you can cooperate (and I suggest you do so) with your colleagues. Moreover, the project is quite open-ended: if there is something you want to try, or you are curious about, contact me and I will do my best to integrate that into the project work.

Google, ChatGPT, and the library (this sounds old-fashioned, but the best sources are still there) are your friends. Solving problems means finding good literature on the subject. The reason I decided to build the project upon some slides is precisely to force you to integrate them with relevant literature, which you have to find (no worries, this can be done easily).

Finally, do not be afraid to contact me. Good luck!

[1] If you can avoid Java and Python, I will appreciate that.