CST Part IB Supervisions

Example Sheet 1

Petar Veličković

Lent Term 2016

Compiler Construction

As you will not have had many lectures yet, these exercises are meant to aid your understanding of what a compiler is and should do on a high level, so that you can be better equipped to tackle the further lectures.

1. Consider the following C++ program:

```
do
{
  x--;
  while (y <= 10)
  {
      if (y >= 5) x = 10;
      else x = 0;
  }
} while (x <= 10);
```

   a) Present a possible abstract syntax tree (AST) representation of this program.

   b) Present a possible list of assembly instructions that would be produced after compiling this code (you may use any instruction set you want as long as you clearly state its semantics).

2. Provide a simple example of a program (in C/C++ or another language with similar features) which is legal under the language's syntax, but does not have defined behaviour. Demonstrate that the program is legal by providing its AST.

3. Imagine that you have designed a new language. As you feel that your language is superior to others, you conclude that the compiler for the language should be written in the language itself. Provide a possible overview (without looking ahead in the lecture notes!) of how this could be successfully done—no need to go into high levels of detail.
   (*Hint:* The way in which you have compiled the compiler is unknown to the public!)

Practical work

I am a very strong proponent of "learning by doing", and for a course such as Compiler Construction, where the topic of discussion is a very tangible piece of software, this is by far the best way to reinforce your knowledge. One of the larger mistakes I made when I was a IB student was focusing entirely on learning the theory—implementing something is the best way to both understand the theory behind it and understand the hidden complexities the lectures might not directly address!

Since last year, Tim Griffin has started using OCaml as the language of choice for the course. It is a useful language to know (especially if you'd like to work for Jane Street at some point), and is similar enough to ML to be learnt relatively quickly. If you haven't had the chance to work with OCaml before and you'd like to learn it for the purposes of this course, I advise you to complete Exercise 1. Note that I will not be constraining you on the language of choice for solving the further exercises—it's the principles that count, not the syntax. Your exercise for this week, while not directly related to compilation, requires constructing a function involving the kind of recursion and pattern-matching required for many components of a compiler.

1. **(OPTIONAL)** Solve all the exercises at `http://try.ocamlpro.com`.
2. Implement a function that takes in a regular expression (expressed using the syntax of the *Discrete Mathematics* course, e.g. $\varepsilon|ab^*$ would be given as `Concat(Union(Null, Sym(a)), Star(Sym(b)))`, and produces a Nondeterministic Finite Automaton (NFA) that recognises it. Also implement a function that takes an NFA and a string, and returns a boolean value signifying whether the string is accepted by the NFA.
3. **(VERY OPTIONAL)** Modify your function from Exercise 2 to produce a Deterministic Finite Automaton (DFA) rather than an NFA.

Group Projects

I believe that, by now, you will have been assigned to your group projects. I would like to discuss the projects' structure with you, and perhaps provide a few useful words of advice on how to approach them. Therefore, for this supervision, prepare a short description of your project and think about which aspect(s) of it you would most likely want to be focusing on (implementation as well as organisation-wise).