Compiler Construction
Exercise Sheet 1

Alistair O'Brien

February 7, 2026

Before attempting the problems

Please complete these exercises and submit your solutions at least 48 hours before our scheduled supervisions to ajo41@cam.ac.uk as a *pdf* attachment. Feel free to choose the format of your answers (handwritten, typed, LaTeX'd)—just ensure that they're (mostly) legible.

Attempt to complete at least 75% of the set exercises—if you get stuck simply make a note of it in your answer and move onto the next question. Many of these questions are taken from the exercise sheet, which has some useful notes.

1 The Gap

1. Write down a *pipeline* of steps involved in compilation. Pay attention to the intermediate representations between each step.
   (Hint: try using the Slang compiler as a running example.)
2. What are *virtual machines* and why are they useful?
   (Hint: do not talk about AWS EC2 instances.)

2 Lexing, NFAs, DFAs

1. For each of these regular expressions r:

$$b(a \vee b)* a$$
$$((\epsilon \vee a)b*)*$$

   (a) Construct an NFA accepting the regular language $L(r)$.

   (b) Construct a corresponding DFA using the *powerset* construction.

2. Given any regular expression r, can you produce a CFG that generates the same language? Can you do this for the above regular expressions?
3. In lexing, what two rules are used to disambiguate multiple token matches out of the same character stream?

3 Context free grammars

1. Consider the grammar:

$$\begin{array}{rcl} T & \rightarrow & R \\ & | & aTc \\ R & \rightarrow & \epsilon \\ & | & RbR \end{array}$$

(a) For the string *aabbbcc*, give (i) a *leftmost* derivation (ii) a *rightmost* derivation.

(b) Is the grammar ambiguous? Justify your answer.

2. The following grammar G_1 is *ambiguous*:

$$\begin{array}{rcl} E & \rightarrow & E + E \\ & | & E * E \\ & | & (E) \\ & | & \text{id} \end{array}$$

(a) Give an *unambiguous* grammar G_2 such that $L(G_1) = L(G_2)$.

(b) Prove that the language of the two grammars coincides.

4 Top-down parsers

1. Consider the grammar:

$$\begin{array}{rcl} S & \rightarrow & \text{id} \\ & | & (S + \text{id}) \end{array}$$

(a) Write a *recursive descent* parser for this grammar in OCaml.

(b) Compute First and Follow sets for the non-terminals of this grammar.

(c) Construct the predictive LL(1) parsing table.

(d) Trace a parsing of `(((z + u) + y) + x)`.

2. 2020 - Paper 4 question 4

5 Bottom-up parsers

1. 2015 - Paper 3 question 3

2. Implement an `ocamllex` lexer and Menhir parser for the following grammar of regular expressions:

$$\begin{array}{rcl} \textit{regexp} & \rightarrow & \textit{regexp} \mid \textit{regexp} \\ & | & \textit{regexp regexp} \\ & | & \textit{regexp}* \\ & | & [\textit{character-set}] \\ & | & (\textit{regexp}) \\ \\ \textit{character-set} & \rightarrow & \textit{character character-set} \\ & | & \textit{character}-\textit{character character-set} \\ & | & \epsilon \end{array}$$

(Hint: read Real World OCaml - Chapter 19.)