Functional Programming

Compiling and Parsing

Exercise 1. (Exercise 6.1 from *"Modern Compiler Implementation in ML"* by Andrew W. Appel.) Using the `ocamlopt` compiler with parameter `-S` and other parameters turning on all possible compiler optimizations, evaluate the compiled programs by these criteria:

1. Are local variables kept in registers? Show on an example.
2. If local variable `b` is live across more than one procedure call, is it kept in a callee-save register? Explain how it would speed up the program:
   `let f a = let b = a+1 in let c = g () in let d = h c in b+c`
3. If local variable `x` is never live across a procedure call, is it properly kept in a caller-save register? Explain how doing thes would speed up the program:
   `let h y = let x = y+1 in let z = f y in f z`

Exercise 2. As above, verify whether escaping variables of a function are kept in a closure corresponding to the function, or in closures corresponding to the local, i.e. nested, functions that are returned from the function (or assigned to a mutable field).

Exercise 3. As above, verify that OCaml compiler performs *inline expansion* of small functions. Check whether the compiler can inline, or specialize (produce a local function to help inlining), recursive functions.

Exercise 4. Write a "`.mll` program" that anonymizes, or masks, text. That is, it replaces identified probable full names (of persons, companies etc.) with fresh shorthands *Mr. A*, *Ms. B*, or *Mr./Ms. C* when the gender cannot be easily determined. The same (full) name should be replaced with the same letter.

- Do only a very rough job of course, starting with recognizing two or more capitalized words in a row.

Exercise 5. In the lexer `EngLexer` we call function `abridged` from the module `EngMorph`. Inline the operation of `abridged` into the lexer by adding a new regular expression pattern for each `if` clause. Assess the speedup on the *Shakespeare* corpus and the readability and either keep the change or revert it.

Exercise 6. Make the lexer re-entrant for the second Menhir example (toy English grammar parser).

Exercise 7. Make the determiner optional in the toy English grammar.

1. * Can you come up with a factorization that would avoid having two more productions in total?

Exercise 8. Integrate into the *Phrase search* example, the *Porter Stemmer* whose source is in the `stemmer.ml` file.

Exercise 9. Revisit the search engine example from lecture 6.

1. Perform optimization of data structure, i.e. replace association lists with hash tables.
2. Optimize the algorithm: perform *query optimization*. Measure time gains for selected queries.
3. For bonus points, as time and interest permits, extend the query language with *OR* and *NOT* connectives, in addition to *AND*.
4. * Extend query optimization to the query language with *AND*, *OR* and *NOT* connectives.

Exercise 10. Write an XML parser tailored to the `shakespeare.xml` corpus provided with the phrase search example. Modify the phrase search engine to provide detailed information for each found location, e.g. which play and who speaks the phrase.