3 Compiler Construction (tgg22)

(a) Suppose we are writing a compiler for an ML-like language and we want to employ the equation

\[(\text{map } f \ l_1) \@ (\text{map } f \ l_2) = \text{map } f \ (l_1 \@ l_2)\]

as a left-to-right rewrite rule for optimisation. The symbol \(\@\) represents list append.

Discuss the merits of this idea. Is it always correct? If so, state clearly what assumptions you are making about \(\@\) and map. [5 marks]

(b) A compiler’s front-end will often expand some syntactic constructs into lower-level representations. Consider the following fragment for the abstract syntax of a SLANG-like language.

\[
\text{type var} = \text{string} \\
\text{type exp} = \\
\quad (\ast \text{abstract syntax } \ast) \quad (\ast \text{concrete syntax } \ast) \\
\quad | \text{Var of var} \quad (\ast x \ast) \\
\quad | \text{Project of int * exp} \quad (\ast \text{proj } i \ e \ast) \\
\quad | \text{Tuple of exp list} \quad (\ast (e_1, e_2, \ldots, e_n) \ast) \\
\quad | \text{Let of var } \ast \text{exp } \ast \text{exp} \quad (\ast \text{let } x = e_1 \text{ in } e_2 \ast) \\
\quad | \text{Apply of exp } \ast \text{exp} \ast \text{exp} \quad (\ast e_1 \ e_2 \ast) \\
\quad | \text{Function of var } \ast \text{arg_pattern } \ast \text{exp} \quad (\ast \text{fun } f \ p = e \ast)
\]

and \(\text{arg_pattern} = \)

\[
| \text{APvar of var} \quad (\ast x \ast) \\
| \text{APtuple of arg_pattern list} \quad (\ast (p_1, p_2, \ldots, p_n) \ast)
\]

This language has general projections for \(n\)-tuples so

\[
\text{proj } i \ (e_1, e_2, \ldots, e_k)
\]

will evaluate to \(v_i\), the value of \(e_i\). If \(i\) is not in the range between 1 and \(k\) there will be a run-time error.

In this language we can write functions with simple (possibly nested) patterns for function arguments:

\[
\text{fun } f \ (a, b, (c, (d, e))) = b \ a
\]

[continued ...]
Now suppose we want our front-end to eliminate such patterns. That is, we want to write a function of type

\[
\text{eliminate\_tuple\_patterns : exp -> exp}
\]

so that the resulting expression contains functions with patterns only of the form \text{APvar} x for some (new) variable x. For example, the code for f above should be translated to a semantically equivalent expression of the form

\[
\text{fun f x = ...}
\]

that contains only simple variable arguments (that is, only \text{APvar} patterns in the abstract syntax).

Your task is to write this function in OCaml. You can assume that you have a function for generating fresh variable strings.

\[
\text{new\_var : unit -> string}
\]

[15 marks]