Compiler project tasks — part 6, due Wednesday 28 November

The goal in this phase is to simplify the abstract syntax tree of the program to make the subsequent code generation easier.

Given input program `primes.m`, the output file should be named `primes.simplified`. This file should be generated by pretty-printing the transformed abstract syntax. Test on all programs we have used so far.

Simplification stages

Try to implement each of these stages as a separate pass over the abstract syntax tree.

Control-flow transformations (intraprocedural)

Transform loops STMTREPEAT (stmt*, expr condition) and STMTWHILE (expr condition, stmt*) into STMTLOOP (stmt*) loops.

Transform each STMTFOR (identifier index, expr from, expr to, expr? by, stmt*) loop into an appropriate block.

Transform the conditional STMTIF (expr condition, stmt* thenpart, elsifclause*, elseclause?) by imposing the constraint that the elsifclause* list is always empty and the elseclause? option is always present.

Environment transformations (intraprocedural)

Transform each procedure so that declarations appear only at the beginning of the procedure and not in blocks nested inside.

Make initialization code explicit

Construct initialization code for all variables declared in a block and prepend it to the statements of the block.

Compute the procedure call graph

Construct a graph where vertices are the procedures of the program, and an edge $a \rightarrow b$ means that procedure a calls procedure b. Produce a viewable representation of the graph in Postscript form under the name `primes.cfg.eps`. (Hint: you can use the tool *dot* to generate a nicely laid out graph from a textual specification.)

Flattenning of procedures

Transform all procedures so that there is no nesting of procedures. All procedures are at the same lexical level, and so is the body of module.