02247 Compiler Construction

Matthias Larsen, s103437 30/03/2016

Assignment 3

I got an epiphany yesterday, for which I have not yet been able to meaningful implementation. Instead, I thought I'd describe my idea on how to solve the problem.

Switching to a module pass would give me access to the full scope. I could then, for each new function call encountered, fetch similar calls (Module::getFunction() and then iterate through its uses). This would enable me to find single-line type 1 errors (Identical code fragments except for variations in whitespace, layout and comments, see http://research.cs.queensu.ca/TechReports/Reports/2007-541.pdf for the types of clones).

Finding identical multi-line blocks by serializing / encoding parts of the code, for instance using graphs as such would be able to detect duplicate code by searching for sub graphs. By encoding the code as one or more graphs, one would also be able to find similar (by instructions, not method arguments) code blocks by searching for sub graphs that could be obtained by removing parts from the graph - type 3 errors (Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments).

Using the above method to only look at similar calls (Module::getFunction()), could be extended to find type 2 errors (Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments.) by analysing the call intstruction - i.e. CallInst->getArgOperand(0..n).

Using the Module::getFunction approach to look for identical single-line, or searching for similar single-lines that might've been changed in error would greatly improve the running time for my implementation compared to my first version. This is because it no longer needs to compare every line against each other, but instead only compare identical instructions against each other.

To keep track of what's already been analyzed, a simple list of instructions that have been processed can be kept. During so, would prevent re-analyzing an instruction when encountered later in the program, similar to my first version with a “checkedCache" for processed lines.