# Interprocedural Variable Liveness Analysis
for Function Signature Recovery

MIGUEL ARAUJO AND AHMED BOUGACHA
{maraujo@cs, ahmed.bougacha@sv}.cmu.edu
Carnegie Mellon University
April 17, 2014

Project Milestone Report

Project webpage `http://cs.cmu.edu/~maraujo/15745/`

Advances

Problems, Changes, and Surprises

We didn't encounter any unexpected major problem, and thus didn't make changes to the original goals.

Achieved Goals

At the milestone, the planned schedule was to have completed the 75% goal (that is, working register liveness analysis). We went further on other fronts: signatures are now recovered, and the signature comparison benchmarking tool is functional. We chose to focus on signature recovery a bit earlier so that the benchmarking tool was complete earlier than originally planned; a side effect is that this made it possible to work in parallel on mostly separate problems; another side effect is that the register liveness analysis algorithm, although working, is still being refined. In effect, this means we achieved most of the 75% goal (and are still improving it), but also the 100% goal.

Signature Recovery

Concretely, we are now able, for a given piece of code (LLVM IR generated by the dagger decompiler), to generate alternate function signatures for each original function, augmented with parameter types and a struct return type (to emulate multiple return values in LLVM IR). The return types are still too conservative (too many returned values, some of them dead at all callsites) because of the aforementioned missing improvements to be made on the liveness analysis algorithm.

Signature Comparison Benchmarking

We also have a tool, which is able to compare function signatures: the one we recovered using our analysis, with one we wrote manually for the same function. One complication is the fact that at the C source code level, functions are named; at the decompiled IR level, functions are identified by their linked address; we translate between the first identifier to the second using the binary object file format's symbolic information (as exposed by tools such as GNU-binutils' nm). It would have been best to fully automate the signature comparison, rather than needing to provide a handwritten signature file. We investigated a way to do that: using DWARF debug information to recover source-level signatures for compiled binary functions. Due to the complexity of the DWARF standard, we chose to avoid this technique, because it would be way too involved for our goals.

Completion Logistics

Remaining Work

We respected our original schedule, with the exception of the remaining improvements to be made on the interprocedural side of the register liveness algorithm. The remaining concrete work to be done revolves around improving the signature, and benchmarking the results. For that, we will need a variety of test cases. We wrote a few to aid in development; we will continue to write or maybe import others: this is the only resource needed to complete the project.

Schedule

Week	Miguel	Ahmed
17/4 - 24/4	Benchmarking, signature transformation, and algorithm improvements	
24/4 - 1/5	Poster and final report	