Interprocedural Variable Liveness Analysis for Function Signature Recovery

MIGUEL ARAUJO AND AHMED BOUGACHA
{maraujo@cs, ahmed.bougacha@sv}.cmu.edu
Carnegie Mellon University
March 20, 2014

Project Proposal

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

Project Description

We studied variable liveness as a key problem for register allocation in procedures. Recently, interprocedural variable liveness grew in importance as the increased number of registers available led to arguments being passed in registers instead of in the stack; tracking the liveness of the arguments and return values then becomes essential for register allocation. Another scenario in which the interprocedural analysis is required is in signature recovery in decompilers, i.e. how to figure out the number and type of each function's parameters and return values?

We plan to build on an existing binary to LLVM IR decompiler, Dagger. Currently, each function takes a single parameter (a register context structure) and returns void. By determining which registers are live at the beginning of a function and which are live after each callsite of the function, we can determine which registers are used respectively as arguments and as return values; our transformation would make the analyzed code more closely match the initial function definition.

Evaluation will be done by comparing the recovered signature with the actual source-level signature: one should be able to recover from unspecified decompiled binary code, the combination of ABI calling convention and function signatures for each analyzed function.

We propose the following goals:

75% Primary objective is to complete register liveness analysis on the decompiled code.

100% Recovering and evaluating the function signatures from the register liveness analysis.

125% Further research directions: code transformations to match the recovered function signatures; robust types recovery; including arguments passed in the stack in the analysis.

Logistics

Schedule

Week	Miguel	Ahmed
20/3 - 27/3	Getting familiar with Dagger	Further research, architecture and algorithm definition
27/3 - 3/4	Basic liveness analysis framework	
3/4 - 10/4	Liveness analysis complete	
10/4 - 17/4	Intermediate report and signature recovery	
17/4 - 24/4	Benchmarking and signature transformation	
24/4 - 1/5	Poster and final report	

Milestone

We plan to have fullfilled our 75% goal, register liveness analysis must be working.

Literature Search

There has been related prior work, mostly on the UQBT and Boomerang decompilers. UQBT [1] also used interprocedural dataflow analysis, though for uses other than function signature recovery. Boomerang expands on UQBT, using SSA form, and also doing generalized function signature recovery [2]. Variants of SSA form, such as SSI, have also been used [3]: the decompiler we work on (Dagger) generates a form similar to SSI, but the need to fully maintain it during transformations wasn't encountered yet. Different approaches for function signature recovery have been proposed [4]. A study of register liveness dataflow analyzes [5] also proposes useful algorithms for our situation. Type recovery has also been explored with good results [6].

Resources and getting started

Ahmed is the original creator of Dagger so he is completely familiar with the decompiler. Progress can start immediately.

References

1. Cristina Cifuentes and K. John Gough. Decompilation of binary programs. *Softw. Pract. Exper.*, 25(7):811–829, July 1995.

2. Mike Van Emmerik. *Static Single Assignment for Decompilation*. PhD thesis, The University of Queensland, 2007.

3. Jeremy Singer. Static program analysis based on virtual register renaming. Technical report, University of Cambridge, 2006.

4. Lukas Durfina, Jakub Kroustek, Petr Zemek, and Bretislav Kabele. Detection and recovery of functions and their arguments in a retargetable decompiler. In *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, WCRE '12, pages 51–60, Washington, DC, USA, 2012. IEEE Computer Society.

5. Robert Muth. Register liveness analysis of executable code. Technical report, 1998.

6. JongHyup Lee, Thanassis Avgerinos, and David Brumley. Tie: Principled reverse engineering of types in binary programs. In *NDSS*, 2011.