Title: Synthesis of Machine Code from Semantics

Link: http://research.cs.wisc.edu/wpis/papers/tr1814.pdf

What are the problems/research questions addressed by this article?

--->
In this paper, the author presented a technique to synthesize machine-code instructions from a semantic specification, given as a Quantifier-Free Bit-Vector (QFBV) logic formula. This technique uses an instantiation of the CounterExample Guided Inductive Synthesis (CEGIS) framework,in combination with search-space pruning heuristics to synthesize instruction-sequences.

What are the existing solutions for this research question/problem?

--->
Counter-Example Guided Inductive Synthesis (CEGIS) framework,Examples of semantics-based rewriting include offline optimization, partial evaluation , and binary translation.Existing approaches either (i) work on small bit-vector languages that do not have all the features of an ISA, or (ii) super optimize instruction-sequences.

What is the research method [s] they have used?

--->

Empirichal research.

What is their proposed solution ?

--->
In this paper, the authors presented a technique to synthesize straight-line machine-code instruction-sequences from a QFBV formula. The synthesized instruction-sequence implements the input QFBV formula (i.e., is equivalent to the QFBV formula). Our technique is parameterized by the ISA of the target instruction-sequence, and is easily adaptable to work on other semantic representations, such as a Universal Assembly Language (UAL).
A machine-code synthesizer allows us to create multiple binary-rewriting tools that use the following recipe:
1. Convert instructions in the binary to QFBV formulas.
2. Use analysis results to transform QFBV formulas.
3. Use the synthesizer to produce an instruction-sequence that implements each

transformed formula.

What are three future directions from this article?

-->
One possile future work is MC-SYNTH to obfuscate/de-obfuscate instruction-sequences in malware. A second direction would be to adapt the algorithms in MCSYNTH to synthesize non-straight-line, but non- looping programs. One approach to loop-free code is to use the ite terms in the QFBV formula to create a loop-free CFG skeleton, and then synthesize an appropriate instruction-sequence for each basic block. A third direction is to create a more accurate test of legality of splits by devising a finergrained handling of Mem in SFP # USE and SFP # KILL.

Concepts that you learnt from this paper?

--->
Superoptimization.
Pruning.a