Near Complete Formal Semantics of X86-64

Abstract

ToDo

1. Introduction

2. Challenges

2.1. Using Strata Results

Following are the challenges in using Strata [1] (or Stoke) formula as is.

- Stoke uses C+-functions which define the semantics of instructions. For example, following is the function to define the semantics of add instruction. The functions are generic in the sense that they can be used to obtain obtain the concrete semantics of any instruction like `add %rax, %rbx`

```
1
2 void add(SymBitVector dest, SymBitVector a,
      SymBitVector b) {
3   set(d, a+b);
4 }
5
```

  The untested assumption here is the generic formula will behave identically for all the variants. We have tested all the formula for each instruction variant.
- Strata gives the concrete semantics for a concrete instructions. For other variants it generalize from the concrete semantics. Assumption is the generalization is correct. Test all the generalization.
- While porting to K rule, we generalize the from a concrete semantics that strata provides. Is this generalization faithful? For instruction like `xchg, xadd, cmpxchg`, the formula is different for different operands. So the general K rule we obtain from `xchgl a,b` may not represent the semantics for `xchgl a, a`. Fortunately there exists different instruction variants if the their semantics might be different and accordingly we might have different K rules. For example, `xchgl_r32_eax and xchgl_r32_r32`. But even for `xchgl_r32_32` semantics could be different for cases `r1 !=r2` and `r1 == r2`. Idea: Once lifted as K rule, test the instruction for all variants.

  Lets consider `xaddb SRC, DEST`, as per manual the semantics is

```
1 Temp = Src + Dest;
2 Src  = Dest;
3 Dest = Temp;
```

  The point to note here is that the register updates follow an order. Strata uses `xaddb %rax, %rbx`, to obtain the semantics and it happened that the ordering is maintained and hence strata can generalize the semantics of `xaddb R1, R1`. But even if the ordering is not maintained the semantics is going to be the same for the case R1 != R2, but the generalization for the `R1 == R1` case will mess up. We cannot trust the above generalization by strata. We need to test the K rule for all possible operands.

3. Implementation

3.1. Porting Strata Formulas to K Rules

Strata could ouput the internal AST, used to model a register state formula, in different formats like smtlib, or prefix notation. We have added another backend to generate K rule.

Validate the Porting The K rules generated using the backend are matched against the ones we obtained via symbolic execution on stratified instructions This had two benefits:

1. Gain confidence in porting. 2. In order to get the exact match we need to port all the verification lemma from K to strata code, which will later help in generating simplied K rules for non-stratified instructions.

References

[1] Stefan Heule, Eric Schkufza, Rahul Sharma, and Alex Aiken. Stratified synthesis: Automatically learning the x86-64 instruction set. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, pages 237–250, New York, NY, USA, 2016. ACM.