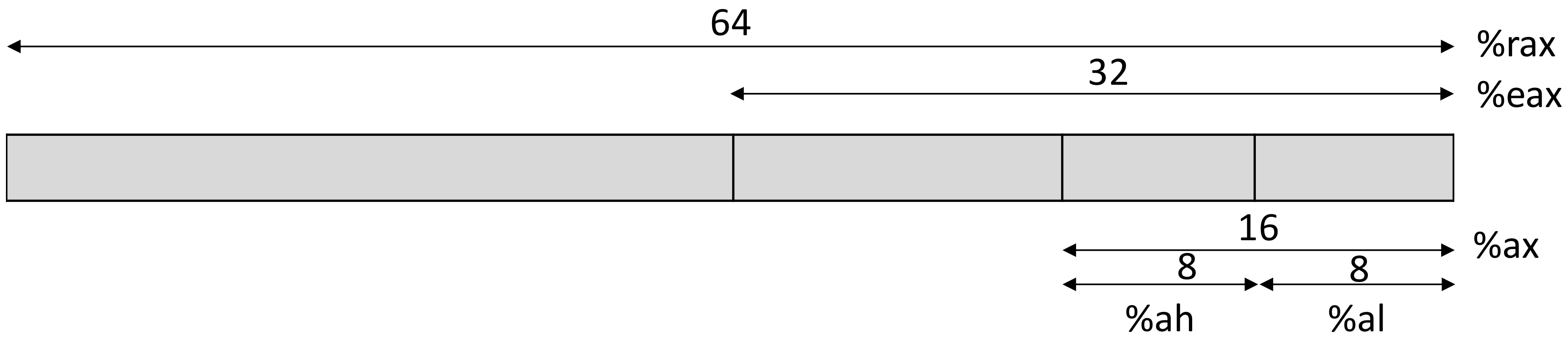
Defining x86-64 Semantics in K

Sandeep Dasgupta
University of Illinois Urbana Champaign
March 01, 2018

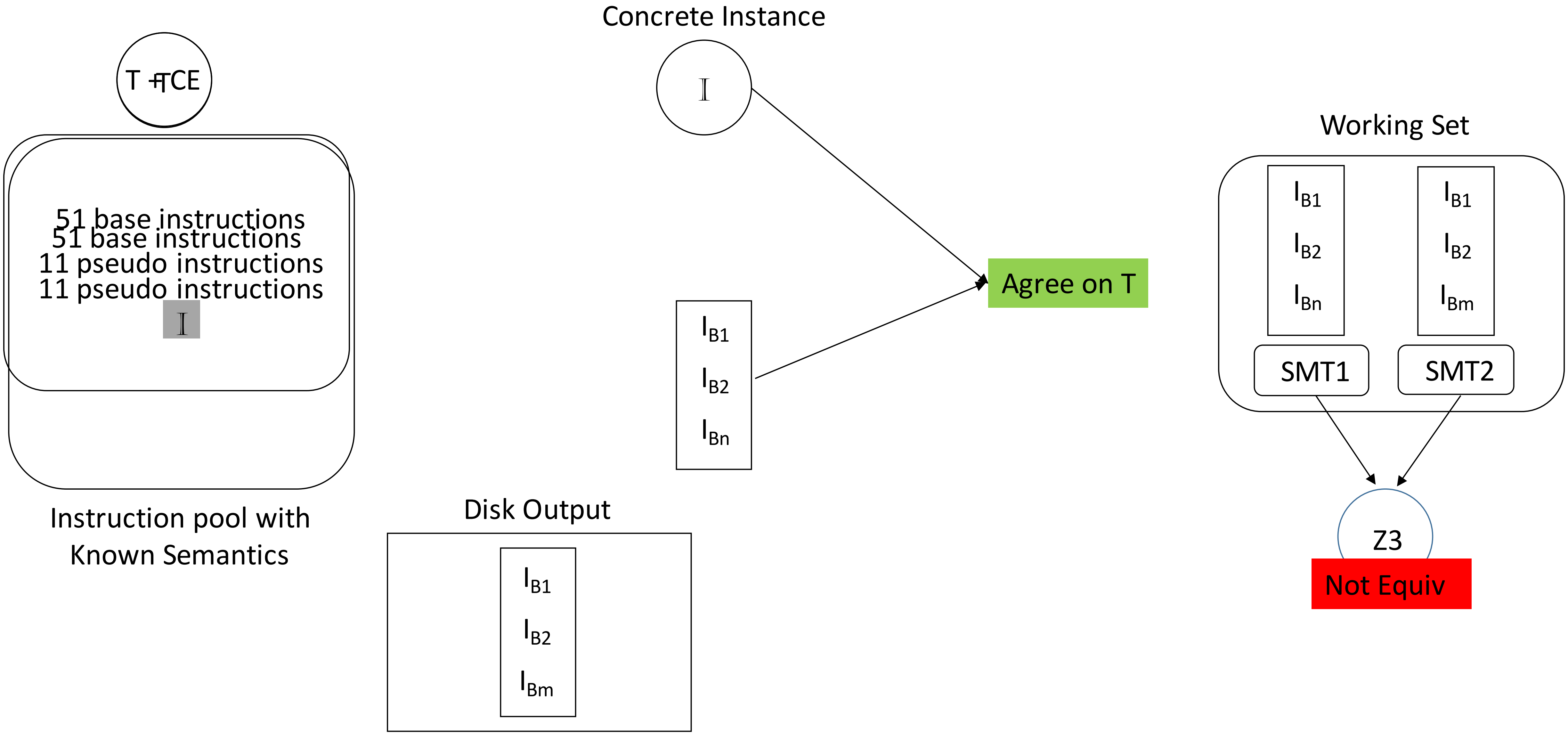
Some minor details on nomenclature



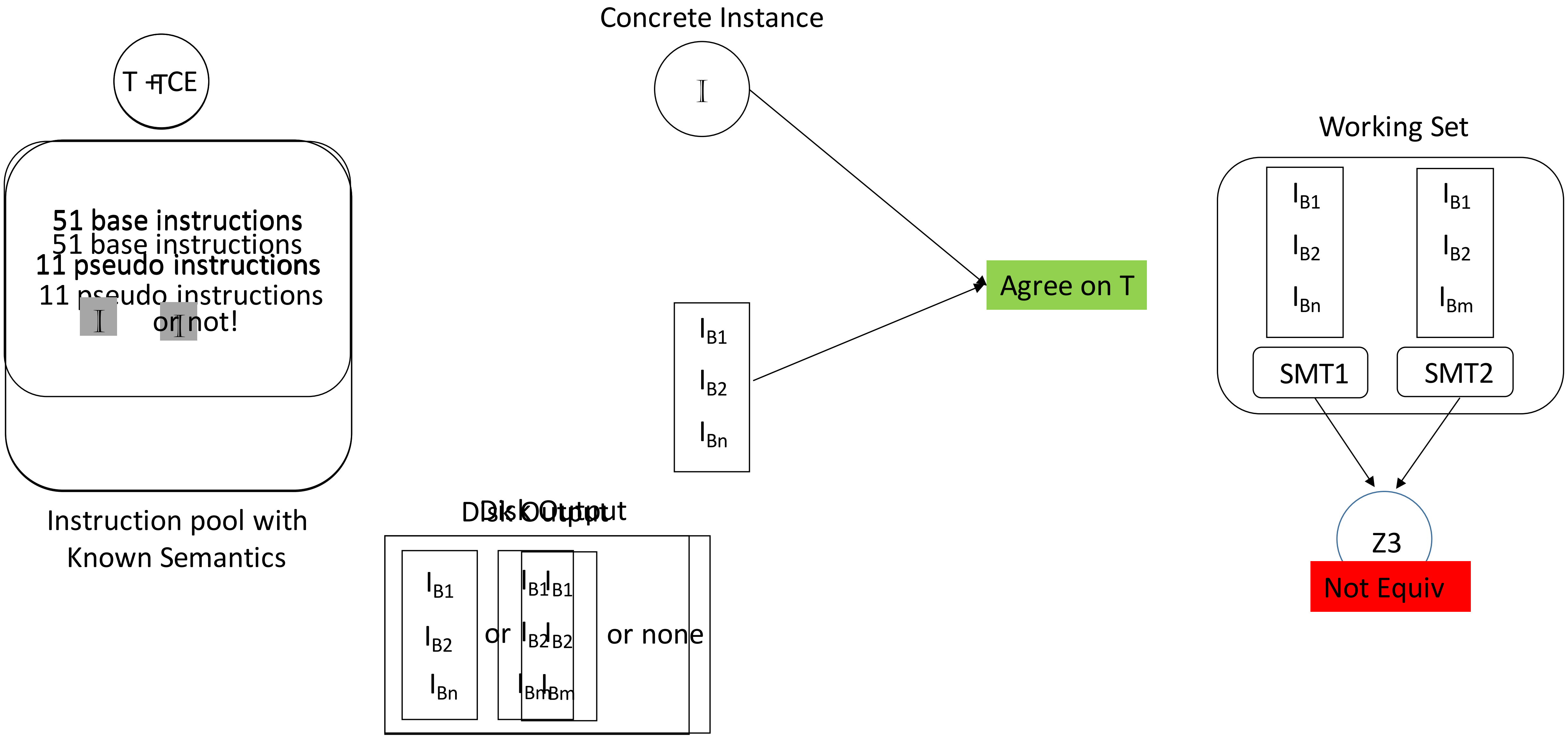
Registers	64 bit	32bit	16 bit	Upper 8	Lower 8
Concrete	%rax %rbx %rcx	%eax %ebx %ecx	%ax %bx %cx	%ah %bh %ch	%al %bl %cl
Generic names	r64	r32	r16	rh	r8

Generic instruction (CODE): incb r8
Concrete Instruction or an instance of above: incb %bl

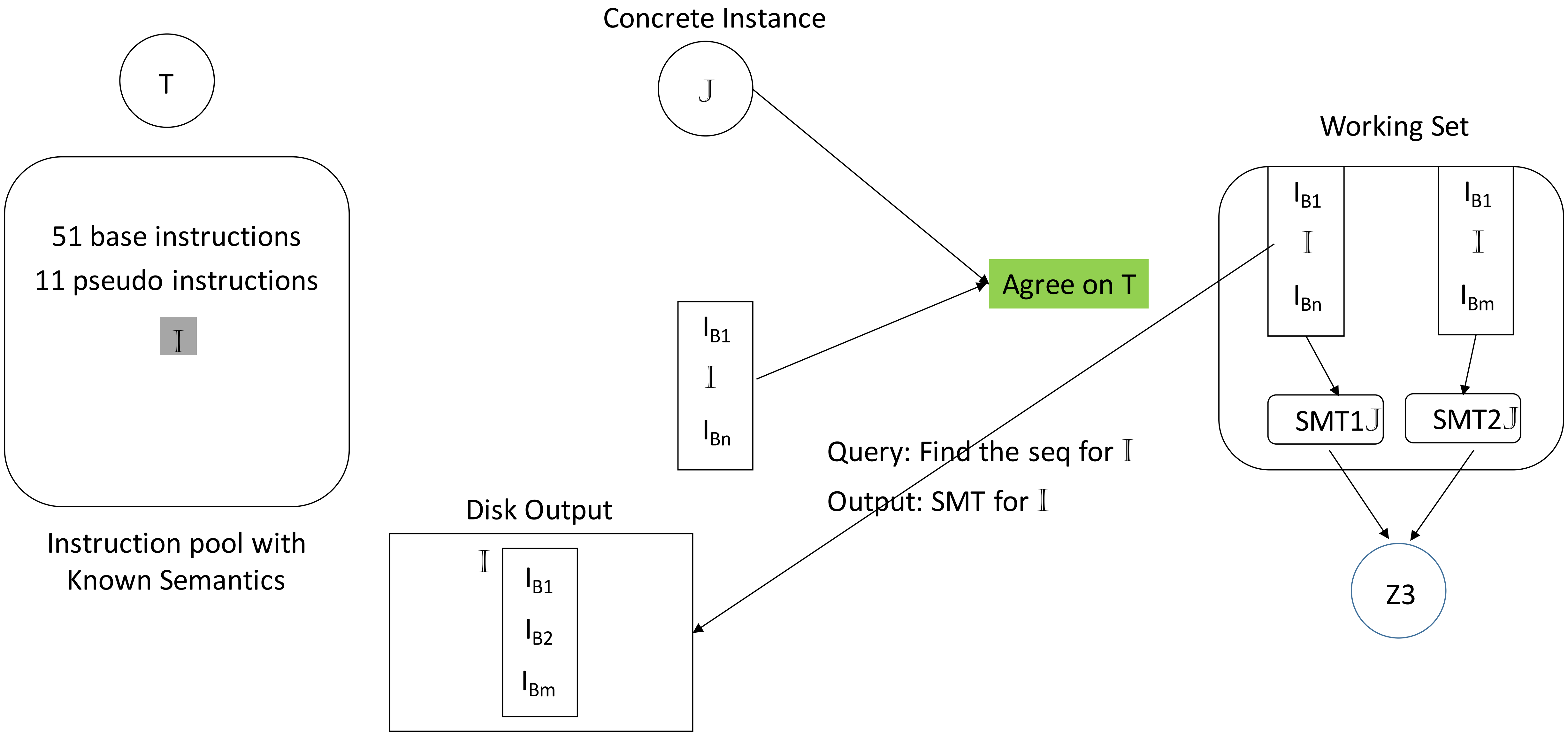
Stratified Synthesis (Stratum 0 instruction I)



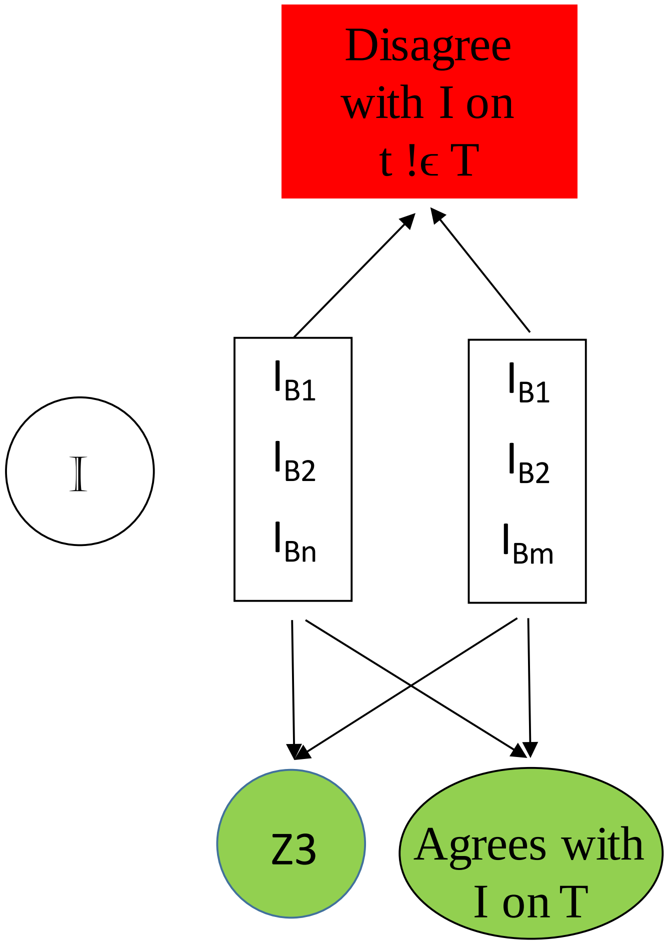
Stratified Synthesis (Stratum 0 instruction I)



Stratified Synthesis (Stratum 1 instruction J)



Strata’s Correctness guarantee



- Proving equivalence with the hand – written formulas in Stoke.
- In case of discrepancy, always the stratified formulas are proven correct either by consulting manuals or by testing on real inputs.

Output of strata

Concrete instruction(CI): **addq %rcx %rbx**

```
orq %rbx, %rbx          #  CODE=orq_r64_r64
adcq %rcx, %rbx         #  CODE=adcq_r64_r64
```

Instruction sequence (IS_{CI})

%rbx : $(0x0_1 \circ \%rcx + 0x0_1 \circ \%rbx)[63:0]$

%cf : $(0x0_1 \circ \%rcx + 0x0_1 \circ \%rbx)[64:64] = 0x1_1$
%zf : $(0x0_1 \circ \%rcx + 0x0_1 \circ \%rbx)[63:0] = 0x0_{64}$
%sf : $(0x0_1 \circ \%rcx + 0x0_1 \circ \%rbx)[63:63] = 0x1_1$

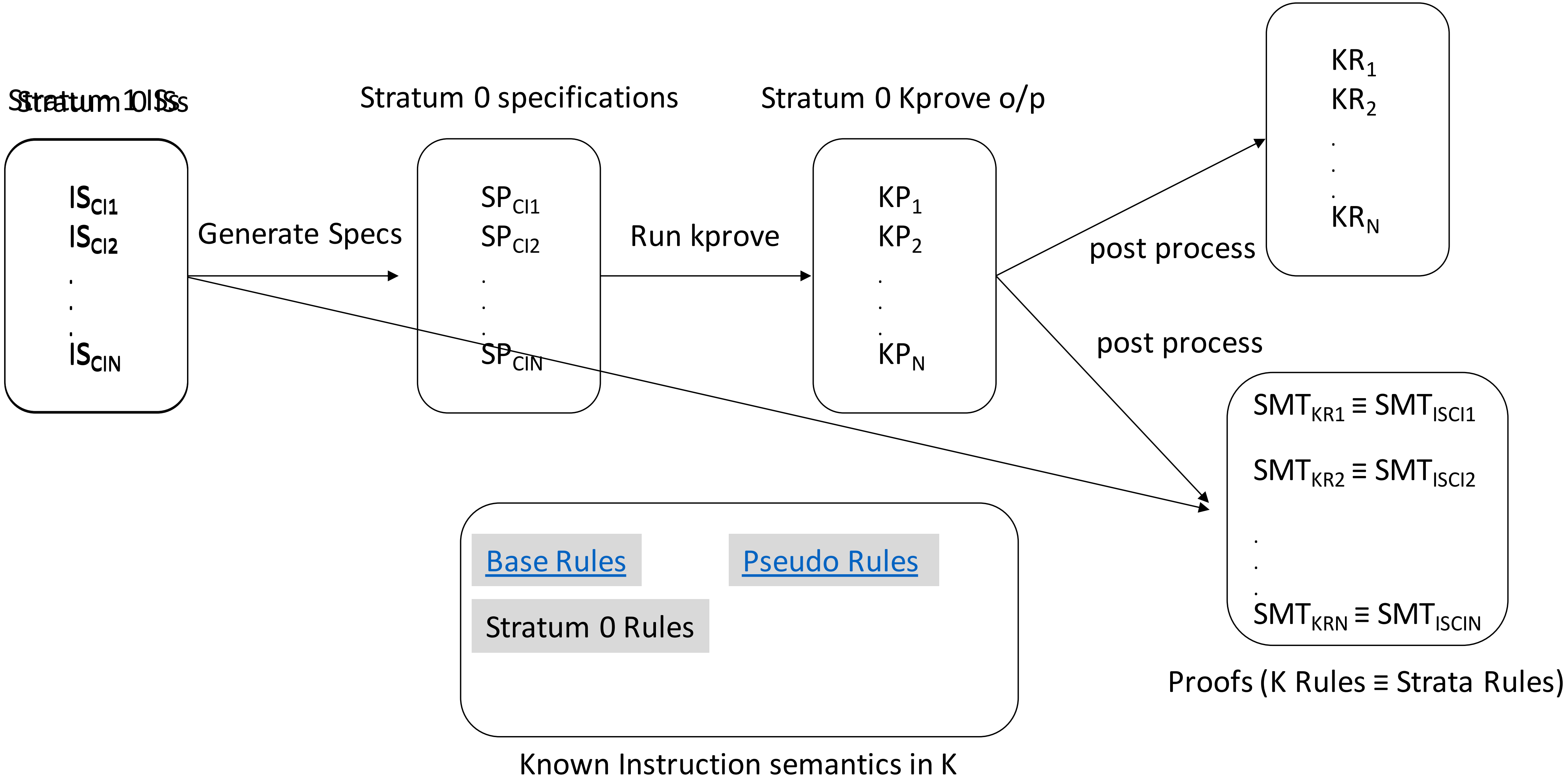
Bitvector Formula

```
%rbx  : (plus (concat <0x0|1> <%rcx|64>) (concat <0x0|1> <%rbx|64>))[63:0]

%cf   : (== (plus (concat <0x0|1> <%rcx|64>) (concat <0x0|1> <%rbx|64>))[64:64] <0x1|1>)
%zf   : (== (plus (concat <0x0|1> <%rcx|64>) (concat <0x0|1> <%rbx|64>))[63:0] <0x0|64>)
%sf   : (== (plus (concat <0x0|1> <%rcx|64>) (concat <0x0|1> <%rbx|64>))[63:63] <0x1|1>)
```

SMT_{ISCI}

Converting IS to K Rule: Overview



Converting IS to K Rule: Demo

- Generating Spec file from a concrete instruction sequence
- Symbolically execute that spec file
- Infer generic K rule.

Challenges

- The synthesized sequences agree with the target instruction only on the target's write set
  - I: Registers not in the read/write set might get clobbered (after before spec after)
  - II: Registers exclusively in read set might get clobbered
  - III: Sub-registers not in write set might get clobbered. (Spec, K rule)
- The generated rules could be extremely complex and huge, which in turn slow down further symbolic execution.
  - Use simplification lemmas. (after before after)

Simplification lemmas (~30)

- BV[I:J] ∘ BV[J:K] => BV[I:K]
- BV[0 : bitwidth(BV)-1] => BV
- (BV1[0:63] ∘ BV2[0:63])[0:31] => BV2[0:31]
- (BV1[0:63] ∘ BV2[0:63])[64:96] => BV1[0:31]
- (BV1[0:63] ∘ BV2[0:63])[32:96] => (BV1[0:31] ∘ BV2[32:63])
- (BV[32:63])[0:8] => BV[32:39]
- (BV1 boolOp BV2)[I:J] => BV1[I:J] & BV2[I:J]
- (cond ? BV1:BV2)[I:J] => (cond ? BV1[I:J]:BV2[I:J])
- BV ∘ (cond ? BV2 : BV3) => (cond ? BV ∘ BV1: BV ∘ BV2)
- (cond ? BV1 : BV2) binOp (cond ? BV3 : BV4) => (cond ? BV1 binOp BV3 : BV2 binOp BV4)

Proving K Rules ≡ Strata Rules

Motivation & Demo

- To expose flaws in strata's symbolic engine's.
  - Upon discrepancy we can use the counter example to test which one is right.
- To gain the same confidence in K rules as we have in strata.
- Verified simplification of K rules. (eg. vmovmskpd_r32_xmm)
- Examples: (eg. x86-cmovnll_r32_r32, vpmovsxwq_ymm_xmm)

Case study when z3 says "failed to prove"

- z3EquivFormulas/x86-shlq_r64_cl.py
- Help fixing a bug in K rule.

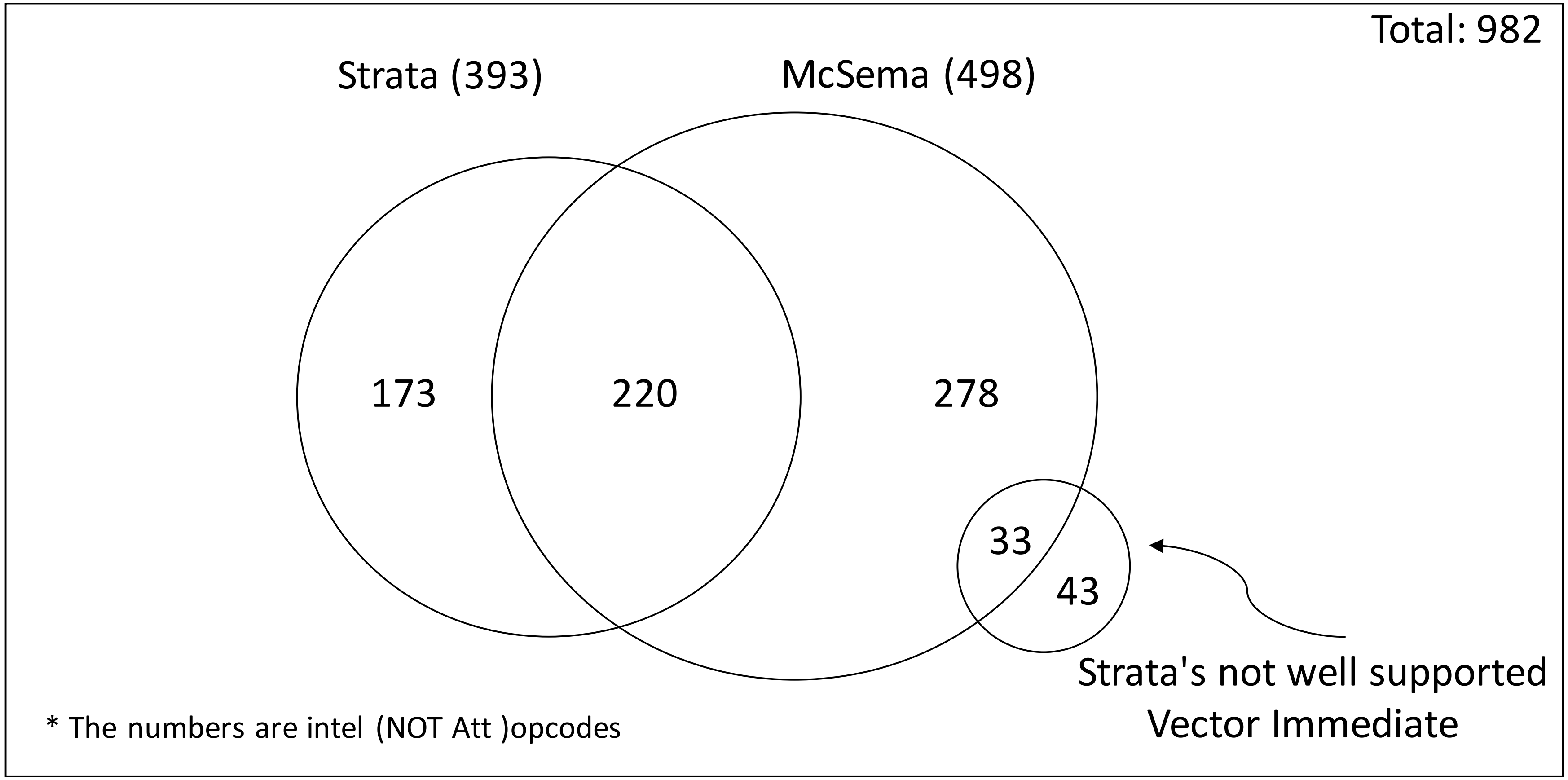
Proving K Rules ≡ Strata Rules : Limitations

- For proving equivalence of floating point instruction variants we need to use UIFs. (eg. x86-vfmadd132sd_xmm_xmm_xmm.py)
- Although the operational semantics have the more precise semantics (as they don't have UIFs), but we cannot very them. But can test!

K Rules Vs Strata Rules

- The formal specification of vector instruction in K are more precise: Does not contain UIFs.
- K rules are executable, so are strata’s instruction sequences (after pretending/appending the save/restore code for scratchpad registers)

Opcode support (Strata Vs McSema)



Going forward

- Borrowing semantics from McSema !
  - Borrow Candidates:
    - Not well supported vector immediate (33)
    - McSema only supported instructions (278)
  - How?
    - We have the LLVM's base instruction semantics (like the semantics for *gptr*, *bitcast*, etc)
    - We can run the sym-ex on LLVM instruction sequence, which at the end write symbolic values to virtual registers (which in our case are the hardware registers or flags)
- Extend stratification instead!
  - Improve base instruction
  - Better heuristics to guide search.
- Generalize the K rules to Immediate (already done) & Memory (TBD).