# Compilation using LLVM

**Juan Manuel Martinez Caamaño (@jmmartinez)**

**Quarkslab**

## Last course

- Dynamic Protections
- Profile-Guided Optimization

## Today's objective

- Symbolic Execution
- Dynamic Symbolic Execution

# Symbolic Execution

# Symbolic Execution

Track the values of instructions as expressions from the program input.

## Symbolic Execution

It's not possible to create a symbolic expression for every possible computed value.
We have to choose an abstract domain (as in dataflow analysis).

- Bit values
- Integer values
- Affine Expressions

# Symbolic Execution

Idea:

- Obtain a symbolic representation of the program
- Use an automatic theorem prover to find an input that makes the program reach a certain state

# Symbolic Execution

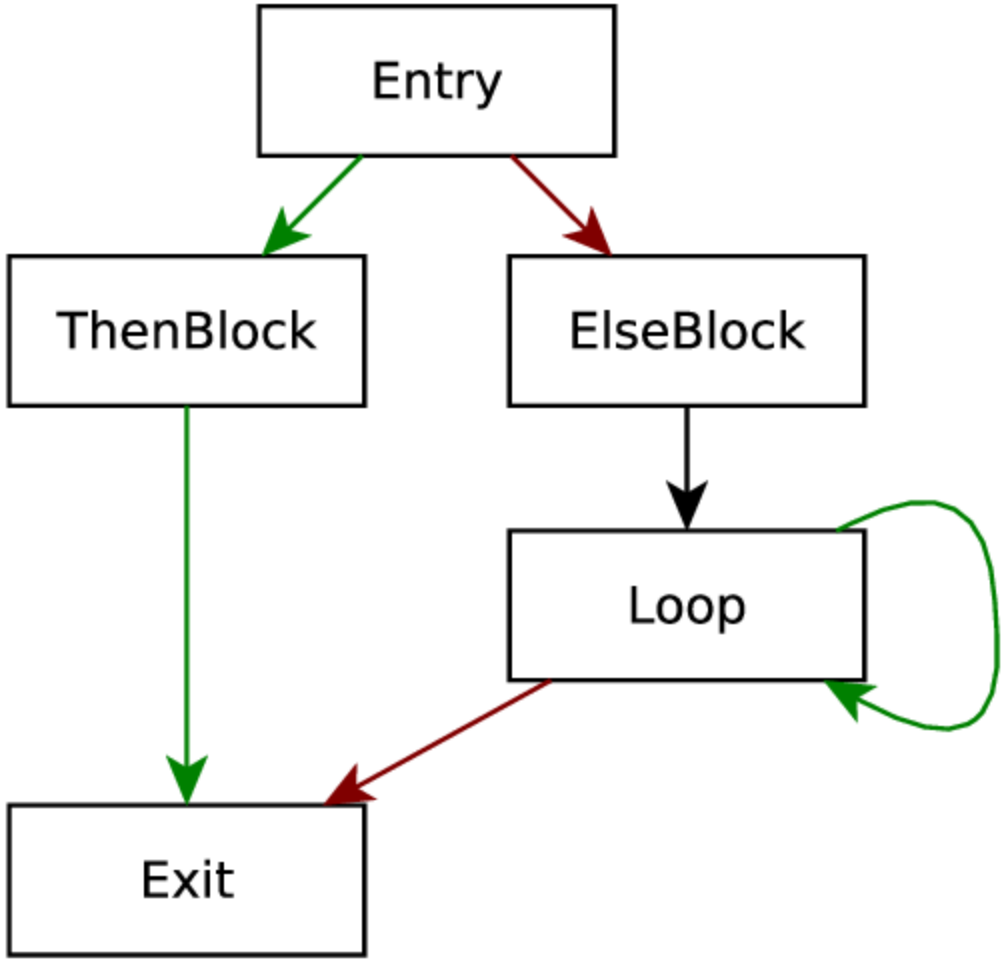
```
%x = add i16 %a 4
%y = xor i16 %b 0xabab
%z = add i16 %x %y
```

LLVM-IR

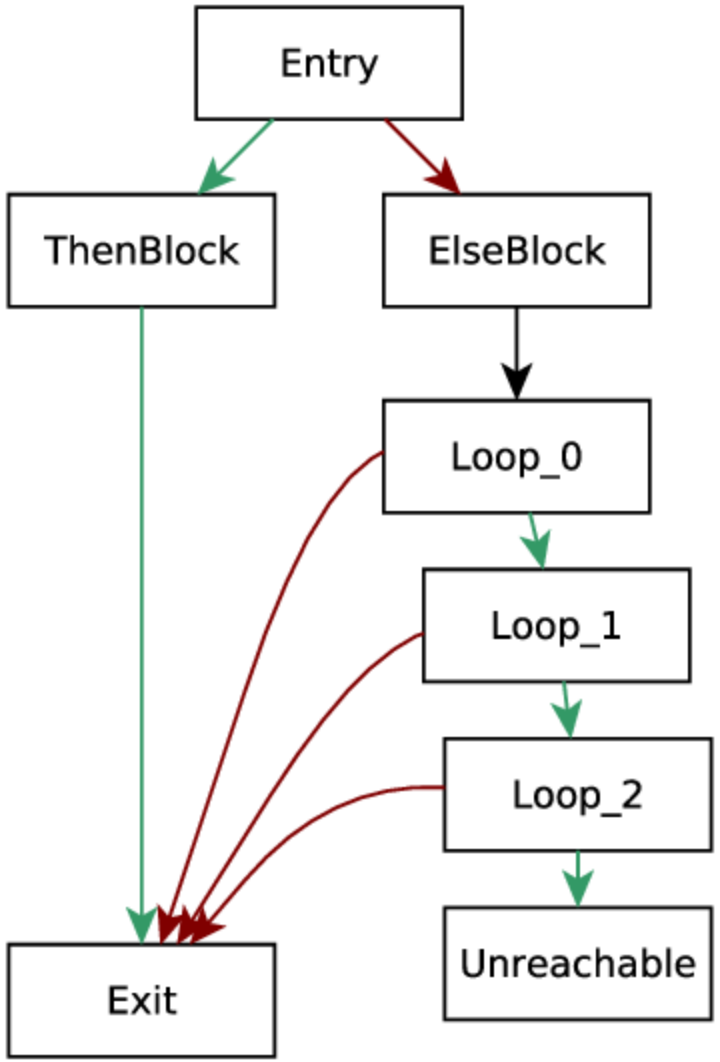
```
a = z3.BitVec('a', 16)
b = z3.BitVec('b', 16)
x = a + 4
y = b ^ 0xabab
z = x + y
```

Python Z3

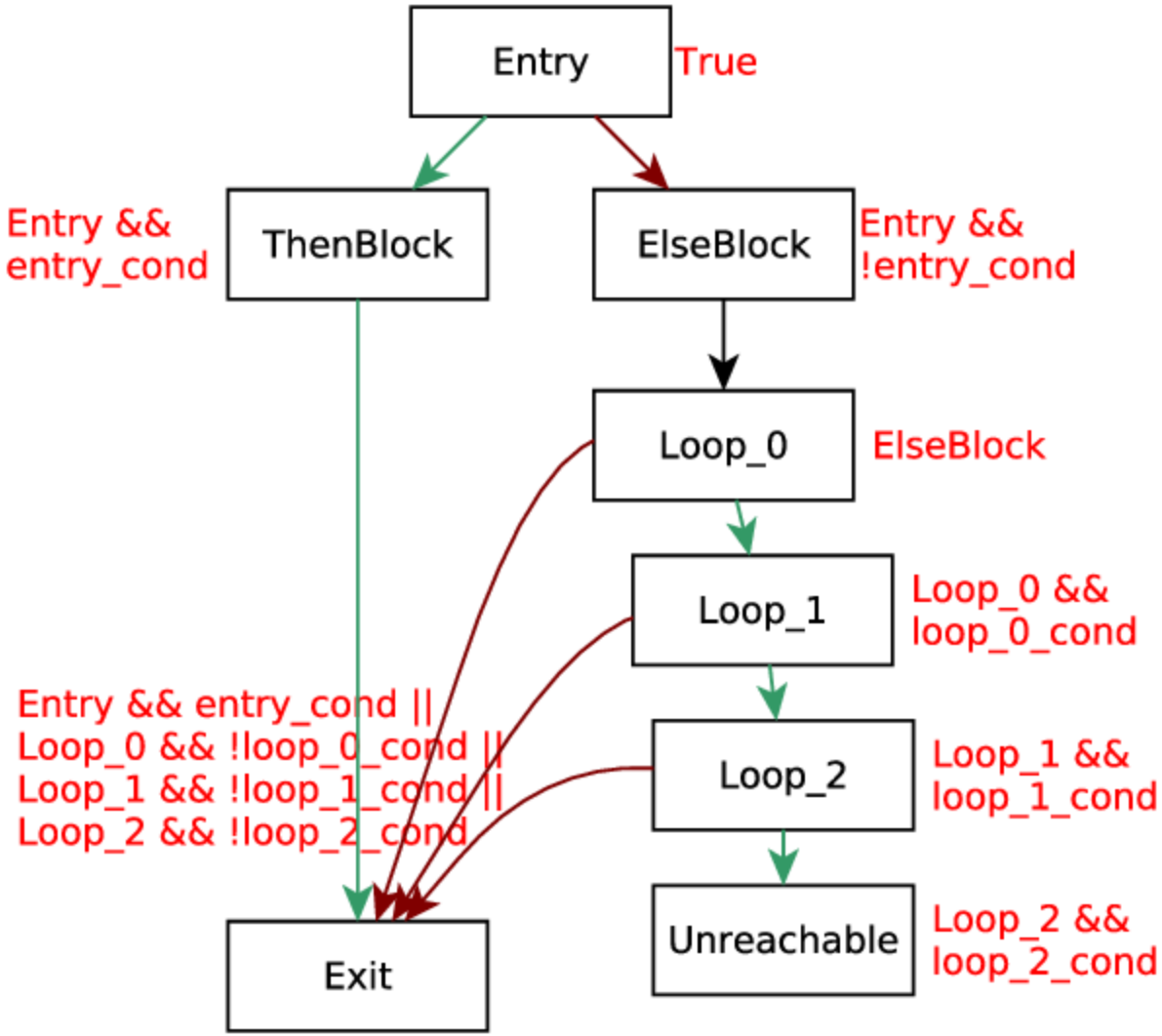
# Symbolic Execution



# Symbolic Execution



# Symbolic Execution



## Symbolic Execution

In reverse engineering it is used for:

- Inversing complex computations (e.g. a hash)
- Produce inputs that cover paths that have not been explored by fuzzing

# Symbolic Execution - Limitations

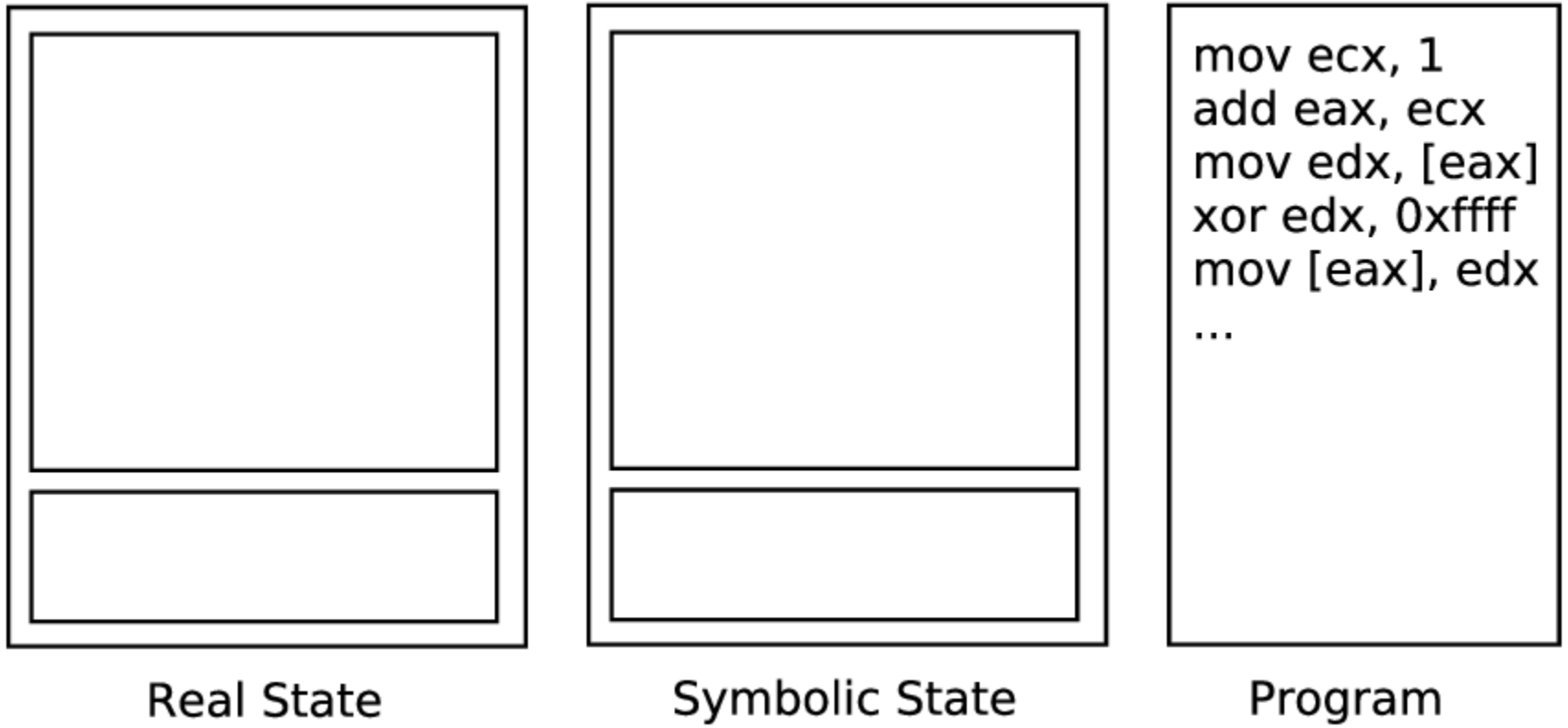
- Modeling Memory: Takes every possible behaviour into account
- Loops and recursion
- Path explosion
- Dealing with complex behaviours: system calls, input/output, concurrency, etc.

# Dynamic Symbolic Execution

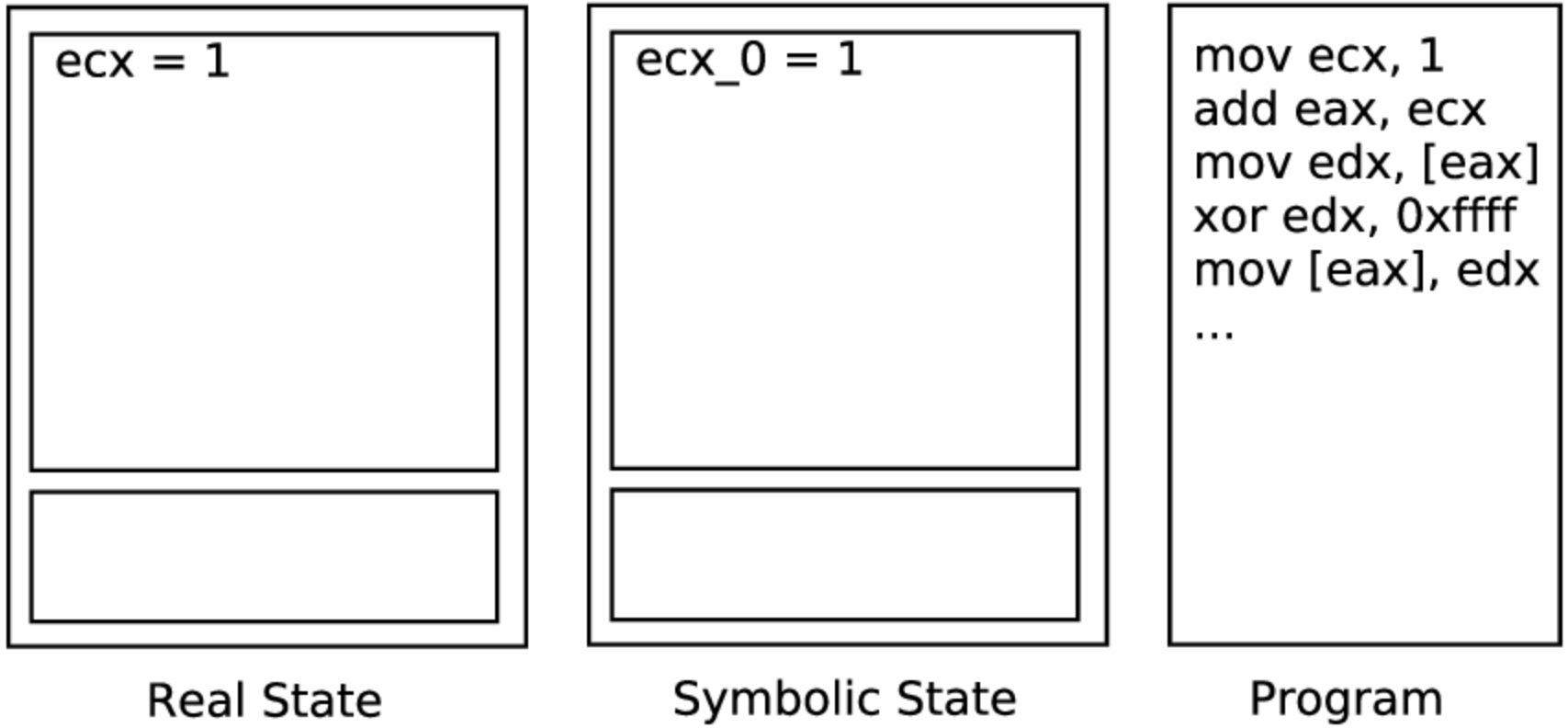
Overcome the explosion of states of pure symbolic execution

- Only take into account relevant paths
- When in trouble, disambiguate using concret execution

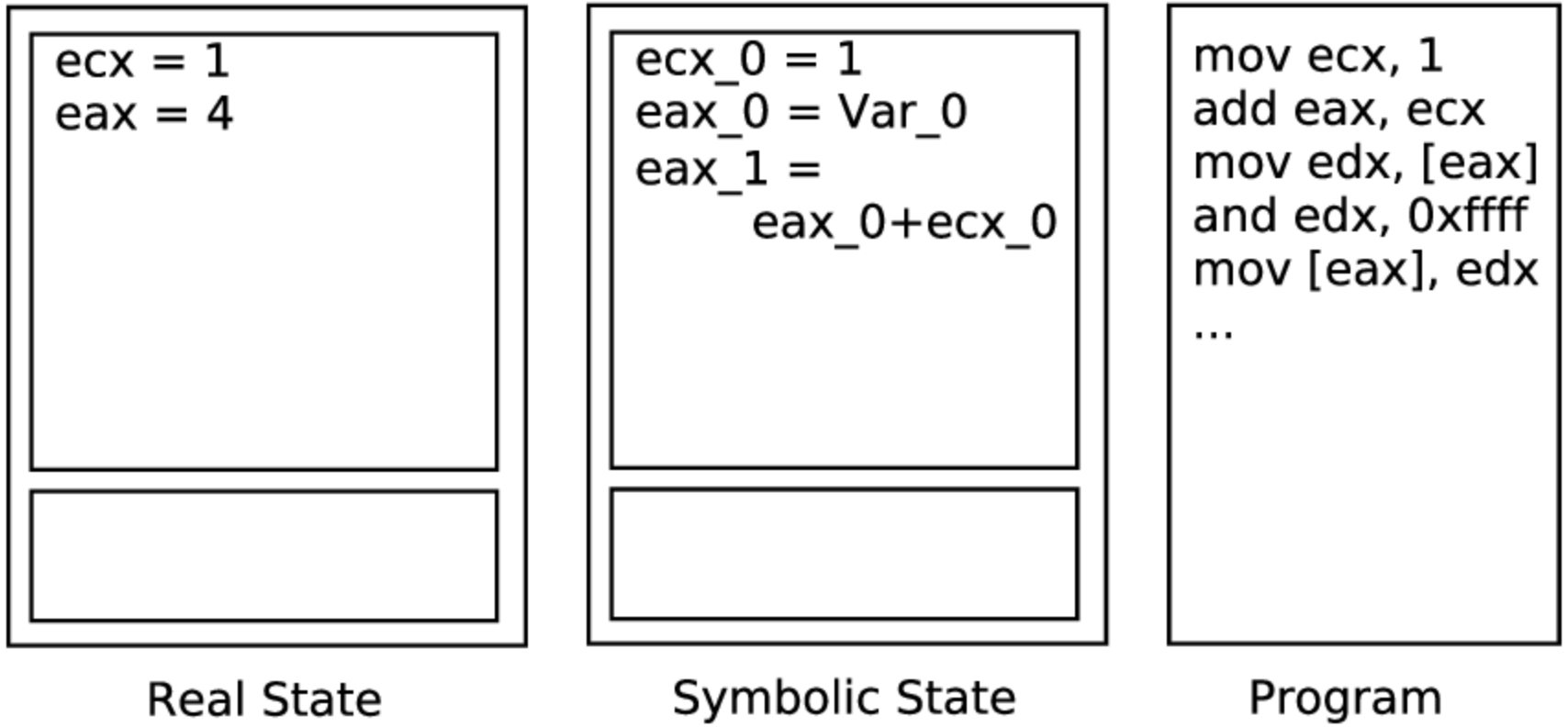
# How it works?


mov ecx, 1
add eax, ecx
mov edx, [eax]
xor edx, 0xffff
mov [eax], edx
...
Real State
Symbolic State
Program

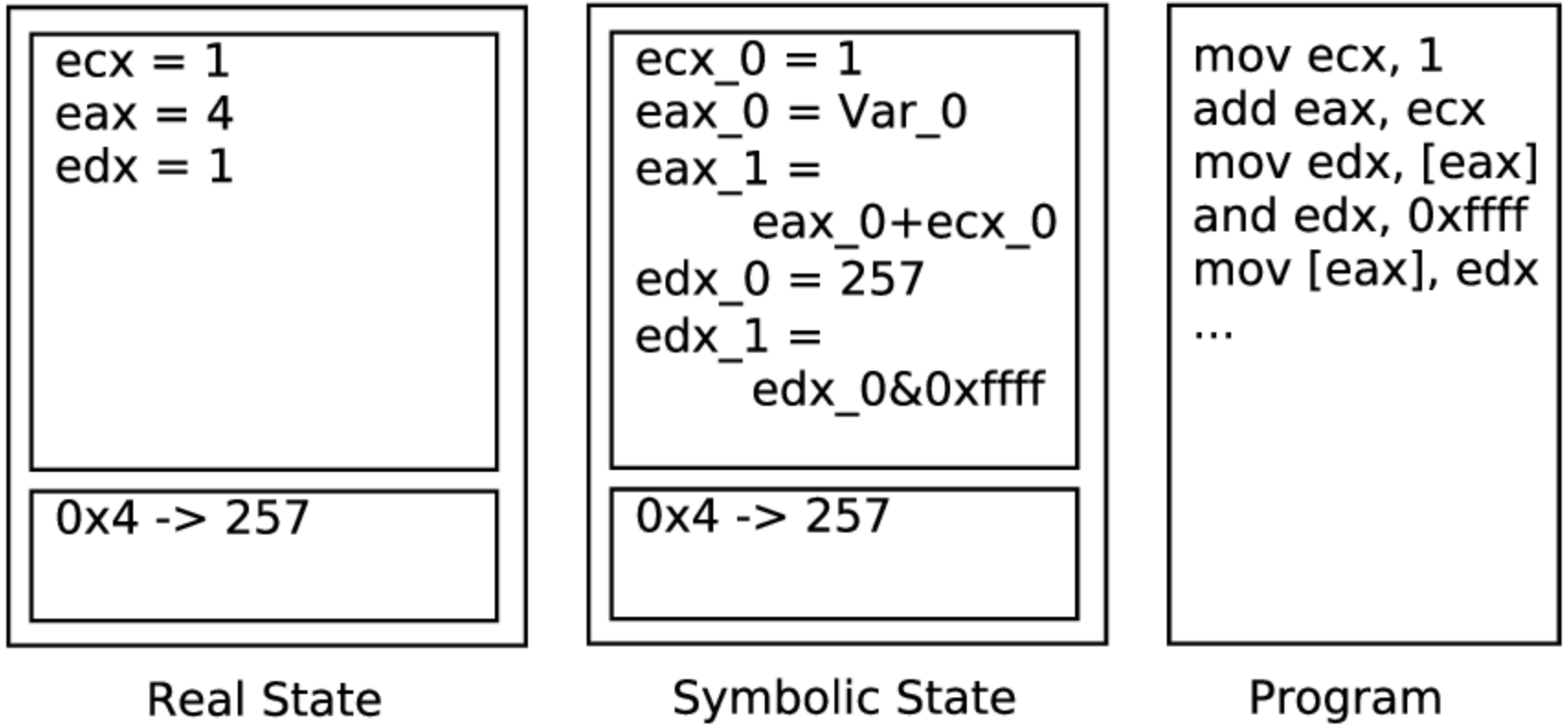
# How it works?


ecx = 1
ecx_0 = 1
mov ecx, 1
add eax, ecx
mov edx, [eax]
xor edx, 0xffff
mov [eax], edx
...
Real State
Symbolic State
Program

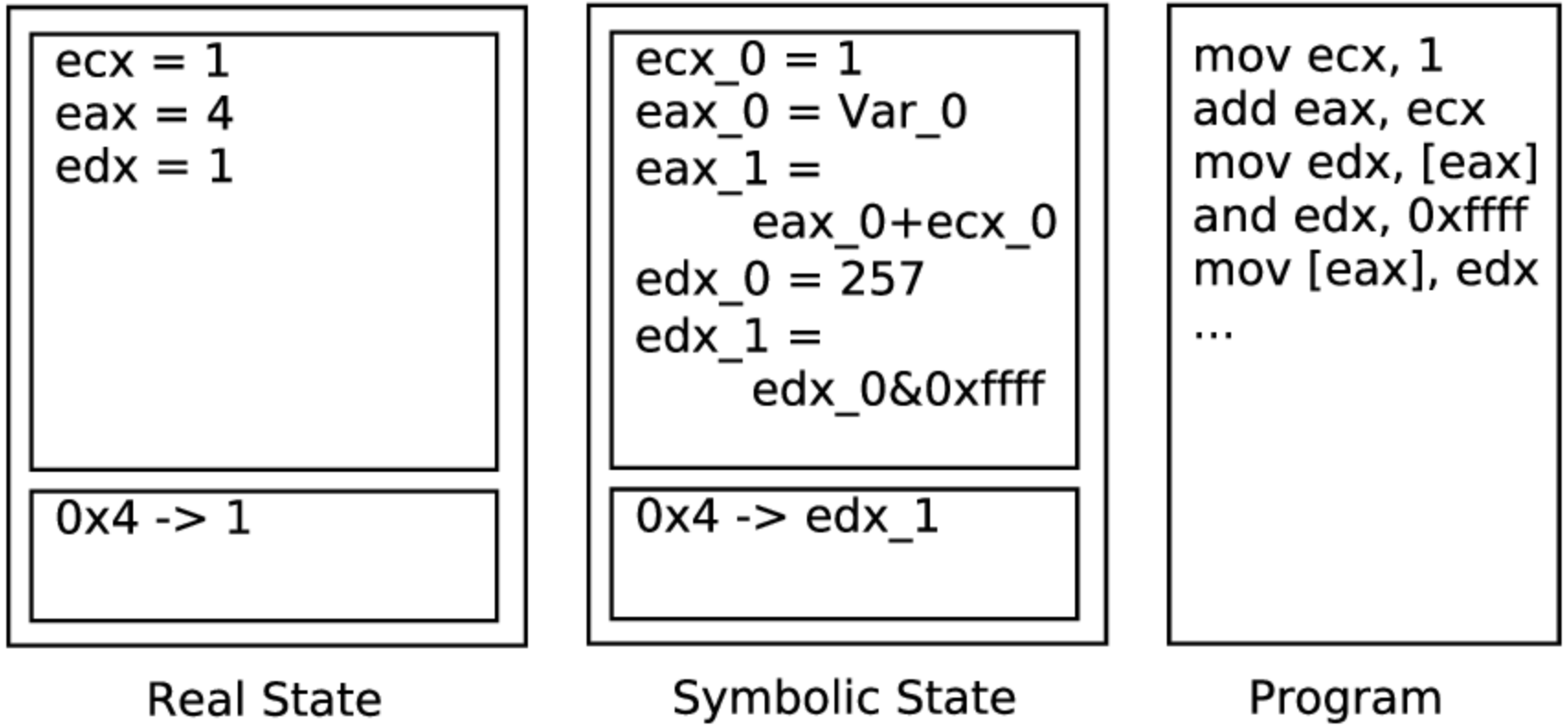
# How it works?


ecx = 1
eax = 4
ecx_0 = 1
eax_0 = Var_0
eax_1 =
eax_0+ecx_0
mov ecx, 1
add eax, ecx
mov edx, [eax]
and edx, 0xffff
mov [eax], edx
...
Real State
Symbolic State
Program

# How it works?


ecx = 1
eax = 4
edx = 1
0x4 -> 257
Real State
ecx_0 = 1
eax_0 = Var_0
eax_1 =
eax_0+ecx_0
edx_0 = 257
edx_1 =
edx_0&0xffff
0x4 -> 257
Symbolic State
mov ecx, 1
add eax, ecx
mov edx, [eax]
and edx, 0xffff
mov [eax], edx
...
Program

# How it works?


ecx = 1
eax = 4
edx = 1
0x4 -> 1
Real State
ecx_0 = 1
eax_0 = Var_0
eax_1 =
eax_0+ecx_0
edx_0 = 257
edx_1 =
edx_0&0xffff
0x4 -> edx_1
Symbolic State
mov ecx, 1
add eax, ecx
mov edx, [eax]
and edx, 0xffff
mov [eax], edx
...
Program

# Dynamic Symbolic Execution - Limitations

Altough the concrete execution helps reducing the scope of the symbolic execution, the problems remain

- Modeling Memory: Aproximation
- Loops and recursion
- Path explosion
- Dealing with complex behaviours: system calls, input/output, concurrency, etc.

# Conclusions

- Symbolic Execution considers every possible execution in the code and, build a formula that represents them
- Dynamic Symbolic Execution considers a subset of executions, and builds a formula that represents them
- They all exhibit flaws from relying on a SMT solver