FORMAL LAND

EVM EQUIVALENCES: PYTHON ↔ ROCQ RUST ↔ ROCQ

MARCH 2025

GOAL

Give a **Rocq** semantics of the **Python** specification of EVM and the **Rust Revm**, and prove them **equivalent**.

FORMAL LAND

PYTHON SPECIFICATION

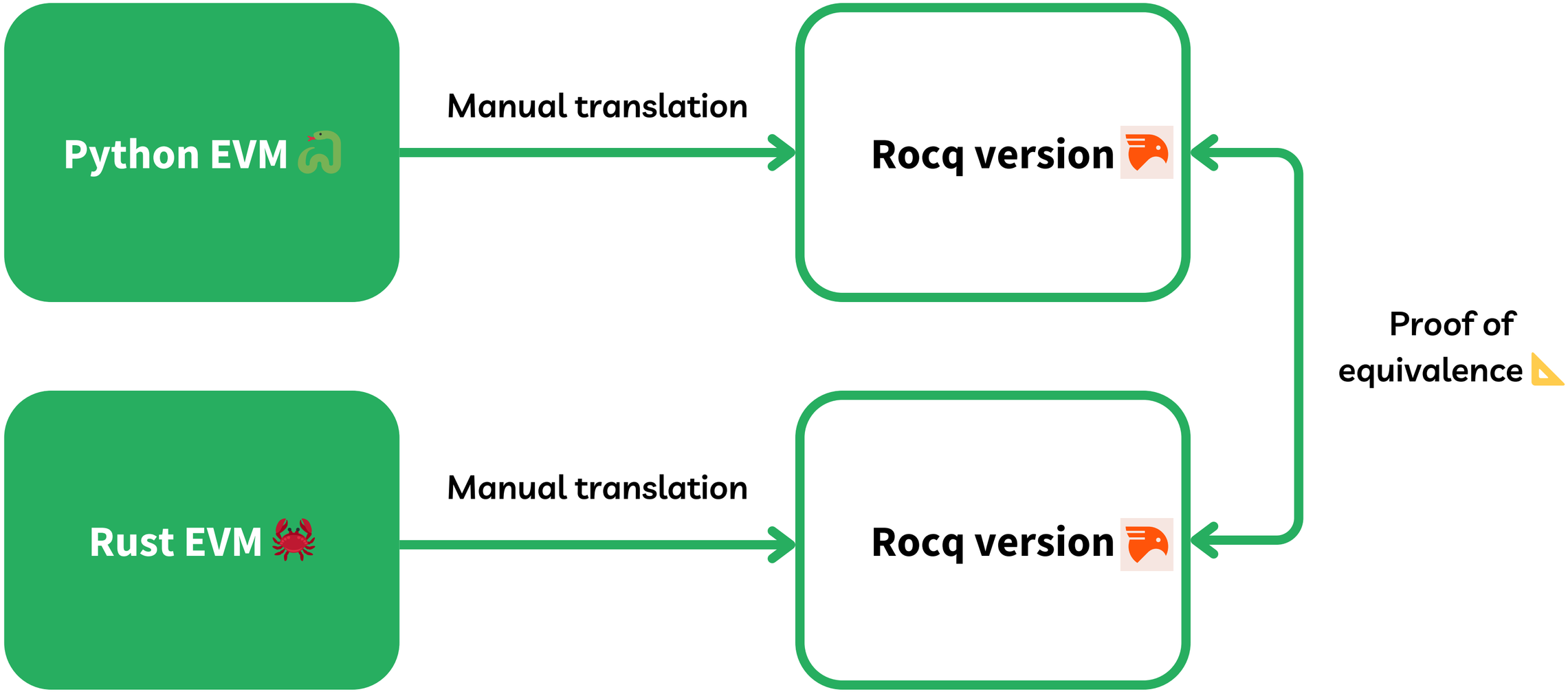
- https://github.com/ethereum/execution-specs
- Reference implementation
- Simple
- 3,000 lines per version

FORMAL LAND

RUST REVM

- https://github.com/bluealloy/revm
- Popular for zkVMs
- Optimized for speed
- 5,000 lines for the “interpreter” folder, 20,000 in total

FORMAL LAND


Python EVM
Manual translation
Rocq version
Proof of equivalence
Rust EVM
Manual translation
Rocq version

FORMAL LAND

PYTHON

```python
def div(evm: Evm) -> None:
    """
    Integer division of the top two elements of the stack. Pushes the result
    back on the stack.

    Parameters
    ----------
    evm :
        The current EVM frame.

    """
    # STACK
    dividend = pop(evm.stack)
    divisor = pop(evm.stack)

    # GAS
    charge_gas(evm, GAS_LOW)

    # OPERATION
    if divisor == 0:
        quotient = U256(0)
    else:
        quotient = dividend // divisor

    push(evm.stack, quotient)

    # PROGRAM COUNTER
    evm.pc += 1
```

ROCQ

```
Definition div : MS? Evm.t Exception.t unit :=
  (* STACK *)
  letS? divident := StateError.lift_lens Evm.Lens.stack pop in
  letS? divisor := StateError.lift_lens Evm.Lens.stack pop in

  (* GAS *)
  letS? _ := charge_gas GAS_LOW in

  (* OPERATION *)

  let division :=
    match (U256.to_Z divident) with
    | 0 =>
      U256.of_Z 0
    | _ =>
      U256.of_Z ((U256.to_Z divisor) / (U256.to_Z divident))
  end in

  let result := division in

  letS? _ := StateError.lift_lens Evm.Lens.stack (push result) in

  (* PROGRAM COUNTER *)
  letS? _ := StateError.lift_lens Evm.Lens.pc (fun pc =>
    (inl tt, Uint.__add__ pc (Uint.Make 1))) in

  returnS? tt.
```

FORMAL LAND

RUST

```rust
pub fn div<H: Host + ?Sized>(
    interpreter: &mut Interpreter
    _host: &mut H
) {
    gas!(interpreter, gas::LOW);
    pop_top!(interpreter, op1, op2);
    if *op2 != U256::ZERO {
        *op2 = op1.wrapping_div(*op2);
    }
}
```

ROCQ

```
Definition div :
    MS? Interpreter.t string unit :=
  letS? _ := gas_macro Gas.LOW in
  letS? '(op1, op2_ref) := pop_top_macro2 in
  liftS? Interpreter.Lens.stack (
    liftS?of!? op2_ref (
      letS? op2 := readS? in
      if U256.eq op2 U256.ZERO
      then returnS? tt
      else writeS? (U256.wrapping_div op1 op2)
    )
  ).
```

FORMAL LAND

ISSUE

- The code is **similar**
- But we can make **mistakes**
- Hard to follow **upgrades**

COQ-OF-RUST

- Tool to import **Rust to Rocq/Coq**
- Automatic
- https://github.com/formal-land/coq-of-rust

FORMAL LAND

EXAMPLE

```
Definition div (ε : list Value.t) (τ : list Ty.t) (α : list Value.t) : M :=
  match ε, τ, α with
  | [], [ H ], [ interpreter; _host ] ⇒
    ltac:(M.monadic
      (let interpreter := M.alloc (| interpreter |) in
      let _host := M.alloc (| _host |) in
      M.catch_return (|
        ltac:(M.monadic
          (M.read (|
            let~ _ :=
              M.match_operator (|
                M.alloc (| Value.Tuple [] |),
                [
                  fun γ ⇒
                    ltac:(M.monadic
                      (let γ :=
                        M.use
                          (M.alloc (|
                            UnOp.not (|
                              M.call_closure (|
                                M.get_associated_function (|
                                  Ty.path "revm_interpreter::gas::Gas",
                                  "record_cost",
                                  []
                                |),
                                [
                                  M.SubPointer.get_struct_record_field (|
                                    M.read (| interpreter |),
                                    "revm_interpreter::interpreter::Interpreter",
                                    "gas"
                                  |);
                                  M.read (|
                                    M.get_constant (| "revm_interpreter::gas::constants::LOW" |)
                                  |)
                                ]
                              |)
                            |)
                          |)) in
                      let _ :=
                        M.is_constant_or_break_match (| M.read (| γ |), Value.Bool true |) in
```

FORMAL LAND

OUTPUT

- Directly the Rust AST (THIR)
- Too low-level
- Too verbose

REFINEMENTS

1. **“Links”** Re-construct type information and name/trait resolution
2. **“Simulations”** From (mutable) references to a state monad

FORMAL LAND

LINKS

- Automation tactic in Rocq
- We handle a few instructions
- We write the types and generate the rest by unification

FORMAL LAND

EXAMPLE

```
Instance run_add
    {WIRE H : Set} `{Link WIRE} `{Link H}
    {WIRE_types : InterpreterTypes.Types.t} `{InterpreterTypes.Types.AreLinks WIRE_types}
    (run_InterpreterTypes_for_WIRE : InterpreterTypes.Run WIRE WIRE_types)
    (interpreter : Ref.t Pointer.Kind.MutRef (Interpreter.t WIRE WIRE_types))
    (_host : Ref.t Pointer.Kind.MutRef H) :
  Run.Trait
    instructions.arithmetic.add [] [ Φ WIRE; Φ H ] [ φ interpreter; φ _host ]
    unit.
Proof.
  constructor.
  cbn.
  eapply Run.Rewrite. {
    repeat erewrite IsTraitAssociatedType_eq by apply run_InterpreterTypes_for_WIRE.
    reflexivity.
  }
  destruct run_InterpreterTypes_for_WIRE.
  destruct run_StackTrait_for_Stack.
  destruct popn_top as [popn_top [H_popn_top run_popn_top]].
  destruct run_LoopControl_for_Control.
  destruct gas as [gas [H_gas run_gas]].
  destruct set_instruction_result as [set_instruction_result [H_set_instruction_result
  run_set_instruction_result]].
  run_symbolic.
Defined.
```

FORMAL LAND

SIMULATIONS

- Still in progress
- A few simple gas helper functions for now
- The goal of relying on automation also to only have to write the types

FORMAL LAND

GOAL

Having “links” + “simulations” for 80% of the Revm at the end of March.

FORMAL LAND

NEXT

- Show that the refinement Revm is equivalent to a manual and idiomatic version of the EVM in Rocq
- Make a rigorous translation from Python to Rocq

FORMAL LAND

THANKS

FORMAL LAND