General algorithmic problem solving competitions with crowd-sourced solution validation

A blockchain-based platform design

Efstathios Aliprantis
Athens University of Economics and Business

GEC 2021, Greece

Algorithm Competitions

I have a problem,
I need an algorithm

Petitioner

I can create algorithms

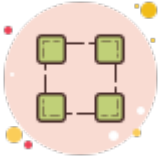
Creator

A Traditional Approach

We judge
submitted algorithms

Experts
Committee

Proposed Blockchain-based Platform



I fully moderate
the competition
& determine
the winner

Blockchain-based
Platform

Icons by Icons8

Example: Factorization

```
Verifier(instance,
         solution?,
         fail_reason*):

  product = 1
  for c in solution?:
    if not c is not prime:
      return False
    product = product * c

  return product == instance
```

~ Made by Petitioner

```
Solver1(instance):

  solution = empty_array()

  for i in 2..instance:
    while instance mod i == 0:
      instance = instance div i
      solution.append(i)

  return solution
```

~ Made by Creator1

```
Solver2(instance):

  if instance mod 2 == 0:
    return [2, instance div 2]

  return [instance]
```

~ Made by Creator2

Instance	Solver1	Verifier
34	[2, 17]	True
25	[5, 5]	True

Instance	Solver2	Verifier
34	[2, 17]	True
25	[25]	**False**

Parameter ***fail_reason* is used whenever necessary to speed up computations on Verifier*

Basic Flow of a Competition

Main Observation

Solver is not correct
<=>
∃ (instance, fail_reason) s.t. **Verifier**(instance, **Solver**(instance), fail_reason) = **False**

Basic Flow of a Competition

1) The **Petitioner** submits the **Verifier**

2) A **Creator** submits a **Solver** and pays a deposit

3) a. A Tester (human being) finds a “counter-instance”
=> Solver rejected, Tester receives the deposit

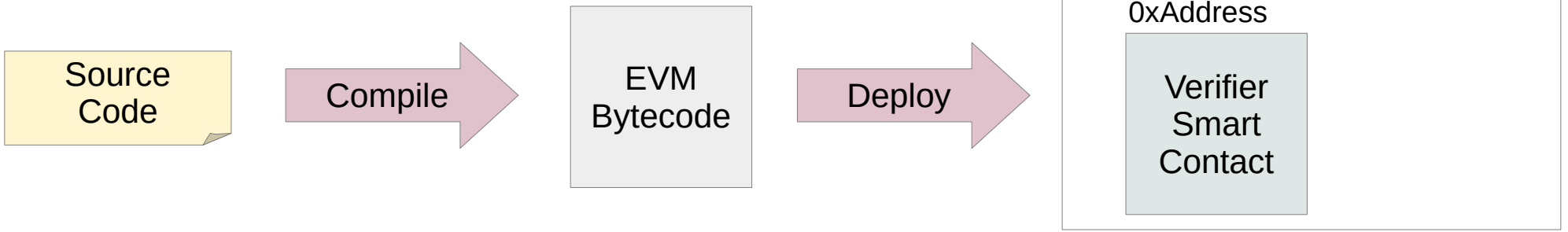
b. No “counter-instance” found within a predefined time period
=> Solver accepted, deposit refunded to Creator

Ethereum can execute code & handle payments automatically!

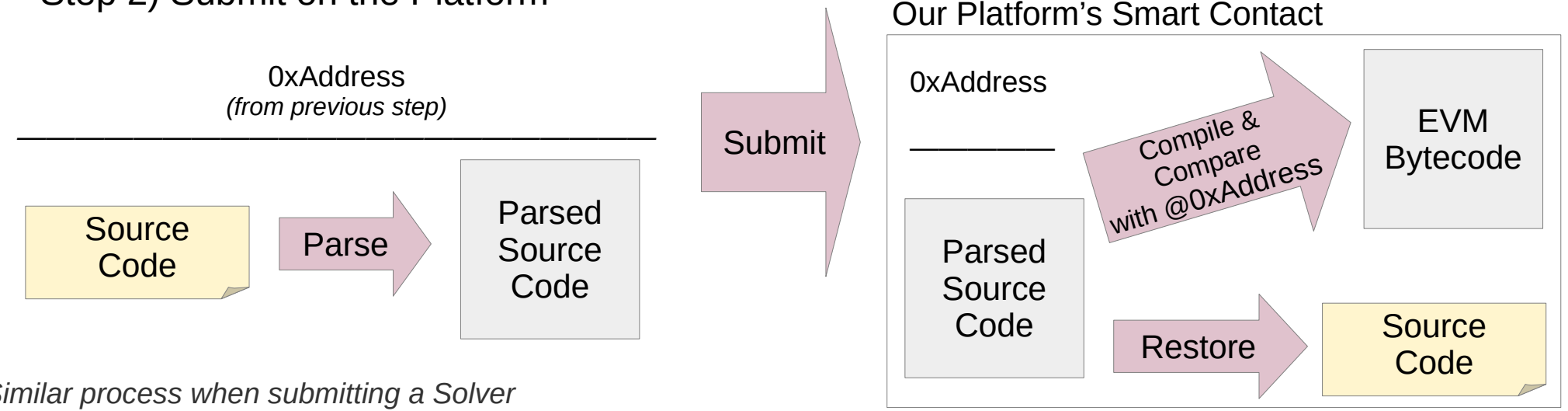
Blockchain ensures fully transparent processes

Submitting a Verifier

Step 1) Deploy on Ethereum



Step 2) Submit on the Platform



Similar process when submitting a Solver

Enforcing Solver Complexity Constraints

- The compiler outputs instructions that measure steps & space used by Solvers
- The Petitioner submits one more function to limit the allowed steps/space of Solvers: `ComplexityConstraints: instance → (max_steps, max_space)`

Limitations

- Verifiers & Solvers must be able to be implemented (no pure theoretic constructs)
- Verifiers & Solvers must run in (gas) attainable time/space
  - **Hint**: Verifiers can utilize the `fail_reason` parameter to reduce complexity
- Solvers can only be judged per instance, not on average

What ‘s Next: Proofs of correctness

The Platform will be able to, also, validate proofs of correctness

- Definitely correct accepted algorithms
- Not relying on Testers
- No need to wait for a predefined time period

Computer-based proof validation will be automated thus error-free

Acknowledgments

I’d like to thank my advisor prof. Eugénie Foustoucos for her constant guidance

Some Related Work

- Dilia Rodriguez, 2016, “Verification Games: Crowd-Sourced Formal Verification”. Air Force Research Laboratory AFRL-RI-RS-TR-2016-096, URL: https://apps.dtic.mil/sti/pdfs/AD1006471.pdf
- Borching Su, 2018, “MathCoin: A Blockchain Proposal that Helps Verify Mathematical Theorems In Public”. IACR Cryptol. ePrint Arch. 2018: 271, URL: https://eprint.iacr.org/2018/271.pdf
- For a list of platforms for “Algorithm Programming Competitions” see https://cs.au.dk/~gerth/code *(NOTE: Our Platform differs from listed ones as it aims to solve problems of interest for Petitioners while the listed ones pose problems already solved as a mean to challenge the contestants)*

References (Ethereum)

- Vitalik Buterin, 2013. “A Next-Generation Smart Contract and Decentralized Application Platform”. URL: https://ethereum.org/en/whitepaper
- Gavin Wood, 2014. “Ethereum: A Secure Decentralised Generalised Transaction Ledger”. URL: http://gavwood.com/Paper.pdf