ERGO

RESEARCH INSTITUTE.

ALEXANDER CHEPURNOY, VASILY KHARIN, DMITRY MESHKOV

SELF-REPRODUCING COINS AS UNIVERSAL TURING MACHINE

Overview

- The paper is online: `https://arxiv.org/abs/1806.10116`

OVERVIEW

- The paper is online: `https://arxiv.org/abs/1806.10116`
- The very general question: how can we do more with less?

QUESTIONS

- Can we achieve Turing completeness without jump (or equivalent) opcode?

QUESTIONS

- Can we achieve Turing completeness without jump (or equivalent) opcode?
- Can we do it in UTXO model?

QUESTIONS

- Can we achieve Turing completeness without jump (or equivalent) opcode?
- Can we do it in UTXO model?
- If so, what is the practical use of it?

OUTLINE

- TURING COMPLETENESS IN BLOCKCHAIN ENVIRONMENT
- SCRIPTING LANGUAGE PREREQUISITES, UTXO'S, ERGO
- GENERAL CONSTRUCTION + PROOF OF TURING COMPLETENESS
- PRACTICAL CASES
- CONCLUSIONS

TURING COMPLETENESS AND BLOCKCHAIN

- A system is Turing complete if it can emulate universal Turing machine

TURING COMPLETENESS AND BLOCKCHAIN

- A system is Turing complete if it can emulate universal Turing machine
- Blockchain requires constant block validation time

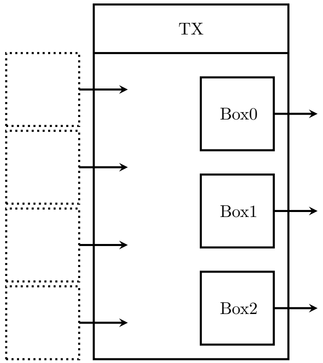
TURING COMPLETENESS AND BLOCKCHAIN

- A system is Turing complete if it can emulate universal Turing machine
- Blockchain requires constant block validation time
- There is a popular tenet in the folklore that if one needs Turing completeness on blockchain, jump, while, or recursion must be present on the script level
- There is a popular tenet in the folklore that if one needs Turing completeness on blockchain, jump, while, or recursion must be present on the script level

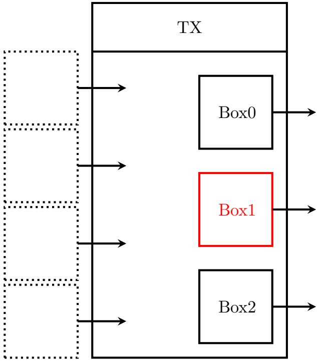
TURING COMPLETENESS AND BLOCKCHAIN

- A system is Turing complete if it can emulate universal Turing machine
- Blockchain requires constant block validation time
- There is a popular tenet in the folklore that if one needs Turing completeness on blockchain, jump, while, or recursion must be present on the script level
- There is a popular tenet in the folklore that if one needs Turing completeness on blockchain, jump, while, or recursion must be present on the script level
- But imagine Ethereum$^{\ominus}$ where there is gas limit per block, and all the state changes are reversed after transaction script execution (and payment amount is not dependent on the program). Then the Ethereum$^{\ominus}$ is not Turing-complete.

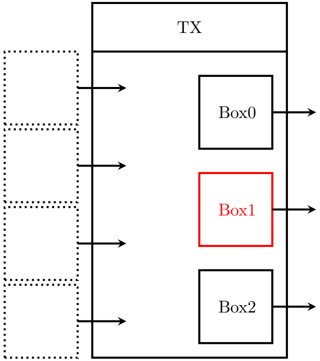
Bitcoin$^{\oplus}$: Boxes, Registers, Scripting

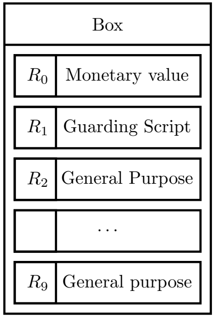


Bitcoin$^{\oplus}$: Boxes, Registers, Scripting

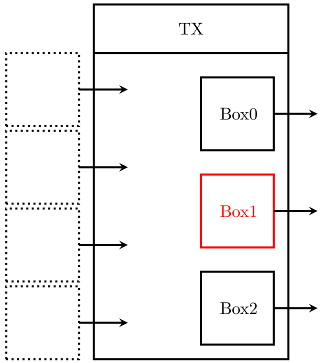


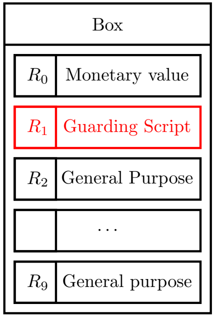
Bitcoin$^{\oplus}$: Boxes, Registers, Scripting



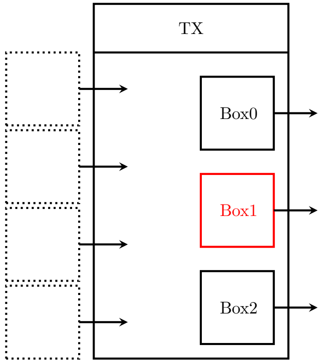


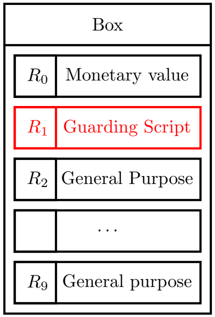
BITCOIN$^{\oplus}$: BOXES, REGISTERS, SCRIPTING





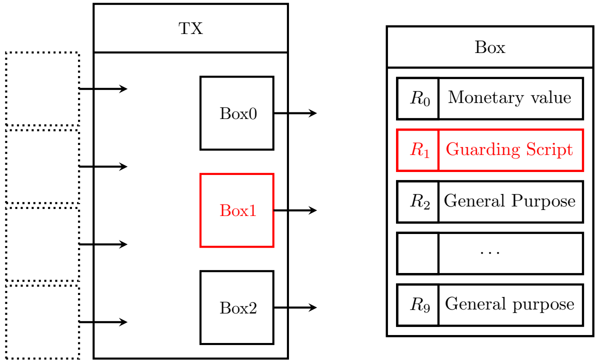
BITCOIN$^{\oplus}$: BOXES, REGISTERS, SCRIPTING





SCRIPTING LANGUAGE

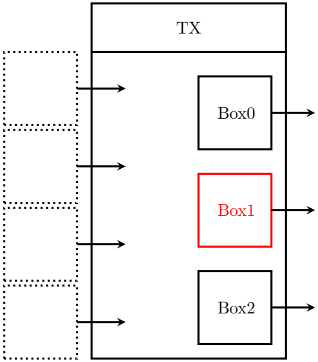
BITCOIN$^{\oplus}$: BOXES, REGISTERS, SCRIPTING

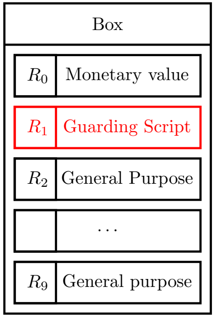


SCRIPTING LANGUAGE

- Basic arithmetics

BITCOIN$^{\oplus}$: BOXES, REGISTERS, SCRIPTING

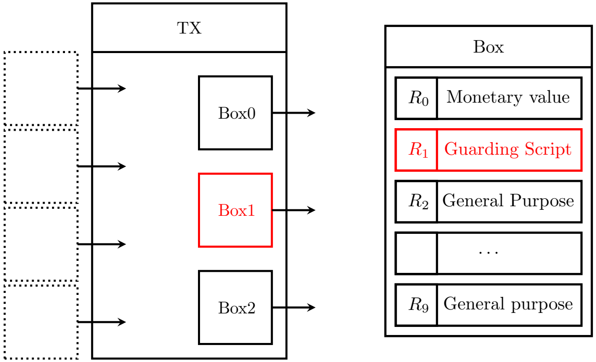




SCRIPTING LANGUAGE

- Basic arithmetics
- If-then-else clause

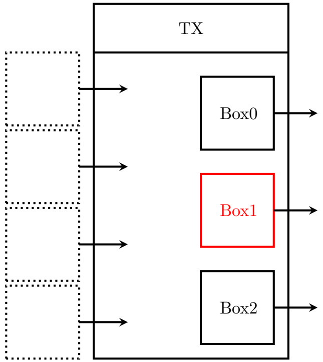
BITCOIN$^{\oplus}$: BOXES, REGISTERS, SCRIPTING

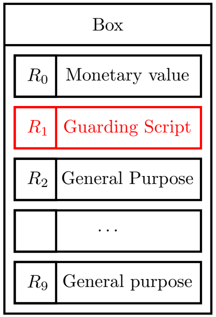


SCRIPTING LANGUAGE

- Basic arithmetics
- If-then-else clause
- Cryptographic primitives

BITCOIN$^{\oplus}$: BOXES, REGISTERS, SCRIPTING

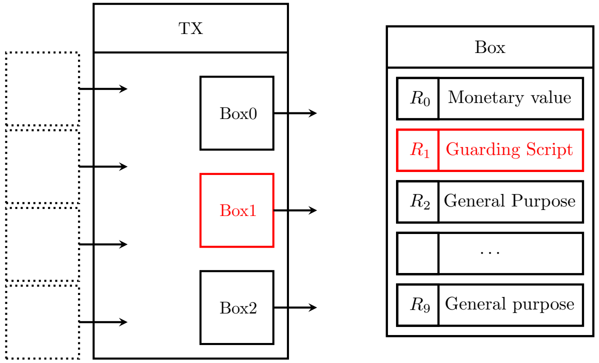




SCRIPTING LANGUAGE

- Basic arithmetics
- If-then-else clause
- Cryptographic primitives
- Array declaration (predefined size) and access operations

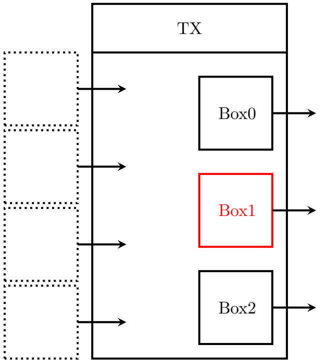
BITCOIN$^{\oplus}$: BOXES, REGISTERS, SCRIPTING

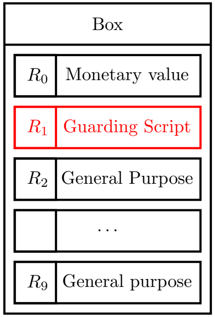


SCRIPTING LANGUAGE

- Basic arithmetics
- If-then-else clause
- Cryptographic primitives
- Array declaration (predefined size) and access operations
- Context data
  - HEIGHT: Int
  - SELF: Box
  - INPUTS: Array[Box]
  - OUTPUTS: Array[Box]

BITCOIN$^{\oplus}$: BOXES, REGISTERS, SCRIPTING





SCRIPTING LANGUAGE

- Basic arithmetics
- If-then-else clause
- Cryptographic primitives
- Array declaration (predefined size) and access operations
- Context data
    - HEIGHT: Int
    - SELF: Box
    - INPUTS: Array[Box]
    - OUTPUTS: Array[Box]

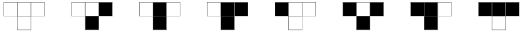
RULE 110 CELLULAR AUTOMATON

Turing completeness can be demonstrated by simulating a simple Turing complete system.

Cook, M.: Universality in elementary cellular automata (`https://git.io/vj6sw`)

RULE 110 CELLULAR AUTOMATON

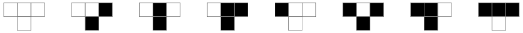
Turing completeness can be demonstrated by simulating a simple Turing complete system.



Cook, M.: Universality in elementary cellular automata (`https://git.io/vj6sw`)

RULE 110 CELLULAR AUTOMATON

Turing completeness can be demonstrated by simulating a simple Turing complete system.

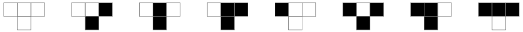


$$\mathsf{x} = \ell \cdot \mathsf{c} \cdot \mathsf{r} + \mathsf{c} \cdot \mathsf{r} + \mathsf{c} + \mathsf{r} \mod 2$$

Cook, M.: Universality in elementary cellular automata (`https://git.io/vj6sw`)

RULE 110 CELLULAR AUTOMATON

Turing completeness can be demonstrated by simulating a simple Turing complete system.



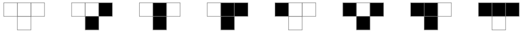
$$x = \ell \cdot c \cdot r + c \cdot r + c + r \mod 2$$

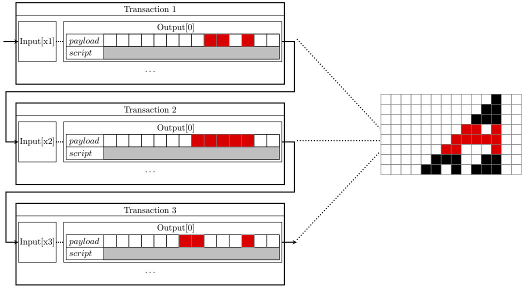
```
let indices: Array[Int]=Array(0, 1, 2, 3, 4, 5)
let inLayer: Array[Byte]=SELF.R3[Array[Byte]].value
fun procCell(i: Int): Byte = {
    let l = inLayer((if(i==0) 5 else (i-1)))
    let c = inLayer(i)
    let r = inLayer((i + 1) % 6)
    ((l * c * r + c * r + c + r) % 2).toByte
}
(OUTPUTS(0).R3[Array[Byte]].value==
            indices.map(procCell)) &&
 (OUTPUTS(0).R1 == SELF.R1)
```

Cook, M.: Universality in elementary cellular automata (`https://git.io/vj6sw`)

RULE 110 CELLULAR AUTOMATON

Turing completeness can be demonstrated by simulating a simple Turing complete system.





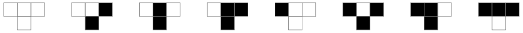
$$x = \ell \cdot c \cdot r + c \cdot r + c + r \mod 2$$

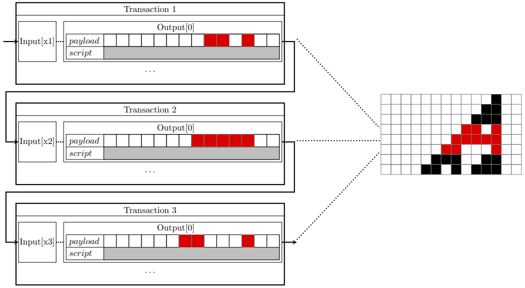
```
let indices: Array[Int]=Array(0, 1, 2, 3, 4, 5)
let inLayer: Array[Byte]=SELF.R3[Array[Byte]].value
fun procCell(i: Int): Byte = {
    let l = inLayer((if(i==0) 5 else (i-1)))
    let c = inLayer(i)
    let r = inLayer((i + 1) % 6)
    ((l * c * r + c * r + c + r) % 2).toByte
}
(OUTPUTS(0).R3[Array[Byte]].value==
            indices.map(procCell)) &&
 (OUTPUTS(0).R1 == SELF.R1)
```

Cook, M.: Universality in elementary cellular automata (`https://git.io/vj6sw`)

RULE 110 CELLULAR AUTOMATON

Turing completeness can be demonstrated by simulating a simple Turing complete system.





Cook, M.: Universality in elementary cellular automata (`https://git.io/vj6sw`)

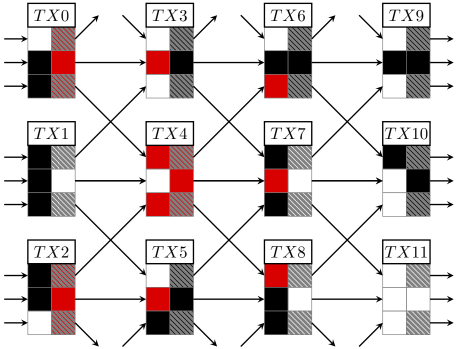
$$x = \ell \cdot c \cdot r + c \cdot r + c + r \mod 2$$

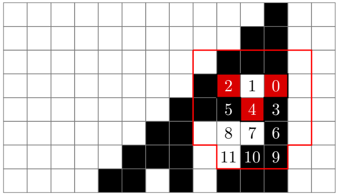
```
let indices: Array[Int]=Array(0, 1, 2, 3, 4, 5)
let inLayer: Array[Byte]=SELF.R3[Array[Byte]].value
fun procCell(i: Int): Byte = {
    let l = inLayer((if(i==0) 5 else (i-1)))
    let c = inLayer(i)
    let r = inLayer((i + 1) % 6)
    ((l * c * r + c * r + c + r) % 2).toByte
}
(OUTPUTS(0).R3[Array[Byte]].value==
            indices.map(procCell)) &&
 (OUTPUTS(0).R1 == SELF.R1)
```

We just allowed recursive calls by granting the script access to an output.

RULE 110 CELLULAR AUTOMATON: INFINITE TAPE

Use transaction chaining to share memory between the transactions





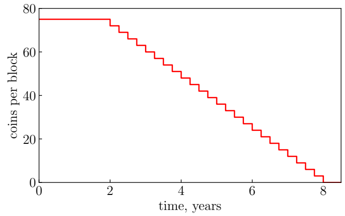
https://git.io/vj6rX

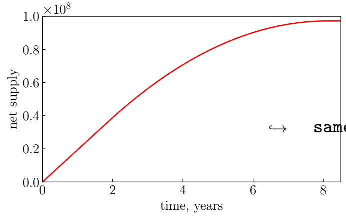
EXAMPLES

- Crowdfunding
- Demurrage currency
- Oracles (w. authenticated state)
- Decentralized exchanges

`https://github.com/ergoplatform/ergo/blob/master/papers/yellow`

Practical case: Ergo token emission





```
let epoch = 1 + (HEIGHT-fixedRatePeriod)/epochLength
let out = OUTPUTS(0)
let coinsToIssue = if(HEIGHT < fixedRatePeriod) fixedRate
else fixedRate - oneEpochReduction*epoch
let correctCoinsConsumed = coinsToIssue==(SELF.value - out.value)
let sameScriptRule = SELF.propositionBytes==out.propositionBytes
let heightIncreased = HEIGHT>SELF.R3[Long].value
let heightCorrect = out.R3[Long].value==HEIGHT
let lastCoins = SELF.value<=oneEpochReduction
(correctCoinsConsumed && heightCorrect && heightIncreased &&
↪ sameScriptRule) || (heightIncreased && lastCoins)
```

https://github.com/ergoplatform/ergo/blob/master/papers/yellow

CONCLUSIONS

- Turing Completeness of the blockchain system can be achieved by unwinding the recursive calls between the transactions. It fully complies with the blockchain requirements, and does not require ad-hoc structures to bypass the halting problem
- We provide the explicit proof of the Turing completeness of Ergo blockchain scripting system. To our knowledge, this is the first proof of that kind
- The construction is explicit, and the functionality is fully implemented
- Self-reproducing coins allow one to make practical constructions. As an example, significant fraction of the validation rules can be brought from the hard-coded form to the scripting layer. Moreover, the logic of arbitrary complexity can be potentially implemented

THANK YOU FOR YOUR ATTENTION