Blockchain Abstract Data Type

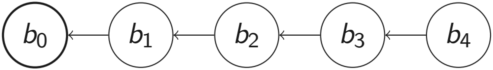
E. Anceaume, A. Del Pozzo, R. Ludinard, M. Potop-Butucaru, S. Tucci-Piergiovanni

Blockchain day, @LINCS

July 12th, 2019

Blockchain: a distributed public ledger

Ideally, the Blockchain is an append-only (immutable) chain of blocks.

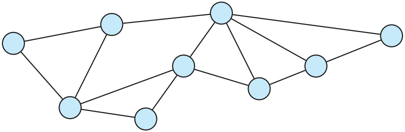


Each **block** contains the hash of the previous block and other application dependent information (as transactions).

Few Important points

Blockchain runs on a distributed system: different nodes are involved

Nodes communicate exchanging messages.



Each node has a local copy of the Blockchain

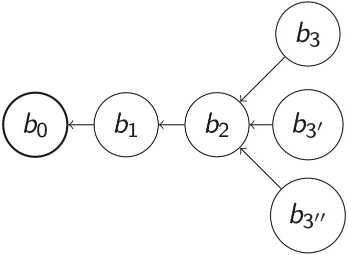
Append a new block

When there is a new block, who appends it?

Append a new block

When there is a new block, who appends it?

We want to preserve a chain shape, so we do not want to have multiple writers per time:



Two main approaches to append

We want one writer per block height.

- Proof-of-Work: a peer in order to append a new block has to provide as a proof the solution of a cryptographic puzzle.
  - it may happen to have more than one peer writing concurrently.

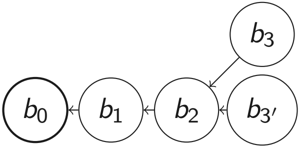
Two main approaches to append

We want one writer per block height.

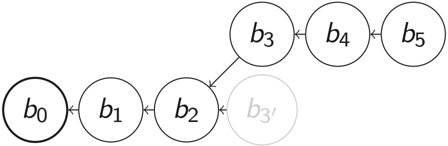
- Proof-of-Work: a peer in order to append a new block has to provide as a proof the solution of a cryptographic puzzle.
  - it may happen to have more than one peer writing concurrently.
- Consensus: peers agree on the next block to append.
  - Consensus does not scale;

Fork

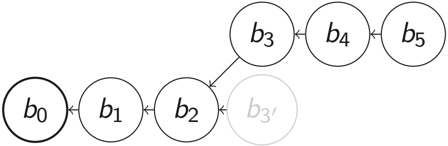
There can be more than one peer that appends, i.e., solves the PoW to append at the same block, in such case we have a **fork**.



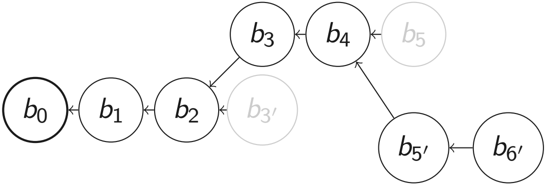
Fork Resolution: the longest chain is the main chain.



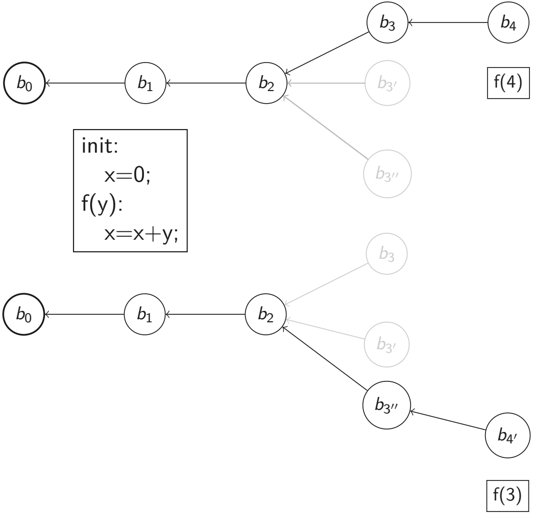
What do we read?



Different peers can have a different version of the Blockchain (due to network delays). **Which kind of consistency is provided?**

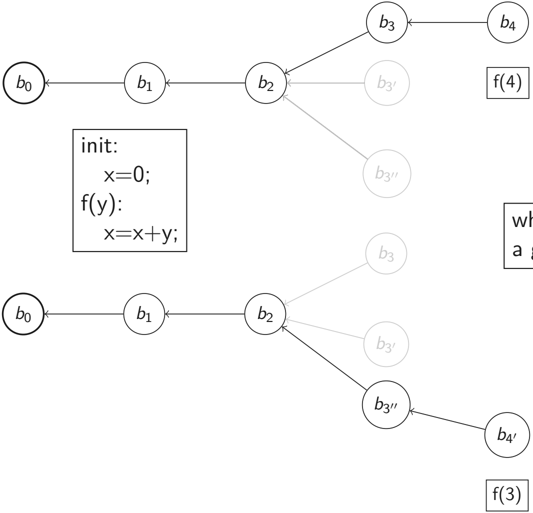


Example: Smart Contracts on Blockchain



```
init:
   x=0;
f(y):
   x=x+y;
```

Example: Smart Contracts on Blockchain



```
init:
    x=0;
f(y):
    x=x+y;
```

what is the value of x at
a generic time t at the two sites?

Blockchain, from the origin to nowadays

2008, ₿ S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

Blockchain, from the origin to nowadays

2008, S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum , Hyperledger

2016, PeerCensus, ByzCoin

2017, RedBelly, Algorand

... and many others

Blockchain, from the origin to nowadays

2008, ₿ S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum , Hyperledger

2016, PeerCensus, ByzCoin

How to formalize them?

2017, RedBelly, Algorand

... and many others

Blockchain, from the origin to nowadays

2008, S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum, Hyperledger

2016, PeerCensus, ByzCoin

How to formalize them?

2017, RedBelly, Algorand

... and many others

2017, A. Girault et al., Why You Can't Beat Blockchains: Consistency and High Availability in Distributed Systems.

2018, A. Fernández Anta et al., Formalizing and implementing distributed ledger objects.

Blockchain, from the origin to nowadays

2008, ₿ S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum , Hyperledger

2016, PeerCensus, ByzCoin

How to formalize them?

2017, RedBelly, Algorand

... and many others

2017, A. Girault et al., Why You Can't Beat Blockchains: Consistency and High Availability in Distributed Systems.

2018, A. Fernández Anta et al., Formalizing and implementing distributed ledger objects.

few attempts to formalize Blockchain as a **list** of records

Blockchain, from the origin to nowadays

2008, S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum , Hyperledger

2016, PeerCensus, ByzCoin

2017, RedBelly, Algorand

... and many others

How to formalize them?

2017, A. Girault et al., Why You Can't Beat Blockchains: Consistency and High Availability in Distributed Systems.

2018, A. Fernández Anta et al., Formalizing and implementing distributed ledger objects.

few attempts to formalize Blockchain as a **list** of records

Blockchain, from the origin to nowadays

2008, S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum , Hyperledger

2016, PeerCensus, ByzCoin

2017, RedBelly, Algorand

... and many others

How to formalize them?

2017, A. Girault et al., Why You Can't Beat Blockchains: Consistency and High Availability in Distributed Systems.

2018, A. Fernández Anta et al., Formalizing and implementing distributed ledger objects.

few attempts to formalize Blockchain as a **list** of records

Blockchain, from the origin to nowadays

2008, S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System

2015, Ethereum, Hyperledger

2016, PeerCensus, ByzCoin

2017, RedBelly, Algorand

... and many others

Our contribution:

A unified construction providing formal specifications capturing forkable and non-forkable blockchains

E. Anceaume et al. *Blockchain Abstract Data Type*. In SPAA 2019

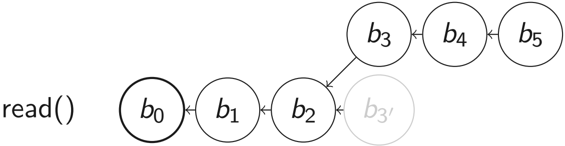
2017, A. Girault et al., Why You Can't Beat Blockchains: Consistency and High Availability in Distributed Systems.

2018, A. Fernández Anta et al., Formalizing and implementing distributed ledger objects.

Abstract Data Type

Our approach:

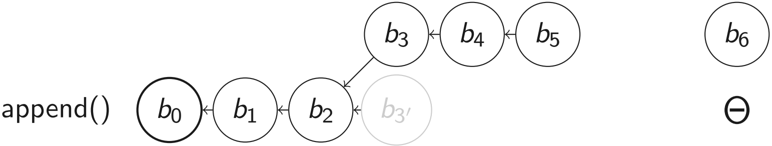
- Blockchain formalized as a **tree** of blocks: BlockTree Abstract Data Type;
- the block generation process is formalized as an Oracle compoundable with the BlockTree: Θ Token Oracle Abstract Data Type.



Abstract Data Type

Our approach:

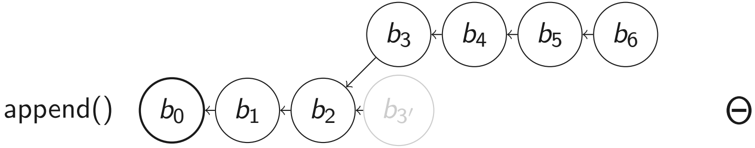
- Blockchain formalized as a **tree** of blocks: BlockTree Abstract Data Type;
- the block generation process is formalized as an Oracle compoundable with the BlockTree: Θ Token Oracle Abstract Data Type.



Abstract Data Type

Our approach:

- Blockchain formalized as a **tree** of blocks: BlockTree Abstract Data Type;
- the block generation process is formalized as an Oracle compoundable with the BlockTree: Θ Token Oracle Abstract Data Type.



©CEA_List 2019

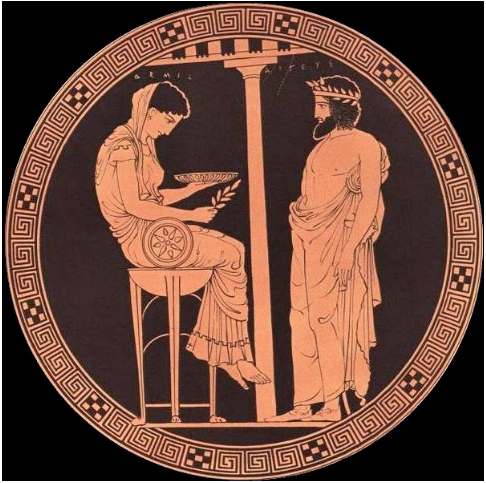
BlockTree Abstract Data Type

The BlockTree Abstract Data Type exposes two operations:

- read(): selects a blockchain in the blocktree;

- append(b): appends the block b to the blocktree if such block is valid, i.e., it satisfies a predicate P.

Token Oracle

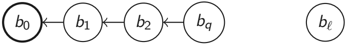


Any process that wants to append a block must call the oracle.

Token Oracle

The Token Oracle Θ_k Abstract Data Type exposes two operations:

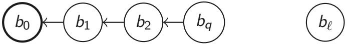
- getToken(b_q, b_ℓ): returns or not the right to extend the block b_k with block b_ℓ.



Token Oracle

The Token Oracle Θ_k Abstract Data Type exposes two operations:

- getToken(b_q, b_ℓ): returns or not the right to extend the block b_k with block b_ℓ.

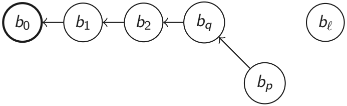
  

- consumeToken$(b_\ell^{b_q})$: allows a valid block to be appended or not, depending on how many blocks already extend b_q.

Frugal and Prodigal Token Oracles

A Frugal Oracle $\Theta_{F,k}$ allows to append at most k blocks to the same block.

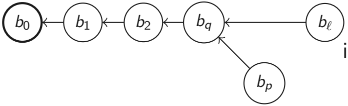
A Prodigal Oracle Θ_P allows to append an unlimited number of blocks to any block.



Frugal and Prodigal Token Oracles

A Frugal Oracle $\Theta_{F,k}$ allows to append at most k blocks to the same block.

A Prodigal Oracle Θ_P allows to append an unlimited number of blocks to any block.

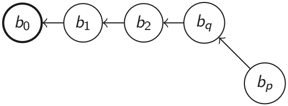


if $\Theta_{F,k>1}$ or Θ_P

Frugal and Prodigal Token Oracles

A Frugal Oracle $\Theta_{F,k}$ allows to append at most k blocks to the same block.

A Prodigal Oracle Θ_P allows to append an unlimited number of blocks to any block.



if $\Theta_{F,k=1}$

BlockTree Abstract Data Type

The BlockTree Abstract Data Type exposes two operations:

- read(): selects a blockchain in the blocktree;
- append(b): appends the block b to the blocktree if such block is valid, i.e., it satisfies a predicate P.

We establish two consistency criteria predicating on the result of the read() operations.

Blockchain Consistency Criteria

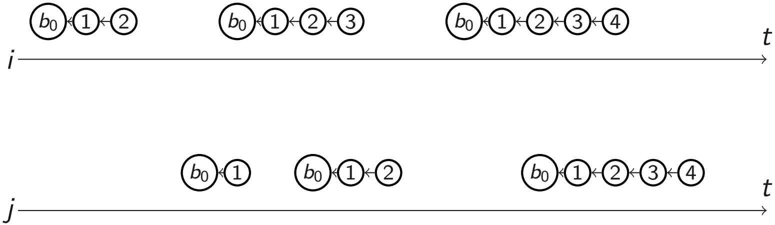
Eventual Consistency Criteria (EC):

- Local Monotonic Read;
- Validity;
- Ever Growing Tree;
- **Eventual Prefix** properties.

Strong Consistency Criteria (SC) :

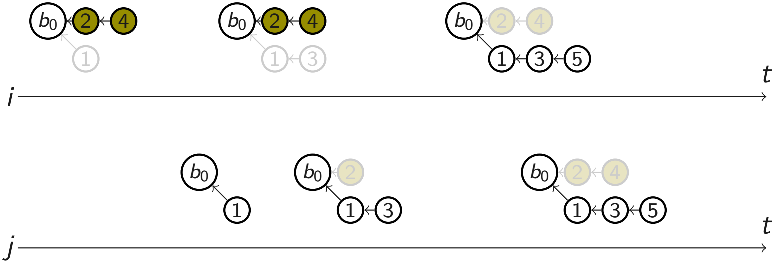
- Local Monotonic Read;
- Validity;
- Ever Growing Tree;
- **Strong Prefix** properties.

Strong Prefix Property



Strong prefix property: for each pair of read() operations, one returns a blockchain that is the prefix of the other or vice versa.

Eventual Prefix Property



Eventual prefix property: For each read blockchain with a score s, eventually all the subsequent read blockchains share a maximum common prefix with a score of at least s.

Blocktree and Oracle ADT hierarchy

$$\begin{tikzcd}
 & \boxed{\mathfrak{R}(\text{BT-ADT}_{SC}, \Theta_{F,k=1})} \arrow[dl] \arrow[dr] & \\
\boxed{\mathfrak{R}(\text{BT-ADT}_{SC}, \Theta_{F,k>1})} \arrow[d] & & \boxed{\mathfrak{R}(\text{BT-ADT}_{SC}, \Theta_{P})} \arrow[ddl] \\
\boxed{\mathfrak{R}(\text{BT-ADT}_{EC}, \Theta_{F,k>1})} \arrow[dr] & & \\
 & \boxed{\mathfrak{R}(\text{BT-ADT}_{EC}, \Theta_{P})} &
\end{tikzcd}$$

$\Theta_{F,k=1}$ has Consensus number ∞

Θ_P has Consensus number 1

We compose the BlockTree ADT and the Oracle ADT as $\mathfrak{R}(\text{BT-ADT}, \Theta)$ in a hierarchy.
In this way, we can state implementability results on the weakest combination and propagate them above.

Impossibilities

- It is not possible to implement a Blockchain satisfying Eventual Consistency if an update message is lost;

Impossibilities

- It is not possible to implement a Blockchain satisfying Eventual Consistency if an update message is lost;
- It is not possible to implement a Blockchain satisfying Strong Consistency if a fork occurs;

Impossibilities

- It is not possible to implement a Blockchain satisfying Eventual Consistency if an update message is lost;
- It is not possible to implement a Blockchain satisfying Strong Consistency if a fork occurs;
  - $\Theta_{F,k=1}$ is necessary;

Impossibilities

- It is not possible to implement a Blockchain satisfying Eventual Consistency if an update message is lost;
- It is not possible to implement a Blockchain satisfying Strong Consistency if a fork occurs;
  - $\Theta_{F,k=1}$ is necessary;
    - Consensus is necessary;

The best we can have in presence of Forks is Eventual Consistency.

Mapping with existing solutions

References	Refinement
Bitcoin	$\mathfrak{R}(BT\text{-}ADT_{EC}, \Theta_P)$
Ethereum	$\mathfrak{R}(BT\text{-}ADT_{EC}, \Theta_P)$
Algorand	$\mathfrak{R}(BT\text{-}ADT_{SC}, \Theta_{F,k=1})$
ByzCoin	$\mathfrak{R}(BT\text{-}ADT_{SC}, \Theta_{F,k=1})$
PeerCensus	$\mathfrak{R}(BT\text{-}ADT_{SC}, \Theta_{F,k=1})$
Redbelly	$\mathfrak{R}(BT\text{-}ADT_{SC}, \Theta_{F,k=1})$
Hyperledger	$\mathfrak{R}(BT\text{-}ADT_{SC}, \Theta_{F,k=1})$
Tendermint	$\mathfrak{R}(BT\text{-}ADT_{SC}, \Theta_{F,k=1})$

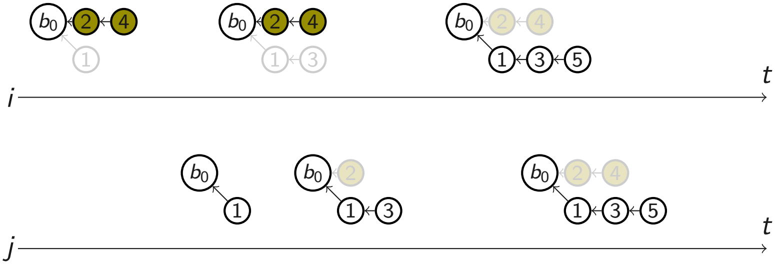
Conclusions and Future Work

- we presented a formal specification for characterizing blockchains;
- and derived conclusion on their implementability in a distributed system.

Future works.

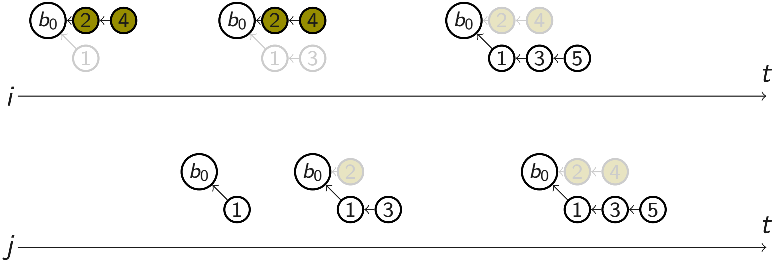
- solvability of Strong and Eventual Prefix in message-passing system;
- fairness properties for oracles;
- . . .

Validity Property



Validity property: all the block read are valid (w.r.t. the application level) and have been appended by some process.

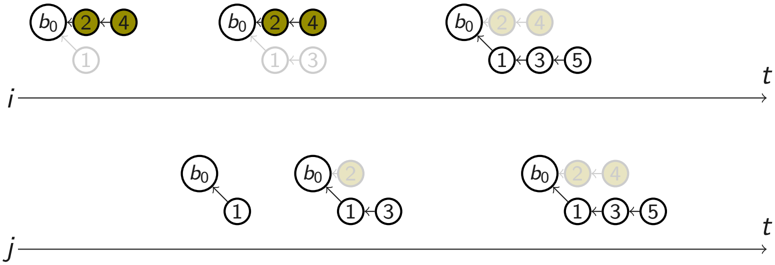
Local Monotonic Read Property



Local monotonic read property: the **score** of the sequence of blockchains read at the same peer never decreases.

score: it can be the length, the weight, etc.., it is a general way to measure and compare blockchains.

Ever Growing Tree Property



Ever growing tree property: the score of returned blockchains eventually grows.

Commissariat à l'énergie atomique et aux énergies alternatives
CEA Tech List
Centre de Saclay — 91191 Gif-sur-Yvette Cedex
www-list.cea.fr

Etablissement public à caractère industriel et commercial — RCS Paris B 775 685 019