TOWARDS A MECHANIZATION OF FRAUD PROOF GAMES IN LEAN

[MARTÍN CERESA] AND CÉSAR SÁNCHEZ

INTRODUCTION

WHAT

A simple **Lean** mechanization of arbitration games.

WHY

L2 Blockchains use *Optimistic Rollups* for **scalability.** Arbitration games ensure correctness via *fraud proofs*.

HOW

Claims are defended and revoked through **authenticated data-structures**.

CONTRIBUTIONS

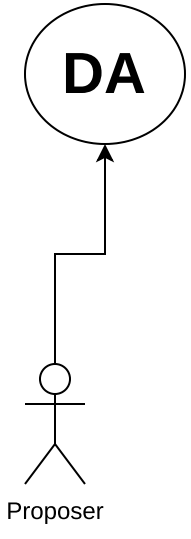
- Proved honest players always win
  - Proposers can defend their claims
  - Challengers can debunk false claims
- Building blocks for L2 Schemes
- Proved Strategies as Verified Oracles

OPTIMISTIC ROLLUPS

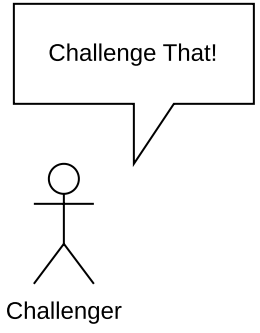
- Instead of computing everything, results are assumed correct **unless** someone challenge them.
- Challenges lead to **arbitration games** between proposers and challengers.
- Arbitration games are deterrents, **not commonly executed**.
- Disputable Assertions (DAs) are registered in the blockchain.
- DAs are $\langle f(x) = y, \textbf{hash}(f(x) = y) \rangle$

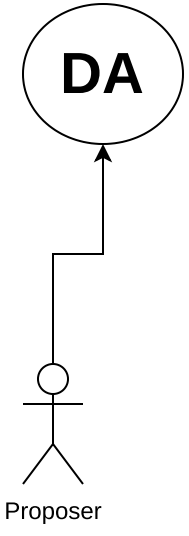
ARBITRATION GAMES

ARBITRATION GAMES

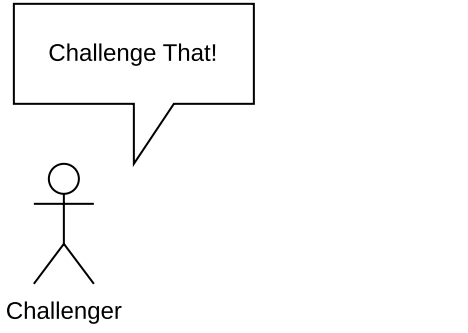


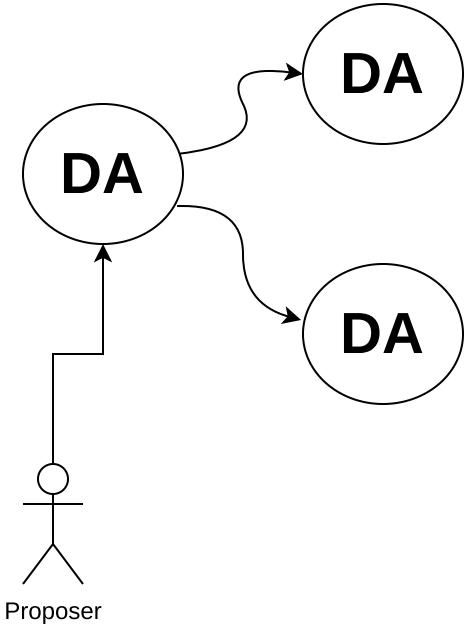
ARBITRATION GAMES



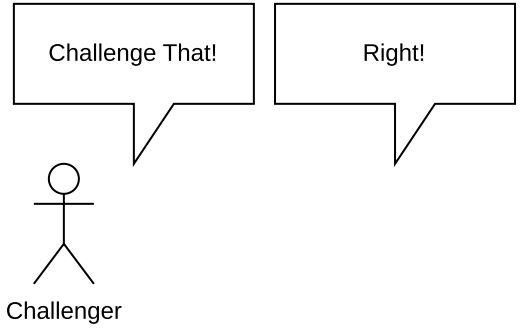


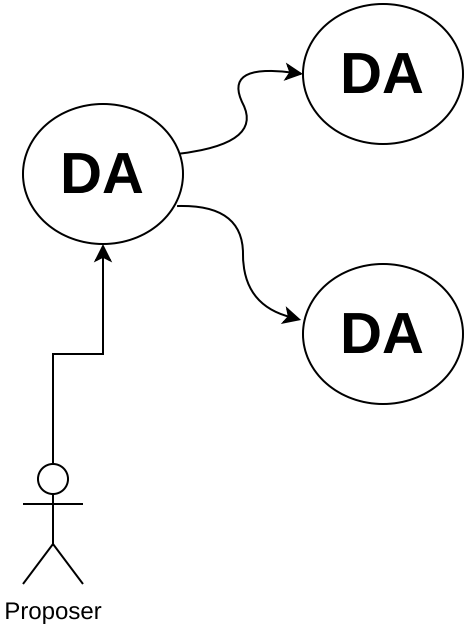
ARBITRATION GAMES



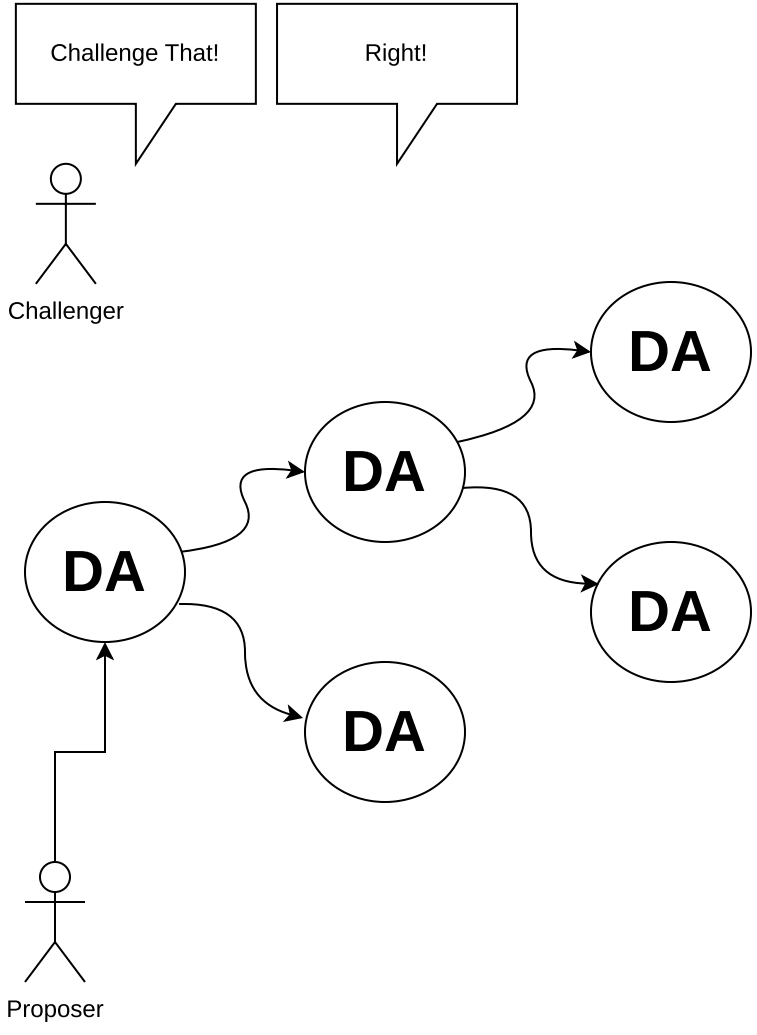


ARBITRATION GAMES

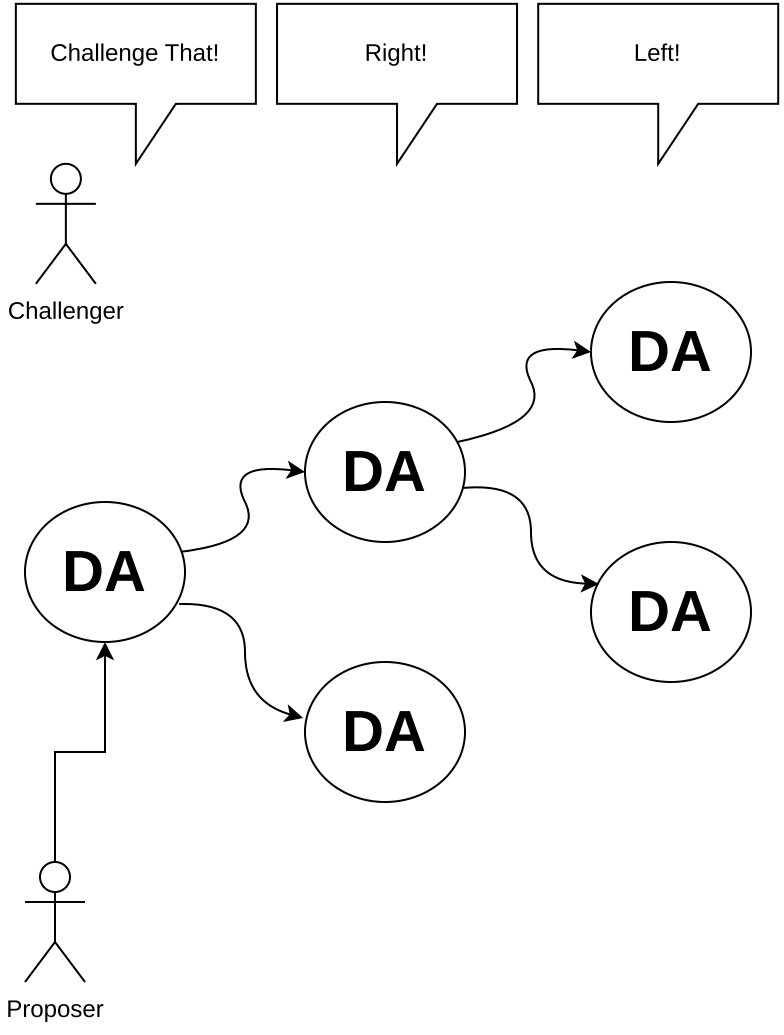




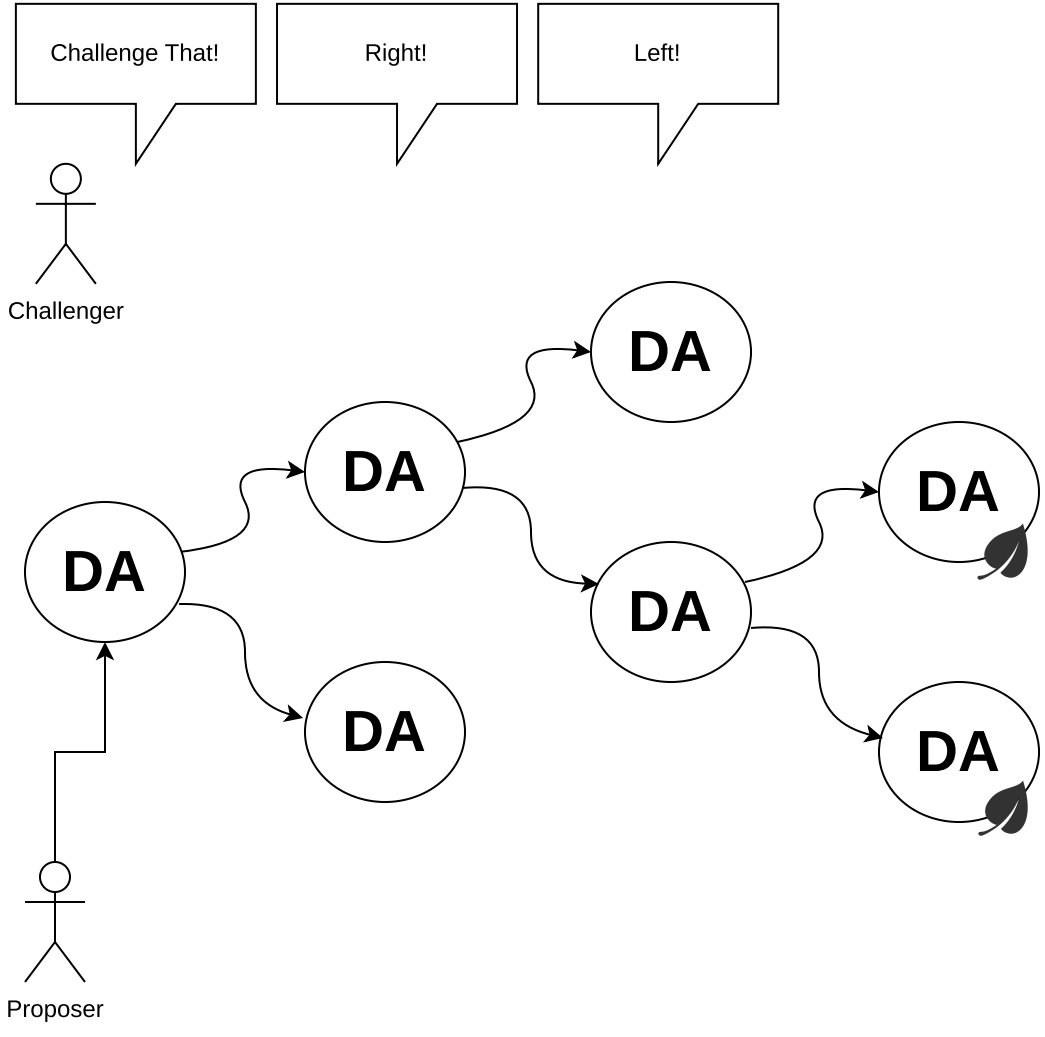
ARBITRATION GAMES



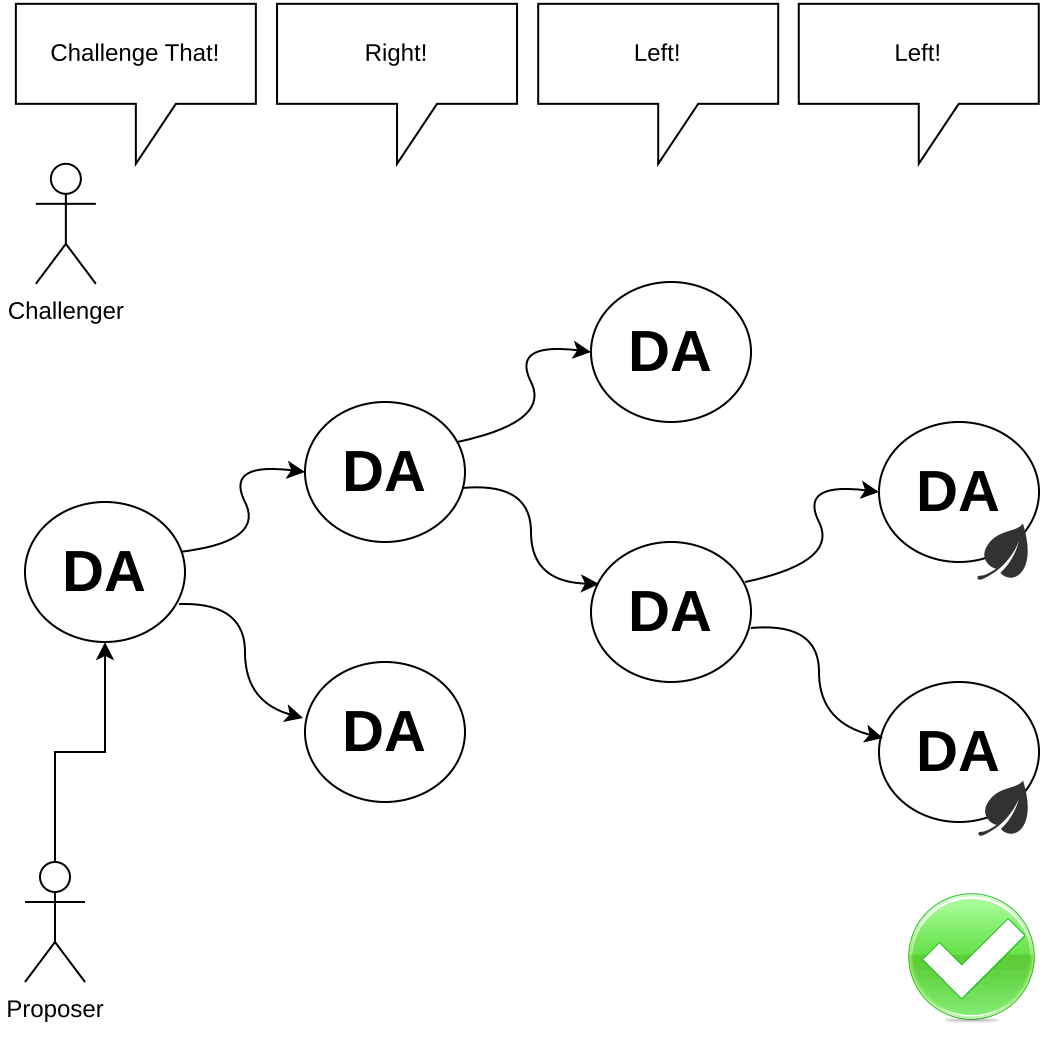
ARBITRATION GAMES



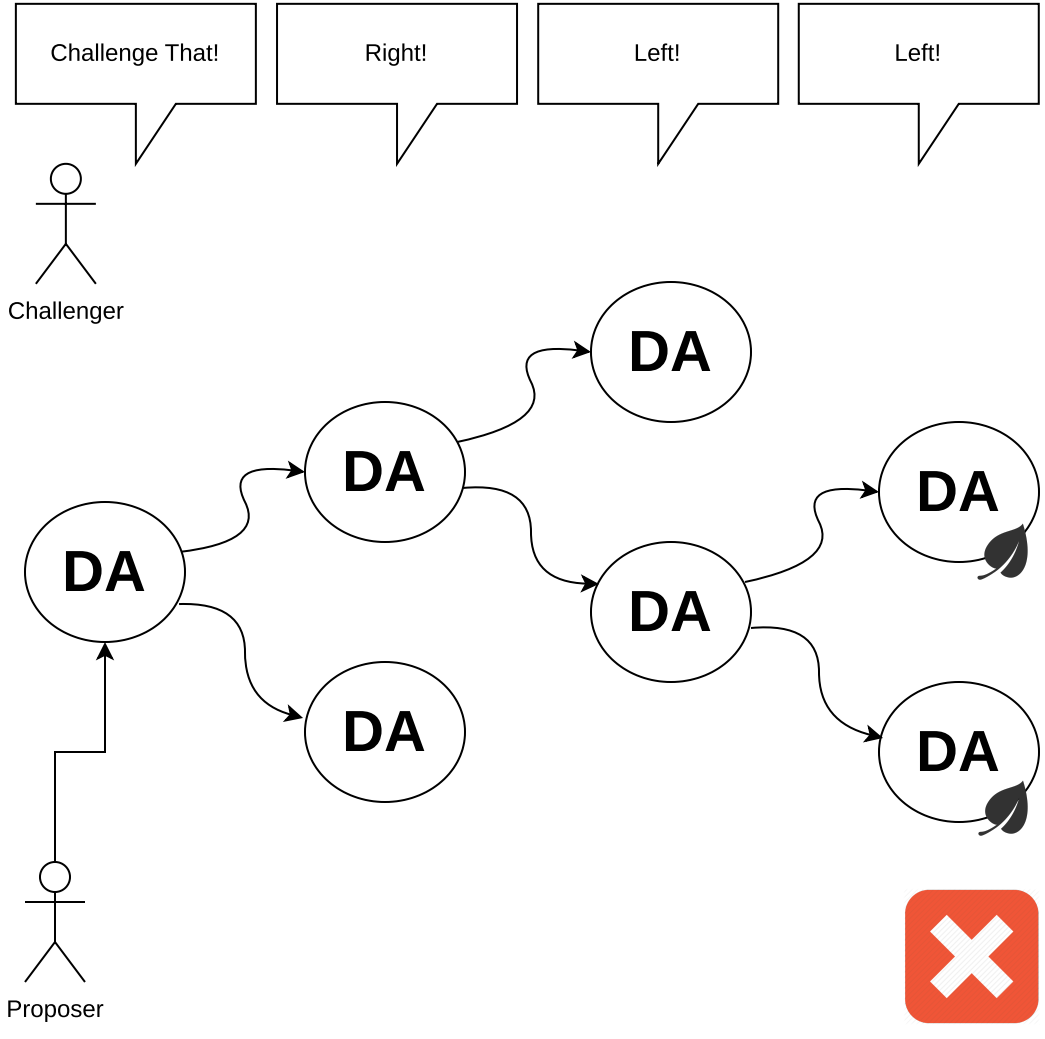
ARBITRATION GAMES



ARBITRATION GAMES



ARBITRATION GAMES



ARBITRATION GAMES

- Turn-based games: one player reveals, the other explores.
- Proposers defend their claim, challengers build fraud-proofs.
- Games are played over Merkle trees.
- All verification steps are done by a L1 smart contract.
- Two kind of frauds:
  - Incorrect hash (structure)
  - Incorrect data (membership)

FORMALIZATION

DISPUTABLE ASSERTIONS

DAs are partial `TraceTree` with their hashes.

```
structure TraceTree (α β γ : Type) where
  data : BinaryTree α β
  res : γ

-- def implicit_assumption
--     (comp : TraceTree α β γ)
--     (leaf_interpretation : α -> γ)
--     (node_intrepretation : β -> γ -> γ -> γ) : Prop
--     := fold leaf_interpretation node_interpretation comp.data = comp.res
```

DISPUTABLE ASSERTIONS

DAs are partial `TraceTree` with their hashes.

```
structure TraceTree (α β γ : Type) where
  data : BinaryTree α β
  res : γ

-- def implicit_assumption
--     (comp : TraceTree α β γ)
--     (leaf_interpretation : α -> γ)
--     (node_intrepretation : β -> γ -> γ -> γ) : Prop
--     := fold leaf_interpretation node_interpretation comp.data = comp.res
```

DISPUTABLE ASSERTIONS

DAs are partial `TraceTree` with their hashes.

```
structure TraceTree (α β γ : Type) where
  data : BinaryTree α β
  res : γ

-- def implicit_assumption
--     (comp : TraceTree α β γ)
--     (leaf_interpretation : α -> γ)
--     (node_intrepretation : β -> γ -> γ -> γ) : Prop
--     := fold leaf_interpretation node_interpretation comp.data = comp.res
```

DISPUTABLE ASSERTIONS

DAs are partial `TraceTree` with their hashes.

```
structure TraceTree (α β γ : Type) where
  data : BinaryTree α β
  res : γ

-- def implicit_assumption
--     (comp : TraceTree α β γ)
--     (leaf_interpretation : α -> γ)
--     (node_intrepretation : β -> γ -> γ -> γ) : Prop
--     := fold leaf_interpretation node_interpretation comp.data = comp.res
```

GAMES : ARBITRATION GAME

```
 1 inductive ChooserMoves where | Now | ContLeft | ContRight
 2
 3 def treeCompArbGame
 4     -- Public Information
 5     (da : TraceTree α β γ)
 6     -- Game Mechanics
 7     (leafCondition : α -> α' -> γ -> Winner)
 8     (midCondition  : β -> γ -> γ -> γ -> Winner)
 9     -- Players
10     (revealer : BinaryTree (Option α') (Option (γ × γ)))
11     (chooser : BinaryTree Unit ((β × γ × γ × γ) -> Option ChooserMoves))
12     : Winner := match da.data, revealer with ...
```

GAMES : ARBITRATION GAME

```
 1 inductive ChooserMoves where | Now | ContLeft | ContRight
 2
 3 def treeCompArbGame
 4     -- Public Information
 5     (da : TraceTree α β γ)
 6     -- Game Mechanics
 7     (leafCondition : α -> α' -> γ -> Winner)
 8     (midCondition  : β -> γ -> γ -> γ -> Winner)
 9     -- Players
10     (revealer : BinaryTree (Option α') (Option (γ × γ)))
11     (chooser : BinaryTree Unit ((β × γ × γ × γ) -> Option ChooserMoves))
12     : Winner := match da.data, revealer with ...
```

GAMES : ARBITRATION GAME

```
 1 inductive ChooserMoves where | Now | ContLeft | ContRight
 2
 3 def treeCompArbGame
 4     -- Public Information
 5     (da : TraceTree α β γ)
 6     -- Game Mechanics
 7     (leafCondition : α -> α' -> γ -> Winner)
 8     (midCondition  : β -> γ -> γ -> γ -> Winner)
 9     -- Players
10     (revealer : BinaryTree (Option α') (Option (γ × γ)))
11     (chooser : BinaryTree Unit ((β × γ × γ × γ) -> Option ChooserMoves))
12     : Winner := match da.data, revealer with ...
```

GAMES : ARBITRATION GAME

```
inductive ChooserMoves where | Now | ContLeft | ContRight

def treeCompArbGame
    -- Public Information
    (da : TraceTree α β γ)
    -- Game Mechanics
    (leafCondition : α -> α' -> γ -> Winner)
    (midCondition  : β -> γ -> γ -> γ -> Winner)
    -- Players
    (revealer : BinaryTree (Option α') (Option (γ × γ)))
    (chooser : BinaryTree Unit ((β × γ × γ × γ) -> Option ChooserMoves))
    : Winner := match da.data, revealer with ...
```

GAMES : ARBITRATION GAME

```
 1 inductive ChooserMoves where | Now | ContLeft | ContRight
 2
 3 def treeCompArbGame
 4     -- Public Information
 5     (da : TraceTree α β γ)
 6     -- Game Mechanics
 7     (leafCondition : α -> α' -> γ -> Winner)
 8     (midCondition  : β -> γ -> γ -> γ -> Winner)
 9     -- Players
10     (revealer : BinaryTree (Option α') (Option (γ × γ)))
11     (chooser : BinaryTree Unit ((β × γ × γ × γ) -> Option ChooserMoves))
12     : Winner := match da.data, revealer with ...
```

GAMES : MEMBERSHIP GAME

```
 1 inductive Side : Type where | Left | Right
 2
 3 structure ElemInMTree (α ℍ : Type) where
 4   elem : α
 5   path : List Side
 6   mtree : ℍ
 7
 8 inductive ChooserSmp : Type where| Now | Continue
 9
10 def arbElem
11     (da : ElemInMTree α ℍ)
12     (proposer : Side List -> Option (ℍ × ℍ))
13     (chooser : Side List -> (ℍ × ℍ -> Option ChooserSmp))
14     : Winner
```

GAMES : MEMBERSHIP GAME

```
 1 inductive Side : Type where | Left | Right
 2
 3 structure ElemInMTree (α ℍ : Type) where
 4   elem : α
 5   path : List Side
 6   mtree : ℍ
 7
 8 inductive ChooserSmp : Type where| Now | Continue
 9
10 def arbElem
11     (da : ElemInMTree α ℍ)
12     (proposer : Side List -> Option (ℍ × ℍ))
13     (chooser : Side List -> (ℍ × ℍ -> Option ChooserSmp))
14     : Winner
```

GAMES : MEMBERSHIP GAME

```
 1 inductive Side : Type where | Left | Right
 2
 3 structure ElemInMTree (α ℍ : Type) where
 4   elem : α
 5   path : List Side
 6   mtree : ℍ
 7
 8 inductive ChooserSmp : Type where| Now | Continue
 9
10 def arbElem
11     (da : ElemInMTree α ℍ)
12     (proposer : Side List -> Option (ℍ × ℍ))
13     (chooser : Side List -> (ℍ × ℍ -> Option ChooserSmp))
14     : Winner
```

GAMES : MEMBERSHIP GAME

- Variants
  - Linear path game (bottom-up & top-down)
  - Logarithmic (bisection) game (defined using `treeCompArbGame`)
- Different variants are equivalent

DECOMPOSING L2 SCHEMES

- Instead of arbitrating over traces of programs, we can arbitrate over properties of specific algorithms.
- Optimistic Rollups =
  - Distributed Sequencer
  - Data Availability Committee
  - State Transition Function

VALID BATCH DEFINITION

Validity

Every transaction request in b is a valid transaction request added by a client.

No Duplicates

No transaction request appears twice in b.

Integrity

No transaction request in b appears in a legal batch tag previously posted by the sequencer.

VALID BATCH DEFINITION

Validity

Every transaction request in b is a valid transaction request added by a client.

No Duplicates

No transaction request appears twice in b.

~~**Integrity**~~

Correct DA

Merkle tree is correct.

VALID BATCH DEFINITION

```
def local_valid {α ℍ : Type}
  (da : BinaryTree α Unit × ℍ)(validity_pred : α -> Bool) : Prop
 -- Merkle Tree is correct
 := da.fst.hash_BTree = da.snd
 -- All elements are |validity_pred| valid
 ∧ (da.fst.fold validity_pred and)
 -- There are no duplicated elements.
 ∧ List.Nodup da.fst.toList
```

PLAYER ACTIONS

- Player 1 : Proposes DAs (Valid or not) : Data and Hash
- Player 2 : Challenge those claims or not:
  - Data does not match hash
  - There is an invalid element
  - There are duplicated elements
  - Valid batch

ONE HONEST CHOOSER PREVENTS INVALID BLOCKS

```
1 theorem honest_chooser_valid
2     [Hash α 𝕎][HashMagma 𝕎][InjectiveHash α 𝕎][InjectiveMagma 𝕎]
3     (validity_pred : α -> Bool) (p1 : P1_Actions α 𝕎)
4     : linear_l2_protocol validity_pred p1 (honest_chooser validity_pred)
5     ↔ local_valid p1.da validity_pred
```

ONE HONEST CHOOSER PREVENTS INVALID BLOCKS

```
1 theorem honest_chooser_valid
2     [Hash α ℍ][HashMagma ℍ][InjectiveHash α ℍ][InjectiveMagma ℍ]
3     (validity_pred : α -> Bool) (p1 : P1_Actions α ℍ)
4     : linear_l2_protocol validity_pred p1 (honest_chooser validity_pred)
5     ↔ local_valid p1.da validity_pred
```

ONE HONEST CHOOSER PREVENTS INVALID BLOCKS

```
1 theorem honest_chooser_valid
2     [Hash α 𝕎][HashMagma 𝕎][InjectiveHash α 𝕎][InjectiveMagma 𝕎]
3     (validity_pred : α -> Bool) (p1 : P1_Actions α 𝕎)
4     : linear_l2_protocol validity_pred p1 (honest_chooser validity_pred)
5     ↔ local_valid p1.da validity_pred
```

CONCLUSIONS

- Formalized Arbitration games
- Defined DA, players, honest players
- Membership games: linear (bottom-up and top-down) and logarithmic.
- Formalized a simpler version of Optimistic Rollups.
- Strategies are executable

FUTURE WORK

Domain Specific Layer-2 Framework

Can we designed a language to decompose Layer-2 protocols into simple games?

Add Time to the model

Time is an attack vector (delay attacks.)

Layer-1 Limitation

Small step verification not fitting in L1-transactions.

Incentives

Why players behave the way they do?

Multiplayer Games

Agents and multiplayer games, tournaments.