FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

Søren Eller Thomsen, Aarhus University
Bas Spitters, Aarhus University
FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

Søren Eller Thomsen, Aarhus University
Bas Spitters, Aarhus University
The diagram illustrates a network with nodes B22, B41, and A12 connected to a central node GB. Alice and Bob are depicted on the left and right sides of the diagram, respectively, with connections to the nodes. The diagram also includes a speech bubble with the text B22, B41, A12.
Alice

Bob

B22

B41

A12

R10

GB

B22

B41

A12

R10

GB

B22

B41

A12

R10

B22, B41, A12, R10
FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

Søren Eller Thomsen, Aarhus University
Bas Spitters, Aarhus University
FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

Søren Eller Thomsen, Aarhus University
Bas Spitters, Aarhus University
PROOF OF STAKE

• Number of tickets are proportional to the amount of stake each player owns.

• A winning ticket can (but shouldn’t!) be used to create multiple different blocks.
FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

Søren Eller Thomsen, Aarhus University
Bas Spitters, Aarhus University
“FORMALIZING” (AKA. CONTRIBUTIONS)

1. We define formal semantics of executions of an abstract PoS NSB in Coq.

2. We give the first mechanized proof of the core combinatorics of this protocol. Specifically we prove:
   a) Chain Growth.
   b) Chain Quality.
   c) Common Prefix (n > 3t).

3. We develop a new methodology for verifying protocols by their abstract functional interfaces.
1. We define formal semantics of executions of an abstract PoS NSB in Coq.

2. We give the first mechanized proof of the core combinatorics of this protocol. Specifically we prove:
   a) Chain Growth.
   b) Chain Quality.
   c) Common Prefix (n > 3t).

3. We develop a new methodology for verifying protocols by their abstract functional interfaces.
MODELLING - OVERVIEW

HONEST PARTIES

• Local State
• Delivery
• Baking

GLOBAL STATE

• Set of parties and states
• State for adversary
• State for network

NETWORK

• Functions on a global state

ADVERSARY

• Opaque adversarial stateful function
Record LocalState :=
mkLocalState
  { tT : treeType
  ; pk : Party
  ; tree : tT }.

Definition honest_bake : Slot -> Transactions -> State LocalState Messages := …

Definition honest_rcv : Slot -> Messages -> State LocalState unit := …
**MODELLING - HONEST PARTIES**

The state monad.

**Definition** honest_bake : Slot -> Transactions -> State LocalState Messages := ...

**Definition** honest_rcv : Slot -> Messages -> State LocalState unit := ...

**Record** LocalState :=

mkLocalState

{ tT : treeType
; pk : Party
; tree : tT }.

**Record** mixin_of T := Mixin

{ extendTree : T -> Block -> T
; bestChain : Slot -> T -> Chain
; allBlocks: T -> BlockPool
; tree0 : T

; : allBlocks tree0 =i [:: GenesisBlock]
; : forall t b, allBlocks (extendTree t b) =i allBlocks t ++ [:: b]
; : forall t s, valid_chain (bestChain s t)
; : forall c s t, valid_chain c -> {subset c <= [seq b <- allBlocks t | sl b <= s]} -> |c| <= bestChain s t
; : forall s t, {subset (bestChain s t) <= [seq b <- allBlocks t | sl b <= s]}).
REACHABLE WORLDS

Init → Ready → Receive → Delivered → Bake → Baked → Inc Round
**Theorem chain_growth:**

\[
\text{forall } w \ N1 \ N2, \\
N0 \downarrow N1 \rightarrow N1 \downarrow N2 \rightarrow \\
w \leq |\text{lucky_slots_worlds } N1 \ N2| \rightarrow \\
|honest_tree_chain N1| + w \leq |honest_tree_chain N2|. 
\]

**Theorem chain_quality:**

\[
\text{forall } N \ p \ l \ b_j \ b_i \ c \ w, \\
\text{let } bc := \text{bestChain } (t\_now N) \ (\text{tree } l) \text{ in} \\
\text{let } f := [:: b_j] ++ c ++ [:: b_i] \text{ in} \\
N0 \downarrow N \rightarrow \\
forging_free N \rightarrow \\
collision_free N \rightarrow \\
\text{has_state } p \ N l \rightarrow \text{is_honest } p \rightarrow \\
\text{prune_time } k \ bc1 \preceq bc2 \lor \\
\exists t1 \ t2, [/\ t1 \leq k \\
, t\_now N1 \leq t2 \leq t\_now N2 \\
\& \ |\text{super_slots_range } t1 \ t2| \\
\leq 2 \times |\text{adv_slots_range } t1 \ t2|]. 
\]
**THEOREMS**

**Theorem chain_quality:**
forall N p l b_j b_i c w, 
let bc := bestChain (t_now N) (tree l) in  
let f := [:: b_j] ++ c ++ [:: b_i] in 
N0 ⇓ N -> 
forging_free N -> 
collision_free N -> 
has_state p N l -> is_honest p -> 
fragment f bc -> 
honest_advantage_ranges_gt w (sl b_j - sl b_i) -> 
w <= |honest_blocks f|.

**Theorem chain_growth:**
forall w N1 N2, 
N0 ⇓ N1 -> N1 ⇓ N2 -> 
w <= |lucky_slots_worlds N1 N2| -> 
|honest_tree_chain N1| + w <= |honest_tree_chain N2|.

**Theorem common_prefix:**
forall k N1 N2, 
N0 ⇓ N1 -> N1 ⇓ N2 -> 
forging_free N2 -> 
collision_free N2 -> 
forall p1 p2 l1 l2, 
let bc1 := bestChain (t_now N1) (tree l1) in 
let bc2 := bestChain (t_now N2) (tree l2) in 
is_honest p1 -> is_honest p2 -> 
has_state p1 N1 l1 -> has_state p2 N2 l2 -> 
prune_time k bc1 ≤ bc2 /
exists t1 t2, [\ t1 ≤ k , t_now N1 ≤ t2 <= t_now N2 
& |super_slots_range t1 t2| <= 2 * | adv_slots_range t1 t2| ].
THEOREMS

Theorem chain_growth :
  forall w N1 N2,
  N0 ⊨ N1 -> N1 ⊨ N2 ->
  w <= |lucky_slots_worlds N1 N2| ->
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.

Theorem chain_quality :
  forall N p l b_j b_i c w,
  let bc := bestChain (t_now N) (tree l) in
  let f := :: b_j ++ c ++ :: b_i in
  N0 ⊨ N ->
  forging_free N ->
  collision_free N ->
  has_state p N l -> is_honest p ->
  fragment f bc ->
  honest_advantage_ranges_gt w (sl b_j - sl b_i) ->
  w <= |honest_blocks f|.

Theorem common_prefix :
  forall k N1 N2,
  N0 ⊨ N1 ->
  N1 ⊨ N2 ->
  forging_free N2 ->
  collision_free N2 ->
  forall p1 p2 l1 l2,
  let bc1 := bestChain (t_now N1) (tree l1) in
  let bc2 := bestChain (t_now N2) (tree l2) in
  is_honest p1 -> is_honest p2 ->
  has_state p1 N1 l1 -> has_state p2 N2 l2 ->
  prune_time k bc1 ≤ bc2 \\
  exists t1 t2, [/\ t1 ≤ k ,
    t_now N1 <= t2 <= t_now N2 
    & |super_slots_range t1 t2| <= 2 * | adv_slots_range t1 t2| ].
Theorem chain_quality :
  forall N p l b_j b_i c w, 
  let bc := bestChain (t_now N) (tree l) in  
  let f := [:: b_j] ++ c ++ [:: b_i] in
  N0 ⇓ N -> 
  forging_free N -> 
  collision_free N -> 
  has_state p N l -> is_honest p -> 
  fragment f bc -> 
  honest_advantage_ranges_gt w (sl b_j - sl b_i) -> 
  w <= |honest_blocks f|.

Theorem chain_growth :
  forall w N1 N2, 
  N0 ⇓ N1 -> N1 ⇓ N2 -> 
  w <= |lucky_slots_worlds N1 N2| -> 
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.

Theorem common_prefix :
  forall k N1 N2, 
  N0 ⇓ N1 -> 
  N1 ⇓ N2 -> 
  forging_free N2 -> 
  collision_free N2 -> 
  forall p1 p2 l1 l2, 
  let bc1 := bestChain (t_now N1) (tree l1) in 
  let bc2 := bestChain (t_now N2) (tree l2) in 
  is_honest p1 -> is_honest p2 -> 
  has_state p1 N1 l1 -> has_state p2 N2 l2 -> 
  prune_time k bc1 ≤ bc2 /
  exists t1 t2, [\ t1 <= k 
  , t_now N1 <= t2 <= t_now N2 
     & |super_slots_range t1 t2| <= 2 * | adv_slots_range t1 t2| ].
Theorem chain_quality : 
  forall N p l b_j b_i c w, 
  let bc := bestChain (t_now N) (tree l) in 
  let f := [:: b_j] ++ c ++ [:: b_i] in 
  N0 ↪ N -> 
  forging_free N -> 
  collision_free N -> 
  has_state p N l -> is_honest p -> 
  fragment f bc -> 
  honest_advantage_ranges_gt w (sl b_j - sl b_i) -> 
  w <= |honest_blocks f|.

Theorem chain_growth : 
  forall w N1 N2, 
  N0 ↪ N1 -> 
  N1 ↪ N2 -> 
  w <= |lucky_slots_worlds N1 N2| -> 
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.

Theorem common_prefix : 
  forall k N1 N2, 
  N0 ↪ N1 -> 
  N1 ↪ N2 -> 
  forging_free N2 -> 
  collision_free N2 -> 
  forall p1 p2 l1 l2, 
  let bc1 := bestChain (t_now N1) (tree l1) in 
  let bc2 := bestChain (t_now N2) (tree l2) in 
  is_honest p1 -> is_honest p2 -> 
  has_state p1 N1 l1 -> has_state p2 N2 l2 -> 
  prune_time k bc1 ≤ bc2 \ /
  exists t1 t2, \ /
  t_now N1 <= t2 <= t_now N2 
  & |super_slots_range t1 t2| <= 2 * |adv_slots_range t1 t2|.

Theorem common_prefix : 
  forall k N1 N2, 
  N0 ↪ N1 -> 
  N1 ↪ N2 -> 
  forging_free N2 -> 
  collision_free N2 -> 
  forall p1 p2 l1 l2, 
  let bc1 := bestChain (t_now N1) (tree l1) in 
  let bc2 := bestChain (t_now N2) (tree l2) in 
  is_honest p1 -> is_honest p2 -> 
  has_state p1 N1 l1 -> has_state p2 N2 l2 -> 
  prune_time k bc1 ≤ bc2 \ /
  exists t1 t2, \ /
  t_now N1 <= t2 <= t_now N2 
  & |super_slots_range t1 t2| <= 2 * |adv_slots_range t1 t2|.

Theorem chain_growth : 
  forall w N1 N2, 
  N0 ↪ N1 -> N1 ↪ N2 -> 
  w <= |lucky_slots_worlds N1 N2| -> 
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.
**THEOREMS**

**Theorem chain_quality:**
\[
\text{forall } N \ p \ l \ b_j b_i c w, \\
\text{let } bc := \text{bestChain } (t_{\text{now }} N) \ (\text{tree } l) \text{ in} \\
\text{let } f := [:: b_j] ++ c ++ [:: b_i] \text{ in} \\
N_0 \Downarrow N \rightarrow \\
\text{forging_free } N \rightarrow \\
\text{collision_free } N \rightarrow \\
\text{has_state } p \ N \ l \rightarrow \text{is_honest } p \rightarrow \\
\text{fragment } f \ bc \rightarrow \\
\text{honest_advantage_ranges_gt } w \ (\text{sl } b_j - \text{sl } b_i) \rightarrow \\
w \leq |\text{honest_blocks } f|.
\]

**Theorem chain_growth:**
\[
\text{forall } w \ N_1 \ N_2, \\
N_0 \Downarrow N_1 \rightarrow N_1 \Downarrow N_2 \rightarrow \\
w \leq |\text{lucky_slots_worlds } N_1 \ N_2| \rightarrow \\
|\text{honest_tree_chain } N_1| + w \leq |\text{honest_tree_chain } N_2|.
\]

**Theorem common_prefix:**
\[
\text{forall } k \ N_1 \ N_2, \\
N_0 \Downarrow N_1 \rightarrow N_1 \Downarrow N_2 \rightarrow \\
forging_free N_2 \rightarrow \\
collision_free N_2 \rightarrow \\
\text{forall } p_1 p_2 l_1 l_2, \\
\text{let } bc_1 := \text{bestChain } (t_{\text{now }} N_1) (\text{tree } l_1) \text{ in} \\
\text{let } bc_2 := \text{bestChain } (t_{\text{now }} N_2) (\text{tree } l_2) \text{ in} \\
\text{is_honest } p_1 \rightarrow \text{is_honest } p_2 \rightarrow \\
\text{has_state } p_1 N_1 \ l_1 \rightarrow \text{has_state } p_2 N_2 \ l_2 \rightarrow \\
\text{prune_time } k \ bc_1 \leq bc_2 \\
\text{exists } t_1 t_2, [\forall t_1 \leq k \\
\text{, } t_{\text{now }} N_1 \leq t_2 \leq t_{\text{now }} N_2 \\
\& |\text{super_slots_range } t_1 t_2| \leq 2 \ast |\text{adv_slots_range } t_1 t_2|].
\]

Condition on abstract lottery!
THEOREMS

Theorem chain_quality :
forall N p l b_j b_i c w,
let bc := bestChain (t_now N) (tree l) in
let f := [:: b_j] ++ c ++ [:: b_i] in
N0 $\Downarrow$ N ->
forging_free N ->
collision_free N ->
has_state p N l -> is_honest p ->
fragment f bc ->
honest_advantage_ranges_gt w (sl b_j - sl b_i) ->
w <= |honest_blocks $\uparrow$|.

Theorem chain_growth :
forall w N1 N2,
N0 $\Downarrow$ N1 -> N1 $\Downarrow$ N2 ->
w <= |lucky_slots_worlds N1 N2| $\Rightarrow$
|honest_tree_chain N1| + w <= |honest_tree_chain N2|.

Theorem common_prefix :
forall k N1 N2,
N0 $\Downarrow$ N1 ->
N1 $\Downarrow$ N2 ->
forging_free N2 ->
collision_free N2 ->
forall p1 p2 l1 l2,
let bc1 := bestChain (t_now N1) (tree l1) in
let bc2 := bestChain (t_now N2) (tree l2) in
is_honest p1 -> is_honest p2 ->
has_state p1 N1 l1 -> has_state p2 N2 l2 ->
prune_time k bc1 <= bc2 $\lor$
exists t1 t2, [/\ t1 <= k , t_now N1 <= t2 <= t_now N2
& |super_slots_range t1 t2| <= 2 * |adv_slots_range t1 t2|].

Condition on abstract lottery!
CONCLUSION

• We provide a formal model of the execution semantics of a NSB PoS and are the first to prove both safety and liveness for any BFT consensus algorithm.

• Details: https://eprint.iacr.org/2020/917

• Code: https://github.com/AU-COBRA/PoS-NSB

• Contact: sethomsen@cs.au.dk