REGENESIS DEFINITION.

Abstract. We define (i) the merkle tree structure and the merkleization rule which give the merkle root, (ii) regenesis active and inactive states including to merge the active state to update the inactive state and the minimum data needed by a witness with respect to regenesis states. Because regenesis is a generalization of pure statelessness (where regenesis period is one block), this definition also covers full statelessness.

1. Merkle root

We define the merkle root of the state. This is done by defining the tree structure and merkleization rule at each tree node.

Definition. Let set of leaves be $\mathfrak{I}$ as described in the yellowpaper appendix D, but I_0 is indexed by bit, not nibble. The *merkle root* of $\mathfrak{I}$ is $\texttt{TRIE}(\mathfrak{I}) = m(\mathfrak{I}, 0)$, computed recursively with the following function.

$$m(\mathfrak{I}, i) = \begin{cases} 0^{32} & \text{if} \quad |\mathfrak{I}| = 0 \quad \text{where } 0^{32} \text{ is a byte array of 32 zeros} \\ \texttt{HASH}(I_0 \| \texttt{HASH}(I_1)) & \text{if} \quad |\mathfrak{I}| = 1 \quad \text{where } (I_0, I_1) \in \mathfrak{I} \\ \texttt{HASH}(m(\mathfrak{I}_{I_0[i+1]=0}, i+1) \| m(\mathfrak{I}_{I_0[i+1]=1}, i+1)) & \text{otherwise where } (I_0, I_1) \in \mathfrak{I}, \\ & \qquad \mathfrak{I}_{I_0[i+1]=0} = \{(I_0, I_1) \in \mathfrak{I} : I_0[i+1] = 0\}, \text{ and} \\ & \qquad \mathfrak{I}_{I_0[i+1]=1} = \{(I_0, I_1) \in \mathfrak{I} : I_0[i+1] = 1\}. \end{cases}$$

2. Active and Inactive State, and Regnesis State

Definition.

- The *account leaves* are a set σ of pairs $(a, \sigma[a])$, i.e. σ is a function with preimage $a \in \mathbb{B}_{20}$ as the addresses and image $\sigma[a]$ is an account. Recall that each account contains a nonce, balance, code, and storage.
- The *storage leaves* for an account with address a is set $\sigma[a]_s$ of pairs $(k, \sigma[a]_s[k])$, with key $k \in \mathbb{B}_{32}$ and value $\sigma[a]_s[k] \in \mathbb{B}_{32}$.
- The *inactive ("snapshot") state* σ_m (shorthand for $\sigma_{0,m}$) is the set of all addresses and account pairs at the end of block m, with each account containing its set of storage leaves which are key-value pairs.
- The *active state* $\sigma_{m,n}$ after block n relative to σ_m, where $m \leq n$, is the set of accounts, along with storage for each account, at the end of block B_n, which were "touched" since the end of block B_m and which are not currently "removed".
- The *removed state* $\bar{\sigma}_{m,n}$ is the set of accounts in σ_m which were "removed" from state since block B_m and are not currently in $\sigma_{m,n}$.
- The *regenesis state* at block n relative to σ_m is the pair $(\sigma_{m,n}, \bar{\sigma}_{m,n})$ of active and removed sets.

3. Operations on Active and Inactive State, Including Merging

Definition. First we define some operations over sets of accounts. These operations resemble traditional set union $\cup$, set intersection $\cap$, and set complement $\setminus$, but are specially defined to do important operations on our active and inactive state. Let A be a set of pairs $(a, b) \in A$. We define notation for the set of the first elements.

$$A^0 = \{a | (a, b) \in A\}$$

Let A and B be sets of address-account pairs like σ_m or $\sigma_{m,n}$ above. Define $\dot\cap$ as follows.

$$\begin{aligned} A \dot\cap B = \{\ (a, C[a]) \quad | \quad & a \in A^0 \text{ and } a \in B^0 \text{ and} \\ & C = B \text{ except } \forall a \in A^0\ C[a]_s = \{(k, v) | \quad ((k, v) \in A[a]_s \text{ and } k \notin B[a]^0_s) \text{ or} \\ & \qquad\qquad\qquad\qquad\qquad\qquad\quad ((k, v) \in B[a]_s \text{ and } k \notin A[a]^0_s) \text{ or} \\ & \qquad\qquad\qquad\qquad\qquad\qquad\quad ((k, v) \in B[a]_s \text{ and } k \in A[a]^0_s)\}\} \end{aligned}$$

Next define $\dot\cup$.

$$\begin{aligned} A \dot\cup B = \ (A \dot\cap B)\ \cup \quad \{\ (a, b) \quad | \quad & ((a, b) \in A \text{ and } a \notin B^0) \text{ or} \\ & ((a, b) \in B \text{ and } a \notin A^0)\} \end{aligned}$$

Next define $\dot\setminus$.

$$\begin{aligned} A \dot\setminus B = \quad & \{(a, b) | (a, b') \in A \text{ and } (a, b'') \in B \text{ and } |b''| > 0 \text{ and } b = b' \text{ except } b_s = \{(k, v) | (k, v) \in b'_s \text{ and } k \notin b''^0_s\}\} \\ \cup \quad & \{(a, b) | (a, b) \in A \text{ and } a \notin B^0\} \end{aligned}$$

Finally, $A \dot\subseteq B$ means that if $a \in A^0$ then $a \in B^0$ and if $k \in A[a]^0_s$ then $k \in B[a]^0_s$.

Remark.

- Intuitively, $\dot\cap$ is the intersection of account addresses, with the accounts values in B overwriting those in A, except the storage values which are a union of the storage values in A and B with values in B overwriting A if there is a shared key.
- Intuitively, $\dot\cup$ is the union of account addresses and storage, with all values of B overwriting those of A if there is a collision.
- Intuitively, $\dot\setminus$ removes all accounts from A which are in B except for accounts in B which have non-empty storage, then remove just the storage items from corresponding accounts in A.
- Intuitively, $\dot\subseteq$ means that all all account addresses in A are also in B.

Definition. The *state merge* of active state with inactive state is the operation $(\sigma_m \dot\setminus \bar\sigma_{m,n}) \dot\cup \sigma_{m,n}$.

Remark.

- For a regensis state merge, the order of applying $\dot\setminus$ and $\dot\cup$ to σ_m is arbitrary. But it may be more convenient to apply $\dot\setminus$ first.
- $\sigma_n = (\sigma_m \dot\setminus \bar\sigma_{m,n}) \dot\cup \sigma_{m,n}$.
- $\sigma_{m,n} \dot\cap \bar\sigma_{m,n} = \emptyset$.
- $\bar\sigma_{m,n} \dot\subseteq \sigma_m$.
- The above four need proof.

Given active state $\sigma_{m,n}$, removed state $\bar{\sigma}_{m,n}$ and all necessary recent uncles and recent block hashes needed to execute the block, we want the witness data needed to execute the block B_{n+1}.

Definition.

- The *missing leaves set* $T_{m,n}$ is the set of accounts and their storage items which are accessed to execute and finalize block B_{n+1}, but which are missing from $\sigma_{m,n}$. Note that $T_{m,n}$ includes empty accounts or storage leaves which were accessed, as well as leaves which were inserted. Also note that each touched account's storage $T_{m,n}[a]_s$ is the set of missing storage leaves touched in this block, and when a storage value is touched, its account is also touched even if it is already in $\sigma_{m,n}$.
- The *memoized hashes set* $H_{m,n}$ is a set of pairs $(H_{\text{path}}, H_{\text{hash}})$, with tree paths $H_{\text{path}} \in \mathbf{b}_{\leq 512}$ (where $\mathbf{b}_{\leq 512}$ is the set of bit-arrays of length at most 512 bits) and merkle hashes $H_{\text{hash}} \in \mathbb{B}_{32}$. If H_{path} has more than 256 bits, then the first 256 bits is the tree path to the account, and the remaining bits are the path to the storage merkle hash.
- The *witness data* for block B_{n+1} with respect to active state $\sigma_{m,n}$ is the pair ($T_{m,n}$,$H_{m,n}$) of missing leaves and memoized hashes, which we compute with the algorithm `WITNESSDATA` below.

ALGORITHM `WITNESSDATA`
INPUTS: $\sigma_m, \sigma_n, \sigma_{m,n}, \bar{\sigma}_{m,n}, B_{n+1}$
OUTPUTS: $T_{m,n}, H_{m,n}$
1. Execute block B_{n+1} by calling the block-level state-transition function $\Pi(\sigma_n, B_{n+1})$, which returns not just the updated state σ_{n+1}, but also the set $T_{n,n+1}$ of touched accounts, where, for each address a of a touched account, $T_{n,n+1}[a]_s$ is the set of touched storage items.
2. The set $T_{m,n+1}$ is $T_{n,n+1} \dot{\setminus} \sigma_{m,n}$.
3. Next, we accumulate memoized hashes $H_{m,n}$. For each account $(a, \sigma_n[a]) \in T_{m,n}$ and for each storage item in each account, we call function $m(\mathfrak{I}, i)$ for specific arguments to accumulate our memoized hash set $H_{m,n}$. First define the function we do it with.
[WARNING: $w()$ IS UNDER CONSTRUCTION]

$$w(\mathfrak{I}, \mathfrak{I}', \mathfrak{I}'', i) = \begin{cases} \emptyset & \text{if} \quad |\mathfrak{I}''| = 0 \\ w(\mathfrak{I}, b, c, j) & \text{else} \quad \text{if} \quad |\mathfrak{I}'| > 0 \\ & \quad \text{where} \quad j = \max(l : \exists \mathbf{I}\ \forall (I_0, I_1) \in \mathfrak{I}\ I_0[0..l] = \mathbf{I}), \\ & \quad\quad \mathfrak{I}_{I_0[j]=0} = \{(I_0, I_1) \in \mathfrak{I} : I_0[j] = 0\}, \\ & \quad\quad \mathfrak{I}_{I_0[j]=1} = \{(I_0, I_1) \in \mathfrak{I} : I_0[j] = 1\}, \text{ and} \\ \{(\mathfrak{I}^0[0..i], m(s, i))\} \cup w(\mathfrak{I}, b, c, k) & \text{else if} \quad |\mathfrak{I}'| = 0 \end{cases}$$

3.1. Set $H_{m,n} = w(\sigma_m, \sigma_{m,n}, T_{m,n}, 0)$.
3.2. For each address $a \in T^0_{m,n}$, get set $H_a = w(\sigma_m[a]_s, \sigma_{m,n}[a]_s, T_{m,n}[a]_s, 0)$. Then prepend the address a to each $H_{prefix} \in H^0_a$, and insert the updated H_a into $H_{m,n}$.
4. Return $T_{m,n}$ and $H_{m,n}$.
