The UTXO Set

The **UTXO Set** contains a list of all unspent transaction outputs in the economy.

This list can get quite large, and has to be updated every time a new block is added, which could be as frequently as once per second or more. Hence, care must be taken that the update process is made as efficient and local as possible.

The PHANTOM algorithm assumes that a node has knowledge of the UTXO Set right before each block was added (at least until reorganization, which will be discussed shortly). The naive implementation is to keep a UTXO Set for every block in the DAG, which is manifestly infeasible. The solution is to keep incremental diff data – each block has a designated child called its Diff Child. Additionally, each block carries a field called UTXO Diff, which holds exactly the data required to restore its UTXO Set given the UTXO Set of its Diff Child. The virtual block is the only exception, its UTXO Diff field points to the actual Full UTXO Set. This allows a node to recursively calculate the UTXO Set relative to any block's past – first, following the list of Diff Children all the way to the virtual block, accumulating a stack of UTXO Diffs, then applying them to the Full UTXO Set one by one. While much improved, this solution is still unacceptable, as it still forces us to clone the UTXO Diff each time a new block is added. In practice, the stack is smashed to a single list, containing exactly the differences between the current state, and the state relative to the block we started with.

Whenever the placement of a new block causes a change to the selected parent chain, the blockDAG must undergo reorganization. This might change the colors of blocks, which might affect the UTXO Set relative to some of the blocks. All blocks need to be updated to reflect the change. To be exact, the UTXO Set of each block should reflect the state at the time the block was included **had the structure of the DAG at the time of its inclusion been is the current structure**. When we reorganize the DAG we "go back in time" and erase the block's memory of our change, tricking it to believe that the DAG has always looked this way.

The following article details the ideas, data structures and algorithms used in our implementation of the above concepts.

- Data Structures
  - UTXOSet
    - UTXOSet.AddTx
    - UTXOSet.Clone
    - UTXOSet.Diff
    - UTXOSet.DiffFromTx
    - UTXOSet.Get
    - UTXOSet.Iterate
    - UTXOSet.WithDiff
  - DiffUTXOSet
    - DiffUTXOSet.base
    - DiffUTXOSet.diff
    - DiffUTXOSet.Diff
  - FullUTXOSet
    - FullUTXOSet.Diff
  - UTXODiff
    - UTXODiff.Diff
    - UTXODiff.Inverted
    - UTXODiff.NewUTXODiff
    - UTXODiff.toAdd
    - UTXODiff.toRemove
    - UTXODiff.WithDiff
  - UTXOCollection
  - UTXOIteratorOutput
- Algorithms
  - restoreUTXO
  - meldToBase

Data Structures

UTXOSet

The UTXOSet interface is an abstraction of the notion of a collection of tx in a state.

Implementations: The UTXO Set#DiffUTXOSet, The UTXO Set#FullUTXOSet

UTXOSet.AddTx

> *"AddTx(tx *Tx) (ok bool)"*

Given a pointer to a Tx, go over all the transactions, and try to apply them.

For a TxIn to be applicable to a UTXOSet, it is required that txIn.OutPoint corresponds to a UTXO (i.e. all the coin that this Tx is trying to spend is indeed spendable).

In this case, all the UTXOs which correspond to the OutPoints in txIn are removed, and all the outputs in tx.txOut are added to the UTXO.

This only happens if **all** TxIns are applicable. If **any** of the inputs is invalid, **none** of them are applied, and ok will return as false.

Note: AddTx does not verify that the transactions in the block make sense (e.g. that the expenditure does not exceed the redeemed transactions), it should only get legal transactions.

UTXOSet.Clone

> *"Clone() UTXOSet"*

Returns a perfect deep copy of `this`.

UTXOSet.Diff

> *"Diff(other UTXOSet) *The UTXO Set#UTXODiff"*

Returns a list of differences that, if applied to `this`, will result in a perfect copy of `other`.

Warning: The implementations of this method in both DiffUTXOSet and FullUTXOSet have non-trivial assumptions, please read the reference before using them.

UTXOSet.DiffFromTx

> *"DiffFromTx(tx *Tx) (*The UTXO Set#UTXODiff, error)"*

Given a transaction, this method calculates the difference between `this` before and after applying the transaction.

If one or more of the inputs in the transactions do not appear as an output from the worldview of `this`, an error is returned.

This has a default implementation in the function called `utxo_set.diffFromTx`. Implementations of UTXOSet should decorate this function by default, choosing not to do so must be explicitly justified in the documentation.

UTXOSet.Get

> *"Get(outPoint OutPoint) (*TxOut, bool)"*

Given an OutPoint, return the corresponding TxOut in the UTXOSet.

If the TxOut is not in the UTXOSet return false in the second argument.

UTXOSet.Iterate

> *"Iterate() The UTXO Set#UTXOIterator"*

Returns a UTXOIterator which iterates over the unspent transactions in the UTXO

UTXOSet.WithDiff

> *"WithDiff(diff *UTXODiff) *The UTXO Set#UTXODiff"*

Given a The UTXO Set#UTXODiff `diff`, return a copy of `this` with the changes in `diff` applied.

DiffUTXOSet

An implementation of UTXOSet, representing the data as some base set and a list of differences from the base set.

DiffUTXOSet.base

> *"*The UTXO Set#FullUTXOSet."*

Pointer to the base relative to which the diff list describes our worldview.

DiffUTXOSet.diff

> *"*The UTXO Set#UTXODiff."*

Points to the list off differences between the base and the described worldview.

DiffUTXOSet.Diff

> *"(u *DiffUTXOSet) Diff(other The UTXO Set#UTXOSet) *The UTXO Set#UTXODiff"*

An implementation of The UTXO Set#UTXOSet.Diff.

DiffUTXOSet.Diff has some assumptions on the input:

- other is an instance of The UTXO Set#DiffUTXOSet, and
- `u.base == other.base`

If the assumptions are not met a panic is raised.

After verifying these assumptions, the actual construction of the output is made with a call to UTXODiff.Diff.

Up to checking the assumptions and raising the required panic, this method is extremely simple:

```
Diff(other UTXOSet)
        o = (DiffUTXOSet) other
        return u.diff.Diff(o.diff)
```

FullUTXOSet

An implementation of UTXOSet simply holding a list of all UTXOs.

FullUTXOSet is a struct with a single field of type The UTXO Set#UTXOCollection.

FullUTXOSet.Diff

> *"(u *FullUTXOSet) Diff(other The UTXO Set#UTXOSet) *The UTXO Set#UTXODiff"*

An implementation of The UTXO Set#UTXOSet.Diff.

FullUTXOSet.Diff has some assumptions on the input:

- other is an instance of The UTXO Set#DiffUTXOSet, and
- `u.base == other.base`

If the assumptions are not met a panic is raised.

If the assumptions are met then the list of differences is actually just `other.diff`.

UTXODiff

Represents the difference between two UTXOSets.

Note that UTXODiff is **antisymmetric** on UTXOSets – UTXODiff describes how to get from one set to the other, and to go the other way around one must *invert* the UTXODiff, i.e. to switch toAdd and toRemove.

Below we refer to the set from which changes are made as the *base*, and to the resulting set as the *result*. Note however that these sets do not exist as far as UTXODiff is concerned. It is up to the user that UTXODiff is used in a meaningful context.

A UTXODiff d is considered:

- *legal* if $\texttt{d.toAdd} \cap \texttt{d.toRemove} = \varnothing$
- *valid* relative to a UTXOSet u if $\texttt{d.toRemove} \subseteq \texttt{u}$ and $\texttt{d.toAdd} \cap \texttt{u} = \varnothing$

Note that being valid implies being legal, and that being legal implies being valid with respect to $\texttt{u} = \texttt{d.toRemove}$.

UTXODiff.Diff

> *"(d *UTXODiff) Diff(other *UTXODiff) *UTXODiff"*

Given two diffs relative to the same base, returns the differences between the two results, where the result of d is considered as the base of the returned UTXODiff.

Has the following assumptions on the input:

- Both d and `other` are legal
- Both d and `other` are valid relative to **the same** The UTXO Set#UTXOSet

`UTXODiff.Diff` iterates over all four collections and handles each UTXO appropriately.

The table below details where a given UTXO should be put in the output according to whether and where it appears in both d and `other`.

other \ d	toAdd	toRemove	neither
toAdd	none	**impossible**	toAdd
toRemove	**impossible**	none	toRemove
neither	toRemove	toAdd	**irrelevant**

The cases marked impossible simply can not happen under the assumption that both diffs are valid relative to the same UTXOSet, since one assumes the UTXO is in the set while the other assumes the opposite, by way of contradiction.

Note that the assumptions on the input hold if and only if none of the impossible cases happen, so that raising an error upon such an impossibility constitutes a verification of validity (however, this is not done in the PoC).

The case marked irrelevant just never happens, as we only iterate over UTXOs in the lists.

UTXODiff.Inverted

> *"(d *UTXODiff) Inverted() *UTXODiff"*

Returns a copy of d with toAdd and toRemove switched.

UTXODiff.NewUTXODiff

UTXODiff.toAdd

> *"The UTXO Set#UTXOCollection"*

The UTXOs which appear in the resulting set but not in the base set.

UTXODiff.toRemove

> *"The UTXO Set#UTXOCollection"*

The UTXOs which appear in the base set set but not in the result set.

UTXODiff.WithDiff

> *"(d *UTXODiff) WithDiff(diff *UTXODiff) *UTXODiff"*

This "concatenates" d with `diff`. It assumes that `diff` is valid with respect to the result of d.

For this to hold, it suffices to assume:

- `d.toAdd`$\cap$`diff.toAdd`$=\emptyset$ (`diff` does not add any UTXO entries already in the result of d)
- `d.toRemove`$\cap$`diff.toRemove`$=\emptyset$ (`diff` does not remove any UTXO entries which are not in the result of d)

UTXODiff.WithDiff creates a diff whose base is the base of d, and whose result is the result of diff relative to the result of d.

`UTXODiff.Diff` iterates over all four collections and handles each UTXO appropriately.

The table below details where a given UTXO should be put in the output according to whether and where it appears in both d and `diff`.

diff \ d	toAdd	toRemove	neither
toAdd	**impossible**	none	toAdd
toRemove	none	**impossible**	toRemove
neither	toAdd	toRemove	**irrelevant**

The cases marked impossible simply can not happen under the assumptions, since it implies that diff adds/removes a UTXO entry which was already added/removed by d, by way of contradiction.

Note that the assumptions on the input hold if and only if none of the impossible cases happen, so that raising an error upon such an impossibility constitutes a verification of validity (however, this is not done in the PoC).

The case marked irrelevant just never happens, as we only iterate over UTXOs in the lists.

UTXOCollection

Represents an arbitrary collection of UTXOs. Unlike UTXOSet, this is an arbitrary collection which needs not be a valid state (i.e. it may hold conflicting UTXOs).

UTXOIteratorOutput

A struct containing an OutPoint and *TxOut.

Used as output to UTXOIterator.

The *TxOut points to the UTXO associated with the OutPoint, out of the UTXOs in the UTXOSet on which this iterator was instantiated.

Algorithms

restoreUTXO

The `restoreUTXO` algorithm gets a block and restores the UTXO list from its worldview.

This is done as follows:

```
restoreUTXO(block B)
        stack        = [B]
        cur                = B

        while cur.DiffChild is not null
                cur                = cur.DiffChild
                stack        = stack.append(cur)

        utxo         = empty UTXOSet

        while stack is not empty
                utxo = utxo.WithDiff(stack.pop.UTXODiff)

        return utxo
```

cf. Block.UTXODiff, Block.DiffChild, UTXOSet.WIthDiff.

Note that this algorithm is agnostic about the type of the UTXOSet in question. In practice this function is always applied to a non-virtual block, and in this case the output is the full list of differences between the input block and the current state.

meldToBase

Given a list of differences in the form of a The UTXO Set#DiffUTXOSet `D`, apply the changes to `D.base` and replace `D.diff` with an empty `The UTXO Set#UTXODiff`.

This simple function should be used **with caution**. Typically D.base will contain a pointer to DAG.UTXOSet, so applying meldToBase will change the global state!