BUTANE: A SECURE EUTXO SYNTHETICS PROTOCOL

v. 544337b48d69f8af0d36a1af5a903a95c637d2a8

Micah Kendall
Butane Team
micah@butane.dev

Jay Taylor
Butane Team
jay@butane.dev

Abstract. The EUTXO paradigm as implemented on the Cardano Babbage-era ledger provides a concurrent, fully sharded, turing-complete model of computation (generalised state machine), with untyped plutus core as a restricting validating component of state transitions. Transactions transition states, while each UTxO is an independent piece of sharded state. Butane implements Synthetics, a fungible derivative financial instrument, in a coherent manner, playing into the strengths of a sharded ledger. Here we outline its design and implementation.

1. Introduction

Synthetic assets are tokenized derivatives which permit exposure to the price movements of an underlying asset without owning it directly. The transformative potential of synthetics is congruent with the principles of blockchain technology – a free and open financial future for all.

The Butane Protocol ("Butane") is a permissionless, non-custodial synthetic assets protocol built on the Cardano blockchain. This document provides a comprehensive specification of Butane. It details the core feature set offered by the protocol's smart contracts, while information such as protocol tokenomics and certain technical implementations of off-chain features (such as the off-chain component of the oracle system) are out of scope. This paper is intended to be a living document, and will be iterated on and updated as Butane matures and evolves.

1.1. **Why Cardano**. Cardano is a third-generation blockchain platform that aims to provide a more secure and scalable infrastructure for the development of decentralized applications. The EUTXO model, as implemented on the Cardano Babbage-era ledger, is well-suited for the implementation of a synthetics protocol. As this document will describe, Butane leverages the deterministic and sharded model of computation offered by Cardano to eliminate contention while facilitating the creation and management of complex financial instruments as lightweight Cardano native tokens ("CNTs") which have the ability to be seamlessly used throughout the Cardano ecosystem. We believe unfettered transparency and fairness are paramount features of user interactions with smart contracts, and that the deterministic EUTXO model provides the best technological foundation to achieve them.

2. The Butane Protocol

Butane facillitates the creation of synthetic assets via the opening of Collateralized Debt Positions ("CDPs") – debt obligations created by a user locking some token value as collateral and minting some amount of a synthetic asset such that the collateral value is strictly greater than the minted value. CDP liquidations in the event of the ratio of collateral value to debt value (the "Collateral Ratio") dropping past a calculated threshold ensure solvency in the system and enforce the invariant that all synthetic assets are sufficiently backed and underlying debt is overcollateralized. In accordance with the objective of liquidity maximisation, mechanisms such as CDP liquidations

and redemptions, CDP interest, and interactions with the protocol treasury act to approach a soft price peg for synthetic assets.

2.1. **Configuration**. Each asset class is individually parameterized with the following settings:

- **collateral_assets:** The sorted list of accepted collateral assets.
- **weights:** The list of "weight" parameters for each collateral asset. Used in the calculation of a CDPs "Health Factor".
- **minimum_outstanding_synthetic:** For each CDP which mints this synthetic asset, the minimum amount the CDP is required to mint.
- **interest_rates:** The list of the past 10 and global maximum interest rates for this synthetic.
- **max_proportions:** The list of the "Maximum Proportion" parameters for each collateral token, where MP_i indicates that the ith collateral token can be used to collateralize at most $\text{MP}_i\%$ of the outstanding debt.
- **max_liquidation_return:** In the event of a liquidation, the maximum proportion of the outstanding debt that can be claimed from the CDP owner.
- **treasury_liquidation_share:** In the event of a liquidation, what % of the excess claimed value is sent to the treasury as a fee.
- **redemption_share:** In the event of a CDP redemption, what % of the repayed value the redeemer can claim from the CDP.
- **fee_token_discount:** What % of payable interest is discounted when a user burns the protocol's fee token in place of synthetics.

These parameters are configurable via governance actions.

2.2. **Price Feeds**. Oracle price feeds provide necessary pricing data for collateral tokens, used in the determination of CDP values. The price feed mechanism is independently upgradable via governance actions.

3. BTN

The Butane fee and governance token ("BTN") is a CNT which can be owned by any user. The tokenomics of BTN are outside of the scope of this document. Within the protocol, BTN has the following use cases:

- **Collateral Token:** BTN may be used as a collateral token for opening CDPs if it is an accepted collateral asset in a synthetic's parameters.
- **Interest Payment:** BTN can be burned in place of synthetics to pay interest on a CDP at a discount.
- **Governance Proposals:** The creation of new proposed governance actions requires the staking of BTN.
- **Governance Voting:** Governance proposals are voted on via the staking of BTN. Stakers may earn governance reward tokens for voting on proposals.

The burning of BTN for repaying interest serves as a deflationary mechanism as the total supply is reduced overtime.

4. Collateralized Debt Positions

CDPs are singular UTxOs which contain locked collateral tokens and track their total outstanding debt. The ADA within CDPs is liquidly staked – users can continue to earn staking rewards on their locked ADA. Key CDP metrics are described below.

Let c_i, p_i, W_i be the amount, price (in terms of the synthetic), and weight of collateral token i in the CDP which mints d amount of synthetic token s. Then the "Borrowing Capacity" and "Health Factor" of the CDP are given as:

$$\text{BC} := \sum_{1 \le i \le n} min\left\{ \frac{c_i \cdot p_i}{W_i}, \frac{d \cdot \text{MP}_i^s}{p_i} \right\}$$

$$\text{HF} := \frac{\text{BC}}{d}$$

, where MP_i^s is the *maximum_proportion* parameter for i'th collateral token of s. A position is considered "unhealthy" when $\text{HF} < 1$. Similarly, the "Collateral Value" and "Collateral Ratio" of a CDP is given as:

$$\mathrm{CV} := \sum_{i=1}^{n} c_i \cdot p_i$$

$$\mathrm{CR} := \frac{\mathrm{CV}}{d}$$

The following CDP actions are supported on the protocol:

Create CDP: A position is opened by a user minting some amount of a synthetic asset and locking sufficient collateral value such that $\mathrm{HF} \geq 1$.

Repay CDP: A CDP owner repays the outstanding debt of one of their CDPs and closes the position. Locked collateral is returned to them.

Liquidate CDP: A position with $\mathrm{HF} < 1$ has its debt partially or fully repaid by any user, and a substantial portion of the locked collateral is claimed from it. In the event of a partial liquidation (i.e., where not all outstanding debt is repaid by the liquidator), a new position with $\mathrm{HF} > 1$ is created with the same owner as the liquidated CDP.

Redeem CDP: A position's debt is partially or fully repaid by any user, and equivalent collateral value is claimed from the position by them minus a premium. The conditions for CDP redemptions are managed by a protocol-wide redemption NFT.

CDP actions are entirely composable, permitting more complexity with seamless user experience. A "CDP Adjustment", whereby an existing CDP has either its collateral portfolio or outstanding debt altered, can be achieved via a composition of a "Repay" and "Create" action within a single transaction (see Figure 4).

4.1. **Interest**. CDPs accrue interest linearly. Interest functions as a mechanism for the protocol to control the incentives for creating and closing CDPs, hence influencing the supply of synthetics and providing an avenue for price corrections in the event of a depeg. Interest per debt token is calculated as

$$I = \frac{1}{T_{\mathrm{year}}} \sum_{i=1}^{n} r_i \cdot \Delta t_i$$

Where $n \in \mathbb{Z}_+$ is the smallest number such that $t_n < t_{\mathrm{open}}$,

$$\Delta t_i = \begin{cases} t_{i-1} - t_{\mathrm{open}}, & \text{if } i = n \\ t_{i-1} - t_i, & \text{otherwise} \end{cases}$$

, and

$$t_0 = t_{\mathrm{close}}$$

, with I being the outstanding interest at the time of closing the CDP, t_{open} and t_{close} the opening and closing timestamps of the CDP in milliseconds, respectively, r_i the i'th most recent interest rate, t_i the timestamp in milliseconds that the i'th most recent interest rate was set, and T_{year} the number of milliseconds in a year. Only the 10 most recent interest rate changes are stored in the synthetic parameters – if a CDPs opening time precedes the oldest change, the global maximum rate is applied for the unaccounted timeframe.

Accumulated interest counts towards the total outstanding debt of a CDP. It is paid when a CDP is closed (either by a repayment, liquidation, or redemption), and there are no fees when a CDP is opened. Interest can be paid either as extra debt (i.e., burning extra synthetic tokens in addition to the initial outstanding amount at the CDPs opening) or by burning an equivalent amount of BTN, with a possible discount. This discount is configured via the *fee_token_discount* parameter.

We have the total outstanding amount of a CDP at closing time as

$$\begin{aligned} S_{\mathrm{close}} &:= s + s \times I \\ &= s(1 + I) \end{aligned}$$

, where s is the initial borrow (mint) amount, and so the health factor at closing time is given as

$$\begin{aligned} \mathrm{HF}_{\mathrm{close}} &:= \frac{\mathrm{BC}}{S_{\mathrm{close}}} \\ &= \frac{\mathrm{BC}}{s(1 + I)} \end{aligned}$$

This implies the health factor of a CDP will tend towards 0 overtime, and so all CDPs will eventually be able to be liquidated and closed.

4.2. **Ownership**. CDP ownership is not necessarily tied to a specific user but to a key or some constraint. Specifically, the following are possible definitions of CDP ownership:

- The payment key credential of a Cardano address
- The current holder of a specific NFT
- The ability to withdraw stake from a specified script
- Authorization via a signature from a specified signing key

This allows for CDPs to be owned by scripts, and for the ownership of a CDP to be transferred to another party via the transfer of an NFT bond, allowing for future interoperability and extension.

5. Liquidations

A CDP is subject to liquidation when it is unhealthy. This involves a liquidator burning some amount of outstanding debt and claiming value of locked collateral up to some value greater than the debt that was repaid.

The *max_liquidation_return* parameter defines the limit on the value a liquidator can claim. $\text{MLR} = 120\%$ signifies that a liquidator who repaid the CDP fully can claim at maximum 120% of the repayed debt's value from the CDP's collateral. If they repaid half, they could claim at maximum 60%.

Following a liquidation, the debt must either be 0 (in which case the CDP is closed) or greater than the *minimum_outstanding* parameter. In the first case, the remaining collateral (if any) is sent to a collection script where the CDP owner can reclaim it. In the second case, leftover debt and collateral is transferred to a new position with the same original owner. This enforces that partial liquidations effectively make the CDP healthy, preventing cases where unhealthy debt exists in the system due to small liquidations causing contention issues and preventing unhealthy debt from being fully repaid. A liquidator cannot increase any of the collateral token amounts within a leftover CDP – this is to prevent a liquidator from exposing a CDP owner to more of a specific collateral token in order to withdraw a greater amount of another token.

The *treasury_liquidation_share* parameter indicates what % of the excess collateral claimed by the liquidator gets sent to the treasury as a fee. The value of this fee can be determined:

$$\text{LF} = \text{TS}\left(1 - \frac{1}{\text{CR}}\right) \cdot V$$

, where V is the value claimed from the CDP owner during the liquidation and TS is the *treasury_liquidation_share* parameter.

The liquidator has full discretion over which assets they claim from the CDP when they liquidate, so long as they adhere to the *max_liquidation_return* constraint and ensure in the event of a partial liquidation that the leftover CDP is healthy. The treasury fee value is of the same token proportions as the value claimed by the liquidator.

6. Redemptions

CDP redemptions involve a user repaying some amount of a CDP's outstanding debt and claiming an equivalent amount of collateral value from it (minus a premium), based on the mark prices provided by the oracle price feed. The user effectively redeems a portion of their synthetics holdings for its underlying value. Redemptions act as a depeg protection mechanism, as in the event of a synthetic's market price falling below its mark price, arbitrage opportunities arise for users to redeem synthetics at values near their mark price, reducing supply and creating buy pressure.

Redemptions don't involve a CDP owner losing total value in their CDP, as the outstanding debt of their position decreases by a greater or equal amount to the value of collateral claimed. This is configured via the *redemption_share* parameter, which scales the amount of collateral that can be claimed per token burned. As an example, $\text{RS} = 97\%$ implies that a redeemer can claim 97% of the repayed value from the CDP's collateral.

Butane is parameterised with an NFT which must be referenced for all redemption actions. All redemption actions are either signed by the owner of the token, or invoke the script owner of the token via a stake withdrawal for authorization.

The redeemers pay interest on behalf of the user. Redeemers can redeem the total amount of synthetics they burn, including interest, as it is treated as extra debt. Hence, as a CDP is open for longer, more of its collateral can be redeemed. If a CDP is fully redeemed (i.e., all of its outstanding debt is paid), the leftover collateral is sent to a collection script for the owner's reclamation.

7. Governance

Changes to Butane are handled through a decentralised governance mechanism. A governance NFT dictates conditions for proposal creations and voting, where the NFT owner (a key or a script) must authorise the action. The NFT is transferrable to permit the upgrading of governance.

7.1. **Actions**. The following are the primary governance actions:

- **New Synthetic:** Defining a new synthetic via the creation of a params UTxO.
- **Params Update:** Updating the value of some parameter for a given synthetic.
- **Treasury Spend:** An arbitrary spend from the treasury. Defines a new output to be created, which allows for the treasury's value to be used for a variety of purposes.
- **Treasury Mint:** The minting of a synthetic token via the treasury. Primary used for minting previously-burned interest.
- **Treasury Debt Creation:** The treasury can be redeemed against by creating arbitrary debt. Allows users to redeem their synthetics for their underlying value against the treasury's holdings.
- **Treasury Stake Update:** Updates the treasury's stake delegation.
- **External Governance:** A governance action handled by an external script.
- **Text Proposal:** An arbitrary proposal containing text to be voted on. It has no programmatic effect on the protocol.

7.2. **Proposal Process**. A user can create a governance proposal to enact a governance action by staking an amount of BTN (an amount configurable as a protocol parameter). Users will vote on the proposal via the staking of their BTN, and if the proposal passes, an action UTxO is created which facillitates the execution of the action. If the proposal fails, the initial locked BTN by the proposer may be burnt.

Details of the proposal mechanism will be included in future versions of this document as this feature is fully realised and implemented within Butane.

8. Validator Design

(See also Section Appendix B for a full protocol type specification) Butane's smart contracts leverage many idiomatic design patterns unique to the EUTXO ledger model in order to provide a secure and maximally efficient user experience.

8.1. **General Patterns and Optimizations**. Benchmark testing of Butane has determined performance figures of up to 50 CDP creations and 42 CDP repayments per transaction. These figures are achievable primarily through the following script implementation details:

8.1.1. *The "Withdraw 0" Trick*. The ledger permits transactions which withdraw 0 rewards from a script staking credential. This allows for arbitrary one-off validation logic to be included within a transaction. Butane leverages this trick extensively when validating CDP transactions – instead of validating the spending of each CDP UTxO individually (which can involve significant duplicate logic as the number of CDP spends in the transaction increase), all validation is handled via a stake withdrawal validator which must be invoked whenever any Butane UTxO is spent or any token is minted. As the spending validation logic for Butane UTxOs are minimal and the validator is invoked for every Butane UTxO spent during the transaction, there is significant potential for performance gain by optimising the spend script (see Section 8.2.3).

The trick also simplifies the process of attaching extra data to the transaction, as in the case of Butane's price feeds. Price data is provided to the script context by using the withdrawal trick on a price feed validator, where the price data is passed into the stake withdrawal redeemer and can then hence be read by other scripts invoked in the same transaction.

8.1.2. *Ordering*. The ledger orders transaction inputs and reference inputs lexicographically,

and orders the withdrawals map in the script context by script hashes. These can effect a suboptimal script execution as control over the ordering of data is lost. Transaction outputs, however, are ordered deterministically by the transaction builder. The following are Butane design details which leverage these facts:

- The order of synthetic parameters (which are referenced inputs) is determined as the transaction is built, and the price feed redeemer is constructed in the same order.
- Created CDP outputs are grouped by synthetic type and ordered the same as the synthetic parameters.
- Spent CDPs which have remaining outputs (i.e. in the case of a partial liquidation or a liquidation with leftovers) have their corresponding outputs at the same index in the Butane output list as the Butane input list.
- CDP repayment and liquidation actions are detailed in a list of records in the stake withdrawal redeemer in the same order as the CDP inputs. As it's impossible to have total control over both the ordering of the transaction's inputs and reference inputs, these records contain the associated index of the parameter UTxO for each CDP's synthetic type.
- The two primary stake withdrawal validators for Butane are the CDP logic script (see Section 8.2.1) and the price feed script (see Section 8.2.4). The CDP script is parameterized with a *salt* value which, during deployment, is brute forced such that the resulting script hash comes before the price feed validator's. This ensures that the CDP validator precedes the price feed validator in the withdrawals map in the script context, allowing for CDP spend validation to be able to search for the script hash more cheaply.

These orderings minimize the need for searches and sorts in the validation logic.

8.1.3. *Burning*. Unnecessary creations of outputs can be costly, as each output increases the transaction budget and must contain at least a certain amount of ADA (see CIP-55). Butane utilises token mints and burns to circumvent these costs. Notably, CDP interest is repaid via the burning of synthetic tokens or BTN, rather than sending the interest as a UTxO to the treasury. The treasury can then mint the synthetics later, effectively collecting all of the interest into a single UTxO ad hoc while not encumbering Butane users with any unnecessary costs.

8.2. **Validators**. Butane is comprised of six validators.

8.2.1. *Synthetics Main (V_{SM}).* A stake withdrawal validator which handles and implements all of the CDP actions as defined in Section 4. Permits the minting and burning of synthetic tokens as CDPs are created and destroyed. The stake withdrawal redeemer is constructed as a list of actions to be performed, where actions are fully composable with one another.

8.2.2. *Synthetics External (V_{SE}).* A stake withdrawal validator which handles governance endpoints. Its design is wholly compatible with V_{SM} but was separated from it into a new validator to conform to transaction size limits when deploying the protocol validators.

8.2.3. *State (V_S).* A very lightweight spend validator which contains most protocol UTxOs, including CDPs, synthetic parameters, and treasury outputs. Whenever a Butane UTxO is spent, V_S enforces that V_{SM} is withdrawn from in the same transaction. V_S is a generic validator design that can work with any script using the "Withdraw 0" trick, and hence can be argued to be a validator design of high importance to Cardano as a whole. Optimization efforts of V_S have been handled publically among the Cardano developer community.

8.2.4. *Price Feed (V_{PF}).* A withdrawal validator which handles the verification of on-chain price feeds. Oracle price feeds are signed values which contain ordered lists of collateral token prices in terms of their associated synthetic tokens. V_{PF} is parameterized by the verification key used to validate price feeds. Transactions that require price feeds withdraw from V_{PF} while passing in the signed feeds into the redeemer, allowing other scripts to access them via the script context. The prices and their signatures themselves can be acquired via an off-chain oracle service which collects and signs prices. The price feed mechanism is deliberately simple to streamline the process

of future protocol upgrades involving new oracle systems.

8.2.5. *Leftovers (V_L).* A simple spend validator which handles leftover collateral in the event of a CDP liquidation where not all collateral can be claimed. It stores collateral value which can be claimed by the original CDP owner at any time. The validation criteria is identical to authorising a CDP repayment (and so leftovers can be owned by scripts, NFTs, etc., the same as CDPs).

8.2.6. *BTN (V_B).* A mint validator which handles the generation of BTN, as well as token burning and the token vesting scheme. It enforces that the genesis event occurs only once and that BTN cannot be minted afterwards, only burned.

8.3. **Dependencies**. Validators form a dependency Directed Acyclic Graph (DAG) as shown in Figure 1. An arrow from a validator x to a validator y indicates that x is directly dependent on y.

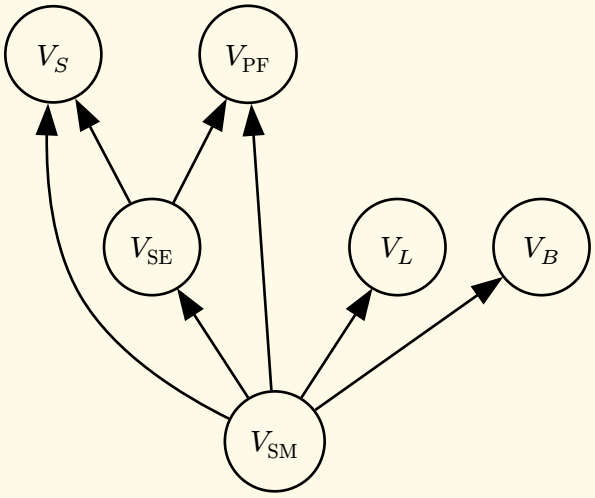

Figure 1: Validators Dependency Graph

8.4. **Upgradability**. Butane is designed to facilitate a seamless transition in the event of a full protocol upgrade. Governance itself is upgradable by virtue of the governance token being transferrable, and the main protocol can accept the wrapping of an upgraded set of protocol tokens via a governance action. This allows a 1:1 swap between new and old synthetics.

A full upgrade (transfer of treasury, acceptance of equivalent synthetics, adjustment of synthetic parameters) may happen in one vote via the governance token being sent to a script which issues governance actions to the original protocol. Parameters may then be adjusted via governance to slowly wean off CDPs and incentivise them to migrate to the updated protocol.

9. Further Work

Butane introduces an efficient and generalisable model for atomicity and composability within the design of a decentralized application. While this document has focused on composability within the protocol itself, the model described extends to a general concept of interoperability within the Cardano ecosystem.

It is a continuing goal of the Butane team to foster an environment among Cardano applications which encourages atomicity and transactional efficiency via the composition of actions between protocols. The model presented here may be extended and defined formally as a set of standards for interoperability on Cardano which protocols can conform to, allowing them to take advantage of the same composability and atomicity features as Butane.

Appendix A. Protocol Transactions

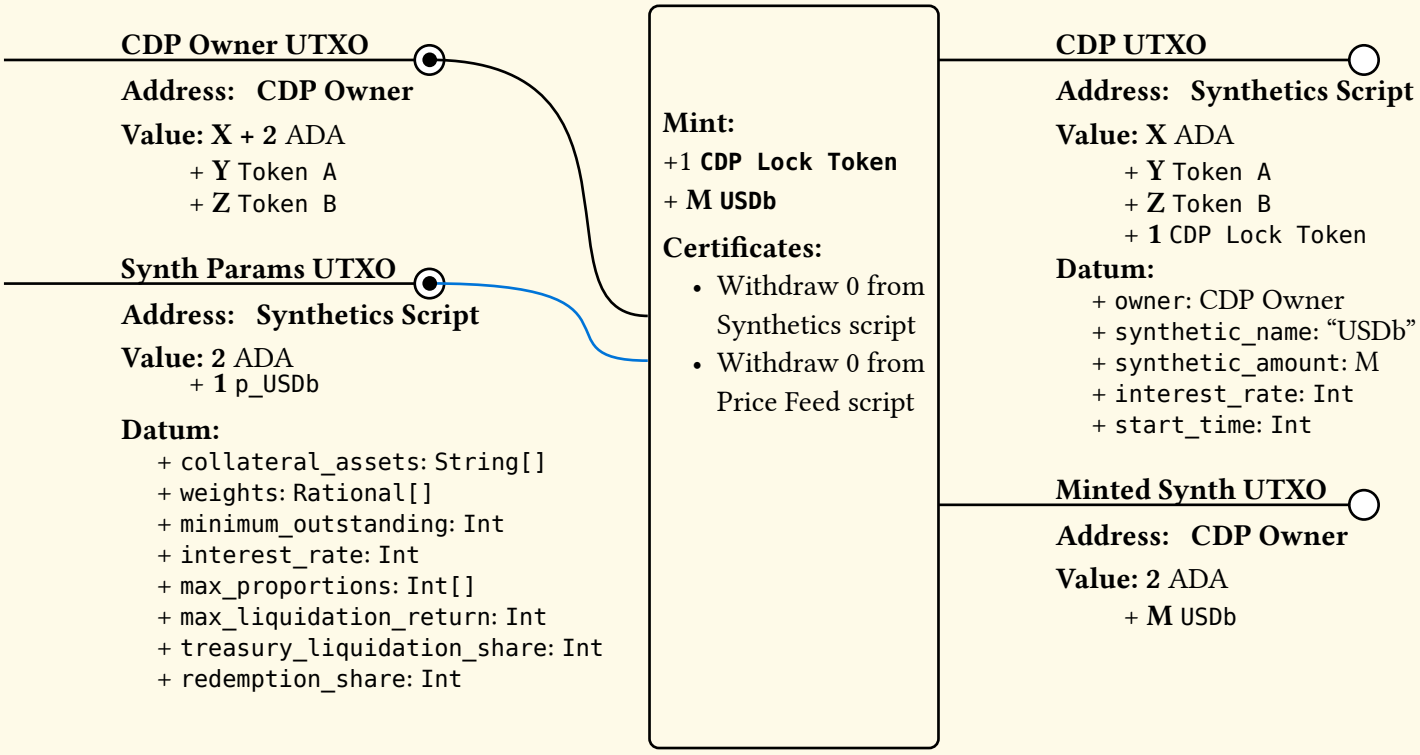

Figure 2: CDP Creation Transaction (USDb)

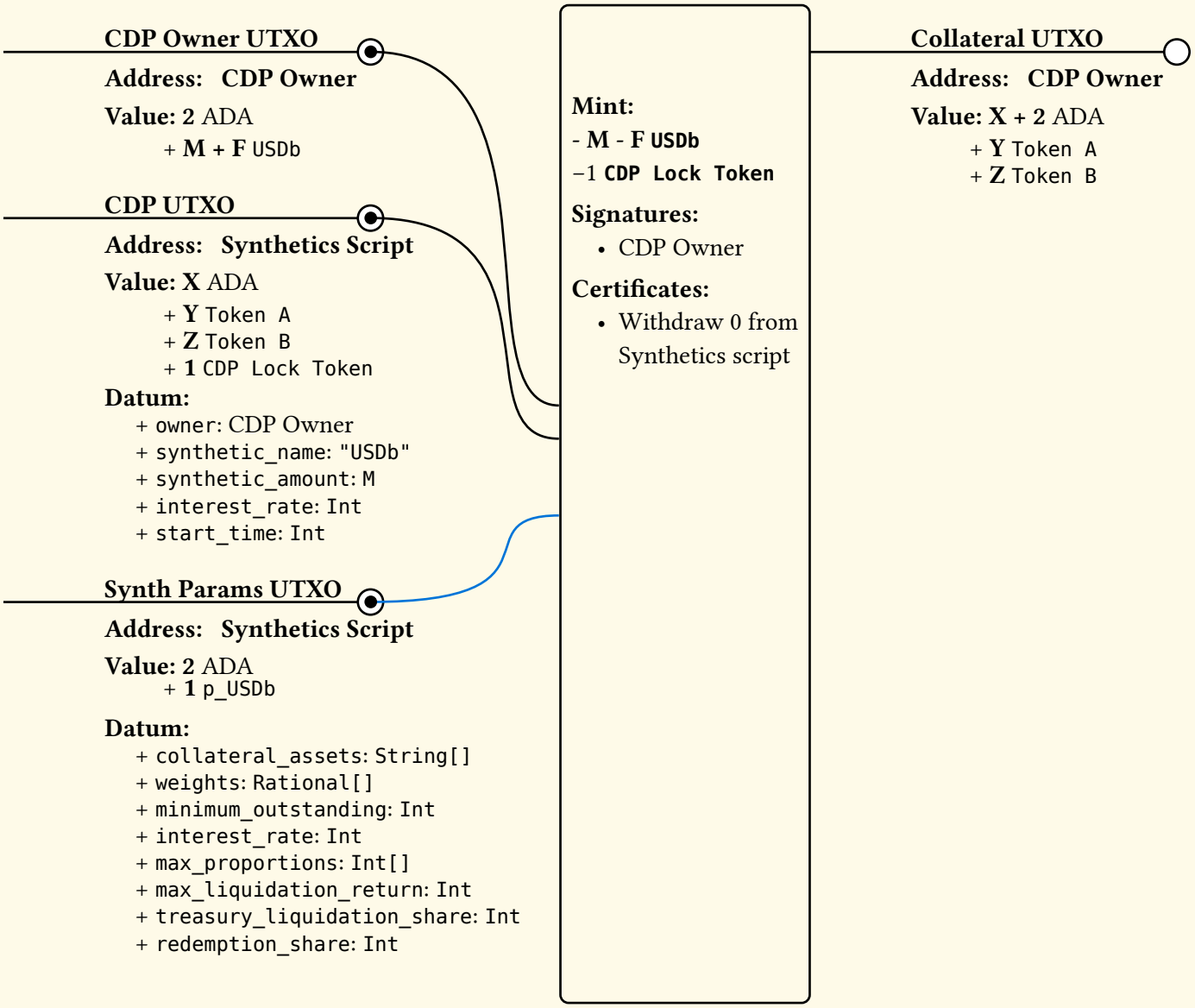

Figure 3: CDP Repayment Transaction (USDb)

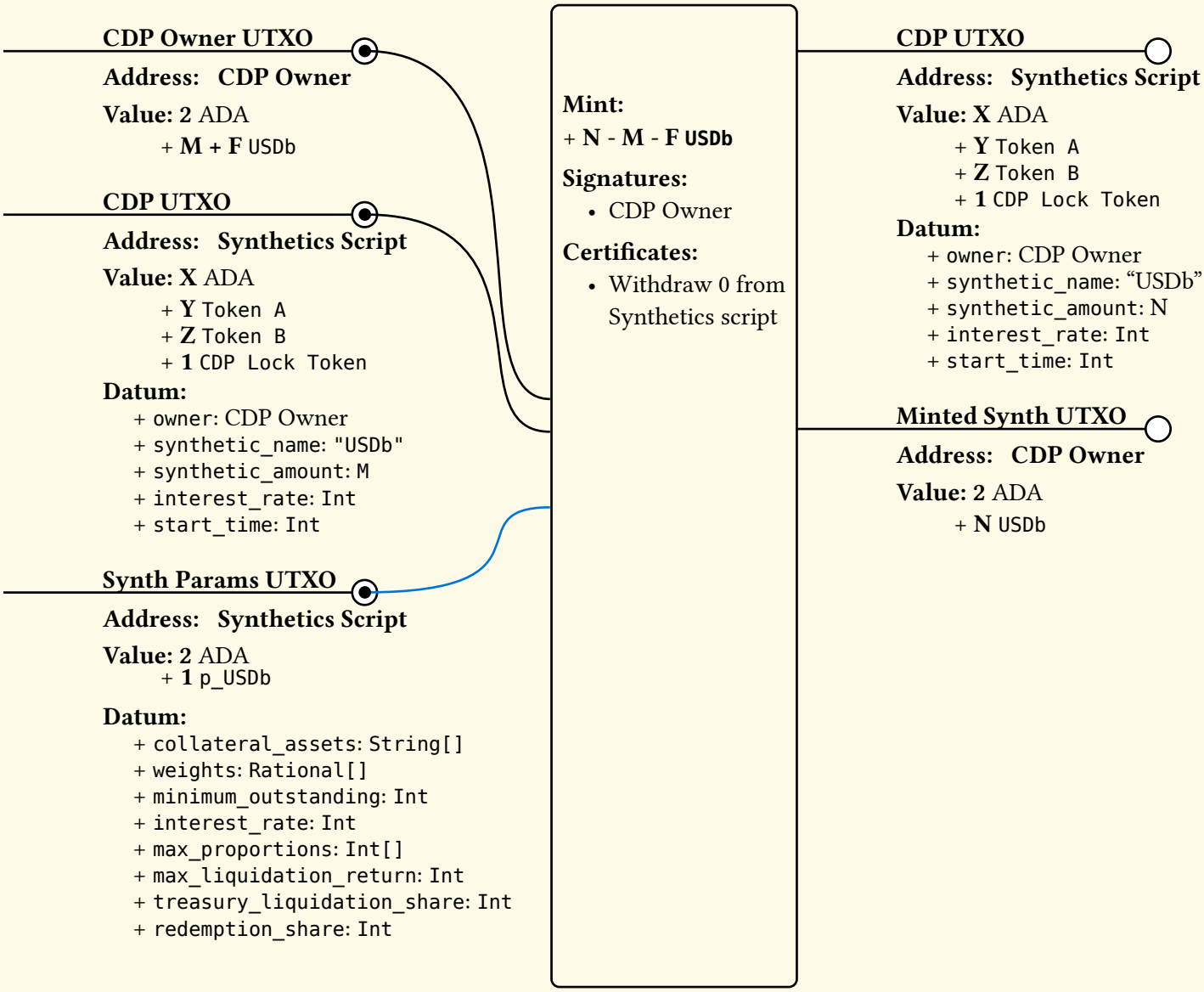

Figure 4: CDP Debt Adjustment Transaction (USDb)

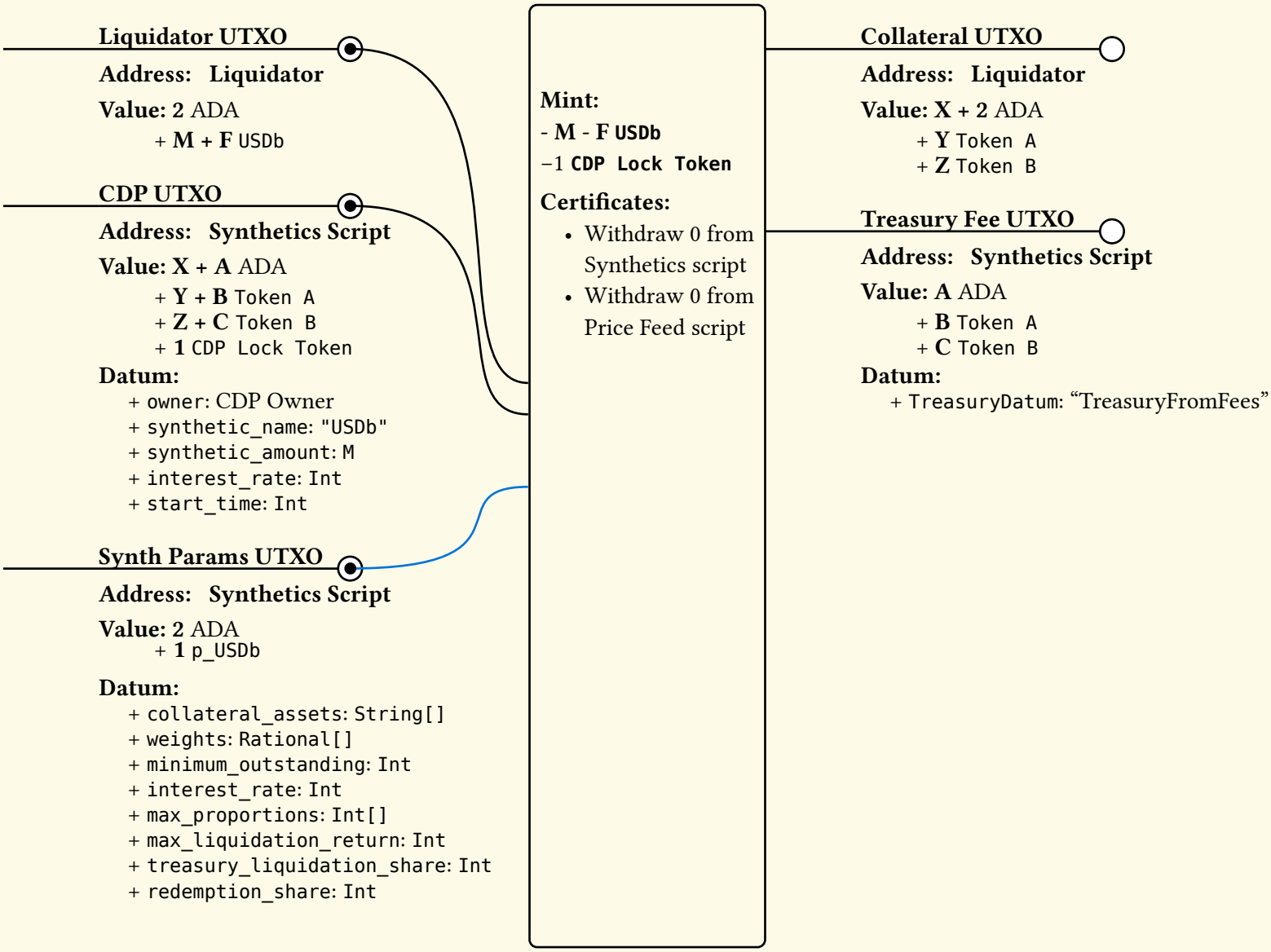

Figure 5: CDP Liquidation Transaction (USDb)

Appendix B. Protocol Definitions

B1. **Validator Definitions**.

Validator Name	Parameters
Btn → Mint	`InitUtxo`: aiken/transaction.OutputReference,
Leftovers → Collect	
PriceFeed → CheckFeed	`VerificationKey`: ByteArray,
State → Spend	

Synthetics → ExternalValidate	`GovNft`: butane/types.AssetClass, `StateScriptHash`: ByteArray, `PriceFeedScriptHash`: ByteArray,
Synthetics → Validate	`FeeToken`: butane/types.AssetClass, `StateScriptHash`: ByteArray, `PriceFeedScriptHash`: ByteArray, `LeftoversScriptHash`: ByteArray, `RedemptionsNft`: butane/types.AssetClass, `ExternScriptHash`: ByteArray, `Salt`: Int,

B2. **Redeemers**.

Validator Name	**Redeemer**
Btn → Mint	btn.Redeemer
Leftovers → Collect	butane/types.CDPCredentialVerifier
PriceFeed → CheckFeed	List<butane/types.Feed<butane/types.PriceFeed, ByteArray>>
State → Spend	Data
Synthetics → ExternalValidate	ByteArray
Synthetics → Validate	butane/types.PolicyRedeemer

B3. **Datums**.

Validator Name	**Datum**
Leftovers → Collect	butane/types.LeftoversDatum
State → Spend	butane/types.LightDatum<butane/types.MonoDatum>

B4. **Definitions**.

Bool := {
  False
| True
}

List<a> $\{x_n \in \text{a}\}_{n=0}^{\infty}$

Option<a> := {
  Some(a)
| None
}

```
aiken/dict.Dict<a, b>  Map<a → b>

aiken/interval.Interval<a>  := {
    Interval {
        lower_bound := aiken/interval.IntervalBound<a>,
        upper_bound := aiken/interval.IntervalBound<a>,
    }
}

aiken/interval.IntervalBound<a>  := {
    IntervalBound {
        bound_type := aiken/interval.IntervalBoundType<a>,
        is_inclusive := Bool,
    }
}

aiken/interval.IntervalBoundType<a>  := {
     NegativeInfinity
   | Finite(a)
   | PositiveInfinity
}

aiken/transaction.Datum  := {
     NoDatum
   | DatumHash(ByteArray)
   | InlineDatum(Data)
}

aiken/transaction.Output  := {
    Output {
        address := aiken/transaction/credential.Address,
        value := aiken/transaction/value.Value,
        datum := aiken/transaction.Datum,
        reference_script := Option<ByteArray>,
    }
}

aiken/transaction.OutputReference  := {
    OutputReference {
        transaction_id := aiken/transaction.TransactionId,
        output_index := Int,
    }
}

aiken/transaction.TransactionId  := {
    TransactionId {
        hash := ByteArray,
    }
}
```

```
aiken/transaction/credential.Address := {
    Address {
        payment_credential := aiken/transaction/credential.Credential,
        stake_credential := Option<aiken/transaction/credential.Referenced<aiken/transaction/
    credential.Credential>>,
    }
}
aiken/transaction/credential.Credential := {
    VerificationKeyCredential(ByteArray)
  | ScriptCredential(ByteArray)
}
aiken/transaction/credential.Referenced<a> := {
    Inline(aiken/transaction/credential.Credential)
  | Pointer {
        slot_number := Int,
        transaction_index := Int,
        certificate_index := Int,
    }
}
aiken/transaction/value.Value Map<ByteArray → aiken/dict.Dict<ByteArray, Int>>
btn.Redeemer := {
    Mint
  | Burn
}
butane/types.AssetClass := {
    AssetClass {
        policy_id := ByteArray,
        asset_name := ByteArray,
    }
}
butane/types.CDPCredential := {
    AuthorizeWithPubKey(ByteArray,ByteArray)
  | AuthorizeWithConstraint(butane/types.Constraint)
}
butane/types.CDPCredentialVerifier := {
    AuthorizingDirectly(butane/types.CDPSubVerifier)
  | AuthorizingOtherWithSignature {
        other := butane/types.ConstraintCredential,
        sub_verifier := butane/types.CDPSubVerifier,
        signature := ByteArray,
    }
}
```

```
butane/types.CDPSubVerifier := {
    AuthorizedWithExtraSigs
  | AuthorizedWithInputsOref(aiken/transaction.OutputReference)
  | AuthorizedWithWithdrawal
}

butane/types.CompatRedeemer := {
    CompatLock {
        oidx := Int,
    }
  | CompatUnlock {
        soidx := Option<Int>,
    }
}

butane/types.Constraint := {
    MustSpendToken(butane/types.AssetClass)
  | MustWithdrawFrom(aiken/transaction/credential.Referenced<aiken/transaction/
    credential.Credential>)
}

butane/types.ConstraintCredential := {
    ConstraintCredential {
        utxo := aiken/transaction.OutputReference,
        interval := aiken/interval.Interval<Int>,
        constraint := butane/types.Constraint,
    }
}

butane/types.FeeType := {
    FeeInSynthetic
  | FeeInFeeToken {
        fee_token_idx := Int,
    }
}

butane/types.Feed<a, b> := {
    Feed {
        data := a,
        extra := b,
    }
}

butane/types.GovAction := {
    NewParamsAuth {
        params := butane/types.Params,
    }
  | UpdateParamsAuth {
        asset := ByteArray,
```

```
        action := butane/types.ParamAction,
      }
    | TreasurySpendAuth {
        out := aiken/transaction.Output,
      }
    | TreasuryMintAuth {
        asset := ByteArray,
        amount := Int,
      }
    | TreasuryCreateDebtAuth {
        debt := butane/types.TreasuryDebt,
      }
    | TextProposalAuth {
        text := ByteArray,
      }
    | TreasuryStakeUpdate {
        delegatee := ByteArray,
      }
    | ExternalScript {
        other_script := ByteArray,
        other_data := Data,
      }
    | GovNewCompat {
        upgrade_policy := ByteArray,
      }
}
butane/types.LeftoversDatum := {
    LeftoversDatum {
      owner := butane/types.CDPCredential,
    }
}
butane/types.LightDatum<a> := {
    LightDatum {
      script_credential := aiken/transaction/credential.Referenced<aiken/transaction/
    credential.Credential>,
      other := butane/types.MonoDatum,
    }
}
butane/types.LightDatumGen<a, b> := {
    LightDatumGen {
      script_credential := a,
      other := a,
    }
}
butane/types.MonoDatum := {
```

```
    ParamsWrapper {
        params := butane/types.Params,
    }
  | CDP {
        owner := butane/types.CDPCredential,
        synthetic_asset := ByteArray,
        synthetic_amount := Int,
        start_time := Int,
    }
  | GovDatum {
        gov := butane/types.GovAction,
    }
  | TreasuryDatum {
        treas := butane/types.TreasuryDatum,
    }
  | CompatLockedTokens
}
butane/types.ParamAction := {
    NewCollateral {
        collateral_asset := butane/types.AssetClass,
        weight_numerator := Int,
        weight_denominator := Int,
    }
  | UpdateWeight {
        collateral_asset_idx := Int,
        weight_numerator := Int,
        weight_denominator := Int,
    }
  | UpdateInterest {
        interest_rate := Int,
    }
  | UpdateMinOutstanding {
        min_outstanding := Int,
    }
  | UpdateMaxProportions {
        max_proportions := List<Int>,
    }
  | UpdateMaxLiquidationReturn {
        max_return := Int,
    }
  | UpdateTreasuryLiquidationShare {
        share := Int,
    }
  | UpdateRedemptionShare {
        share := Int,
    }
```

```
    | UpdateFeeTokenDiscount {
        discount := Int,
      }
}
butane/types.Params := {
    Params {
      collateral_assets := List<butane/types.AssetClass>,
      weights := List<Int>,
      denominator := Int,
      minimum_outstanding_synthetic := Int,
      interest_rates := List<(Int, Int)>,
      max_proportions := List<Int>,
      max_liquidation_return := Int,
      treasury_liquidation_share := Int,
      redemption_share := Int,
      fee_token_discount := Int,
    }
}
butane/types.PolicyRedeemer := {
    SyntheticsMain {
      spends := List<butane/types.SpendAction>,
      creates := List<Int>,
      fee_out_idx := Int,
    }
  | ParamsMint {
      oidx := Int,
    }
  | ParamsUpdate {
      oidx := Int,
      reverse_order := Bool,
    }
  | GovernanceIssue {
      output_idx := Int,
    }
  | ExternalGovernance
  | TreasurySpend {
      expected_output_idx := Int,
      gov_inp_idx := Int,
    }
  | TreasuryMint {
      expected_output_idx := Int,
      gov_inp_idx := Int,
    }
  | TreasuryCreateDebt {
      expected_output_idx := Int,
      gov_inp_idx := Int,
    }
```

```
    | TreasuryRedemption {
        expected_output_idx := Int,
        gov_inp_idx := Int,
      }
    | Compatibility {
        inner := butane/types.CompatRedeemer,
      }
    | BadDebt {
        treasury_out_idx := Int,
      }
}
butane/types.PriceFeed := {
    PriceFeed {
      collateral_prices := List<Int>,
      synthetic := ByteArray,
      denominator := Int,
      validity := aiken/interval.Interval<Int>,
    }
}
butane/types.SpendAction := {
    SpendAction {
      spend_type := butane/types.SpendType,
      params_idx := Int,
      fee_type := butane/types.FeeType,
    }
}
butane/types.SpendType := {
    LiquidateCDP {
      repay_amount := Int,
    }
  | RepayCDP(butane/types.CDPCredentialVerifier)
  | RedeemCDP
}
butane/types.TreasuryDatum := {
    TreasuryWithDebt {
      debt := butane/types.TreasuryDebt,
    }
  | TreasuryFromGenesis
  | TreasuryFromFees
  | TreasuryOnlyRedeem
  | TreasuryOnlySpend
}
butane/types.TreasuryDebt := {
```

```
    TreasuryDebt {
        amount := Int,
        asset := ByteArray,
    }
}
```