Djed: A Formally Verified Crypto-Backed Pegged Algorithmic Stablecoin

Joachim Zahnentferner[1], Dmytro Kaidalov[1], Jean-Frédéric Etienne[1], and Javier Díaz[1 2]

[1] Input Output Global*
{dmytro.kaidalov,jean-frederic.etienne,javier.diaz}@iohk.io
[2] Atix Labs
jdiaz@atixlabs.com

August 17, 2021

Abstract. This paper describes Djed, an algorithmic stablecoin protocol that behaves like an autonomous bank that buys and sells stablecoins for a price in a range that is pegged to a target price. It is crypto-backed in the sense that the bank keeps a volatile cryptocurrency in its reserve. The reserve is used to buy stablecoins from users that want to sell them. And revenue from sales of stablecoins to users are stored in the reserve. Besides stablecoins, the bank also trades reservecoins in order to capitalize itself and maintain a reserve ratio significantly greater than one. To the best of our knowledge, this is the first stablecoin protocol where stability claims are precisely and mathematically stated and proven. Furthermore, the claims and their proofs are formally verified using two different techniques: bounded model checking, to exhaustively search for counter-examples to the claims; and interactive theorem proving, to build rigorous formal proofs using a proof assistant with automated theorem proving features.

1 Introduction

In the narrowest sense, a *stablecoin* is a cryptocurrency that has its price pegged to a fiat currency (e.g. **USD**) and is fully backed by reserves denominated in the same fiat currency. More broadly, a *stablecoin* can be defined as a digital asset that has mechanisms to maintain a low deviation of its price from a target price. Since

* We would like to thank Alexander Chepurnoy, Amitabh Saxena, Nicolas Arqueros and Robert Kornacki for discussions of this stablecoin protocol and for its implementation in ErgoScript; and we are grateful to the Ergo community for deploying this implementation on the Ergo blockchain. We are thankful to Alan McSherry, Ferdi Kurt and Jann Müller for their ongoing implementations in, respectively, OpenStar, Solidity and Plutus.

the invention of bitcoin and other cryptocurrencies, a major obstacle for their wider adoption and recognition as proper currencies, particularly as means of exchange and units of account, has been the instability (volatility) of their price in relation to fiat currencies. Stablecoins aim to overcome this obstacle.

Various mechanisms can contribute to a stablecoin's stability. Ultimately, they are all grounded on the basic economic principles of supply and demand. If demand for buying/selling stablecoins is higher than the current supply of sale/purchase orders, this supply must be increased to avoid an increase/decrease in the stablecoin's price.

Backing the stablecoins by reserves and using these reserves to actively buy and sell stablecoins for prices close to the target price is currently the most common mechanism. This mechanism is not exclusive to stablecoins; it can be seen in pegged national currencies such as the **HKD** (Hong Kong Dollar). Centrally operated fiat-pegged fiat-backed stablecoins such as **USDT** and **USDC** use variations of this mechanism that may differ, for instance, on the actual composition of the reserves and with whom the operator interacts to buy and sell stablecoins. As long as the stablecoin is fully backed by reserves in the currency to which it is pegged and the operator can react quickly enough to variations in the demand, it is easy to see that stability will be guaranteed. Typically, the reserves will not be kept all in cash in a vault, safe or bank account, but rather at least partly in interest-bearing financial instruments such as bonds. The returns from such investments provide revenue for the operator. The risks associated with these investments may imply that the stablecoin may eventually lose its full-backing, compromising the stability in the long-term. Lack of liquidity of these investments may cause the operator to be unable to react quickly enough to changes in demand, compromising the stability in the short-term. Another source of revenue are the fees or spread practiced when buying and selling the stablecoin. For example, if the operator sells **USDT** for 1.005 **USD** and buys **USDT** for 0.995 **USD**, it has a revenue of 1 cent for every **USDT** that it buys and then sells, while keeping the price stable within the range from 0.995 and 1.005.

The main drawback of fiat-backed stablecoins is that they require trust on the entities keeping the reserves. This is not only a theoretical concern. Lack of transparency about the reserves and skepticism about its full-backing claim, combined with inefficient stabilization measures by Tether, have actually already caused **USDT** to trade for at least as low as 0.91 USD.

Interestingly, issues related to transparency of the reserves do not arise when the backing asset is a cryptocurrency on a public blockchain. Furthermore, issues related to the inefficient and unreliable execution of stabilization measures can be eliminated by implementing the stabilization mechanisms as smart contracts that are automatically, reliably and transparently executed.

This paper describes Djed: a crypto-backed algorithmic stablecoin contract. Djed acts as an autonomous bank, keeping a *reserve* R of *BaseCoins* (BCs), and minting and burning *StableCoins* (SCs) and *ReserveCoins* (RCs). It maintains the peg of the SCs to a target price by buying and selling SCs, using its reserve. While doing so, it charges fees and accumulates them in its reserve. The beneficiaries of this revenue stream are ultimately the RC holders, who contribute with additional funds to the reserve and take the risk of price fluctuation.

First a simpler version, Minimal Djed, is defined in Section 2. This version is designed to be as intuitive and straightforward as possible, while still being stable, safe and secure, as shown in Section 3. However, as discussed in Section 4, it still suffers from some non-critical minor issues. A more complex version, Extended Djed, is defined in Section 5. It addresses some of the known minor issues from Minimal Djed, but is

more complex. Several stability properties are stated and proven as theorems in Section 3 . Sections 6 and 7 describe and discuss the formal verification of the theorems and proofs using, respectively, model checking and interactive theorem proving techniques. Current implementations of Djed are briefly discussed in Section 8. Section 9 discusses related work, with a focus on three algorithmic stablecoins that were influential in the design of Djed.

2 Minimal Djed

The *target price* of an SC is denoted P^t_{SC}. For example[3], in the case of a stablecoin pegged to some *peg currency* PC (e.g. EUR, USD, ...):

$$P^t_{SC} = X^{PC}_{BC}\ BC \tag{1}$$

where X^{PC}_{BC} is the price of 1 unit of the peg currency in BCs.

Because Djed's reserve may be insufficient to buy back all stablecoins for the target price, Djed sets the actual *price* P_{SC} of SCs according to the following equation:

$$P_{SC} = \begin{cases} P^t_{SC} & \text{if } N_{SC} = 0 \\ min(P^t_{SC}, \frac{R}{N_{SC}}) & \text{otherwise} \end{cases} \tag{2}$$

where N_{SC} is the number of stablecoins in circulation.

The portion of Djed's reserve that would need to be used to buy back all stablecoins is known as its *liabilities*:

$$L(N_{SC}) = N_{SC} P_{SC} \tag{3}$$

Because P_{SC} is volatile, Djed strives to keep a high *reserve ratio*:

$$r(R, N_{SC}) = \frac{R}{L(N_{SC})} \tag{4}$$

Djed does so by having a *minimum reserve ratio* r_{min} and disallowing users from buying SCs or selling back RCs if, after these actions, $r(R, N_{SC}) < r_{min}$. To prevent dilution for the RC holders, Djed also has a *maximum reserve ratio* r_{max} and disallows users from buying more RCs if, before or after the purchase, $r(R, N_{SC}) > r_{max}$; unless $N_{SC} < N^*_{SC}$, where N^*_{SC} is the *threshold number of stablecoins* parameter[4]. Note that, even though

[3] Djed does not need to be pegged to a fiat currency. It just needs a target price. The target price could be a weighted average of the price of a volatile asset, a stock index, an inflation index, ...

[4] The main purpose of this parameter is to allow users to buy reservecoins, and hence contribute to the quick growth of the reserve, soon after initialization. If it were not for this, it is easy to see that users would be able to buy neither stablecoins nor reservecoins right after initialization when $N_{SC} = 0$ and $N_{RC} = 0$. Moreover, this parameter also allows a quick re-initialization in the unlikely event that all stablecoins and reservecoins are sold back and a state where $N_{SC} = 0$ and $N_{RC} = 0$ is reached again.

Djed disallows some types of purchases and sales when the reserve ratio is above the maximum or below the minimum, the reserve ratio may still go above the maximum or below the minimum due to price fluctuations.

The reserve surplus is Djed's *equity*:

$$E(R, N_{SC}) = R - L(N_{SC}) \tag{5}$$

Djed's equity is shared equally among RC holders, and thus the *target price* of RCs is:

$$P^t_{RC}(R, N_{SC}, N_{RC}) = \frac{E(R, N_{SC})}{N_{RC}} \tag{6}$$

where N_{RC} is the number of RCs in circulation.

However, the target price is undefined when $N_{RC} = 0$ and a price equal to 0 when $E(R, N_{SC}) = 0$ would be problematic, because users would be able to buy an arbitrary number of RCs without any cost. Therefore, Djed sets the actual *buying price* P^b_{RC} according to the following equation:

$$P^b_{RC}(R, N_{SC}, N_{RC}) = \begin{cases} max(P^t_{RC}, P^{min}_{RC}) & \text{if } P^t_{RC} \text{ is defined} \\ P^{min}_{RC} & \text{otherwise} \end{cases} \tag{7}$$

where P^{min}_{RC} is a parameter of Djed.

From the user's point of view, there are 4 actions:

Action	**User Sends**	**User Receives**	**Condition**[5][6]
Buy SCs	$n(1 + fee)P_{SC}$	n SCs	$r(R, N_{SC}) \geq r_{min}$
Sell SCs	n SCs	$n(1 - fee)P_{SC}$	
Buy RCs	$n(1 + fee)P^b_{RC}$	n RCs	$r(R, N_{SC}) \leq r_{max}$ or $N_{SC} < N^*_{SC}$
Sell RCs	n RCs	$n(1 - fee)P^t_{RC}$	$r(R, N_{SC}) \geq r_{min}$

Djed has 3 state variables (R, N_{SC}, N_{RC}) and 5 parameters (r_{min}, r_{max}, fee, N^*_{SC}, P^{min}_{RC}). The parameters are assumed to be set to values that satisfy the following constraints: $r_{min} > 1 + fee$; $r_{max} \geq r_{min}$; $0 < fee \leq 1$; $N^*_{SC} > 0$; $P^{min}_{RC} > 0$.

Djed also depends on one (and only one) external variable: the oracle exchange rate X^{PC}_{BC}. The models and formalizations assume that $X^{PC}_{BC} > 0$. This is a reasonable assumption[7], because $X^{PC}_{BC} = 0$ (or, worse, $X^{PC}_{BC} < 0$) would imply that the value of the peg currency has collapsed. In such cases, a stablecoin pegged to it would be worthless anyway.

[5] The conditions must hold *before* and *after* the actions.

[6] The conditions in the table are stated in a way to ease readability and intuitive understanding. However, note that the calculation of the reserve ratio potentially involves a division by zero. Implementations should circumvent this issue by using the standard technique of multiplying both sides of the inequation by the denominator of the reserve ratio. Thus, for instance, the condition $r(R, N_{SC}) \geq r_{min}$ should be understood and implemented as $R \geq L(N_{SC})r_{min}$.

[7] In practice, depending on how much the implementation trusts the oracle, it would be important to check this assumption when receiving data from the oracle.

3 Stability Properties

Minimal Djed enjoys several stability properties. The first one is that, in the normal reserve ratio range where purchases and sales are not restricted[8], users have no incentive to trade stablecoins outside the peg range in a secondary market.

Theorem 1 (Peg Maintenance - Upper Bound). *If $r(R, N_{SC}) > r_{min} + \epsilon$ (for a sufficiently large ϵ) and a user u wants to sell a stablecoin in the secondary market for a price P such that $P > (1 + fee)P^t_{SC}$, then there is no rational user u^* who would buy from u.*

Proof. Assume, for the sake of contradiction, that such a rational u^* exists. Then u^* would have had two options: to buy a stablecoin from u for P; or to buy a stablecoin directly from the bank for P' where $P' = (1 + fee)P_{SC}$. Note that the second option is available because, for a sufficiently large ϵ, the post-action condition for the action of buying stablecoins from the bank holds. By definition of P_{SC}, $P' < (1 + fee)P^t_{SC}$. Therefore, buying directly from the bank for P' would have been less costly than buying from u for P, and would have been the preferred option for a rational user. Hence u^* is irrational. □

Theorem 2 (Peg Maintenance - Lower Bound). *If $r(R, N_{SC}) > 1$ and a user u wants to buy a stablecoin in the secondary market for a price P such that $P < (1 - fee)P^t_{SC}$, then there is no rational user u^* who would sell to u.*

Proof. Assume, for the sake of contradiction, that such a rational u^* exists. Then u^* would have had two options: to sell a stablecoin to u for P; or to sell a stablecoin directly to the bank for P' where $P' = (1 - fee)P_{SC}$. By definition of P_{SC} and the fact that $r(R, N_{SC}) > 1$, $P' = (1 - fee)P^t_{SC}$. Therefore, selling directly to the bank for P' would have been more profitable than selling to u for P, and would have been the preferred option for a rational user. Hence u^* is irrational. □

The second stability property is that stablecoins remain pegged despite market crashes up to a magnitude that depends on the reserve ratio.

Theorem 3 (Peg Robustness during Market Crashes). *If $N_{SC} > 0$, r is the current reserve ratio, x and x' are the exchange rates before and after the crash, $r > 1$, then the bank can tolerate a basecoin price crash of $\frac{r-1}{r}$ and the new stablecoin price would still be P^t_{SC}.*

Proof. By the definition of P_{SC}, the peg is maintained as long as $P^t_{SC} \leq \frac{R}{N_{SC}}$. By the definition of P^t_{SC}, reserve ratio and liabilities, this inequation can be simplified to $x' \leq rx$. Since x and x' are the exchange rates of 1 unit of pegged currency in BCs, the exchange rates of 1 BC in pegged currency are $\frac{1}{x}$ and $\frac{1}{x'}$. The inequation can then be rewritten as $y' \geq \frac{y}{r}$. Therefore, $\frac{y-y'}{y} \leq \frac{r-1}{r}$. □

Another crucial property is that the bank never becomes insolvent, which is a condition defined by having negative equity.

[8] In practice, there are other factors beyond the reserve ratio that may restrict interaction of the user with the bank, such as blockchain congestion and high blockchain transaction fees. Such factors would have an effect on the ability of the bank to maintain the peg range in secondary markets.

Theorem 4 (No Insolvency). *In all bank states and for any exchange rate,* $E(R, N_{SC}) \geq 0$.

Proof. By definition, $E(R, N_{SC}) = R - L(N_{SC}) = R - N_{SC}P_{SC}$. By the definition of P_{SC}, there are 3 cases to consider:

- $N_{SC} = 0$: in this case, $E(R, N_{SC}) = R - 0P^t_{SC} = R$ and $R \geq 0$.
- $N_{SC} \neq 0$ and $\frac{R}{N_{SC}} \leq P^t_{SC}$: in this case, $E(R, N_{SC}) = R - N_{SC}\frac{R}{N_{SC}} = 0$.
- $N_{SC} \neq 0$ and $\frac{R}{N_{SC}} > P^t_{SC}$: in this case, $E(R, N_{SC}) = R - N_{SC}P^t_{SC}$. Since $\frac{R}{N_{SC}} > P^t_{SC}$, it must be the case that $E(R, N_{SC}) > R - N_{SC}\frac{R}{N_{SC}}$. Therefore, $E(R, N_{SC}) > 0$.

□

Another important property is that, provided that the exchange remains constant, the bank is never in a state susceptible to *bank runs* where stablecoin holders would feel incentivized to race against each other to sell their stablecoins.

Theorem 5 (No Bank Runs for Stablecoins). *Let* u_1 *and* u_2 *be two stablecoin holders. Let* P^1_{SC} *be the price of stablecoins obtained by* u_1 *by selling its stablecoins back to the bank. Let* P^2_{SC} *be the price of stablecoins obtained by* u_2 *by selling its stablecoins back to the bank. Assume, without loss of generality, that* u_2*'s sale occurs after* u_1*'s sale. Then, in all bank states and for any constant exchange rate,* $P^2_{SC} \geq P^1_{SC}$.

Proof. Firstly, note that the sale of stablecoins back to the bank is never restricted by a post-action condition. Therefore, it cannot be the case that the sale by u_1 would bring the bank to a state where u_2's sale would be blocked. Let q be the quantity of stablecoins sold by u_1 and R_1 be the reserve after the sale by u_1. Then, following the definition of P_{SC}, there are two cases to consider:

- $P^1_{SC} = P^t_{SC}$: in this case, $P^t_{SC} \leq \frac{R}{N_{SC}}$ and $R_1 = R - q(1 - fee)P^t_{SC}$. Then:

$$P^2_{SC} = min\left(P^t_{SC}, \frac{R - q(1 - fee)P^t_{SC}}{N_{SC} - q}\right)$$

  Using the fact that $P^t_{SC} \leq \frac{R}{N_{SC}}$, we have that:

$$\frac{R - q(1 - fee)P^t_{SC}}{N_{SC} - q} \geq \frac{R - q(1 - fee)\frac{R}{N_{SC}}}{N_{SC} - q}$$

  Simplifying, we obtain:

$$\frac{R - q(1 - fee)P^t_{SC}}{N_{SC} - q} \geq \frac{R}{N_{SC}}\frac{N_{SC} - q + qfee}{N_{SC} - q}$$

  And hence:

$$\frac{R - q(1 - fee)P^t_{SC}}{N_{SC} - q} \geq \frac{R}{N_{SC}}$$

  Therefore:

$$P^2_{SC} = P^t_{SC} = P^1_{SC}$$

- $P^1_{SC} = \frac{R}{N_{SC}}$: in this case, $P^t_{SC} \geq \frac{R}{N_{SC}}$ and $R_1 = R - q(1 - fee)\frac{R}{N_{SC}}$. Then:

$$P^2_{SC} = min\left(P^t_{SC}, \frac{R - q(1 - fee)\frac{R}{N_{SC}}}{N_{SC} - q}\right)$$

  We now have two cases:

- $P^2_{SC} = P^t_{SC}$: in this case, $P^2_{SC} > P^1_{SC}$.
- $P^2_{SC} = \frac{R - q(1-fee)\frac{R}{N_{SC}}}{N_{SC}-q}$: in this case, simplifying, we have $P^2_{SC} = \frac{R}{N_{SC}}\frac{N_{SC}-q+qfee}{N_{SC}-q}$ and therefore $P^2_{SC} \geq P^1_{SC}$.

□

The following theorem shows that, provided that the exchange rate remains constant, the equity per reservecoin always increases.

Theorem 6 (Monotonically Increasing Equity per Reservecoin). *Assuming that the exchange rate remains constant and $N_{RC} > 0$, for every action a, $\frac{E(R^a, N^a_{SC})}{N^a_{RC}} \geq \frac{E(R, N_{SC})}{N_{RC}}$, where R^a, N^a_{SC}, $N^a_{RC} > 0$ are respectively, the reserve, number of stablecoins and number of reservecoins after action a.*

Proof. There are 4 cases:

- **Buy n SCs**. In this case, $\frac{E(R^a, N^a_{SC})}{N^a_{RC}} = \frac{R + n(1+fee)P_{SC} - (N_{SC}+n)P^a_{SC}}{N_{RC}}$. By the assumption of constant exchange rate and given that $P^a_{SC} = P_{SC}$ (due to restriction of the buying operation if $P_{SC} < P^t_{SC}$) we have that $\frac{E(R^a, N^a_{SC})}{N^a_{RC}} = \frac{R - N_{SC}P_{SC} + nfeeP_{SC}}{N_{RC}} > \frac{R - N_{SC}P_{SC}}{N_{RC}} = \frac{E(R, N_{SC})}{N_{RC}}$.
- **Sell n SCs**. In this case, $\frac{E(R^a, N^a_{SC})}{N^a_{RC}} = \frac{R - n(1-fee)P_{SC} - (N_{SC}-n)P^a_{SC}}{N_{RC}}$. It is easy to see that inequality $\frac{R - n(1-fee)P_{SC} - (N_{SC}-n)P^a_{SC}}{N_{RC}} \geq \frac{R - N_{SC}P_{SC}}{N_{RC}} = \frac{E(R, N_{SC})}{N_{RC}}$ holds if $n(1-fee)P_{SC} + (N_{SC}-n)P^a_{SC} \leq N_{SC}P_{SC}$. We will prove the latter for 2 different cases:
  - If $P_{SC} = \frac{R}{N_{SC}} \leq P^t_{SC}$: given that $P^a_{SC} \leq \frac{R^a}{N^a_{SC}}$, we can define $P^a_{SC} = \frac{R^a}{N^a_{SC}}\gamma$ for some $\gamma \in (0, 1]$. Provided that $R^a = R - n(1-fee)\frac{R}{N_{SC}}$ and doing substitutions we have:

$$n(1-fee)\frac{R}{N_{SC}} + (N_{SC}-n)\frac{R - n(1-fee)\frac{R}{N_{SC}}}{N_{SC}-n}\gamma \leq N_{SC}\frac{R}{N_{SC}},$$

$$\frac{n(1-fee)}{N_{SC}}(1-\gamma) \leq 1-\gamma,$$

    which holds due to $N_{SC} > n(1-fee)$.
  - If $P_{SC} < \frac{R}{N_{SC}}$: first let's show that $\frac{R}{N_{SC}} < \frac{R^a}{N^a_{SC}}$. Given that $P_{SC} < \frac{R}{N_{SC}}$ we can define $P_{SC} = \frac{R}{N_{SC}}\sigma$ for some $\sigma \in (0, 1)$. Hence, $\frac{R^a}{N^a_{SC}} = \frac{R - n(1-fee)\frac{R}{N_{SC}}\sigma}{N_{SC}-n}$. Then we have:

$$\frac{R}{N_{SC}} < \frac{R - n(1-fee)\frac{R}{N_{SC}}\sigma}{N_{SC}-n}, \qquad 1 < \frac{N_{SC} - n(1-fee)\sigma}{N_{SC}-n},$$

    which is true due to $n > n(1-fee)\sigma$. From $\frac{R}{N_{SC}} < \frac{R^a}{N^a_{SC}}$ follows that $P_{SC} = P^a_{SC} = P^t_{SC}$, hence we have that:

$$n(1-fee)P^t_{SC} + (N_{SC}-n)P^t_{SC} \leq N_{SC}P^t_{SC},$$

    which holds due to $N_{SC} - n + n(1-fee) \leq N_{SC}$.

– **Buy** n **RCs**. First let's show that $R < R^a$. It is easy to see given that $R^a = R + n(1 + fee)P^b_{RC}$, where $P^b_{RC} \geq P^{min}_{RC} > 0$. Then, by definition of P_{SC}, it holds that $P_{SC} = min(P^t_{SC}, \frac{R}{N_{SC}}) \leq P^a_{SC} = min(P^t_{SC}, \frac{R^a}{N_{SC}})$. Now we will prove the theorem for 2 different cases:
  - If $P_{SC} = \frac{R}{N_{SC}} \leq P^t_{SC}$: in this case $P^a_{SC} \leq \frac{R^a}{N^a_{SC}}$, hence we can define $P^a_{SC} = \frac{R^a}{N^a_{SC}}\gamma$ for some $\gamma \in (0, 1]$. Note that from $P_{SC} = \frac{R}{N_{SC}}$ follows that $E(R, N_{SC}) = R - N_{SC}\frac{R}{N_{SC}} = 0$. Making substitutions we have:

$$\frac{E(R^a, N^a_{SC})}{N^a_{RC}} = \frac{R^a - N^a_{SC}P^a_{SC}}{N^a_{RC}} \geq \frac{E(R, N_{SC})}{N_{RC}},$$

$$\frac{R + n(1 + fee)P^b_{RC} - N_{SC}\frac{R+n(1+fee)P^b_{RC}}{N_{SC}}\gamma}{N_{RC} + n} \geq 0,$$

$$\frac{(R + n(1 + fee)P^b_{RC})(1 - \gamma)}{N_{RC} + n} \geq 0,$$

    which holds due to $(1 - \gamma) \geq 0$.
  - If $P_{SC} < \frac{R}{N_{SC}}$: in this case $P_{SC} = P^a_{SC} = P^t_{SC}$. By definition $P^b_{RC} \geq \frac{R - N_{SC}P_{SC}}{N_{RC}}$ and given that $R - N_{SC}P_{SC} > 0$ we can define $P^b_{RC} = \frac{R - N_{SC}P_{SC}}{N_{RC}}\alpha$ for some $\alpha \geq 1$. Making substitutions we have:

$$\frac{E(R^a, N^a_{SC})}{N^a_{RC}} = \frac{R + n(1 + fee)P^b_{RC} - N_{SC}P_{SC}}{N_{RC} + n} \geq \frac{R - N_{SC}P_{SC}}{N_{RC}} = \frac{E(R, N_{SC})}{N_{RC}},$$

$$\frac{R - N_{SC}P_{SC}}{N_{RC} + n} + \frac{n(1 + fee)\frac{R - N_{SC}P_{SC}}{N_{RC}}\alpha}{N_{RC} + n} \geq \frac{R - N_{SC}P_{SC}}{N_{RC}},$$

$$\frac{N_{RC}}{N_{RC} + n} + \frac{n(1 + fee)\alpha}{N_{RC} + n} \geq 1, \qquad \frac{N_{RC} + n(1 + fee)\alpha}{N_{RC} + n} \geq 1,$$

    which holds given that $\alpha \geq 1$.
– **Sell** n **RCs**. Given that $R^a = R - n(1 - fee)P^t_{RC}$, where $P^t_{RC} = \frac{R - N_{SC}P_{SC}}{N_{RC}} \geq 0$, it follows that $R \geq R^a$. Then, by definition of P_{SC}, it holds that $P_{SC} = min(P^t_{SC}, \frac{R}{N_{SC}}) \geq P^a_{SC} = min(P^t_{SC}, \frac{R^a}{N_{SC}})$. Note that the Sell RC operation is allowed only if $r(R^a, N^a_{SC}) \geq r_{min}$, hence $P^a_{SC} = P^t_{SC}$ and $P_{SC} = P^t_{SC}$. Making substitutions we get:

$$\frac{E(R^a, N^a_{SC})}{N^a_{RC}} = \frac{R - n(1 - fee)P^t_{RC} - N_{SC}P^t_{SC}}{N_{RC} - n} \geq \frac{R - N_{SC}P^t_{SC}}{N_{RC}} = \frac{E(R, N_{SC})}{N_{RC}},$$

$$\frac{R - N_{SC}P^t_{SC}}{N_{RC} - n} - \frac{n(1 - fee)\frac{R - N_{SC}P^t_{SC}}{N_{RC}}}{N_{RC} - n} \geq \frac{R - N_{SC}P^t_{SC}}{N_{RC}},$$

$$\frac{N_{RC}}{N_{RC} - n} - \frac{n(1 - fee)}{N_{RC} - n} \geq 1, \qquad \frac{N_{RC} - n(1 - fee)}{N_{RC} - n} \geq 1,$$

  which holds due to $n(1 - fee) \leq n$.

□

As a corollary, the bank is therefore never subject to *draining* of its reserves by malicious users, provided that the exchange rate remains constant.

Theorem 7 (No Reserve Draining). *Assuming that the exchange rate remains constant, for any initial bank state* $(R^0, N^0_{SC}, N^0_{RC})$*, there is no sequence of actions* $a_1, a_2, \ldots, a_n$ *that would bring the bank to a state* $(R^n, N^n_{SC}, N^n_{RC})$ *such that* $R^n < R^0$*,* $N^n_{SC} = N^0_{SC}$ *and* $N^n_{RC} = N^0_{RC}$*, assuming* $N^i_{SC} > 0, N^i_{RC} > 0$ *for all* $i \in \{0..n-1\}$*.*

Proof. We will consider two separate cases:

- If $E(R^0, N^0_{SC}) > 0$: assume, for the sake of contradiction, that such a sequence exists. Then note that the equity per reservecoin after the sequence of actions would be smaller than the equity per reservecoin before the sequence of actions. Therefore, the assumption leads to a contradiction with Theorem 6.
- If $E(R^0, N^0_{SC}) = 0$: note that in this case $P^0_{SC} = \frac{R^0}{N^0_{SC}}$. Let's first prove that for any action a it holds that $\frac{R^a}{N^a_{SC}} \geq \frac{R^0}{N^0_{SC}}$. There are 4 different actions to consider:
  - *Sell RCs* and *Buy SCs* are disallowed if $E(R^0, N^0_{SC}) = 0$ (due to $r < r_{min}$).
  - *Buy RCs*: in this case $R^a > R^0$ and $N^a_{SC} = N^0_{SC}$, hence $\frac{R^a}{N^a_{SC}} > \frac{R^0}{N^0_{SC}}$ holds.
  - *Sell SCs*: in this case $R^a = R^0 - n(1-fee)\frac{R^0}{N^0_{SC}}$, where n is the number of sold SCs, hence

$$\frac{R^0 - n(1-fee)\frac{R^0}{N^0_{SC}}}{N^0_{SC} - n} \geq \frac{R^0}{N^0_{SC}}, \qquad \frac{N^0_{SC} - n(1-fee)}{N^0_{SC} - n} \geq 1,$$

    which holds due to $n(1-fee) \leq n$.

  Now, let assume, for the sake of contradiction, that there is a sequence $a_1, a_2, \ldots, a_n$ such that $R^n < R^0$, $N^n_{SC} = N^0_{SC}$ and $N^n_{RC} = N^0_{RC}$. In this case $\frac{R^n}{N^0_{SC}} < \frac{R^0}{N^0_{SC}}$, which contradicts the just proved statement.

  Note that if the sequence contains $a_i, i \in [1, n]$ such that equity becomes $E(R^i, N^i_{SC}) > 0$, any further $a^j, j > i$ would make it zero again as it contradicts Theorem 6. Hence, $R^n < R^0$ and $N^n_{SC} = N^0_{SC}$ is impossible in this case due to $E(R^n, N^n_{SC}) > E(R^0, N^0_{SC})$.

□

And, finally, the dilution to which reservecoin holders may be subject is bounded.

Theorem 8 (Bounded Dilution). *Assuming that* $N_{SC} \geq N^*_{SC}$ *and provided that the exchange rate and the number of stablecoins* $N_{SC} > 0$ *remains constant, if* P^b_{RC} *is the current price of reservecoins,* $N_{RC} > 0$ *is the current number reservecoins and* R *is the current reserve, then the maximum number of RCs that can be bought additionally is* $\frac{r_{max} N_{SC} P^t_{SC} - R}{(1+fee)P^b_{RC}}$ *(or* 0 *if this number is negative).*

Proof. Let's consider an operation a to buy n reservecoins such that $r^a = r_{max}$ is the reserve ratio after the operation. Given that $r^a = \frac{R^a}{N_{SC} P^a_{SC}}$, where $R^a = R + n(1+fee)P^b_{RC}$, we can find the maximum number n of RCs to be bought from the equation $\frac{R+n(1+fee)P^b_{RC}}{N_{SC}P^a_{SC}} = r_{max}$. Provided that $r^a = r_{max} > 1$ it follows that $P^a_{SC} = P^t_{SC}$ and, thus, $n = \frac{r_{max} N_{SC} P^t_{SC} - R}{(1+fee)P^b_{RC}}$. Note that splitting operation a into two (or more) operations will not yield more reservecoins as the price P^b_{RC} monotonically grows for every operation (see Theorem 6). □

4 Known Minor Issues of Minimal Djed

The design of Minimal Djed described in Section 2 is the result of a non-trivial compromise between attempting to satisfy many desirable properties and taming the complexity that tends to arise from such an attempt. The outcome is an intentionally simple solution that addresses major stability concerns, as demontrated in Section 3. However, this simple solution is susceptible to a few known minor issues. These minor issues are addressed in Extended Djed, described in Section 5, which is significantly more complex, but retains the same stability principles as Minimal Djed.

The known minor issues are:

- **Reserve Draining Attack with Price Foresight:** a variation of Theorem 7 dropping the assumption of constant exchange rate does not hold. A malicious user who can foresee how the exchange rate will evolve, perhaps because of an excessive oracle delay or active price manipulation, can perform sequences of actions that will drain the bank's reserves.
- **Wholesale Discount:** in Minimal Djed the price is fixed before the action. But every action changes the balance of reserves and SCs/RCs and hence affects the future price. Therefore, the total price paid (or received) for a given quantity of SCs or RCs depends on how this quantity is bought or sold. For example, assuming constant exchange rate, buying 10 RCs at once is cheaper than buying 10 RCs in two consecutive purchase of 5 RCs each.
- **Zero equity:** when the reserve ratio falls to one, the equity falls to zero making the target price of RCs also zero. To avoid purchase of an unlimited number of RCs with price equal to zero, a minimal price was introduced. The problem of this simple fix is that the artificially set minimal price might be inconsistent with the market price, discouraging users from buying RCs in times it is most needed. For example, if $P^t_{RC} = 5$ BCs and $P^{min}_{RC} = 10$ BCs, then users might not want to pay 10 BCs for a coin that is backed by 5 BCs.
- **Rigid Fees:** the pricing model doesn't allow to increase or decrease fees smoothly to encourage operations that drive the reserve ratio to an optimal level and to discourage operations that drive it away from the optimum.
- **"Haircut" for Stablecoin Holders:** when the peg is lost, the SC holders suffer financial losses. Minimal Djed does not have mechanisms to cover these losses.
- **RC Bank Runs:** the analogue of Theorem 5 for reservecoins does not hold. When the reserve ratio is close to r_{min}, RC holders may feel encouraged to race against each other to sell their RCs, because every RC sale brings the reserve ratio closer to r_{min} and further sales of RC would be blocked when r_{min} is reached, and they do not want to be blocked.

5 Extended Djed

Section 4 raises a number of minor issues with the Minimal Djed construction. In this section we present Extended Djed aiming to overcome those issues. The core principles remain the same: the contract keeps a reserve R of basecoins allowing minting and burning stablecoins and reservecoins. The difference with Minimal Djed is the modified pricing model that introduces additional features helping to solve identified problems.

The same notations and definitions are used as in Section 2 except a few differences. Previously, the liabilities were defined as $L(N_{SC}) = N_{SC}P_{SC}$ (Eq. (3)). In this

section we will call $L(N_{SC})$ normalized liabilities. In contrast, we define target liabilities $L^t(N_{SC}) = N_{SC}P^t_{SC}$, which represent the full amount of debt owed to stablecoins holders if the bank were to buy back stablecoins for the target price instead of the actual price. Importantly, the reserve ratio is redefined as $r = \frac{R}{L^t}$. Additionally, we introduce system parameters r_{opt} and r_{peg} ($r_{opt} > r_{peg} \geq 1$) denoting, correspondingly, the optimal reserve ratio and the minimal reserve ratio for which the stablecoin's price peg holds. Finally, fee_0 defines the base fee in the system (analogous to *fee* in Minimal Djed).

In the following sections we often omit arguments of functions and simply write, for instance, L^t or L for better readability.

We start by defining equations for the *nominal price* of stablecoins and reservecoins. These equations are the base for pricing specific operations. The nominal price shows amount of basecoins worth of one stablecoin or one reservecoin respectively.

The nominal price of a stablecoin is defined as follows:

$$P_{SC} = k \cdot P^t_{SC}, \qquad k = \min(1, \frac{r}{r_{peg}}), \tag{8}$$

where $k \in (0, 1]$ is a coefficient that maintains the peg between the value of the stablecoin and corresponding pegged currency. When $k = 1$, stablecoins are worth their intended target price. If $r < r_{peg}$, k becomes less than one meaning that at this point the peg is lost. If r_{peg} is configured to be more than one, the stablecoins start to lose their peg even before the bank becomes under-capitalized.

The nominal price of a reservecoin is defined as follows:

$$P_{RC} = \frac{R - L}{N_{RC}}, \qquad L = N_{SC}P_{SC}, \tag{9}$$

where L is normalized liabilities. Note that $L = k \cdot L^t$.

Given that $E = R - L$ is the equity of the bank shared among reservecoin holders, we can see that once the reserve ratio falls below r_{peg}, a part of the target liabilities L^t (determined by k) is converted to equity. This is done to prevent equity and, correspondingly, the reservecoin price falling to zero when the peg is lost eliminating the need to introduce a minimal price for reservecoins. Fig. 1 shows the dependence of the stablecoin and reservecoin nominal prices on the reserve ratio. As can be seen, the dependence is refracted at point $r = r_{peg}$ so that the reservecoin price approaches zero gradually as r approaches zero.

The ultimate goal of the pricing model is to incentivize users to keep the reserve ratio at an optimal level. In this case, the system is considered to be at an equilibrium that provides robust stability for stablecoin holders and attractive rewards for reservecoin holders. To achieve this goal, the extended model introduces dynamic fees that are increased for operations that shift the reserve ratio away from the optimum and decreased for operations that bring it closer to the optimum.

Recall that there are four types of operations:

1. **Buy SCs** – mints new stablecoins increasing reserves but decreasing reserve ratio.
2. **Sell SCs** – burns stablecoins paying back from reserves; increases the reserve ratio.
3. **Buy RCs** – mints new reservecoins increasing both reserves and reserve ratio, but dilutes relative shares of existing reservecoin holders.
4. **Sell RCs** – burns reservecoins paying back from reserves; decreases the reserve ratio and enlarges relative shares of existing reservecoin holders.

The following subsections define precise prices for particular types of operations.

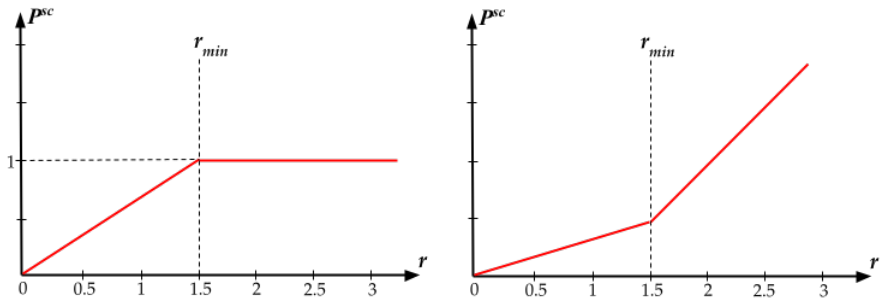

Fig. 1. Stablecoin (on the left) and reservecoin (on the right) prices depending on the reserve ratio. The price is normalized by the target price (i.e., $P^{sc} = \frac{P_{SC}}{P^t_{SC}}$, $P^{rc} = \frac{P_{RC}}{P^t_{SC}}$). It is assumed the reserve ratio is changed due to exchange rate fluctuations.

5.1 Price for Buying Reservecoins

The price of buying one reserve coin is defined as follows:

$$P^{buy}_{rc} = P_{RC} \cdot (1 + fee(R)), \tag{10}$$

$$fee(R) = \begin{cases} fee_0 + k_{rm} \cdot \frac{R - L^t \cdot r_{opt}}{L^t \cdot r_{opt}}, & \text{if } r \geq r_{opt}, \\ fee_0, & \text{if } r < r_{opt}, \end{cases}$$

where k_{rm} is a system parameter defining a linear correlation coefficient between the reserves deviation from the optimal level and fee deviation from fee_0.

$fee(R)$ defines the dynamic fee dependent on the current reserve ratio. Provided that the reserve ratio is increased when RCs are bought, the fee is minimized when $r < r_{opt}$ and linearly grows if $r \geq r_{opt}$ (see Fig. 2). By carefully configuring k_{rm} it is possible to discourage reservecoins buying by imposing a high fee (up to 100%) when the reserve ratio grows too much, thus eliminating the need to have an explicit r_{max} bound as in Minimal Djed.

Given that the nominal price of a reservecoin depends on the current amount of reserves and amount of already issued reservecoins, it means that it is changed after each new reservecoin is bought.

Thus, if a user wants to buy N reservecoins, the price for each coin $i \in [0, N-1]$ should be calculated iteratively:

$$P^{buy}_{rc,i} = \frac{R_i - L_i}{N_{RC_i}}(1 + fee(R_i)),$$

$$L_i = \begin{cases} L^t, & \text{if } r_i \geq r_{peg}, \\ \frac{R_i}{r_{peg}}, & \text{if } r_i < r_{peg}, \end{cases} \qquad N_{RC_i} = N_{RC_0} + i, \qquad R_{i+1} = R_i + P^{buy}_{rc,i},$$

where

- R_i – the amount of basecoins in reserve just before buying coin i;
- N_{RC_i} – the amount of reservecoins just before buying coin i;
- r_i – the reserve ratio just before buying coin i ($r_i = \frac{R_i}{L^t}$);

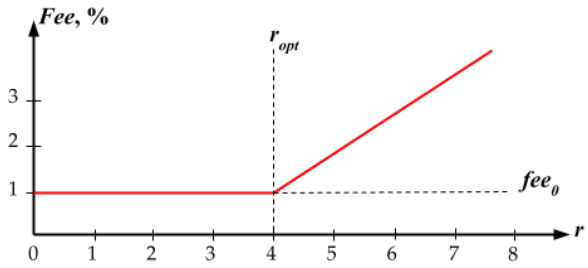

Fig. 2. The dependence of a reservecoin buying fee on the reserve ratio. Because a buying operation increases the reserve ratio, the fee linearly increases when the current reserve ratio is above the optimum and stays at the base level fee_0 when it is below.

– R_0, N_{RC_0} – the initial amounts of reserves and reservecoins before the operation begins.

Then, the total price for N coins is:

$$P_{rc}^{buy}(N) = \sum_{i=0}^{N-1} P_{rc,i}^{buy} = \sum_{i=0}^{N-1} \frac{R_i - L_i}{N_{RC_i}}(1 + fee(R_i)). \tag{11}$$

Such iterative price recalculation for every coin allows to prevent price manipulations by combining several buying operations together, thus making the overall operation cheaper. But iterative calculation might be expensive, especially if we consider its implementation in a smart contract. Reducing the equation to a closed form without the cycle in the discrete setting seems infeasible, but can be done in the continuous setting. The basic idea is to assume that each coin is divisible into infinite number of pieces (so that each piece approaches zero) and to recalculate the price after buying each smallest piece. In such a setting we are in the field of mathematical analysis which allows to derive the needed function. See the details of deriving formulas in the continuous setting in Appendix A.

5.2 Price for Selling Reservecoins

When reservecoins are sold, price calculation follows the same pattern as for buying. The base price is defined as follows:

$$P_{rc}^{sell} = P_{RC} \cdot (1 - fee(R)), \tag{12}$$

$$fee(R) = \begin{cases} fee_0 + k_{rr} \cdot \frac{L^t \cdot r_{opt} - R}{L^t \cdot r_{opt}}, & \text{if } r < r_{opt}, \\ fee_0, & \text{if } r \geq r_{opt}, \end{cases}$$

where k_{rr} is a system parameter.

The sold reservecoins are burned and corresponding amount of basecoins is returned to a user from the reserve.

As in the buying case, $fee(R)$ defines the dynamic fee dependent on the current reserve ratio. But in this case the fee linearly grows if the reserve ratio is below the optimum, because the operation further decreases it (see Fig. 3).

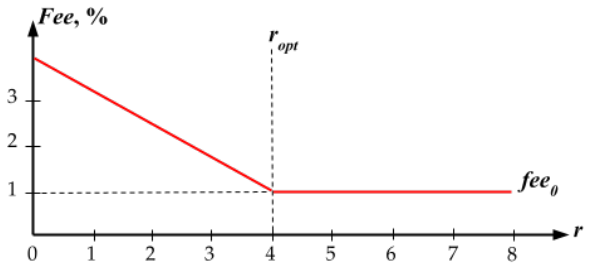

Fig. 3. The dependence of the reservecoin selling fee on the reserve ratio. Because the operation decreases reserves, the fee linearly decreases when the current reserve ratio is below the optimum and stays at the minimal level fee_0 when it is above.

If a user sells N reservecoins, the price for each coin $i \in [0, N-1]$ is calculated as follows:

$$P_{rc,i}^{sell} = \frac{R_i - L_i}{N_{RC_i}}(1 - fee(R_i)),$$

$$L_i = \begin{cases} L^t, & \text{if } r_i \geq r_{peg}, \\ \frac{R_i}{r_{peg}}, & \text{if } r_i < r_{peg}, \end{cases} \qquad N_{RC_i} = N_{RC_0} - i, \qquad R_{i+1} = R_i - P_{rc,i}^{buy},$$

Then, the total amount of returned basecoins for selling N reservecoins is:

$$P_{rc}^{sell}(N) = \sum_{i=0}^{N-1} P_{rc,i}^{sell} = \sum_{i=0}^{N-1} \frac{R_i - L_i}{N_{RC_i}}(1 - fee(R_i)). \tag{13}$$

Equation (13) requires iterative price calculation, we can apply similar techiques and consider reservecoins selling in the continuous setting to simplify the equation. See full details of derivation in Appendix B.

5.3 Price for Buying Stablecoins

The base price of buying one stablecoin is defined as follows:

$$P_{sc}^{buy} = P_{SC} \cdot (1 + fee(R, N_{SC})), \tag{14}$$

$$fee(R, N_{SC}) = \begin{cases} fee_0 + k_{sm} \cdot \frac{L^t \cdot r_{opt} - R}{L^t \cdot r_{opt}}, & \text{if } r < r_{opt}, \\ fee_0, & \text{if } r \geq r_{opt}, \end{cases}$$

where k_{sm} is a system parameter and $L^t = N_{SC} P_{SC}^t$.

Similarly to other operations, $fee(R, N_{SC})$ defines the dynamic fee. In case of buying stablecoins, the reserve ratio is decreased, so the fee is increased when the current ratio is below the optimum and stays at the minimum level fee_0 otherwise (the behaviour of $fee(R, N_{SC})$ is analogous to as in the case of selling reservecoins (see Fig. 3)).

Assuming that buying stablecoins is not allowed if $r < r_{peg}$ (either due to raising fees to 100% or due to a hard stop limit), the equation can be simplified:

$$P_{sc}^{buy} = P_{SC}^t \cdot (1 + fee(R, N_{SC})), \tag{15}$$

Even though the nominal price does not depend on reserves (if $r \geq r_{peg}$), the dynamic fee calculation does, so the fee is slightly changed as each subsequent coin is bought. Thus, if a user wants to buy N stablecoins, the price for each coin $i \in [0, N-1]$ is calculated iteratively:

$$P^{buy}_{sc,i} = P^t_{SC} \cdot (1 + fee(R_i, N_{SC_i})), \qquad R_{i+1} = R_i + P^{buy}_{sc,i}, \qquad N_{SC_i} = N_{SC_0} + i$$

where R_0 is the initial amount of reserves.

Then, the total price for buying N stablecoins is:

$$P^{buy}_{sc}(N) = \sum_{i=0}^{N-1} P^{buy}_{sc,i} = \sum_{i=0}^{N-1} P^t_{SC} \cdot (1 + fee(R_i, N_{SC_i})). \tag{16}$$

The continuous setting is considered in Appendix C.

5.4 Price for Selling Stablecoins

The amount of basecoins to be returned for selling one stablecoin is defined as follows:

$$P^{sell}_{sc} = P_{SC} \cdot (1 - fee(R, N_{SC})), \tag{17}$$

$$fee(R, N_{SC}) = \begin{cases} fee_0 + k_{sr} \cdot \frac{R - L^t \cdot r_{opt}}{L^t \cdot r_{opt}}, & \text{if } r > r_{opt}, \\ fee_0, & \text{if } r \leq r_{opt}, \end{cases}$$

where k_{sr} is a system parameter and $L^t = N_{SC} P^t_{SC}$.

Given that stablecoins selling increases the reserve ratio, $fee(R, N_{SC})$ grows if the ratio is above the optimal level to discourage further distancing (the behaviour of $fee(R, N_{SC})$ is the same as in the case of reservecoins buying, see Fig. 2).

Note that the nominal price of stablecoins remains constant while $r \geq r_{peg}$. If $r < r_{peg}$, the stablecoins lose their value according to the coefficient k defined by (8).

Similarly to other operations, the price of selling N stablecoin is calculated iteratively for each coin $i \in [0, N-1]$:

$$P^{sell}_{sc,i} = k(r_i) \cdot P^t_{SC} \cdot (1 - fee(R_i, N_{SC_i})), \tag{18}$$

$$k(r_i) = \min(1, \frac{r_i}{r_{peg}}) = \min(1, \frac{R_i}{N_{SC_i} \cdot P^t_{SC} \cdot r_{peg}}),$$

$$R_{i+1} = R_i - P^{sell}_{sc,i}, \qquad N_{SC_i} = N_{SC_0} - i.$$

Then, the total amount of returned basecoins is:

$$P^{sell}_{sc}(N) = \sum_{i=0}^{N-1} P^{sell}_{sc,i} = \sum_{i=0}^{N-1} k(r_i) \cdot P^t_{SC} \cdot (1 - fee(R_i, N_{SC_i})). \tag{19}$$

The continuous setting is considered in Appendix D.

5.4.1 Debt-for-equity swaps. Since keeping the value of stablecoins is an ultimate goal of the system, it is undesirable to cut it when $r < r_{peg}$ without any compensation. Thus, it is suggested to compensate a user with reservecoins on the equivalent amount of retained basecoins. This exchange resembles what is known as "debt-for-equity swaps" in the traditional financial world.

The number of compensated reservecoins for one sold stablecoin is defined as follows:

$$S_{sc}^{sell} = \frac{(1 - k(r)) \cdot P_{SC}^{t}}{P_{RC}} \cdot (1 - fee_0), \tag{20}$$

where $k(r) = \min(1, \frac{r}{r_{peg}})$ as defined by Eq. (8).

Note that if $r \geq r_{peg}$, a user will receive zero reservecoins as the stablecoin value is fully returned. If $1 \leq r < r_{peg}$, the compensated amount of reservecoins is fully backed by retained basecoins so its value and nominal price is preserved. But if $r < 1$, to compensate full value of a stablecoin, the system will have to mint new reservecoins without sufficient backing by retained basecoins which will dilute its value (and, correspondingly, decrease the nominal price).

The amount of compensated reservecoins should be calculated iteratively for every sold stablecoin. Thus, if a user wants to sell N stablecoins, the number of reservecoins returned for each stablecoin $i \in [0, N-1]$ is calculated as follows:

$$S_{sc,i}^{sell} = \frac{(1 - k(r_i)) \cdot P_{SC}^{t}}{P_{RC,i}} \cdot (1 - fee_0),$$

$$P_{RC,i} = \frac{R_i - L_i}{N_{RC_i}}, \qquad L_i = k(r_i) \cdot P_{SC}^{t} \cdot N_{SC_i}, \qquad N_{RC\,i+1} = N_{RC\,i} + S_{sc,i}^{sell},$$

where R_i, $k(R_i)$ and N_{SC_i} are defined by (18).

Then, the total amount of compensated reservecoins for selling N stablecoins is:

$$S_{sc}^{sell}(N) = \sum_{i=0}^{N-1} S_{sc,i}^{sell} = \sum_{i=0}^{N-1} \frac{(1 - k(r_i)) \cdot P_{SC}^{t} \cdot N_{RC_i}}{R_i - k(r_i) \cdot P_{SC}^{t} \cdot N_{SC_i}} \cdot (1 - fee_0). \tag{21}$$

Due to dilution of reservecoins with every sold stablecoin, in this case it is even more important to consider the continuous setting (see details in Appendix D).

6 Model-Checking Djed's Stability Properties

In order to ensure that the Minimal Djed requirements are sufficient to guarantee stability, the stablecoin contract is formalized in an extension of the Lustre synchronous dataflow language [1] as an infinite-state system. The Kind2 SMT-based model checker [2] is used to validate the stability properties. It takes as input one or more Lustre files annotated with the properties to be verified and outputs whether each property is satisfied or not. In particular, Bounded Model Checking (BMC) [3] is used for detecting any property violation. The search for an execution trace leading to a violation starts from the set of possible initial states of the system. A counterexample is generated whenever a property is not satisfied. Concretely, BMC determines whether a counterexample at a given discrete time step k exists for a property P by verifying the following propositional formula:

$$bmc(k) \equiv I(\vec{x}_0) \wedge \bigwedge_{i=0}^{k-1} \left(C(\vec{x}_i) \wedge T(\vec{x}_i, \vec{x}_{i+1}) \right) \wedge C(\vec{x}_k) \wedge \neg P(\vec{x}_k) \tag{22}$$

Here, $\vec{x}$ represents the set of state variables. Notation $\vec{x}_0, \ldots, \vec{x}_k$ is used to represent vector copies of $\vec{x}$ at each time step $i \in [0..k]$. Quantifier-free formula $I(\vec{x}_0)$ specifies the set of initial states. Quantifier-free formula $C(\vec{x_i})$ characterizes any assumption imposed on $\vec{x}$, while formula $T(\vec{x}_i, \vec{x}_{i+1})$ represents the transition relation between $\vec{x}_i$ and $\vec{x}_{i+1}$ over one time step. Finally, formula $P(\vec{x_k})$ denotes the property (or set of properties) to be satisfied.

For invariant satisfaction, a proof by induction strategy called K-Induction is used [4]. Intuitively, the base case aims at determining whether a property holds from the initial states of the system. As for the induction step, it aims at proving that if a property holds for any arbitrary state of the system then, it must also hold in the next possible reachable states. When both cases are proved, the property is considered as an invariant of the system. Induction at a given discrete time step k is therefore characterized by the following formulae:

$$base(k) \equiv I(\vec{x}_0) \wedge \bigwedge_{i=0}^{d-1} (C(\vec{x}_i) \wedge T(\vec{x}_i, \vec{x}_{i+1})) \wedge C(\vec{x}_d) \rightarrow P(\vec{x}_d), \quad for\ 0 \leq d \leq k \quad (23)$$

$$step(k) \equiv \bigwedge_{i=0}^{k-1} (\ P(\vec{x}_i) \wedge C(\vec{x}_i) \wedge T(\vec{x}_i, \vec{x}_{i+1}) \ ) \wedge C(\vec{x}_k) \rightarrow P(\vec{x}_k) \quad (24)$$

It can be observed that the base case is verified from time step 0. The step case is quite similar to $base(k)$, except that the initial constraint (i.e., $I(\vec{x}_0)$) is removed and property P is assumed to hold on all states up to time step $k-1$ [9]. Formula $step(k)$ may also fail to hold, especially when conjunction $P(\vec{x}_i) \wedge C(\vec{x}_i)$ is not sufficient to establish induction (i.e., it may correspond to an over-approximation of states reachable by the system). In this case, an induction trace leading to the falsification of $step(k)$ is produced by KIND2. This induction trace is generally referred to as a *counterexample to induction* (CTI) and can be analyzed to deduce strong enough invariants about the system's states establishing the inductiveness of property P.

6.1 Data Types

The following data types are used to specify the input, output and state variables maintained by the stablecoin contract in LUSTRE.

Listing 1: StableCoin Data Types

```
type Order = enum { MintSC, MintRC, NoOrder };
type Proceed = enum { MintedSC, MintedRC, RedeemedSC, RedeemedRC, Error, NoReply };
type ErrorInfo = enum { Min_Ratio_Violated, Max_Ratio_Violated, Invalid_Mint_Value, None };

type InputMsg = struct { order: Order; qnt: AmountType };

type OutputMsg = struct { ack: Proceed; err: ErrorInfo; price: ReserveType };

type Parameters = struct { r_min: RatioType; r_max: RatioType; fee: ReserveType; n_sc_s: N_Type; p_min: ReserveType };

function imported params () returns (out: Parameters );
```

The user input to the contract is defined as a structure specifying the type of order submitted to the contract and the number of coins to be bought (i.e., positive value) or sold (i.e., negative value). The `NoOrder` literal is introduced to model the non-invocation of

[9] Note that formula $step(k)$ is reduced in proving whether $P(\vec{x}_0)$ can solely be inferred from $C(\vec{x}_0)$ when $k = 0$, i.e., the proof by induction really starts when $k = 1$.

the contract's functions. The contract also produces a structure as output to indicate whether or not an order was processed successfully. On a successful order, the `price` field indicates the actual buying price (i.e., value increasing reserve) or selling price (i.e., value decreasing reserve). Field `ErrorInfo` is significant only when an order is aborted and indicates the type of failure encountered. Otherwise, it is set to `None`. Alias types `AmountType`, `ReserveType`, `N_Type` and `RatioType` are introduced to add some genericity and are defined with concrete ones according to the implementation to be verified. For instance, the stability properties have been proved w.r.t. an implementation using unbounded integer representation. Finally, type `Parameters` specifies the parameters characterizing the Djed specification and abstract function `params` (i.e., imported directive) is introduced to provide a global access to parameters.

6.2 Infinite-State System Specification

The code excerpt in Listing 2 illustrates how the stablecoin state-transition system is specified in Lustre. The full specification is provided in Appendix F. In Lustre, the notion of *node* is used to model transition systems. Plainly speaking, a node is like a function except that it can make use of temporal combinators to characterize state variables or to specify stateful behaviours. Function definitions (see Listing 3) are also supported in Lustre. However, functions should be stateless in the sense that: outputs and local variables should be non-temporal combinations of inputs; a function is not allowed to call a node but only other functions.

Listing 2: StableCoin State-Transition System

```
node StableCoin(i_msg: InputMsg; rate: RateType)
     returns (p_reserve: ReserveType; p_sc: N_Type; p_rc: N_Type; reserve: ReserveType;
              n_sc: N_Type; n_rc: N_Type; o_msg: OutputMsg)
let
 p_reserve = 0 -> pre reserve;
 p_sc = 0 -> pre n_sc;
 p_rc = 0 -> pre n_rc;

 o_msg = if i_msg.order = NoOrder then
          NullReply
         else if i_msg.order = MintSC then
          mintSC(i_msg.qnt, rate, p_reserve, p_sc)
         else
          mintRC(i_msg.qnt, rate, p_reserve, p_sc, p_rc, t_bootstrap);

 reserve = if o_msg.ack = Error then p_reserve else p_reserve + o_msg.price;
 n_sc = if o_msg.ack = MintedSC or o_msg.ack = RedeemedSC then p_sc + i_msg.qnt else p_sc;
 n_rc = if o_msg.ack = MintedRC or o_msg.ack = RedeemedRC then p_rc + i_msg.qnt else p_rc;
tel
```

The stablecoin node takes two arguments as input, namely: the user's order and the exchange rate. Operator `->` ("followed by") is used to initialize the state variables, while operator `pre` makes reference to the previous value of an expression. Variables `reserve`, `n_sc` and `n_rc` respectively correspond to the bank reserve, number of stablecoins in circulation and number of reservecoins in circulation. They are updated only when an order is successfully processed. Otherwise, they retain their previous value. Note that, node `StableCoin` (see Listing 2) corresponds to the unfolded version of node `StableCoin_InitState` (see Listing 6 in Appendix F) to ease comprehension. Node `StableCoin_InitState` has been introduced to unify as much as possible the formalization of the various aforementioned theorems.

The minting and redeeming logics for both stablecoins and reservecoins are respectively specified in functions `mintSC` and `mintRC`. Note that buying and selling of reservecoins are authorized when there are no stablecoins in circulation (see Section 6.4). Moreover, all minting and redeeming actions are not authorized when the

exchange rate is not greater than zero (see Section 6.4). Selling of stabecoins (resp. reservecoins) is also forbidden when the number of coins being sold is greater than the number of stablecoins (resp. reservecoins) in circulation. Listing 3 shows how function `mintSC` is defined.

Listing 3: Minting and Redeeming StableCoin

```
function mintSC(d_sc: N_Type; rate: RateType; reserve: ReserveType; n_sc: N_Type)  returns (o_msg: OutputMsg)
 var s_price, fee, t_price, t_reserve: ReserveType;
let
  s_price = price_sc(reserve, n_sc, rate) * d_sc;
  t_price = computeFee(s_price);
  t_reserve = reserve + t_price;
  o_msg = if d_sc > 0 then
           if rate > 0 and t_reserve >= (n_sc + d_sc) * rate * params().r_min
           then
            OutputMsg { ack = MintedSC; err = None; price = t_price }
           else
            ErrorCode1
          else if -d_sc <= n_sc and rate > 0 then
            OutputMsg { ack = RedeemedSC; err = None; price = t_price }
          else
           ErrorCode3;
tel
```

6.3 Property Specification

Each stability property is formalized in a specific Lustre node so as not only to facilitate individual proof analysis but also to explicitly identify the necessary inductive lemmas required for each one of them. Listing 4 shows how Theorem 5 is formally specified. Properties about the stablecoin contract are expressed in a black-box fashion (i.e., only formulated in terms of inputs and outputs of the contract). Hence, each property node is parameterized with the inputs necessary to invoke the contract. The `assert` directive is used to specify assumptions under which the property hold. The two main assumptions concern the contract's parameters, namely: parameters remain constant from one time step to the next; and parameters should satisfy the min and max values necessary to guarantee stability.

Data structure `SCSellerType` is introduced to model SC holders that may sell their coins back to the bank, with field `sold_once` indicating whether at least one SC was sold and field `price_per_sc` denoting the price obtained per SC. In node `Theorem5`, state variable `p_seller` is used to indicate whether an SC holder has previously sold SCs to the bank (i.e., `sold_once` set to **`true`**). Variable `c_seller` is updated with the current seller information on each successful selling order (see node `update_SCSeller`). Otherwise, it is assigned to the value of the previous seller. Note that the Lustre formalization is even stronger than the formulation in Section 3, as it also covers situations whereby there can be an arbitrary number of actions between two successful selling SC events.

Directive `check` is used to specify properties to be verified. An optional identifier can be provided for each property. When several properties are specified, they are implicitly conjoined and proved as a whole. This feature is practical when inductive lemmas are required to establish invariant satisfaction. When a counterexample is detected, the violated properties are automatically removed from the conjunction and the satisfaction of the remaining ones is reassessed. In the formalization, property `"THEOREM_5"` states the following:

> *If the exchange rate remains constant and at least one SC was already sold then, on a successful selling order, the current selling price per SC is greater than or equal to the previously applied price.*

The inductive lemmas necessary to prove `"THEOREM_5"` are also provided. In addition to the theorems listed in Section 3, the following ones were also proved on the stablecoin specification:

```
check "THEOREM_10" o_msg.ack = MintedSC => o_msg.price = sc_price_pc;
check "THEOREM_11" o_msg.ack = RedeemedSC and p_reserve >= p_sc * rate => o_msg.price = sc_price_pc;
check "THEOREM_12" o_msg.ack = RedeemedRC => equity(p_reserve, p_sc, rate) > 0;
check "THEOREM_13" reserve >= 0 and n_sc >= 0 and n_rc >= 0;
check "THEOREM_14" n_rc > 0 => reserve > 0;
check "THEOREM_15" n_sc > 0 => reserve > 0;
```

Property `"THEOREM_10"` states that the buying price for SCs is always 1 PC in BCs, while `"THEOREM_11"` states that the selling price for SCs is always 1 PC in BCs when $R \geq L(N_{SC})$. As for `"THEOREM_12"`, it states that RCs can be sold only when $E(R, N_{SC}) > 0$. The three remaining ones are self-explanatory.

Listing 4: Theorem 5 Formalization

```
type SCSellerType = struct { sold_once: bool; price_per_sc: ReserveType };

node update_SCSeller (p_reserve: ReserveType; p_sc: N_Type; rate: RateType; o_msg: OutputMsg; seller: SCSellerType)
     returns (out: SCSellerType)
let
 out = if o_msg.ack = RedeemedSC then
        SCSellerType { sold_once = true; price_per_sc = min(p_reserve div p_sc, rate) }
       else
         seller;
tel

const defaultSeller_SC = SCSellerType { sold_once = false; price_per_sc = 0 };

node Theorem5 (i_msg: InputMsg; rate: RateType )  returns (o_msg: OutputMsg)
 var reserve: ReserveType; n_sc: N_Type; n_rc: N_Type; p_reserve: ReserveType; p_sc: N_Type;
     p_rc: N_Type; constant_rate: bool; p_seller: SCSellerType; c_seller: SCSellerType;
 let
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming that parameters remain constant
  assert true ->
          params().r_min = pre params().r_min and
          params().r_max = pre params().r_max and
          params().fee = pre params().fee and
          params().n_sc_s = pre params().n_sc_s and
          params().p_min = pre params().p_min;

  -- Assuming min and max value constraints
  assert params().r_min > 1 and params().r_max >= params().r_min and
         params().fee > 0 and params().fee <= 100 and
         params().n_sc_s > 0 and params().p_min > 0;

  constant_rate = true -> pre constant_rate and rate = pre rate;
  p_seller = defaultSeller_SC -> pre c_seller;
  c_seller = update_SCSeller(p_reserve, p_sc, rate, o_msg, p_seller);

  -- Stability Property
  check "THEOREM_5"
   (constant_rate and p_seller.sold_once and o_msg.ack = RedeemedSC) => c_seller.price_per_sc >= p_seller.price_per_sc;

  -- Lemmas
  check (constant_rate and p_seller.sold_once) => p_seller.price_per_sc <= rate;
  check (constant_rate and p_seller.sold_once and p_sc > 0) => p_seller.price_per_sc <= min(p_reserve div p_sc, rate);
tel
```

6.4 Improvements from Falsified Properties

Several improvements to earlier versions of the Minimal Djed specification were made due to various property violations detected by BMC. Many of these improvements involved making explicit various assumptions that had previously been left implicit. The minimal parameter constraints required to guarantee stability were also validated via counterexample detection. Some of these improvements are discussed below.

Reserve Draining. The following counterexample was detected when attempting to prove `"THEOREM_13"` on the initial Djed specification, with $r_{min} = 4$, $r_{max} = 8$, $fee = 1\%$ and $P_{RC}^{min} = 100$:

$cycle$	:	0	1	2	3
$i\_msg.order$	:	$MintRC$	$MintSC$	$MintSC$	$MintRC$
$i\_msg.qnt$	:	6	100	-100	-4
P_{SC}^{t}	:	0	2	5	5
P_{SC}	:	0	2	5	5
P_{RC}^{b}	:	100	101	$51.333\ldots$	$52.1666\ldots$
R	:	606	808	313	-83
N_{SC}	:	0	100	0	0
N_{RC}	:	6	6	6	2
$o\_msg.ack$	:	$MintedRC$	$MintedSC$	$RedeemedSC$	$RedeemedRC$
$o\_msg.err$	:	$None$	$None$	$None$	$None$
$o\_msg.price$	:	606	202	-495	-396

In fact, an earlier version of the Minimal Djed specification specified P_{RC}^{b} as both buying and selling prices for RCs. The above counterexample shows that this configuration can lead to a reserve draining scenario, especially when P_{RC}^{t} is less than P_{RC}^{min} on a selling RC action. This particular situation arises at time step 3 with P_{RC}^{t} evaluating to $52.1666\ldots$ (i.e., $313 \div 6$). The default price of 100 is therefore considered when selling the 4 RCs, thus leading to a negative bank reserve. It should be noted that, with the exception of price computation[10], the value for each state variable in the above execution trace corresponds to the value obtained at the end of each time step. It is also worth recalling that R, N_{SC} and N_{RC} are all initialized to zero. (see Listing 2). The same interpretation should be applied in the remaining of this section.

Theorem 7 Violation. Another reserve draining situation was detected when attempting to prove the initial formulation of Theorem 7. The counterexample obtained shows that there may exist a sequence of n possible actions for which $R^n < R^0$, $N_{SC}^n = N_{SC}^0$ and $N_{RC}^n = N_{RC}^0$, when solely assuming a constant exchange rate. Such a sequence of actions is illustrated by the execution trace given below, with $r_{min} = 3$, $r_{max} = 5$, $fee = 5\%$ and $P_{RC}^{min} = 10$. Notation $\perp$ is used for undefined values. Indeed, P_{RC}^{t} cannot be computed when N_{RC} is zero. As can be seen, the initial bank state (i.e., R^0, N_{SC}^0 and N_{RC}^0) has a reserve value that largely backs the number of RCs, even though there are no stabecoins in circulation. This situation is possible due to the accumulation of fees over a significant number of buying/selling orders. At time step 1, it can also be observed that RCs can still be sold even when there are no SCs in circulation. Indeed, the *minimum reserve ratio* check cannot be performed in this particular case as N_{SC} is zero. The draining situation arises mainly because default price P_{RC}^{min} is used when buying back the 4 RCs in the last time step (i.e., target price P_{RC}^{t} is undefined).

[10] P_{SC}^{t}, P_{RC}^{b} and P_{SC}^{t} are computation w.r.t. the value of the bank state at the begining of each time step

$cycle$	:	0	1	2
$i\_msg.order$	:	$NoOrder$	$MintRC$	$MintRC$
$i\_msg.qnt$	:	0	-4	4
P^b_{RC}	:	430	430	10
P^t_{RC}	:	430	430	$\perp$
R	:	1720	86	128
N_{SC}	:	0	0	0
N_{RC}	:	4	0	4
R^0	:	1720	1720	1720
N^0_{SC}	:	0	0	0
N^0_{RC}	:	4	4	4
$o\_msg.ack$	:	$NoReply$	$RedeemedSC$	$MintedRC$
$o\_msg.err$	:	$None$	$None$	$None$
$o\_msg.price$	:	0	-1634	42

Liveness issues. The formal modelling of an earlier version of the Minimal Djed specification in Lustre has also led to the detection of some liveness issues mainly related to the potential re-initilization of the stablecoin smart contract. For instance, when N_{RC} is set back to zero[11], selling SCs may be the only feasible actions once the bank reserve is no more sufficient to satisfy the *minimum reserve ratio* for any buying SC action. This is the expected behaviour of the stablecoin algorithm due to the mechanisms put in place to guarantee stability.

However, the blocking situation arises when N_{SC} reverts to zero. Indeed, the earlier specification authorizes the buying of RCs only when the *maximum reserve ratio* is satisfied or when the buying action happens within an initilization period (i.e., unlimited buying of RCs to capitalize the reserve). However, once the initialization period is over and after a sequence of buying and selling actions leading to a state where $N_{SC} = 0$ and $r < r_{min}$, we can end in a blocking situation where no action is authorized: any buying RC action to bring in more reserve will be rejected as condition $R \leq L(N_{SC})r_{max}$ will no more be satisfied [12]; no futher action will be authorized once there are no more RCs in circulation[13]. Parameter N^*_{SC} was therefore introduced to palliate this blocking situation. It mainly authorizes the buying of RCs when the number of stabelcoins in circulation is below a certain threshold and irrespective of the *maximum reserve ratio*.

Null Exchange Rate Prohibition. With the Minimal Djed specification, it is conceivable to buy SCs during the initilization period without having any RCs in circulation, while still having post-condition $r \geq r_{min}$ satisfied. For instance, 14 SCs can be bought at $P^t_{SC} = 200$ with $fee = 50\%$ and $r_{min} = 1.5$. The resulting R is therefore 4200

[11] This scenario is possible mainly due to the accumulation of fees. In particular, all RCs can be sold while the *minimum reserve ratio* is still preserved and there are still stablecoins in circulation.

[12] Once at least one buying/selling action has been authorized, R can never be set back to zero even when there are no more SCs and RCs in circulation. This is mainly due to the fees accumulated on each action. Hence, condition $R \leq L(N_{SC})r_{max}$ always evaluates to *false* when N_{SC} is set back to zero.

[13] Selling of RCs is still authorized as condition $R \geq L(N_{SC})r_{min}$ is always satisfied when $N_{SC} = 0$.

BCs with r evaluating to 1.5 (i.e., $4200 \div (14 \times 200)$). However, if a null exchange rate is authorized then, a large amount of SCs might be bought without any accumulation of fees. This scenario was confirmed by a counterexample detected when attempting to prove `"THEOREM_15"`, whereby R is still set to zero with n amount of SCs in circulation.

Minimum Fee Percentage. Another counterexample, detected when attempting to prove `"THEOREM_14"`, explicitly confirms that the minimum fee percentage must be greater than zero. Otherwise, we may end in a situation where there is no more reserves to back RCs in circulation. The following execution trace illustrates this scenario, with $r_{min} = 2$, $r_{max} = 8$, $fee = 0\%$ and $P_{RC}^{min} = 2$:

$cycle$	0	1	2
$i\_msg.order$	$MintRC$	$MintSC$	$MintSC$
$i\_msg.qnt$	1	2	-2
P_{SC}^{t}	1	1	5
P_{SC}	0	1	2
R	2	4	0
N_{SC}	0	2	0
N_{RC}	1	1	1
$o\_msg.ack$	$MintedRC$	$MintedSC$	$RedeemedSC$
$o\_msg.err$	$None$	$None$	$None$
$o\_msg.price$	1	2	4

As can be seen, the absence of accumulated fees during the buying actions occurring at the first two time steps causes a complete depletion of the reserve when all SCs are sold within the last time step.

Minimum Reserve Ratio (r_{min}). BMC also confirms that r_{min} should be greater than $1 + fee$ for stability to be guaranteed. Indeed, there is a possibility to buy SCs for less than 1 *peg currency* in BCs even when r_{min} is set to one. The following execution trace exhibits this scenario, with $r_{min} = 1$, $r_{max} = 2$, $fee = 96\%$ and $P_{RC}^{min} = 2$:

$cycle$	0	1
$i\_msg.order$	$MintSC$	$MintSC$
$i\_msg.qnt$	3	1
P_{SC}^{t}	8	17
P_{SC}	8	15.68
R	47.04	77.773
N_{SC}	3	4
N_{RC}	0	0
$o\_msg.ack$	$MintedSC$	$MintedSC$
$o\_msg.err$	$None$	$None$
$o\_msg.price$	47.04	30.733

It can be noticed that the resulting reserve ratio is greater than one (i.e., ≈ 1.1437) even when $\frac{R}{N_{SC}}$ is used as the actual buying price for SCs in the last time step.

Default RC Price (P_{RC}^{min}). As stated in Section 2, default price P_{RC}^{min} is used for RCs when the *target price* is undefined (i.e., $N_{RC} = 0$). However, if a null default price is specified then, a large amount of RCs might be bought without any accumulation of fees. This scenario was confirmed by a counterexample obtained when attempting to prove `"THEOREM_14"`. In particular, a null default price may induce a situation whereby there is no reserves to back RCs in circulation.

7 Formalization in Isabelle/HOL

Isabelle [5] is a generic proof assistant. It allows mathematical formulas to be expressed in a formal language and provides tools for proving those formulas in a logical calculus. The main application is the formalization of mathematical proofs and in particular formal verification of computer hardware and software. Isabelle can cope with a large class of object-logics, including several first-order logics (intuitionistic and classical), Martin–Löf's Type Theory, and Zermelo–Fraenkel set theory. Each new logic is formalized within Isabelle's meta-logic, which is an intuitionistic fragment of Church's type theory. The most widespread instance of Isabelle nowadays is Isabelle/HOL [6], which provides a higher-order logic theorem proving environment that is ready to use for big applications.

In this work, Isabelle/HOL is used to formalize parts of Djed's informal description. The formalization covers Minimal Djed as defined in Section 2, including machine-checked proofs for the stability properties stated in Section 3 (see the formalization details in Appendix E). Remarkably, the effort spent on the formalization (roughly four weeks) appears to be cost-effective.

7.1 Design Considerations

We highlight the following important design decisions of our Isabelle formalization:

- *Isabelle/HOL.* Isabelle/HOL is by far the most developed object-logic, including powerful specification tools and a vast library of formally verified mathematics, thus being the logic of choice for our work.
- *Isabelle/Isar.* Our proofs are conducted in the structured proof language Isar, allowing for proof texts naturally understandable for both humans and computers, while preserving the structure of the Djed's informal proofs to aid the reader.
- *Locales.* We use locales (Isabelle's module system) to bundle Djed's parameters and their constraints. This also permits the specification of a proof context shared by all theorems, thus avoiding duplication.
- *Exact real arithmetic.* The Isabelle/HOL library includes exact real arithmetic, which we use for our formalization. This allows to formally reason at the same level of abstraction as Djed's informal description, ignoring implementation issues such as rounding errors.
- *Transition systems.* Using Isabelle's principle of inductive predicates, we define transition systems that model the execution of individual Djed actions and sequences thereof.
- *Notational convenience.* Isabelle provides excellent notational support: new notations can be introduced using normal mathematical symbols. We leverage such support and define special syntax to match that used by Djed's informal description, thus improving readability.

7.2 Uncovered Issues

The Isabelle/HOL formalization uncovered numerous issues in earlier versions of the informal description of Minimal Djed and in earlier drafts of the informal proofs of the stability properties. The process of formalizing Minimal Djed and its stability theorems in Isabelle contributed to the more precise description and proofs that can now be found in Sections 2 and 3. To illustrate the classes of issues uncovered by the formalization, we can enumerate the following:

- Implicit and/or missing constraints on variables/actions in the informal model: Lower and upper bounds (e.g., $fee \in [0, 1)$), non-negativity assumptions (e.g., $N_{SC} > 0$), and pre-action conditions (e.g., $n \leq N_{SC}$ for a 'Sell n SCs' action).
- Implicit and/or missing assumptions in the informal proofs: E.g., the exchange rates must be constant in Theorem 5, and $N_{SC} > 0$ must be an assumption in Theorem 8 .
- Falsifiable formulas in the informal proofs: E.g., the conclusion of Theorem 6 had to be relaxed to $\geq$ due to impossible cases in the informal proof, which in turn caused the proof of Theorem 7 to be reworked.
- Superfluous assumptions in theorems: E.g., in Theorem 3, $P = P^t_{SC}[x]$ is redundant since it is implied by $r > 1$.
- Missing case distinctions in the informal proofs required to address specific cases that do not hold in the general case.
- Steps in the informal proofs that are missing or found to be non trivial, particularly in the proofs of Theorems 6 and 7 .
- Ambiguous wording: E.g, $N_{SC} > 0$ and $N_{RC} > 0$ in the context of the statement of Theorem 7 .

8 Implementations

Minimal and Extended Djed already have the following implementations:

- **SigmaUSD on Ergo:**[14] Minimal Djed was implemented in ErgoScript. It was deployed anonymously with the name SigmaUSD by the Ergo community in early Q1 2021 on Ergo with a fee of 1% and an oracle that updated the exchange rate every hour. This initial version was subject to the reserve draining attack mentioned in Section 4 by an anonymous user who owned a large number of ERGs (the base coin on Ergo). The attack was ultimately unsuccessful and it is estimated that the user lost approximately 100000 USD trying to perform the attack. To further discourage such attacks, this initial deployment of Minimal Djed was replaced by a variant where the fee was set to 2%, the oracle[15] is updated every 12 minutes and every oracle update is allowed to change the price by at most 0.49%, unless the price difference is greater than 50%. This variant has not been attacked. Furthermore, its success is evidenced by the following indicators: SigmaUSD's reserve captured 6% of the entire supply of ERG; the stablecoin market cap quickly rose to a peak of 10 million USD; there are approximately 30 transactions per day buying or selling stablecoins or reservecoins from or to the bank, and this amounts to approximately

[14] https://sigmausd.io/ and https://github.com/Emurgo/age-usd.

[15] https://explorer.ergoplatform.com/en/oracle-pool-state/ergusd, https://github.com/ergoplatform/oracle-core/blob/master/src/api.rs#L41-L67.

1% of the total number of transactions per day on Ergo; the reserve ratio is being kept consistently at healthy levels between 300% and 400%; and the stablecoin was able to maintain its peg during the market crash of May 2021, when the price of ERG (along with most other cryptocurrencies) fell 44% in 1 day and 60% in 5 days. Notably, SigmaUSD is, to the best of our knowledge, the first stablecoin to be implemented on a blockchain that uses UTxO-style accounting and contracts.

- **Implementation in Solidity:** Minimal Djed was implemented in Solidity in two different variants: one where the basecoin is the native currency of the blockchain; and another one where the basecoin can be any ERC20-compliant coin. The implementations still need to be audited, but they have already been deployed to the following testnets[16]: Binance Smart Chain's testnet; Avalanche's Fuji; Polygon's Mumbai; Ethereum's Kovan; Ethereum's Rinkeby; RSK's testnet.
- **Preliminary Implementation in Plutus for Cardano:** An implementation of an earlier version of Minimal Djed in the Plutus language[17] [7–9] is available[18]. This version uses monetary policies [10] and the native assets (i.e. stablecoins and reservecoins) are uniquely identified by the hash of the monetary policy [11]. Therefore, the contract is not update-able after deployment. This may be an advantage or disadvantage, depending on one's preferences regarding immutability. This implementation also assumes a single oracle entity that provides signed exchange rate data in an off-chain manner directly to the transactions instead of posting data to the chain as usual.
- **Ongoing OpenStar Implementation:** OpenStar is a framework for private permissioned blockchains developed in Scala. The implementation of Djed using OpenStar is following the idea of off-chain smart contract execution [12] in order to have a stablecoin on Cardano that does not depend on smart contracts being executed on-chain on Cardano.
- **Prototype in Scala:**[19] Both Minimal and Extended Djed have prototype implementations in Scala that were used, besides other things, for various economic simulations. They also implement test suites for checking price calculation formulas. The prototype can be used as a reference for future production implementations.

9 Related Work

There are hundreds of stablecoins and dozens of algorithmic stablecoins nowadays. The three stablecoin protocols that were most influential in the design of Djed were:

[16] https://testnet.bscscan.com/../0x1dc9ff8b018a091bee9dc908c5b89fb687d21a63, https://testnet.bscscan.com/../0x77b6b5cf465836b856b30078590dd8355cd6e3c4, https://testnet.avascan.info/../0xb1bfad591c8de6fcd19da8338b9771d333ef8515, https://polygon-explorer-mumbai.chainstacklabs.com/tx/0x74c7c9a9bf6b155eb422fd192400095b77a5a40df18886c9043bc49400e32e2c/internal-transactions, https://kovan.etherscan.io/address/0x8450b2a8eede87f605bfa837ea7f3c686218d0c0, https://rinkeby.etherscan.io/../0xd617065ddd8023cad50320c639b6c9f3f2861052, https://explorer.testnet.rsk.co/../0x14feeb2fff4ef99eafe07f4ea5ee598771d87de7.

[17] https://testnets.cardano.org/en/programming-languages/plutus/overview/, https://github.com/input-output-hk/plutus.

[18] https://github.com/input-output-hk/plutus/blob/master/plutus-use-cases/src/Plutus/Contracts/Stablecoin.hs.

[19] https://github.com/input-output-hk/djed-stablecoin-prototype.

Seigniorage Shares: One of the earliest examples of algorithmic stablecoin is the *seigniorage shares* protocol [13]. Its stabilization mechanisms are similar to the mechanisms used by most central banks. The protocol aims to adjust the circulating supply. If the price is too high, the circulating supply will be increased. If it is too low, the circulating supply will be decreased. To take stablecoins out of circulation, users are encouraged to sell their stablecoins in exchange for seigniorage shares; conversely, when the supply needs to be increased, stablecoins are issued and distributed to seigniorage share holders. In this way, the seigniorage (i.e. the revenue from printing money) goes to seigniorage share holders. Djed's reservecoins are similar to seigniorage shares, but Djed is more like a different kind of central bank, whose currency is pegged to and fully backed by another one, and hence its revenue comes from market making instead of seigniorage.

DAI:[20] Crypto-collateralized stablecoins, of which **DAI** is currently the most prominent example, are similar to crypto-backed stablecoins, with the subtle but important difference, however, that cryptocurrencies are used to provide collateral for stablecoin-denominated loans instead of reserves. Similarly to how a house that serves as collateral in a mortgage does not count as reserves in the bank that is providing the loan, the collateral provided for the issuance of a crypto-collateralized stablecoin does not count as reserve. Due to the lack of reserves, pure crypto-collateralized stablecoins resemble an extreme form of fractional reserve banking (with a fraction of 0%). Unlike commercial bank money (i.e. money in a bank account), which can be exchanged at any time by anyone holding it for central bank money by withdrawing cash from the bank's reserves, crypto-collateralized stablecoins cannot be exchanged for anything by withdrawing from the stablecoin contract's reserves, simply because there are no reserves. When interacting with the contract, users can only use stablecoins to repay their own loans and recover their own collateral or to bid in auctions for liquidated collateral from others.

Staticoin:[21] The Staticoin protocol has a *riskcoin* whose function is very similar to Djed's reservecoin. The main difference, from a user's perspective, is that riskcoin's appeal rests in the leverage it provides with respect to base coins, whereas reservecoin's appeal rests mostly in the market making revenue through fees, even though it also provides some leverage. Djed also has minimum and maximum reserve ratio thresholds, which protect stablecoin holders from market crashes and reservecoin holders from dilution, whereas Staticoin does not.

10 Conclusions and Future Work

Djed is an algorithmic crypto-backed pegged stablecoin that strives to keep a reserve ratio significantly greater than one in order to guarantee the peg. This was an intentionally conservative design decision, because stability is of paramount importance and achieving stability with fractional reserves or even with no reserve (or collateralization) at all appears to be a much more elusive goal. This decision paid off. As shown in this paper, Djed is not only provably stable in theory but its implementation and deployment on Ergo, which survived the market crash of 2021 with the peg unscathed, has already provided strong empirical evidence of its stability in practice as well.

[20] https://makerdao.com/en/.
[21] http://staticoin.com.

Minimal Djed was intentionally designed with simplicity as a goal. As it is fully defined in just 1.5 pages (cf. Section 2), this goal can arguably be considered achieved.

Thanks to its simplicity, it is highly amenable to the use of formal methods, such as model checking and interactive theorem proving. It took only roughly 4 weeks of work to obtain the results discussed in Sections 6 and 7. Therefore, the work presented here can also be considered a success story for the discipline of formal methods, which has a reputation to be time-consuming.

The commitment to simplicity required some tough compromises. Consequently, Minimal Djed is subject to a few minor issues, which we fully disclosed. We have partially addressed these issues in Extended Djed. It is important to note, however, that the application of formal methods to Extend Djed remains for future work.

With the benefit of hindsight, the decision to focus on a crypto-backed stablecoin with reserve ratio greater than one also turned out to be wise, considering that recent stablecoin projects that attempted to achieve stability with fractional reserves (e.g. FEI [22] and Iron [23]) failed.

Despite the difficulty of the challenge, we believe that designing a stablecoin with fractional reserve (or no reserve at all) is a worthy endeavor, especially because, as is well-known, it is not economically efficient to leave capital idling in the reserve. Although backing and collateralization facilitate stability and tend to increase the public's trust on the stability, it is not impossible to achieve stability without them. To see this, consider that a central bank digital currency (CBDC) would be pegged to the central bank's fiat currency and, like the fiat currency itself, would not need to be backed by anything, by definition of *fiat*. And, despite this, it would nevertheless be stable in relation to the prices of goods and services, at least in the short term, just like their associated fiat currencies in countries that enjoy low inflation.

In the case of Djed, a path of future work towards fractional reserves could start by exploring more deeply the debt for equity swap mechanism described in Subsection 5.4.1. In the meantime, until a provably stable fractional or unbacked stablecoin is created, the concern of reserve capital inefficiency can be partially addressed at the implementation level by staking or lending the reserve, depending on the availability of reliable, safe and secure decentralized staking or lending protocols on the chains where the stablecoin is deployed.

References

1. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language lustre," in *Proceedings of the IEEE*, vol. 79, no. 9, 1991, pp. 1305–1320.
2. A. Champion, A. Mebsout, C. Sticksel, and C. Tinelli, "The kind 2 model checker," in *Computer Aided Verification*. Springer International Publishing, 2016, pp. 510–517.
3. A. Biere, "Bounded Model Checking," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, vol. 185, pp. 457–481.

[22] https://www.coindesk.com/1b-fei-stablecoins-rocky-start-is-a-wake-up-call-for-defi-investors

[23] https://cointelegraph.com/news/iron-finance-bank-run-stings-investors-a-lesson-for-all-stablecoins

4. N. Een and N. Sörensson, "Temporal Induction by Incremental SAT Solving," *Electronic Notes in Theoretical Computer Science*, vol. 89, pp. 543–560, December 2003.
5. L. Paulson and T. Nipkow, *Isabelle: A Generic Theorem Prover*, ser. Lecture Notes in Computer Science. Springer, 1994. [Online]. Available: https://books.google.com.ar/books?id=RxlhqG0-cGwC
6. T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, ser. LNCS. Springer, 2002, vol. 2283.
7. M. Peyton Jones and R. Kireev, "The plutus platform," 2021. [Online]. Available: https://hydra.iohk.io/build/6641339/download/1/plutus.pdf
8. M. P. Jones, V. Gkoumas, R. Kireev, K. MacKenzie, C. Nester, and P. Wadler, "Unraveling recursion: Compiling an IR with recursion to system F," in *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*, ser. Lecture Notes in Computer Science, G. Hutton, Ed., vol. 11825. Springer, 2019, pp. 414–443. [Online]. Available: https://doi.org/10.1007/978-3-030-33636-3_15
9. J. Chapman, R. Kireev, C. Nester, and P. Wadler, "System F in agda, for fun and profit," in *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings*, ser. Lecture Notes in Computer Science, G. Hutton, Ed., vol. 11825. Springer, 2019, pp. 255–297. [Online]. Available: https://doi.org/10.1007/978-3-030-33636-3_10
10. J. Zahnentferner, "Multi-currency ledgers," Cryptology ePrint Archive, Report 2020/895, 2020, https://eprint.iacr.org/2020/895.
11. M. M. T. Chakravarty, J. Chapman, K. MacKenzie, O. Melkonian, J. Müller, M. P. Jones, P. Vinogradova, P. Wadler, and J. Zahnentferner, "$\text{Utxo}_{\text{sf ma}}$: UTXO with multi-asset support," in *Leveraging Applications of Formal Methods, Verification and Validation: Applications - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part III*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds., vol. 12478. Springer, 2020, pp. 112–130. [Online]. Available: https://doi.org/10.1007/978-3-030-61467-6_8
12. J. Zahnentferner, "Eliminating contract execution redundancy (short note)," 2020.
13. R. Sams, "A note on cryptocurrency stabilisation: Seigniorage shares," 2014. [Online]. Available: https://blog.bitmex.com/wp-content/uploads/2018/06/A-Note-on-Cryptocurrency-Stabilisation-Seigniorage-Shares.pdf

A Price for buying reservecoins in the continuous setting

As was discussed in Section [5.1 Price for Buying Reservecoins], calculation of the price using iterative equation (11) is inefficient. Reducing the equation to a closed form with constant complexity is possible by switching to the continuous setting. The basic idea is to assume that each coin is divisible into infinite number of pieces (so that each piece approaches zero) and to recalculate the price after buying each smallest piece. In such a setting we are in the field of mathematical analysis which allows to derive the necessary function. Moreover, in this case the price calculation is more precise compared to the discrete setting making impossible to manipulate the price by splitting/combining operations.

Note that the price of buying N reservecoins is a difference between the amount of

reserves before and after the operation:

$$P_{rc}^{buy}(N) = R_N - R_0, \qquad R_N = R_0 + P_{rc}^{buy}(N).$$

Let $R(t), N_{RC}(t)$ be continuous functions on $[0, N]$ defining the amount of reserves and reservecoins in circulation at time t (i.e., after buying t reservecoins), where $R(0) = R_0$ and $N_{RC}(0) = N_{RC_0}$ are the initial amounts. Then, the price $P_{rc}^{buy}(t)$ of buying t coins can be defined as a derivative of the function $R(t)$:

$$\frac{dR(t)}{dt} = P_{rc}^{buy}(t),$$

$$\frac{dR(t)}{dt} = \frac{R(t) - L(t)}{N_{RC}(t)}(1 + fee(t)), \qquad N_{RC}(t) = N_{RC_0} + t, \tag{25}$$

$$L(t) = \begin{cases} L^t, & \text{if } r(t) \geq r_{peg}, \\ \frac{R(t)}{r_{peg}}, & \text{if } r(t) < r_{peg}, \end{cases}$$

$$fee(t) = \begin{cases} fee_0 + k_{rm} \cdot \frac{R(t) - L^t \cdot r_{opt}}{L^t \cdot r_{opt}}, & \text{if } r(t) \geq r_{opt}, \\ fee_0, & \text{if } r(t) < r_{opt}, \end{cases}$$

where $r(t) = \frac{R(t)}{L^t}$ is a reserve ratio after buying t coins.

Equation (25) is a first-order separable differential equation with two variables t and $R(t)$. The solution provides the unknown function $R(t)$. Then, the price of buying N reservecoins can be calculated as follows:

$$P_{rc}^{buy}(N) = R(N) - R_0. \tag{26}$$

We solve (25) by moving different variables into different sides and integrating each side separately on the interval $[0, N]$:

$$\frac{dR(t)}{(R(t) - L(t))(1 + fee(t))} = \frac{dt}{N_{RC}(t)},$$

$$\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t) - L(t))(1 + fee(t))} = \int_0^N \frac{dt}{N_{RC_0} + t}. \tag{27}$$

Note that $L(t)$ and $fee(t)$ are piecewise linear functions depending on the variable $R(t)$. To solve the integral on the left-hand side of (27), we need to decompose it into intervals defined by $L(t)$ and $fee(t)$. Depending on what intervals R_0 and $R(N)$ fall into, the integral might be decomposed in 6 different ways:

$$\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t)-L(t))(1+fee(t))} =$$

$$\begin{cases}
\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t)-\frac{R(t)}{r_{peg}})(1+fee_0)}, & \text{if } R_0 < R(N) < r_{peg} \cdot L^t, \\
\int_{R(0)}^{r_{peg} \cdot L^t} \frac{dR(t)}{(R(t)-\frac{R(t)}{r_{peg}})(1+fee_0)} + & \\
\quad + \int_{r_{peg} \cdot L^t}^{R(N)} \frac{dR(t)}{(R(t)-L^t)(1+fee_0)}, & \text{if } R_0 < r_{peg} \cdot L^t \leq R(N) < r_{opt} \cdot L^t, \\
\int_{R(0)}^{r_{peg} \cdot L^t} \frac{dR(t)}{(R(t)-\frac{R(t)}{r_{peg}})(1+fee_0)} + & \\
\quad + \int_{r_{peg} \cdot L^t}^{r_{opt} \cdot L^t} \frac{dR(t)}{(R(t)-L^t)(1+fee_0)} + & \\
\quad + \int_{r_{opt} \cdot L^t}^{R(N)} \frac{dR(t)}{(R(t)-L^t)\left(1+fee_0+k_{rm}\frac{R(t)-L^t \cdot r_{opt}}{L^t \cdot r_{opt}}\right)}, & \text{if } R_0 < r_{peg} \cdot L^t < r_{opt} \cdot L^t \leq R(N), \\
\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t)-L^t)(1+fee_0)}, & \text{if } r_{peg} \cdot L^t \leq R_0 < R(N) < r_{opt} \cdot L^t, \\
\int_{R(0)}^{r_{opt} \cdot L^t} \frac{dR(t)}{(R(t)-L^t)(1+fee_0)} + & \\
\quad + \int_{r_{opt} \cdot L^t}^{R(N)} \frac{dR(t)}{(R(t)-L^t)\left(1+fee_0+k_{rm}\frac{R(t)-L^t \cdot r_{opt}}{L^t \cdot r_{opt}}\right)}, & \text{if } r_{peg} \cdot L^t \leq R_0 < r_{opt} \cdot L^t \leq R(N), \\
\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t)-L^t)\left(1+fee_0+k_{rm}\frac{R(t)-L^t \cdot r_{opt}}{L^t \cdot r_{opt}}\right)}, & \text{if } r_{peg} \cdot L^t < r_{opt} \cdot L^t \leq R_0 < R(N).
\end{cases}$$

Solving all variants separately we obtain the following solutions for $R(N)$:

$$R(N) = \begin{cases}
R_1(N), & \text{if } R_0 < R(N) < r_{peg} \cdot L^t, \\
R_2(N), & \text{if } R_0 < r_{peg} \cdot L^t \leq R(N) < r_{opt} \cdot L^t, \\
R_3(N), & \text{if } R_0 < r_{peg} \cdot L^t < r_{opt} \cdot L^t \leq R(N), \\
R_4(N), & \text{if } r_{peg} \cdot L^t \leq R_0 < R(N) < r_{opt} \cdot L^t, \\
R_5(N), & \text{if } r_{peg} \cdot L^t \leq R_0 < r_{opt} \cdot L^t \leq R(N), \\
R_6(N), & \text{if } r_{peg} \cdot L^t < r_{opt} \cdot L^t \leq R_0 < R(N),
\end{cases} \tag{28}$$

where $R_1(N), .., R_6(N)$ are defined by (29), (30), (31), (32), (33), and (34) correspondingly. We omit full details of solving the equations as it is straightforward using well-known integration techniques.

Since we do not know beforehand what exactly interval the final result $R(N)$ will fall into, we have to calculate all possible variants and pick the correct one by analyzing the results. For example, if initial reserves $R_0 < r_{peg} \cdot L^t$, we need to calculate $R_1(N)$, $R_2(N)$, and $R_3(N)$. Then, by observing where final results appear relatively to $r_{peg} \cdot L^t$ and $r_{opt} \cdot L^t$, we pick the correct one and neglect the other two.

The algorithm (1) defines the procedure of calculating new reserves $R(N)$. Then, the price of buying N reservecoins can be calculated using (26).

Algorithm 1 Calculating new reserves when buying reservecoins

```
1: procedure CalculateNewReserves
2:     R_0 ← initial amount of reserves
3:     if R_0 < r_peg · L^t then
4:         R_1 ← R_1(N)
5:         if R_1 < r_peg · L^t then
6:             newReserves ← R_1
7:         else
8:             R_2 ← R_2(N)
9:             if r_peg · L^t ≤ R_2 and R_2 < r_opt · L^t then
10:                newReserves ← R_2
11:            else
12:                newReserves ← R_3(N)
13:    else if r_peg · L^t ≤ R_0 and R_0 < r_opt · L^t then
14:        R_4 ← R_4(N)
15:        if R_4 < r_opt · L^t then
16:            newReserves ← R_4
17:        else
18:            newReserves ← R_5(N)
19:    else
20:        newReserves ← R_6(N)
21: return newReserves
```

A.1 The first variant ($R_0 < R(N) < r_{peg} \cdot L^t$)

$$R_1(N) = R_0 \cdot \left(\frac{N_{RC_0} + N}{N_{RC_0}}\right)^X, \tag{29}$$

where $X = (1 - \frac{1}{r_{peg}})(1 + fee_0)$.

A.2 The second variant ($R_0 < r_{peg} \cdot L^t \leq R(N) < r_{opt} \cdot L^t$)

$$R_2(N) = \left(\frac{N_{RC_0} + N}{N_{RC_0}}\right)^{1+fee_0} \cdot \left(\frac{r_{peg} \cdot L^t}{R_0}\right)^{-X} \cdot \left(r_{peg} \cdot L^t - L^t\right) + L^t, \tag{30}$$

where $X = \frac{1}{1-\frac{1}{r_{peg}}}$.

A.3 The third variant ($R_0 < r_{peg} \cdot L^t < r_{opt} \cdot L^t \leq R(N)$)

$$R_3(N) = \frac{Z \cdot E + V \cdot L^t}{V - Z\frac{k_{rm}}{L^t \cdot r_{opt}}}, \tag{31}$$

where

$$Z = \left(\frac{(N_{RC_0} + N)(R_0)^X (r_{peg} - 1)^Y}{N_{RC_0} \cdot (r_{peg} \cdot L^t)^X (r_{opt} - 1)^Y}\right)^{\frac{1}{A}}, \qquad X = \frac{Y}{1 - \frac{1}{r_{peg}}}, \qquad Y = \frac{1}{1 + fee_0},$$

$$V = \frac{1 + fee_0}{r_{opt} \cdot L^t - L^t}, \qquad E = 1 + fee_0 - k_{rm}, \qquad A = \frac{r_{opt}}{r_{opt} \cdot E + k_{rm}}.$$

A.4 The fourth variant ($r_{peg} \cdot L^t \leq R_0 < R(N) < r_{opt} \cdot L^t$)

$$R_4(N) = (R_0 - L^t)\left(\frac{N_{RC_0} + N}{N_{RC_0}}\right)^{1+fee_0} + L^t. \tag{32}$$

A.5 The fifth variant ($r_{peg} \cdot L^t \leq R_0 < r_{opt} \cdot L^t \leq R(N)$)

$$R_5(N) = \frac{Z \cdot E + V \cdot L^t}{V - Z\frac{k_{rm}}{L^t \cdot r_{opt}}}, \tag{33}$$

where

$$V = \frac{1 + fee_0}{r_{opt} \cdot L^t - L^t}, \qquad E = 1 + fee_0 - k_{rm},$$

$$Z = \left(\frac{N_{RC_0} + N}{N_{RC_0}} \cdot \left(\frac{R_0 - L^t}{r_{opt} \cdot L^t - L^t}\right)^{\frac{1}{1+fee_0}}\right)^{\frac{1}{A}}, \qquad A = \frac{r_{opt}}{r_{opt} \cdot E + k_{rm}}.$$

A.6 The sixth variant ($r_{peg} \cdot L^t < r_{opt} \cdot L^t \leq R_0 < R(N)$)

$$R_6(N) = \frac{Z \cdot E + V \cdot L^t}{V - Z\frac{k_{rm}}{L^t \cdot r_{opt}}}. \tag{34}$$

where

$$V = \frac{1 + fee_0 + k_{rm}\frac{R_0 - L^t \cdot r_{opt}}{L^t \cdot r_{opt}}}{R_0 - L^t}, \qquad Z = \left(\frac{N_{RC_0} + N}{N_{RC_0}}\right)^{\frac{1}{A}},$$

$$E = 1 + fee_0 - k_{rm}, \qquad A = \frac{r_{opt}}{r_{opt} \cdot E + k_{rm}}.$$

B Price for selling reservecoins in the continuous setting

The same techniques as for buying are used to calculate the reservecoins selling price. We omit to repeat some notations and explanations as they are practically the same as in Appendix [A]. Read Appendix [A] first for better understanding.

The price $P_{rc}^{sell}(t)$ of selling t coins can be defined as a derivative of the function $R(t)$ in the following way:

$$\frac{dR(t)}{dt} = -P_{rc}^{sell}(t),$$

$$\frac{dR(t)}{dt} = -\frac{R(t) - L(t)}{N_{RC}(t)}(1 - fee(t)), \qquad N_{RC}(t) = N_{RC_0} - t, \tag{35}$$

$$L(t) = \begin{cases} L^t, & \text{if } r(t) \geq r_{peg}, \\ \frac{R(t)}{r_{peg}}, & \text{if } r(t) < r_{peg}, \end{cases}$$

$$fee(t) = \begin{cases} fee_0 + k_{rr} \cdot \frac{L^t \cdot r_{opt} - R(t)}{L^t \cdot r_{opt}}, & \text{if } r(t) \leq r_{opt}, \\ fee_0, & \text{if } r(t) > r_{opt}. \end{cases}$$

Equation (35) is a first-order separable differential equation with two variables t and $R(t)$. The solution provides the unknown function $R(t)$. Then, the price of selling N reservecoins is calculated as follows:

$$P_{rc}^{sell}(N) = R_0 - R(N). \tag{36}$$

The solution of (35) is similar to the one of (25). The variables are moved into different sides and integrated separately on the interval $[0, N]$:

$$\frac{dR(t)}{(R(t) - L(t))(1 - fee(t))} = -\frac{dt}{N_{RC}(t)},$$

$$\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t) - L(t))(1 - fee(t))} = -\int_0^N \frac{dt}{N_{RC_0} + t}. \tag{37}$$

The integral on the left-hand side of (37) has to be decomposed into intervals defined by $L(t)$ and $fee(t)$. Depending on what intervals R_0 and $R(N)$ fall into, it might be decomposed in 6 different ways:

$$\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t) - L(t))(1 - fee(t))} =$$

$$\begin{cases}
\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t)-\frac{R(t)}{r_{peg}})\left(1-fee_0-k_{rr}\frac{L^t\cdot r_{opt}-R(t)}{L^t\cdot r_{opt}}\right)}, & \text{if } R(N) < R_0 < r_{peg}\cdot L^t < r_{opt}\cdot L^t, \\
\int_{R(0)}^{r_{peg}\cdot L^t} \frac{dR(t)}{(R(t)-L^t)\left(1-fee_0-k_{rr}\frac{L^t\cdot r_{opt}-R(t)}{L^t\cdot r_{opt}}\right)} + & \\
\quad + \int_{r_{peg}\cdot L^t}^{R(N)} \frac{dR(t)}{(R(t)-\frac{R(t)}{r_{peg}})\left(1-fee_0-k_{rr}\frac{L^t\cdot r_{opt}-R(t)}{L^t\cdot r_{opt}}\right)}, & \text{if } R(N) < r_{peg}\cdot L^t \le R_0 < r_{opt}\cdot L^t, \\
\int_{R(0)}^{r_{opt}\cdot L^t} \frac{dR(t)}{(R(t)-L^t)(1-fee_0)} + & \\
\quad + \int_{r_{peg}\cdot L^t}^{r_{opt}\cdot L^t} \frac{dR(t)}{(R(t)-L^t)\left(1-fee_0-k_{rr}\frac{L^t\cdot r_{opt}-R(t)}{L^t\cdot r_{opt}}\right)} + & \\
\quad + \int_{r_{peg}\cdot L^t}^{R(N)} \frac{dR(t)}{(R(t)-\frac{R(t)}{r_{peg}})\left(1-fee_0-k_{rr}\frac{L^t\cdot r_{opt}-R(t)}{L^t\cdot r_{opt}}\right)}, & \text{if } R(N) < r_{peg}\cdot L^t < r_{opt}\cdot L^t \le R_0, \\
\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t)-L^t)\left(1-fee_0-k_{rr}\frac{L^t\cdot r_{opt}-R(t)}{L^t\cdot r_{opt}}\right)}, & \text{if } r_{peg}\cdot L^t \le R(N) < R_0 < r_{opt}\cdot L^t, \\
\int_{R(0)}^{r_{opt}\cdot L^t} \frac{dR(t)}{(R(t)-L^t)(1-fee_0)} + & \\
\quad + \int_{r_{opt}\cdot L^t}^{R(N)} \frac{dR(t)}{(R(t)-L^t)\left(1-fee_0-k_{rr}\frac{L^t\cdot r_{opt}-R(t)}{L^t\cdot r_{opt}}\right)}, & \text{if } r_{peg}\cdot L^t \le R(N) < r_{opt}\cdot L^t \le R_0, \\
\int_{R(0)}^{R(N)} \frac{dR(t)}{(R(t)-L^t)(1-fee_0)}, & \text{if } r_{peg}\cdot L^t < r_{opt}\cdot L^t \le R(N) < R_0.
\end{cases}$$

Solving all variants separately we obtain the following solutions for $R(N)$:

$$R(N) = \begin{cases}
R_1(N), & \text{if } R(N) < R_0 < r_{peg}\cdot L^t < r_{opt}\cdot L^t, \\
R_2(N), & \text{if } R(N) < r_{peg}\cdot L^t \le R_0 < r_{opt}\cdot L^t, \\
R_3(N), & \text{if } R(N) < r_{peg}\cdot L^t < r_{opt}\cdot L^t \le R_0, \\
R_4(N), & \text{if } r_{peg}\cdot L^t \le R(N) < R_0 < r_{opt}\cdot L^t, \\
R_5(N), & \text{if } r_{peg}\cdot L^t \le R(N) < r_{opt}\cdot L^t \le R_0, \\
R_6(N), & \text{if } r_{peg}\cdot L^t < r_{opt}\cdot L^t \le R(N) < R_0,
\end{cases} \tag{38}$$

where $R_1(N), .., R_6(N)$ are defined by (39), (40), (41), (42), (43), and (44) correspondingly. We omit full details of solving the equations as it is straightforward using well-known integration techniques.

The algorithm (2) defines the procedure of calculating new reserves $R(N)$. Then, the price of selling N reservecoins can be calculated using (36).

Algorithm 2 Calculating new reserves when selling reservecoins

```
1: procedure CalculateNewReserves
2:     R_0 ← initial amount of reserves
3:     if R_0 < r_peg · L^t then
4:         newReserves ← R_1(N)
5:     else if r_peg · L^t ≤ R_0  and  R_0 < r_opt · L^t then
6:         R_4 ← R_4(N)
7:         if R_4 ≥ r_peg · L^t then
8:             newReserves ← R_4
9:         else
10:            newReserves ← R_2(N)
11:    else
12:        R_6 ← R_6(N)
13:        if R_6 ≥ r_opt · L^t then
14:            newReserves ← R_6
15:        else
16:            R_5 ← R_5(N)
17:            if R_5 ≥ r_peg · L^t then
18:                newReserves ← R_5
19:            else
20:                newReserves ← R_3(N)
21: return newReserves
```

B.1 The first variant ($R(N) < R_0 < r_{peg} \cdot L^t$)

$$R_1(N) = \frac{Z \cdot E}{V - Z\frac{k_{rr}}{L^t \cdot r_{opt}}}, \tag{39}$$

where

$$Z = \left(\frac{N_{RC_0} - N}{N_{RC_0}}\right)^{\frac{(r_{peg}-1)\cdot E}{r_{peg}}}, \qquad E = 1 - fee_0 - k_{rr}, \qquad V = \frac{E + R_0\frac{k_{rr}}{L^t \cdot r_{opt}}}{R_0}.$$

B.2 The second variant ($R(N) < r_{peg} \cdot L^t \leq R_0 < r_{opt} \cdot L^t$)

$$R_2(N) = \frac{Z \cdot E}{V - Z\frac{k_{rr}}{L^t \cdot r_{opt}}}, \tag{40}$$

where

$$Z = \left(\frac{N_{RC_0} - N}{N_{RC_0}} \cdot \left(\frac{(R_0 - L^t)\left(E + \frac{r_{peg} \cdot k_{rr}}{r_{opt}}\right)}{(r_{peg} \cdot L^t - L^t)\left(E + R_0 \frac{k_{rr}}{L^t \cdot r_{opt}}\right)} \right)^C \right)^{E - \frac{E}{r_{peg}}}$$

$$E = 1 - fee_0 - k_{rr}, \qquad C = \frac{r_{opt}}{r_{opt} \cdot E + k_{rr}}, \qquad V = \frac{E + \frac{r_{peg} \cdot k_{rr}}{r_{opt}}}{r_{peg} \cdot L^t}.$$

B.3 The third variant ($R(N) < r_{peg} \cdot L^t < r_{opt} \cdot L^t \leq R_0$)

$$R_3(N) = \frac{Z \cdot E}{V - Z \frac{k_{rr}}{L^t \cdot r_{opt}}}, \tag{41}$$

where

$$Z = \left(\frac{N_{RC_0} - N}{N_{RC_0}} \cdot \left(\frac{R_0 - L^t}{(r_{opt} - 1)L^t} \right)^{(1 - fee_0)^{-1}} \cdot \left(\frac{r_{opt} - 1}{r_{peg} - 1} \cdot \frac{E + \frac{r_{peg} \cdot k_{rr}}{r_{opt}}}{1 - fee_0} \right)^C \right)^{E - \frac{E}{r_{peg}}},$$

$$E = 1 - fee_0 - k_{rr}, \qquad C = \frac{r_{opt}}{r_{opt} \cdot E + k_{rr}}, \qquad V = \frac{E + \frac{r_{peg} \cdot k_{rr}}{r_{opt}}}{r_{peg} \cdot L^t}.$$

B.4 The fourth variant ($r_{peg} \cdot L^t \leq R(N) < R_0 < r_{opt} \cdot L^t$)

$$R_4(N) = \frac{Z \cdot E + V \cdot L^t}{V - Z \frac{k_{rr}}{L^t \cdot r_{opt}}}, \tag{42}$$

where

$$Z = \left(\frac{N_{RC_0} - N}{N_{RC_0}} \right)^{\frac{r_{opt} \cdot E + k_{rr}}{r_{opt}}}, \qquad E = 1 - fee_0 - k_{rr}, \qquad V = \frac{E + R_0 \frac{k_{rr}}{L^t \cdot r_{opt}}}{R_0 - L^t}.$$

B.5 The fifth variant ($r_{peg} \cdot L^t \leq R(N) < r_{opt} \cdot L^t \leq R_0$)

$$R_5(N) = \frac{Z \cdot E + V \cdot L^t}{V - Z \frac{k_{rr}}{L^t \cdot r_{opt}}}, \tag{43}$$

where

$$Z = \left(\left(\frac{N_{RC_0} - N}{N_{RC_0}} \right) \cdot \left(\frac{R_0 - L^t}{r_{opt} \cdot L^t - L^t} \right)^{\frac{1}{1 - fee_0}} \right)^{\frac{1}{C}},$$

$$E = 1 - fee_0 - k_{rr}, \qquad C = \frac{r_{opt}}{r_{opt} \cdot E + k_{rr}}, \qquad V = \frac{1 - fee_0}{r_{opt} \cdot L^t - L^t}.$$

B.6 The sixth variant ($r_{peg} \cdot L^t < r_{opt} \cdot L^t \leq R(N) < R_0$)

$$R_6(N) = (R_0 - L^t)\left(\frac{N_{RC_0} - N}{N_{RC_0}} \right)^{1 - fee_0} + L^t. \tag{44}$$

C Price for buying stablecoins in the continuous setting

Recall that the price of buying N stablecoins in the discrete setting was defined by (16) as:

$$P_{sc}^{buy}(N) = \sum_{i=0}^{N-1} P_{sc,i}^{buy} = \sum_{i=0}^{N-1} P_{SC}^{t} \cdot (1 + fee(R_i, N_{SC_i})),$$

$$fee(R_i, N_{SC_i}) = \begin{cases} fee_0, & \text{if } r_i \geq r_{opt}, \\ fee_0 + k_{sm} \cdot \frac{L_i^t \cdot r_{opt} - R_i}{L_i^t \cdot r_{opt}}, & \text{if } r_i < r_{opt}, \end{cases}$$

where $L_i^t = N_{SC_i} P_{SC}^t$, $N_{SC_i} = N_{SC_0} + i$, and $r_i = \frac{R_i}{L_i^t}$.

Recall also that the price is a difference between the amount of reserves before and after the operation:

$$R_N = R_0 + P_{sc}^{buy}(N).$$

Let $R(t)$, $N_{SC}(t)$ be continuous functions on $[0, N]$ defining the amount of reserves and issued stablecoins at time t (i.e., after buying t stablecoins), where $R(0) = R_0$ and $N_{SC}(0) = N_{SC_0}$ are the initial amounts. Then, the price $P_{sc}^{buy}(t)$ of buying t coins is a derivative of the function $R(t)$:

$$\frac{dR(t)}{dt} = P_{sc}^{buy}(t), \qquad N_{SC}(t) = N_{SC_0} + t,$$

$$\frac{dR(t)}{dt} = P_{SC}^{t}(1 + fee(t)), \tag{45}$$

where

$$fee(t) = \begin{cases} fee_0, & \text{if } r(t) \geq r_{opt}, \\ fee_0 + k_{sm} \cdot \frac{L^t(t) \cdot r_{opt} - R(t)}{L^t(t) \cdot r_{opt}}, & \text{if } r(t) < r_{opt}, \end{cases}$$

$$L^t(t) = N_{SC}(t) \cdot P_{SC}^t, \qquad r(t) = \frac{R(t)}{L^t(t)}.$$

The solution of the differential equation (45) provides the unknown function $R(t)$. It is solved separately for every interval defined by $fee(t)$:

$$\begin{cases} \frac{dR(t)}{dt} = P_{SC}^{t}(1 + fee_0), & \text{if } r(t) \geq r_{opt}, \\ \frac{dR(t)}{dt} = P_{SC}^{t}(1 + fee_0 + k_{sm} \cdot \frac{L^t(t) \cdot r_{opt} - R(t)}{L^t(t) \cdot r_{opt}}), & \text{if } r(t) < r_{opt}. \end{cases} \tag{46}$$

Note that we do not consider $r < r_{peg}$ because buying of stablecoins is not allowed in this case.

Solving (46) we will obtain:

$$R(t) = \begin{cases} R_1(t) = t \cdot P_{SC}^t(1 + fee_0) + R_0, & \text{if } r(t) \geq r_{opt}, \\ R_2(t, R_0, N_{SC_0}) = P\frac{N_{SC_0} + t}{1+K} + \frac{\left(R_0 - P\frac{N_{SC_0}}{1+K}\right)(N_{SC_0})^K}{(N_{SC_0} + t)^K}, & \text{if } r(t) < r_{opt}, \end{cases} \tag{47}$$

where $P = P_{SC}^t \cdot (1 + fee_0 + k_{sm})$ and $K = \frac{k_{sm}}{r_{opt}}$. We omit full details of solving the equations noting only that the first one is a simple separable differential equation and the other one is a first-order non-homogeneous linear differential equation solved by using variation of parameters method[24].

[24] www.sfu.ca/math-coursenotes/../sec_first_order_homogeneous_linear.html

The equation (47) can be used to calculate reserves only when the reserve ratio belongs strictly to one of the intervals, i.e., either $r(t) \geq r_{opt}$ or $r(t) < r_{opt}$ for every $t \in [0..N]$.

Given that the reserve ratio is decreased when stablecoins are bought it is possible that starting with the initial ratio $r(0) > r_{opt}$, at some point $X < N$ the ratio will drop to $r(X) = r_{peg}$. In this case, the new reserve should be calculated using both intervals defined by (47). Let $R_3(N)$ define the reserve in such a case:

$$R_3(N) = R_1(X) + R_2(N', R_0', N_{SC_0}^{'}) - R_0, \tag{48}$$

where

$$N' = N - X, \qquad R_0' = R_1(X), \qquad N_{SC_0}^{'} = N_{SC_0} + X.$$

X is basically an amount of stablecoins that should be bought until the optimal reserve ratio is reached. It can be found from the following equation:

$$r_{opt} \cdot L^t(X) = R_1(X), \qquad L^t(X) = P_{SC}^t(N_{SC_0} + X),$$

$$r_{opt} \cdot P_{SC}^t(N_{SC_0} + X) = X \cdot P_{SC}^t(1 + fee_0) + R_0,$$

$$X = \frac{R_0 - r_{opt} \cdot P_{SC}^t \cdot N_{SC_0}}{P_{SC}^t(r_{opt} - 1 - fee_0)}. \tag{49}$$

Putting everything together, the function $R(N)$ that defines reserves after buying N stablecoins is the following:

$$R(N) = \begin{cases} R_1(N), & \text{if } r(0) > r(N) \geq r_{opt}, \\ R_2(N, R_0, N_{SC_0}), & \text{if } r(N) < r(0) \leq r_{opt}, \\ R_3(N), & \text{if } r(N) < r_{opt} < r(0). \end{cases}$$

where $R_1(N)$, $R_2(N)$ are defined by (47) and $R_3(N)$ is defined by (48).

The algorithm (3) defines the procedure of selecting correct equation for calculating new reserves $R(N)$. Then, the price of buying N stablecoins is $P_{sc}^{buy}(N) = R(N) - R_0$.

Algorithm 3 Calculating new reserves when buying stablecoins

```
1: procedure CalculateNewReserves
2:     R_0 ← initial amount of reserves
3:     r_0 ← initial reserves ratio
4:     N_SC0 ← initial amount of stablecoins
5:     N ← number of stablecoins to be bought
6:     if r_0 > r_opt then
7:         X ← number of stablecoins to be bought until optimum is reached
8:         if N ≤ X then
9:             newReserves ← R_1(N)
10:        else
11:            newReserves ← R_3(N)
12:    else
13:        newReserves ← R_2(N, R_0, N_SC0)
14: return newReserves
```

D Price for selling stablecoins in the continuous setting

The amount of returned basecoins for selling N stablecoins in the discrete setting was defined by (19) as:

$$P_{sc}^{sell}(N) = \sum_{i=0}^{N-1} P_{sc,i}^{sell} = \sum_{i=0}^{N-1} k(r_i) \cdot P_{SC}^t \cdot (1 - fee(R_i, N_{SC_i})).$$

$$fee(R_i, N_{SC_i}) = \begin{cases} fee_0 + k_{sr} \cdot \frac{R_i - L_i^t \cdot r_{opt}}{L_i^t \cdot r_{opt}}, & \text{if } r_i > r_{opt}, \\ fee_0, & \text{if } r_i \leq r_{opt}, \end{cases} \qquad k(r_i) = \min(1, \frac{r_i}{r_{peg}})$$

where $L_i^t = N_{SC_i} \cdot P_{SC}^t$, $N_{SC_i} = N_{SC_0} - i$, and $r_i = \frac{R_i}{L_i^t}$.

Let $R(t)$, $N_{SC}(t)$ be continuous functions on $[0, N]$ defining the amount of reserves and issued stablecoins at time t (i.e., after selling t stablecoins), where $R(0) = R_0$ and $N_{SC}(0) = N_{SC_0}$ are the initial amounts. Then, the price $P_{sc}^{sell}(t)$ of selling t coins is a derivative of the function $R(t)$:

$$\frac{dR(t)}{dt} = -P_{sc}^{sell}(t), \qquad N_{SC}(t) = N_{SC_0} - t,$$

$$\frac{dR(t)}{dt} = -k(r(t)) \cdot P_{SC}^t (1 - fee(t)). \tag{50}$$

The solution of the differential equation (50) provides the unknown function $R(t)$. It is solved separately for every interval defined by $fee(t)$ and $k(r(t))$:

$$\begin{cases} \frac{dR(t)}{dt} = -P_{SC}^t (1 - fee_0 - k_{sr} \cdot \frac{R(t) - L^t(t) \cdot r_{opt}}{L^t(t) \cdot r_{opt}}), & \text{if } r(t) > r_{opt}, \\ \frac{dR(t)}{dt} = -P_{SC}^t (1 - fee_0), & \text{if } r_{peg} \leq r(t) \leq r_{opt}, \\ \frac{dR(t)}{dt} = -\frac{r(t)}{r_{peg}} P_{SC}^t (1 - fee_0), & \text{if } r(t) < r_{peg}. \end{cases} \tag{51}$$

Solving (51) provides:

$$R(t) = \begin{cases} R_1(t, R_0, N_{SC_0}) = \\ \quad -P\frac{N_{SC_0} - t}{1+K} + \frac{\left(R_0 + P\frac{N_{SC_0}}{1+K}\right)(N_{SC_0})^K}{(N_{SC_0} - t)^K}, & \text{if } r(t) > r_{opt}, \\ R_2(t, R_0) = & \\ \quad R_0 \ - \ t \cdot P_{SC}^t (1 - fee_0), & \text{if } r_{peg} \leq r(t) \leq r_{opt}, \\ R_3(t) = R_0 \cdot \left(\frac{N_{SC_0} - t}{N_{SC_0}}\right)^{\frac{1 - fee_0}{r_{peg}}}, & \text{if } r(t) < r_{peg}. \end{cases} \tag{52}$$

where $P = -P_{SC}^t \cdot (1 - fee_0 + k_{sm})$ and $K = \frac{k_{sm}}{r_{opt}}$. We omit full details of solving the equations, they are solved analogously to the equations in the previous sections using well-known mathematical techniques.

Note that the equation (52) can be used to calculate reserves only when reserve ratio belongs strictly to one of the intervals for every $t \in [0..N]$. Given that the reserve ratio is increased when stablecoins are sold it is possible that it will stretch to several intervals in which case we need to use different equations to calculate the price for each particular interval.

(53) shows all possible variants of calculating reserves depending on the dynamics of the reserve ratio.

$$R(N) = \begin{cases} R_1(N, R_0, N_{SC_0}), & \text{if } r_{peg} < r_{opt} \leq r(0) < r(N), \\ R_2(N, R_0), & \text{if } r_{peg} \leq r(0) < r(N) \leq r_{opt}, \\ R_3(N), & \text{if } r(0) < r(t) < r_{peg} < r_{opt}, \\ R_4(N, R_0, N_{SC_0}) = & \\ \quad R_2(X, R_0) - R_0 + & \\ \quad + R_1(N - X, R_0', N_{SC_0}^{'}) & \text{if } r_{peg} \leq r(0) < r_{opt} < r(N), \\ R_5(N) = R_3(Y) - R_0 + & \\ \quad + R_4(N - Y, R_0'', N_{SC_0}^{''}), & \text{if } r(0) < r_{peg} < r_{opt} < r(N), \\ R_6(N) = R_3(Y) - R_0 + & \\ \quad + R_2(N - Y, R_0''), & \text{if } r(0) < r_{peg} < r(N) \leq r_{opt}. \end{cases} \tag{53}$$

where

X – amount of stablecoins to be sold until optimal reserve ratio is reached;
Y – amount of stablecoins to be sold until minimal reserve ratio (r_{peg}) is reached;
$R_0' = R_2(X)$, $N_{SC_0}^{'} = N_{SC_0} - X$;
$R_0'' = R_3(Y)$, $N_{SC_0}^{''} = N_{SC_0} - Y$.

X can be found from the following equation:

$$\begin{gathered} r_{opt} \cdot L^t(X) = R_2(X), \qquad L^t(X) = P_{SC}^t(N_{SC_0} - X), \\ r_{opt} \cdot P_{SC}^t(N_{SC_0} - X) = R_0 - X \cdot P_{SC}^t(1 - fee_0), \\ X(R_0, N_{SC_0}) = \frac{R_0 - r_{opt} \cdot P_{SC}^t \cdot N_{SC_0}}{P_{SC}^t(1 - fee_0 - r_{opt})}. \end{gathered} \tag{54}$$

Y can be found from the following equation:

$$\begin{gathered} r_{peg} \cdot L^t(Y) = R_3(Y), \\ r_{peg} \cdot P_{SC}^t(N_{SC_0} - Y) = R_0 \cdot \left(\frac{N_{SC_0} - Y}{N_{SC_0}}\right)^{\frac{1 - fee_0}{r_{peg}}}, \\ Y(R_0, N_{SC_0}) = N_{SC_0} - \left(\frac{r_{peg} \cdot P_{SC}^t \cdot (N_{SC_0})^d}{R_0}\right)^{\frac{1}{d-1}}, \qquad d = \frac{1 - fee_0}{r_{peg}}. \end{gathered} \tag{55}$$

The algorithm (4) defines the procedure of calculating new reserves $R(N)$ after selling N stablecoins. Then, the amount of returned basecoins to the user is $P_{sc}^{sell}(N) = R_0 - R(N)$.

Algorithm 4 Calculating new reserves when selling stablecoins

procedure CalculateNewReserves
    $R_0 \leftarrow$ initial amount of reserves
    $r_0 \leftarrow$ initial reserves ratio
    $N_{SC_0} \leftarrow$ initial amount of stablecoins
    $N \leftarrow$ number of stablecoins to be sold
    if $r_0 \geq r_{opt}$ **then**
        $newReserves \leftarrow R_1(N, R_0, N_{SC_0})$
    else
        if $r_0 < r_{peg}$ **then**
            $newReserves \leftarrow InitRatioBelowPeg(N, R_0, N_{SC_0})$
        else
            $newReserves \leftarrow InitRatioAbovePeg(N, R_0, N_{SC_0})$
    return *newReserves*

procedure InitRatioAbovePeg(N',R_0',$N_{SC_0}^{'}$)
    $X \leftarrow X(R_0', N_{SC_0}^{'})$ – number of stablecoins to sell until r_{opt} is reached
    if $N' \leq X$ **then**
        $res \leftarrow R_2(N', R_0', N_{SC_0}^{'})$
    else
        $R_0'' \leftarrow R_2(X, R_0', N_{SC_0}^{'})$
        $N_{SC_0}^{''} \leftarrow N_{SC_0}^{'} - X$
        $res \leftarrow R_0'' - R_0' + R_1(N' - X, R_0'', N_{SC_0}^{''})$
    return *res*

procedure InitRatioBelowPeg(N',R_0',$N_{SC_0}^{'}$)
    $Y \leftarrow Y(R_0', N_{SC_0}^{'})$ – number of stablecoins to sell until r_{peg} is reached
    if $N' \leq Y$ **then**
        $res \leftarrow R_3(N', R_0', N_{SC_0}^{'})$
    else
        $R_0'' \leftarrow R_3(Y, R_0', N_{SC_0}^{'})$
        $N_{SC_0}^{''} \leftarrow N_{SC_0}^{''} - Y$
        $res \leftarrow R_0'' - R_0' + InitRatioAbovePeg(N' - Y, R_0'', N_{SC_0}^{''})$
    return *res*

D.1 Debt-for-equity swap in the continuous setting

As was discussed in Section [5.4.1], a user is compensated with reservecoins for selling stablecoins in case $r < r_{peg}$. The amount of reservecoins for N stablecoins in the discrete setting was defined by (21) as:

$$S_{sc}^{sell}(N) = \sum_{i=0}^{N-1} S_{sc,i}^{sell} = \sum_{i=0}^{N-1} \frac{(1 - k(r_i)) \cdot P_{SC}^t \cdot N_{RC_i}}{R_i - k(r_i) \cdot P_{SC}^t \cdot N_{SC_i}} \cdot (1 - fee_0).$$

In this section we describe how to calculate this amount in the continous setting.

Let $R(t)$, $N_{SC}(t)$, and $N_{RC}(t)$ be continuous functions on $[0, N]$ defining the amount of reserves, amount of issued stablecoins, and amount of issued reservecoins correspondingly at time t (i.e., after selling t stablecoins), where $R(0) = R_0$, $N_{SC}(0) = N_{SC_0}$, and $N_{RC}(0) = N_{RC_0}$ are the initial amounts. Then, the amount of compensated reservecoins $S_{sc}^{sell}(N)$ for N stablecoins can be defined as a derivative of the function $N_{RC}(t)$:

$$\frac{dN_{RC}(t)}{dt} = S_{sc}^{sell}(t), \qquad S_{sc}^{sell}(t) = \frac{(1 - k(r(t))) \cdot P_{SC}^t \cdot N_{RC}(t)}{R(t) - k(r(t)) \cdot P_{SC}^t \cdot N_{SC}(t)}(1 - fee_0),$$

$$N_{SC}(t) = N_{SC_0} - t, \qquad k(r(t)) = \begin{cases} 1, & \text{if } r(t) \geq r_{peg}, \\ \frac{r(t)}{r_{peg}}, & \text{if } r(t) < r_{peg}. \end{cases}$$

Note that if $k(r(t)) = 1$ then $S_{sc}^{sell}(t) = 0$, so we need to solve the equation only for the case $k(r(t)) < 1$ (i.e., $r(t) < r_{peg}$):

$$\frac{dN_{RC}(t)}{dt} = \frac{(1 - \frac{R(t)}{P_{SC}^t \cdot N_{SC}(t) \cdot r_{peg}}) P_{SC}^t \cdot N_{RC}(t)}{R(t) - \frac{R(t)}{r_{peg}}}(1 - fee_0), \tag{56}$$

where $R(t)$ is defined by (52) for the interval $r(t) < r_{peg}$.

(56) is a first-order separable differential equation. The solution provides the unknown function $N_{RC}(t)$:

$$N_{RC}(t) = e^A \cdot N_{RC_0}, \tag{57}$$

where

$$A = \frac{1 - fee_0}{1 - \frac{1}{r_{peg}}} \left(\frac{1}{r_{peg}} \ln \frac{N_{SC_0} - t}{N_{SC}} - \frac{P_{SC}^t (N_{SC_0})^d ((N_{SC_0} - t)^{1-d} - (N_{SC_0})^{1-d})}{R_0(1 - d)} \right),$$

$$d = \frac{1 - fee_0}{r_{peg}}.$$

Since the reserves ratio grows when stablecoins are sold and the equation calculates reservecoins only for the case $r(t) < r_{peg}$ we need to apply it only for the amount $t = Y$, where Y is the number of stablecoins to be sold until r_{peg} is reached. Y is defined by (55).

The algorithm (5) defines the full procedure of calculating amount of compensated reservecoins $S_{sc}^{sell}(N)$.

Algorithm 5 Calculating amount of compensated reservecoins

procedure CalculateReservecoins
  $r_0 \leftarrow$ initial reserves ratio
  $N_{RC_0} \leftarrow$ initial amount of reservecoins
  $N \leftarrow$ number of stablecoins to be sold
  if $r_0 < r_{peg}$ **then**
    $Y \leftarrow$ number of stablecoins to be sold until r_{peg} is reached
    if $N < Y$ **then**
      reservecoins $\leftarrow N_{RC}(N) - N_{RC_0}$
    else
      reservecoins $\leftarrow N_{RC}(Y) - N_{RC_0}$
  else
    reservecoins $\leftarrow 0$
return *reservecoins*

E Isabelle/HOL Formalization

```
theory StableCoin
  imports Complex-Main
begin

type-synonym base-coin = real

type-synonym stable-coin = real

type-synonym reserve-coin = real

type-synonym exchange-rate = real

locale stablecoin =
  fixes r_min :: real
    and r_max :: real
    and fee :: real
    and N_SC-threshold :: stable-coin (‹N*_SC›)
    and p-min-rc :: base-coin (‹P^min_RC›)
  assumes r_min-lower-bound: r_min > 1 + fee
    and r_min-upper-bound: r_max ≥ r_min
    and fee-is-percentage: fee ∈ {0<..1}
    and N_SC-threshold-positivity: N*_SC > 0
    and p-min-rc-positivity: P^min_RC > 0
begin

abbreviation stable-coin-target-price :: exchange-rate ⇒ base-coin (‹P^t_SC[-]›)
where
  P^t_SC[X] ≡ X

definition stable-coin-actual-price :: exchange-rate ⇒ base-coin ⇒ stable-coin
⇒ base-coin (‹P_SC'(-,-,-')›) where
```

$P_{SC}(X,\ R,\ N_{SC}) = (\mathit{if}\ N_{SC} = 0\ \mathit{then}\ {P^t}_{SC}[X]\ \mathit{else}\ \mathit{min}\ {P^t}_{SC}[X]\ (R\ /\ N_{SC}))$

definition *liabilities* :: *exchange-rate* ⇒ *base-coin* ⇒ *stable-coin* ⇒ *base-coin* (‹L'(-,-,-')›) **where**
$L(X,\ R,\ N_{SC}) = N_{SC} * P_{SC}(X,\ R,\ N_{SC})$

definition *equity* :: *exchange-rate* ⇒ *base-coin* ⇒ *stable-coin* ⇒ *base-coin* (‹E'(-,-,-')›) **where**
$E(X,\ R,\ N_{SC}) = R - L(X,\ R,\ N_{SC})$

definition *reserve-ratio* :: *exchange-rate* ⇒ *base-coin* ⇒ *stable-coin* ⇒ *real* (‹r'(-,-,-')›) **where**
$r(X,\ R,\ N_{SC}) = R\ /\ L(X,\ R,\ N_{SC})$

definition *reserve-coin-target-price* :: *exchange-rate* ⇒ *base-coin* ⇒ *stable-coin* ⇒ *reserve-coin* ⇒ *base-coin* (‹${P^t}_{RC}$'(-,-,-,-')›) **where**
${P^t}_{RC}(X,\ R,\ N_{SC},\ N_{RC}) = E(X,\ R,\ N_{SC})\ /\ N_{RC}$

definition *reserve-coin-buying-price* :: *exchange-rate* ⇒ *base-coin* ⇒ *stable-coin* ⇒ *reserve-coin* ⇒ *base-coin* (‹${P^b}_{RC}$'(-,-,-,-')›) **where**
${P^b}_{RC}(X,\ R,\ N_{SC},\ N_{RC}) = (\mathit{if}\ N_{RC} = 0\ \mathit{then}\ {P^{min}}_{RC}\ \mathit{else}\ \mathit{max}\ {P^t}_{RC}(X,\ R,\ N_{SC},\ N_{RC})\ {P^{min}}_{RC})$

type-synonym *bank-state* = *base-coin* × *stable-coin* × *reserve-coin*

fun *is-valid-bank-state* :: *bank-state* ⇒ *bool* **where**
is-valid-bank-state $(R,\ N_{SC},\ N_{RC}) = (R \geq 0 \wedge N_{SC} \geq 0 \wedge N_{RC} \geq 0)$

datatype *action* = *BuySCs stable-coin* | *SellSCs stable-coin* | *BuyRCs reserve-coin* | *SellRCs reserve-coin*

type-synonym *transaction* = *action* × *exchange-rate*

fun *tx-rate* :: *transaction* ⇒ *exchange-rate* **where**
tx-rate $(\text{-},\ X) = X$

fun *is-valid-transaction* :: *transaction* ⇒ *bank-state* ⇒ *bool* **where**
is-valid-transaction $(\mathit{BuySCs}\ n,\ X)\ (R,\ N_{SC},\ N_{RC}) = (X > 0 \wedge n > 0 \wedge r(X,\ R,\ N_{SC}) \geq r_{min} \wedge r(X,\ R + n * (1 + \mathit{fee}) * P_{SC}(X,\ R,\ N_{SC}),\ N_{SC} + n) \geq r_{min})$
| *is-valid-transaction* $(\mathit{SellSCs}\ n,\ X)\ (R,\ N_{SC},\ N_{RC}) = (X > 0 \wedge n > 0 \wedge n * (1 - \mathit{fee}) * P_{SC}(X,\ R,\ N_{SC}) \leq R \wedge n \leq N_{SC})$
| *is-valid-transaction* $(\mathit{BuyRCs}\ n,\ X)\ (R,\ N_{SC},\ N_{RC}) = (X > 0 \wedge n > 0 \wedge (r(X,\ R,\ N_{SC}) \leq r_{max} \vee N_{SC} < {N^*}_{SC}) \wedge r(X,\ R + n * (1 + \mathit{fee}) * {P^b}_{RC}(X,\ R,\ N_{SC},\ N_{RC}),\ N_{SC}) \leq r_{max})$
| *is-valid-transaction* $(\mathit{SellRCs}\ n,\ X)\ (R,\ N_{SC},\ N_{RC}) = (X > 0 \wedge n > 0 \wedge n * (1 - \mathit{fee}) * {P^t}_{RC}(X,\ R,\ N_{SC},\ N_{RC}) \leq R \wedge n \leq N_{RC} \wedge r(X,\ R,\ N_{SC}) \geq r_{min} \wedge r(X,\ R - n * (1 - \mathit{fee}) * {P^t}_{RC}(X,\ R,\ N_{SC},\ N_{RC}),\ N_{SC}) \geq r_{min})$

inductive *transition* :: *bank-state* ⇒ *transaction* ⇒ *bank-state* ⇒ *bool* (‹- →⦃-⦄ -› [*51*, *0*, *51*] *50*) **where**
  buy-scs: $\mathcal{S} \rightarrow\{\!|tx|\!\}\ \mathcal{S}'$
    if *is-valid-transaction tx* $\mathcal{S}$
      and *is-valid-bank-state* $\mathcal{S}$
      and (*BuySCs n*, *X*) = *tx*
      and (R, N_{SC}, N_{RC}) = $\mathcal{S}$
      and $R' = R + n * (1 + fee) * P_{SC}(X, R, N_{SC})$
      and $N'_{SC} = N_{SC} + n$
      and $N'_{RC} = N_{RC}$
      and $\mathcal{S}' = (R', N'_{SC}, N'_{RC})$
| *sell-scs*: $\mathcal{S} \rightarrow\{\!|tx|\!\}\ \mathcal{S}'$
    if *is-valid-transaction tx* $\mathcal{S}$
      and *is-valid-bank-state* $\mathcal{S}$
      and (*SellSCs n*, *X*) = *tx*
      and (R, N_{SC}, N_{RC}) = $\mathcal{S}$
      and $R' = R - n * (1 - fee) * P_{SC}(X, R, N_{SC})$
      and $N'_{SC} = N_{SC} - n$
      and $N'_{RC} = N_{RC}$
      and $\mathcal{S}' = (R', N'_{SC}, N'_{RC})$
| *buy-rcs*: $\mathcal{S} \rightarrow\{\!|tx|\!\}\ \mathcal{S}'$
    if *is-valid-transaction tx* $\mathcal{S}$
      and *is-valid-bank-state* $\mathcal{S}$
      and (*BuyRCs n*, *X*) = *tx*
      and (R, N_{SC}, N_{RC}) = $\mathcal{S}$
      and $R' = R + n * (1 + fee) * {P^b}_{RC}(X, R, N_{SC}, N_{RC})$
      and $N'_{SC} = N_{SC}$
      and $N'_{RC} = N_{RC} + n$
      and $\mathcal{S}' = (R', N'_{SC}, N'_{RC})$
| *sell-rcs*: $\mathcal{S} \rightarrow\{\!|tx|\!\}\ \mathcal{S}'$
    if *is-valid-transaction tx* $\mathcal{S}$
      and *is-valid-bank-state* $\mathcal{S}$
      and (*SellRCs n*, *X*) = *tx*
      and (R, N_{SC}, N_{RC}) = $\mathcal{S}$
      and $R' = R - n * (1 - fee) * {P^t}_{RC}(X, R, N_{SC}, N_{RC})$
      and $N'_{SC} = N_{SC}$
      and $N'_{RC} = N_{RC} - n$
      and $\mathcal{S}' = (R', N'_{SC}, N'_{RC})$

inductive *sequence-transition* :: *bank-state* ⇒ *transaction list* ⇒ *bank-state* ⇒ *bool* (‹- →*⦃-⦄ -› [*51*, *0*, *51*] *50*) **where**
  tx-seq-base:
    $\mathcal{S} \rightarrow^*\{\!|[]|\!\}\ \mathcal{S}$
| *txs-seq-ind*:
    $\mathcal{S} \rightarrow^*\{\!|txs\ @\ [tx]|\!\}\ \mathcal{S}'$
      if $\mathcal{S} \rightarrow^*\{\!|txs|\!\}\ \mathcal{S}''$
      and $\mathcal{S}'' \rightarrow\{\!|tx|\!\}\ \mathcal{S}'$
      and $\mathcal{S}'' = (R'', N''_{SC}, N''_{RC})$
      and $N''_{SC} > 0$
      and $N''_{RC} > 0$

lemma *sequence-transition-cons*:
assumes $\mathcal{S} \rightarrow^* \{\!|tx \# txs|\!\} \mathcal{S}'$
shows $\exists \mathcal{S}'' R N_{SC} N_{RC}. \mathcal{S} = (R, N_{SC}, N_{RC}) \wedge \mathcal{S} \rightarrow \{\!|tx|\!\} \mathcal{S}'' \wedge \mathcal{S}'' \rightarrow^* \{\!|txs|\!\} \mathcal{S}' \wedge N_{SC} > 0 \wedge N_{RC} > 0$
using *assms*
proof (*induction txs arbitrary*: $\mathcal{S}'$ *rule*: *rev-induct*)
case *Nil*
then have $\mathcal{S} \rightarrow \{\!|tx|\!\} \mathcal{S}'$
by (*metis append.simps*(1) *snoc-eq-iff-butlast sequence-transition.cases*)
then show *?case*
by (*metis Nil.prems append-Nil sequence-transition.cases snoc-eq-iff-butlast tx-seq-base*)
next
case (*snoc* tx' *txs*)
from *snoc.prems* **have** $\mathcal{S} \rightarrow^* \{\!|(tx \# txs) @ [tx']|\!\} \mathcal{S}'$
by *auto*
then obtain $\mathcal{S}''$ R'' N''_{SC} N''_{RC} **where** $\mathcal{S}'' = (R'', N''_{SC}, N''_{RC})$ $\mathcal{S} \rightarrow^* \{\!|tx \# txs|\!\} \mathcal{S}''$ **and** $\mathcal{S}'' \rightarrow \{\!|tx'|\!\} \mathcal{S}'$ **and** $N''_{SC} > 0$ **and** $N''_{RC} > 0$
by (*auto intro*: *sequence-transition.cases*)
from $\langle \mathcal{S} \rightarrow^* \{\!|tx \# txs|\!\} \mathcal{S}'' \rangle$ **obtain** $\mathcal{S}'''$ R N_{SC} N_{RC} **where** $\mathcal{S} = (R, N_{SC}, N_{RC})$ **and** $\mathcal{S} \rightarrow \{\!|tx|\!\} \mathcal{S}'''$ **and** $\mathcal{S}''' \rightarrow^* \{\!|txs|\!\} \mathcal{S}''$ **and** $N_{SC} > 0$ **and** $N_{RC} > 0$
using *snoc.IH* **by** *blast*
from $\langle N''_{RC} > 0 \rangle$ **and** $\langle N''_{SC} > 0 \rangle$ **and** $\langle \mathcal{S}'' = (R'', N''_{SC}, N''_{RC}) \rangle$ **and** $\langle \mathcal{S}''' \rightarrow^* \{\!|txs|\!\} \mathcal{S}'' \rangle$ **and** $\langle \mathcal{S}'' \rightarrow \{\!|tx'|\!\} \mathcal{S}' \rangle$ **have** $\mathcal{S}''' \rightarrow^* \{\!|txs @ [tx']|\!\} \mathcal{S}'$
by (*simp add*: *txs-seq-ind*)
with $\langle N_{RC} > 0 \rangle$ **and** $\langle N_{SC} > 0 \rangle$ **and** $\langle \mathcal{S} = (R, N_{SC}, N_{RC}) \rangle$ **and** $\langle \mathcal{S} \rightarrow \{\!|tx|\!\} \mathcal{S}''' \rangle$ **show** *?case*
by *blast*
qed

lemma *sequence-transition-alt*:
assumes $\mathcal{S} \rightarrow^* \{\!|txs|\!\} \mathcal{S}'$
obtains Γ
where *length* Γ = *length txs* + 1
and $\mathcal{S} = \Gamma$! 0
and $\mathcal{S}' = \Gamma$! (*length txs*)
and $\forall i \in \{0..<\textit{length txs}\}. \Gamma ! i \rightarrow \{\!|txs ! i|\!\} \Gamma ! (i + 1)$
using *assms*
proof (*induction txs arbitrary*: $\mathcal{S}'$ *thesis rule*: *rev-induct*)
case *Nil*
let *?*Γ = [$\mathcal{S}$]
have $\mathcal{S}' = \mathcal{S}$
by (*blast intro*: *sequence-transition.cases Nil.prems*)
moreover have *length ?*Γ = *length* [] + 1
by *simp*
moreover have $\mathcal{S}$ = *?*Γ ! 0
by *simp*
moreover have $\mathcal{S}$ = *?*Γ ! *length* []

```
    by simp
  moreover have ∀ i ∈ {0..<length []}. ?Γ ! i →⦇[] ! i⦈ ?Γ ! (i + 1)
    by simp
  ultimately show ?case
    using Nil.prems by blast
next
  case (snoc tx txs)
  from ‹S →*⦇txs @ [tx]⦈ S'› obtain S'' where S →*⦇txs⦈ S'' and S''
→⦇tx⦈ S'
    by (blast intro: sequence-transition.cases)
  from ‹S →*⦇txs⦈ S''› and snoc.IH obtain Γ where length Γ = length txs
+ 1 and S = Γ ! 0 and S'' = Γ ! length txs and ∀ i ∈ {0..<length txs}. Γ
! i →⦇txs ! i⦈ Γ ! (i + 1)
    by blast
  let ?Γ = Γ @ [S']
  from ‹length Γ = length txs + 1› have length ?Γ = length (txs @ [tx]) + 1
    by simp
  moreover from ‹S = Γ ! 0› and ‹length ?Γ = length (txs @ [tx]) + 1›
have S = ?Γ ! 0
    by (simp add: nth-append)
  moreover from ‹length Γ = length txs + 1› have S' = ?Γ ! length (txs @
[tx])
    by (metis Suc-eq-plus1 length-append-singleton nth-append-length)
  moreover from ‹S'' = Γ ! length txs› and ‹S'' →⦇tx⦈ S'› and ‹∀ i ∈
{0..<length txs}. Γ ! i →⦇txs ! i⦈ Γ ! (i + 1)› ‹length Γ = length txs + 1›
have ∀ i ∈ {0..<length (txs @ [tx])}. ?Γ ! i →⦇(txs @ [tx]) ! i⦈ ?Γ ! (i + 1)
     by (metis (no-types, lifting) Suc-eq-plus1 Suc-leI add-diff-cancel-right'
add-less-cancel-left atLeastLessThan-iff length-append-singleton nat-less-le nth-append
nth-append-length plus-1-eq-Suc)
  ultimately show ?case
    using snoc.prems(1) by blast
qed

datatype market-offer = BuySCOffer | SellSCOffer

type-synonym secondary-market = market-offer × base-coin

datatype market-choice = Bank | Secondary

fun one-sc-transaction :: exchange-rate ⇒ market-offer ⇒ transaction where
  one-sc-transaction X SellSCOffer = (BuySCs 1, X) — a 'Buy 1 SC' transaction
| one-sc-transaction X BuySCOffer = (SellSCs 1, X) — a 'Sell 1 SC' transaction

fun rational-choice :: bank-state ⇒ exchange-rate ⇒ secondary-market ⇒
market-choice where
  rational-choice S X (action, price) = (
    let
      (R, N_SC, N_RC) = S;
      tx = one-sc-transaction X action
    in
```

if ¬ (*is-valid-bank-state* $\mathcal{S}$ ∧ *is-valid-transaction tx* $\mathcal{S}$)
then
Secondary — Bank does not allow the transaction, thus secondary market is chosen
else
case action of
SellSCOffer ⇒ (*if price* > (*1* + *fee*) ∗ $P_{SC}(X, R, N_{SC})$ *then Bank else Secondary*)
| *BuySCOffer* ⇒ (*if price* < (*1* − *fee*) ∗ $P_{SC}(X, R, N_{SC})$ *then Bank else Secondary*))

theorem *peg-maintenance-upper-bound*:
assumes *action* = *SellSCOffer*
and *tx* = *one-sc-transaction X action* — a 'Buy 1 SC' transaction
and $\mathcal{S} \rightarrow\!\!\{\!|tx|\!\}\ \mathcal{S}'$ — the transaction is allowed
and *secondary-market* = (*action*, *price*)
and *price* > (*1* + *fee*) ∗ ${P^t}_{SC}[X]$
shows ¬ *rational-choice* $\mathcal{S}$ *X secondary-market* = *Secondary*
proof −
obtain *R* **and** N_{SC} **and** N_{RC} **where** $(R, N_{SC}, N_{RC}) = \mathcal{S}$
by (*metis* (*no-types*) *is-valid-bank-state.cases*)
let *?P′* = (*1* + *fee*) ∗ $P_{SC}(X, R, N_{SC})$
have *?P′* ≤ (*1* + *fee*) ∗ ${P^t}_{SC}[X]$
unfolding *stable-coin-actual-price-def* **using** *fee-is-percentage* **by** *auto*
with ⟨*price* > (*1* + *fee*) ∗ ${P^t}_{SC}[X]$⟩ **have** *?P′* < *price*
by *linarith*
moreover from ⟨$\mathcal{S} \rightarrow\!\!\{\!|tx|\!\}\ \mathcal{S}'$⟩ **have** *is-valid-bank-state* $\mathcal{S}$ **and** *is-valid-transaction tx* $\mathcal{S}$
by (*blast intro*: *transition.cases*)+
ultimately have *rational-choice* $\mathcal{S}$ *X secondary-market* = *Bank*
using *assms*(*1*,*2*,*4*) **and** ⟨$(R, N_{SC}, N_{RC}) = \mathcal{S}$⟩ **by** (*simp split*: *prod.splits*) *blast*
then show *?thesis*
by *simp*
qed

lemma *peg-when-reserve-ratio-greater-than-one*:
assumes $r(X, R, N_{SC})$ > *1*
shows $P_{SC}(X, R, N_{SC}) = {P^t}_{SC}[X]$
proof −
from *assms* **have** *R* / (N_{SC} ∗ $P_{SC}(X, R, N_{SC})$) > *1*
using *liabilities-def* **and** *reserve-ratio-def* **by** *simp*
then have *R* / N_{SC} > $P_{SC}(X, R, N_{SC})$
unfolding *stable-coin-actual-price-def* **using** *less-divide-eq-1* **by** *fastforce*
then show *?thesis*
unfolding *stable-coin-actual-price-def* **by** *auto*
qed

corollary *peg-when-reserve-ratio-within-lower-bound*:
assumes $r(X, R, N_{SC}) \geq r_{min}$

shows $P_{SC}(X,\ R,\ N_{SC}) = P^t{}_{SC}[X]$
using *assms* **and** r_{min}*-lower-bound* **and** *fee-is-percentage* **and** *peg-when-reserve-ratio-greater-than-one*
by *auto*

theorem *peg-maintenance-lower-bound*:
assumes *action* = *BuySCOffer*
and *tx* = *one-sc-transaction X action* — a 'Sell 1 SC' transaction
and $\mathcal{S} \rightarrow\!\!\{\!|tx|\!\}\ \mathcal{S}'$ — the transaction is allowed
and *secondary-market* = (*action*, *price*)
and *price* < (*1* − *fee*) ∗ $P^t{}_{SC}[X]$
and $(R,\ N_{SC},\ N_{RC}) = \mathcal{S}$
and $r(X,\ R,\ N_{SC}) > 1$
shows ¬ *rational-choice* $\mathcal{S}$ *X secondary-market* = *Secondary*
proof −
let $?P' = (1 - fee) * P_{SC}(X,\ R,\ N_{SC})$
from ‹$r(X,\ R,\ N_{SC}) > 1$› **have** $?P' = (1 - fee) * P^t{}_{SC}[X]$
using *peg-when-reserve-ratio-greater-than-one* **by** *simp*
with ‹*price* < (*1* − *fee*) ∗ $P^t{}_{SC}[X]$› **have** $?P' >$ *price*
by *linarith*
moreover from ‹$\mathcal{S} \rightarrow\!\!\{\!|tx|\!\}\ \mathcal{S}'$› **have** *is-valid-bank-state* $\mathcal{S}$ **and** *is-valid-transaction tx* $\mathcal{S}$
by (*blast intro*: *transition.cases*)+
ultimately have *rational-choice* $\mathcal{S}$ *X secondary-market* = *Bank*
using *assms*(*1*,*2*,*4*) **and** ‹$(R,\ N_{SC},\ N_{RC}) = \mathcal{S}$› **by** (*simp split*: *prod.splits*) *blast*
then show *?thesis*
by *simp*
qed

theorem *peg-robustness-during-market-crashes*:
assumes *X* > *0*
and $X' > 0$
and $N_{SC} > 0$
and $r = r(X,\ R,\ N_{SC})$
and $r > 1$ — The peg is maintained before the crash
and $Y = 1\ /\ X$ — BC price before the crash
and $Y' = 1\ /\ X'$ — BC price after the crash
and $(Y - Y')\ /\ Y \leq (r - 1)\ /\ r$ — BC price crash of at most $(r - 1)\ /\ r$
shows $P_{SC}(X',\ R,\ N_{SC}) = P^t{}_{SC}[X']$ — The peg is maintained after the crash
proof −
have $P^t{}_{SC}[X'] \leq R\ /\ N_{SC}$
proof −
from ‹$r = r(X,\ R,\ N_{SC})$› **and** ‹$r > 1$› **have** $P_{SC}(X,\ R,\ N_{SC}) = P^t{}_{SC}[X]$
using *peg-when-reserve-ratio-greater-than-one* **by** *simp*
with ‹$X > 0$› **have** $P^t{}_{SC}[X'] \leq R\ /\ N_{SC} \longleftrightarrow P^t{}_{SC}[X'] \leq (R\ /\ N_{SC}) * (P^t{}_{SC}[X]\ /\ P^t{}_{SC}[X])$
by *auto*

```
  also from ‹P_SC(X, R, N_SC) = P^t_SC[X]› and ‹r = r(X, R, N_SC)› have
... ⟷ P^t_SC[X'] ≤ r * P^t_SC[X]
    unfolding liabilities-def and reserve-ratio-def by simp
  also from ‹X > 0› and ‹X' > 0› and ‹r > 1› have ... ⟷ 1 / P^t_SC[X']
≥ 1 / (r * P^t_SC[X])
    by (smt divide-cancel-left frac-le mult-less-cancel-right1)
  also have ... ⟷ 1 / P^t_SC[X'] ≥ (1 / r) * (1 / P^t_SC[X])
    by simp
  also from ‹X > 0› and ‹X' > 0› and ‹Y = 1 / X› and ‹Y' = 1 / X'›
have ... ⟷ Y' ≥ Y / r
    by (simp add: mult.commute)
  also have ... ⟷ Y' ≥ Y * ((r + 1 - r) / r)
    by simp
  also have ... ⟷ Y' ≥ Y * ((r / r) + ((1 - r) / r))
    by (metis add-diff-cancel-left' add-divide-distrib diff-add-cancel)
  also from ‹r > 1› have ... ⟷ Y' ≥ Y * (1 + ((1 - r) / r))
    by force
  also have ... ⟷ Y' ≥ Y + Y * ((1 - r) / r)
    by (simp add: distrib-left)
  also have ... ⟷ Y' - Y ≥ Y * ((1 - r) / r)
    by linarith
  also from ‹X > 0› have ... ⟷ (Y' - Y) * X ≥ (Y * ((1 - r) / r))
* X
    by (metis mult.commute mult-le-cancel-left-pos)
  also from ‹X > 0› and ‹Y = 1 / X› have ... ⟷ (Y' - Y) / Y ≥ (1
- r) / r
    by auto
  also have ... ⟷ (Y - Y') / Y ≤ (r - 1) / r
    by (smt divide-minus-left)
  finally show ?thesis
    using ‹(Y - Y') / Y ≤ (r - 1) / r› by blast
 qed
 then show ?thesis
  unfolding stable-coin-actual-price-def by auto
qed

theorem no-insolvency:
 assumes X ≥ 0 and R ≥ 0 and N_SC ≥ 0
 shows E(X, R, N_SC) ≥ 0
proof −
 consider (a) N_SC = 0 | (b) N_SC ≠ 0 and R / N_SC ≤ P^t_SC[X] | (c)
N_SC ≠ 0 and R / N_SC > P^t_SC[X]
  by linarith
 then show ?thesis
 proof cases
  case a
  from ‹N_SC = 0› have E(X, R, N_SC) = R
   unfolding liabilities-def and equity-def by simp
  with ‹R ≥ 0› show ?thesis
   by auto
```

```
  next
    case b
    from ‹N_SC ≠ 0› and ‹R / N_SC ≤ P^t_SC[X]› have E(X, R, N_SC) = R
− N_SC * (R / N_SC)
      unfolding liabilities-def and equity-def and stable-coin-actual-price-def
by (simp add: min.absorb2)
    also from ‹N_SC ≠ 0› have ... = 0
      by simp
    finally show ?thesis
      by simp
  next
    case c
    from ‹N_SC ≠ 0› have 0 = R − N_SC * (R / N_SC)
      by simp
    also from ‹N_SC ≥ 0› and ‹N_SC ≠ 0› and ‹R / N_SC > P^t_SC[X]› have
... < R − N_SC * P^t_SC[X]
      by (smt mult-le-cancel-left-pos)
    also from ‹N_SC ≠ 0› and ‹R / N_SC > P^t_SC[X]› have ... = E(X, R,
N_SC)
      unfolding liabilities-def and equity-def and stable-coin-actual-price-def
by simp
    finally show ?thesis
      by simp
  qed
qed

theorem no-bank-runs-for-stable-coins:
  assumes S →⦇tx_1⦈ S_1
    and S_1 →⦇tx_2⦈ S_2
    and tx_1 = (SellSCs n_1, X_1)
    and tx_2 = (SellSCs n_2, X_2)
    and S = (R, N_SC, N_RC)
    and S_1 = (R', N'_SC, N'_RC)
    and P1_SC = P_SC(X_1, R, N_SC)
    and P2_SC = P_SC(X_2, R', N'_SC)
    and X_1 = X_2
  shows P2_SC ≥ P1_SC
proof −
  from assms(1−6) have N_SC > 0 and N'_SC > 0 and n_1 > 0 and N'_SC
= N_SC − n_1
    using transition.simps and stablecoin-axioms by auto
  from assms(1,5) and ‹N_SC > 0› have R / N_SC ≥ 0
    by (metis divide-nonneg-pos is-valid-bank-state.simps transition.cases)
  have simpl: R / N_SC ≤ (R − n_1 * (1 − fee) * (R / N_SC)) / (N_SC − n_1)
  proof −
    have R / N_SC ≤ (R / N_SC) * ((N_SC − n_1 + n_1 * fee) / (N_SC − n_1))
    proof −
      from ‹N'_SC > 0› and ‹n_1 > 0› ‹N'_SC = N_SC − n_1› have (N_SC − n_1
+ n_1 * fee) / (N_SC − n_1) ≥ 1
        using fee-is-percentage by auto
```

```
      with ‹R / N_SC ≥ 0› show ?thesis
        using mult-le-cancel-left1 by fastforce
    qed
    also have ... = (R − n_1 * (1 − fee) * (R / N_SC)) / (N_SC − n_1)
    proof −
      have (R / N_SC) * ((N_SC − n_1 + n_1 * fee) / (N_SC − n_1)) = (R /
N_SC) * ((N_SC − n_1 * (1 − fee)) / (N_SC − n_1))
        by (simp add: diff-add-eq diff-diff-eq2 right-diff-distrib′)
      also have ... = (R − n_1 * (1 − fee) * (R / N_SC)) / (N_SC − n_1)
      proof −
        {
          fix x y z w :: real
          assume z ≠ 0
          then have (x − y * (x / z)) / w = (x / z) * ((z − y) / w)
            by (simp add: assms diff-divide-eq-iff right-diff-distrib′)
        }
        with ‹N_SC > 0› show ?thesis
          by auto
      qed
      finally show ?thesis
        by blast
    qed
    finally show ?thesis .
  qed
  show ?thesis
  proof (cases P^t_SC[X_1] ≤ R / N_SC)
    case True
    from ‹P^t_SC[X_1] ≤ R / N_SC› have P_SC(X_1, R, N_SC) = P^t_SC[X_1]
      using stable-coin-actual-price-def by auto
    with assms(1,3,5,6) have R′ = R − n_1 * (1 − fee) * P^t_SC[X_1]
      using transition.cases by fastforce
    with ‹N′_SC > 0› and ‹N′_SC = N_SC − n_1› and assms(8) have P2_SC
= min P^t_SC[X_2] ((R − n_1 * (1 − fee) * P^t_SC[X_1]) / (N_SC − n_1))
      unfolding stable-coin-actual-price-def by simp
    moreover have R / N_SC ≤ (R − n_1 * (1 − fee) * P^t_SC[X_1]) / (N_SC
− n_1)
    proof −
      from simpl have R / N_SC ≤ (R − n_1 * (1 − fee) * (R / N_SC)) /
(N_SC − n_1) .
      also have ... ≤ (R − n_1 * (1 − fee) * P^t_SC[X_1]) / (N_SC − n_1)
      proof −
        from ‹n_1 > 0› have n_1 * (1 − fee) ≥ 0
          using fee-is-percentage by auto
        moreover have N_SC − n_1 > 0
          using ‹N′_SC > 0› and ‹N′_SC = N_SC − n_1› by blast
        ultimately show ?thesis
        using ‹P^t_SC[X_1] ≤ R / N_SC› and divide-le-cancel and mult-left-mono
by fastforce
      qed
      finally show ?thesis .
```

```
    qed
    ultimately have P2_SC = P^t_SC[X_2]
      using ⟨P^t_SC[X_1] ≤ R / N_SC⟩ and assms(9) by linarith
     with assms(7,9) and ⟨P_SC(X_1, R, N_SC) = P^t_SC[X_1]⟩ have P2_SC =
P1_SC
      by auto
    then show ?thesis
      by simp
  next
    case False
    from ⟨¬ P^t_SC[X_1] ≤ R / N_SC⟩ and ⟨N_SC > 0⟩ have P_SC(X_1, R, N_SC)
= R / N_SC
      unfolding stable-coin-actual-price-def by auto
    with assms(1,3,5,6) have R' = R − n_1 ∗ (1 − fee) ∗ (R / N_SC)
      using transition.cases by fastforce
    with ⟨N'_SC > 0⟩ and ⟨N'_SC = N_SC − n_1⟩ and assms(8) have ∗: P2_SC
= min P^t_SC[X_2] ((R − n_1 ∗ (1 − fee) ∗ (R / N_SC)) / (N_SC − n_1))
      unfolding stable-coin-actual-price-def by simp
    show ?thesis
    proof (cases P^t_SC[X_2] ≤ ((R − n_1 ∗ (1 − fee) ∗ (R / N_SC)) / (N_SC −
n_1)))
      case True
      from ⟨P^t_SC[X_2] ≤ ((R − n_1 ∗ (1 − fee) ∗ (R / N_SC)) / (N_SC − n_1))⟩
and ∗ have P2_SC = P^t_SC[X_2]
        by linarith
       also from ⟨¬ P^t_SC[X_1] ≤ R / N_SC⟩ and ⟨P_SC(X_1,R,N_SC) = R /
N_SC⟩ and assms(7,9) have ... > P1_SC
        by linarith
      finally show ?thesis
        by simp
    next
      case False
      from assms(7) and ⟨P_SC(X_1, R, N_SC) = R / N_SC⟩ have P1_SC = R
/ N_SC
        by simp
      also from simpl have ... ≤ (R − n_1 ∗ (1 − fee) ∗ (R / N_SC)) / (N_SC
− n_1) .
      also from ⟨¬ P^t_SC[X_2] ≤ ((R − n_1 ∗ (1 − fee) ∗ (R / N_SC)) / (N_SC
− n_1))⟩ and ∗ have ... = P2_SC
        by auto
      finally show ?thesis .
    qed
  qed
qed

lemma buy-scs-actual-price-invariancy:
  assumes S →⟦tx⟧ S'
    and S = (R, N_SC, N_RC)
    and S' = (R', N'_SC, N'_RC)
    and (BuySCs n, X) = tx
```

shows $P_{SC}(X,\ R,\ N_{SC}) = P_{SC}(X,\ R',\ N'_{SC})$ (**is** $?P_{SC} = ?P'_{SC}$)
proof −
from *assms* **have** *is-valid-transaction tx* $\mathcal{S}$ **and** $R' = R + n * (1 + fee) * ?P_{SC}$ **and** $N'_{SC} = N_{SC} + n$
by (*blast intro*: *transition.cases*)+
with *assms*(*2*,*4*) **have** $r(X,\ R,\ N_{SC}) \geq r_{min}$ **and** $r(X,\ R',\ N'_{SC}) \geq r_{min}$
using *is-valid-transaction.simps*(*1*) **by** (*blast*, *blast*)
then show *?thesis*
using *peg-when-reserve-ratio-within-lower-bound* **by** *presburger*
qed

lemma *unpegged-stable-coin-actual-price-correction*:
assumes $X > 0$
and $N_{SC} > 0$
obtains α
where $P_{SC}(X,\ R,\ N_{SC}) = (R\ /\ N_{SC}) * \alpha$
and $\alpha \in \{0<..1\}$
proof −
let $?P_{SC} = P_{SC}(X,\ R,\ N_{SC})$
assume $*$: $\bigwedge\gamma.\ [\![?P_{SC} = R\ /\ N_{SC} * \gamma;\ \gamma \in \{0<..1\}]\!] \Longrightarrow thesis$
from *assms*(*1*) **have** $\forall x.\ 0 < x \vee X \geq x$
by *linarith*
moreover have $\forall x\ y\ z.\ (x::real) * z\ /\ y\ /\ (z\ /\ y) = x \vee z\ /\ y = 0$
by *fastforce*
moreover have $\forall x\ y\ z.\ ((x::real)\ /\ z \in \{y<..1\} \vee x > z) \vee y \geq x\ /\ z \vee 0 \geq z$
by *auto*
moreover from *assms*(*2*) **have** $N_{SC} \neq 0$
by *auto*
ultimately have $(N_{SC} \neq 0 \wedge (X < R\ /\ N_{SC} \longrightarrow ?P_{SC} \neq X)) \vee (N_{SC} \neq 0 \wedge (X \leq R\ /\ N_{SC} \longrightarrow (\exists\alpha.\ ?P_{SC} = R\ /\ N_{SC} * \alpha \wedge \alpha \in \{0<..1\})))$
using *assms*(*1*) **by** (*metis* (*no-types*, *hide-lams*) *not-le divide-eq-0-iff divide-strict-right-mono mult.commute times-divide-eq-right*)
then obtain $\alpha :: real$ **where** $N_{SC} \neq 0 \wedge \alpha \in \{0<..1\} \wedge ((X < R\ /\ N_{SC} \longrightarrow ?P_{SC} \neq X) \vee (X \leq R\ /\ N_{SC} \longrightarrow ?P_{SC} = R\ /\ N_{SC} * \alpha))$
using *fee-is-percentage* **by** *blast*
with $*$ **show** *?thesis*
using *greaterThanAtMost-iff* **and** *mult.commute* **and** *mult.left-neutral*
unfolding *stable-coin-actual-price-def* **and** *min-def*
by (*metis* (*no-types*, *hide-lams*) *divide-le-eq-1 dual-order.order-iff-strict eq-divide-eq-1 times-divide-eq-left*)
qed

theorem *monotonically-increasing-equity-per-reserve-coin*:
assumes $\mathcal{S} \rightarrow\!\!\{\!|tx|\!\!\}\ \mathcal{S}'$
and $X = tx\text{-}rate\ tx$
and $\mathcal{S} = (R,\ N_{SC},\ N_{RC})$
and $\mathcal{S}' = (R',\ N'_{SC},\ N'_{RC})$
and $X = X'$
and $N'_{RC} > 0$

```
    and N_RC > 0
    and N'_SC > 0
  shows E(X', R', N'_SC) / N'_RC ≥ E(X, R, N_SC) / N_RC
proof −
  from assms(1) show ?thesis
  proof cases
    case (buy-scs n X R N_SC N_RC R' N'_SC N'_RC)
    let ?P_SC = P_SC(X, R, N_SC)
    let ?P'_SC = P_SC(X', R', N'_SC)
    from assms(2,3,5,7) and buy-scs(3,4) have X = X' and N_RC > 0
      by auto
    from assms(1) and buy-scs(3,4,8) and ‹X = X'› have ?P'_SC = ?P_SC
      using buy-scs-actual-price-invariancy by force
    from buy-scs(5,6,7) have E(X, R', N'_SC) / N'_RC = E(X, R + n * (1
+ fee) * ?P_SC, N_SC + n) / N_RC
      by simp
    also from buy-scs(5,6) and ‹X = X'› have ... = (R + n * (1 + fee) *
?P_SC − (N_SC + n) * ?P'_SC) / N_RC
      unfolding equity-def and liabilities-def by presburger
    also have ... = (R + n * ?P_SC + n * fee * ?P_SC − N_SC * ?P'_SC − n
* ?P'_SC) / N_RC
      by (simp add: distrib-left distrib-right)
    also from ‹?P'_SC = ?P_SC› have ... = (R − N_SC * ?P_SC + n * fee *
?P_SC) / N_RC
      by auto
   also from ‹N_RC > 0› and buy-scs(1,3,4) have ... > (R − N_SC * ?P_SC)
/ N_RC
     using fee-is-percentage and is-valid-transaction.simps(1) and peg-when-reserve-ratio-within-lower-bound
by (smt divide-strict-right-mono greaterThanAtMost-iff mult-pos-pos)
    finally show ?thesis
     using assms(3−5) and buy-scs(4,8) and ‹X = X'› unfolding equity-def
and liabilities-def by auto
  next
    case (sell-scs n X R N_SC N_RC R' N'_SC N'_RC)
    let ?P_SC = P_SC(X, R, N_SC)
    let ?P'_SC = P_SC(X', R', N'_SC)
    from sell-scs(1,3,4) have N_SC > 0
      by auto
    from assms(2−5,7−8) and sell-scs(3,4,8) have X = X' and N_RC >
0 and N'_SC > 0
      by auto
    from assms(3) and sell-scs(1,3) have X > 0
      by auto
    from assms(1) and sell-scs(3) have n > 0
      by (auto intro: transition.cases)
    from sell-scs(2,4) have R ≥ 0
      using is-valid-bank-state.simps by blast
    from ‹N'_SC > 0› and sell-scs(6) have N_SC − n > 0
      by blast
    have ∗: n * (1 − fee) * ?P_SC + (N_SC − n) * ?P'_SC ≤ N_SC * ?P_SC
```

proof (*cases* $?P_{SC} = R \;/\; N_{SC}$)
case *True*
from ‹$N'_{SC} > 0$› **and** ‹$X > 0$› **and** ‹$X = X'$› **obtain** γ **where** $?P'_{SC} = (R' \;/\; N'_{SC}) * \gamma$ **and** $\gamma \in \{0<..1\}$
using *unpegged-stable-coin-actual-price-correction* **by** *blast*
from ‹$N_{SC} - n > 0$› **and** ‹$n > 0$› **have** $N_{SC} > n * (1 - fee)$
by (*smt fee-is-percentage greaterThanAtMost-iff mult-left-le*)
with ‹$N_{SC} > 0$› **and** ‹$\gamma \in \{0<..1\}$› **have** $((n * (1 - fee)) \;/\; N_{SC}) * (1 - \gamma) \leq 1 - \gamma$
by (*smt greaterThanAtMost-iff le-divide-eq-1-pos mult-less-cancel-right1*)
with ‹$R \geq 0$› **have** $n * (1 - fee) * (R \;/\; N_{SC}) * (1 - \gamma) \leq (1 - \gamma) * R$
by (*simp add*: *divide-inverse mult.commute mult.left-commute mult-left-mono*)
then have $n * (1 - fee) * (R \;/\; N_{SC}) * (1 - \gamma) + \gamma * R \leq R$
by (*simp add*: *mult.commute right-diff-distrib'*)
then have $n * (1 - fee) * (R \;/\; N_{SC}) + \gamma * R - n * (1 - fee) * (R \;/\; N_{SC}) * \gamma \leq R$
by (*smt distrib-left mult-cancel-left1*)
then have $n * (1 - fee) * (R \;/\; N_{SC}) + (R - n * (1 - fee) * (R \;/\; N_{SC})) * \gamma \leq R$
by (*smt mult.commute ring-class.ring-distribs*(*2*))
with ‹$N_{SC} - n > 0$› **and** ‹$N_{SC} > 0$› **have** $n * (1 - fee) * (R \;/\; N_{SC}) + (N_{SC} - n) * ((R - n * (1 - fee) * (R \;/\; N_{SC})) \;/\; (N_{SC} - n)) * \gamma \leq N_{SC} * (R \;/\; N_{SC})$
by *auto*
with *True* **and** ‹$?P'_{SC} = (R' \;/\; N'_{SC}) * \gamma$› **and** *sell-scs*(*5*,*6*) **show** *?thesis*
by *fastforce*
next
case *False*
with *sell-scs*(*1*,*3*,*4*) **have** $?P_{SC} < R \;/\; N_{SC}$
unfolding *stable-coin-actual-price-def* **by** *fastforce*
then have $?P_{SC} = P^t{}_{SC}[X]$
unfolding *stable-coin-actual-price-def* **by** *fastforce*
have $R \;/\; N_{SC} < R' \;/\; N'_{SC}$
proof −
from ‹$N_{SC} > 0$› **and** ‹$X > 0$› **obtain** σ **where** $?P_{SC} = (R \;/\; N_{SC}) * \sigma$ **and** $\sigma \in \{0<..1\}$
using *unpegged-stable-coin-actual-price-correction* **by** *blast*
with *False* **have** $\sigma < 1$
by *force*
with ‹$n > 0$› **have** $n > n * (1 - fee) * \sigma$
by (*smt fee-is-percentage greaterThanAtMost-iff mult-le-cancel-left1 mult-less-cancel-left mult-not-zero*)
then have $N_{SC} - n < N_{SC} - n * (1 - fee) * \sigma$
by *simp*
with ‹$N_{SC} - n > 0$› **have** $1 < (N_{SC} - n * (1 - fee) * \sigma) \;/\; (N_{SC} - n)$
using *less-divide-eq-1-pos* **by** *blast*
moreover from *sell-scs*(*2*,*4*) **have** $R \;/\; N_{SC} \geq 0$

by *fastforce*
ultimately have $R \,/\, N_{SC} < (R \,/\, N_{SC}) * ((N_{SC} - n * (1 - fee) * \sigma) \,/\, (N_{SC} - n))$
using *False* **and** ‹*?P*$_{SC}$ = $(R \,/\, N_{SC}) * \sigma$› **by** (*metis* (*no-types*) *less-eq-real-def linorder-not-less mult-eq-0-iff mult-le-cancel-left2*)
then have $R \,/\, N_{SC} < ((R \,/\, N_{SC}) * (N_{SC} - n * (1 - fee) * \sigma)) \,/\, (N_{SC} - n)$
by *simp*
then have $R \,/\, N_{SC} < ((R \,/\, N_{SC}) * N_{SC} - n * (1 - fee) * (R \,/\, N_{SC}) * \sigma) \,/\, (N_{SC} - n)$
by (*simp add*: *mult.assoc mult.left-commute right-diff-distrib'*)
with *False* **and** ‹*?P*$_{SC}$ = $(R \,/\, N_{SC}) * \sigma$› **have** $R \,/\, N_{SC} < (R - n * (1 - fee) * (R \,/\, N_{SC}) * \sigma) \,/\, (N_{SC} - n)$
by *auto*
moreover from ‹*?P*$_{SC}$ = $(R \,/\, N_{SC}) * \sigma$› **and** *sell-scs*(*5*,*6*) **have** $R' \,/\, N'_{SC} = (R - n * (1 - fee) * (R \,/\, N_{SC}) * \sigma) \,/\, (N_{SC} - n)$
by *simp*
ultimately show *?thesis*
by *simp*
qed
with ‹*?P*$_{SC}$ = $P^t{}_{SC}[X]$› **and** ‹*?P*$_{SC}$ < $R \,/\, N_{SC}$› **have** $P^t{}_{SC}[X] < R' \,/\, N'_{SC}$
by *linarith*
with ‹$X = X'$› **have** *?P*$'_{SC}$ = $P^t{}_{SC}[X]$
unfolding *stable-coin-actual-price-def* **by** *auto*
then have $n * (1 - fee) * P^t{}_{SC}[X] + (N_{SC} - n) * P^t{}_{SC}[X] = (n * (1 - fee) + N_{SC} - n) * P^t{}_{SC}[X]$
by (*metis add-diff-eq distrib-left mult.commute*)
also have ... = $(N_{SC} - n * fee) * P^t{}_{SC}[X]$
by (*simp add*: *right-diff-distrib'*)
also from ‹$X > 0$› **and** ‹$n > 0$› **have** ... ≤ $N_{SC} * P^t{}_{SC}[X]$
using *fee-is-percentage* **by** *auto*
finally have $n * (1 - fee) * P^t{}_{SC}[X] + (N_{SC} - n) * P^t{}_{SC}[X] \leq N_{SC} * P^t{}_{SC}[X]$ **.**
with ‹*?P*$_{SC}$ = $P^t{}_{SC}[X]$› **and** ‹*?P*$'_{SC}$ = $P^t{}_{SC}[X]$› **show** *?thesis*
by *presburger*
qed
with ‹$N_{RC} > 0$› **have** $(R - n * (1 - fee) * ?P_{SC} - (N_{SC} - n) * ?P'_{SC}) \,/\, N_{RC} \geq (R - N_{SC} * ?P_{SC}) \,/\, N_{RC}$
by (*fastforce simp add*: *divide-le-cancel*)
with *assms*(*3*−*5*) **and** *sell-scs*(*4*−*8*) ‹$X = X'$› **show** *?thesis*
unfolding *equity-def* **and** *liabilities-def* **by** *fast*
next
case (*buy-rcs n X R* N_{SC} N_{RC} R' N'_{SC} N'_{RC})
let *?P*$_{SC}$ = $P_{SC}(X, R, N_{SC})$
let *?P*$'_{SC}$ = $P_{SC}(X', R', N'_{SC})$
let *?Pb*$_{RC}$ = $P^b{}_{RC}(X, R, N_{SC}, N_{RC})$
from *assms*(*4*,*8*) **and** *local.buy-rcs*(*6*,*8*) **have** $N_{SC} > 0$
by *blast*

from *assms*(2−5,7,8) **and** *buy-rcs*(3,4,8) **have** $X = X'$ **and** $N_{RC} > 0$ **and** $N'_{SC} > 0$
by *auto*
from *assms*(3) **and** *buy-rcs*(1,3) **have** $X > 0$
using *is-valid-transaction.simps*(3) **by** *blast*
from *assms*(1) **and** *buy-rcs*(3) **have** $n > 0$
using *is-valid-transaction.simps*(3) **by** (*blast intro*: *transition.cases*)
have $R < R'$
proof −
have $?Pb_{RC} \geq P^{min}{}_{RC}$
unfolding *reserve-coin-buying-price-def* **by** *simp*
then have $?Pb_{RC} > 0$
using *p-min-rc-positivity* **by** *linarith*
with ‹$n > 0$› **have** $n * (1 + fee) * ?Pb_{RC} > 0$
using *fee-is-percentage* **by** *fastforce*
with *buy-rcs*(5) **show** *?thesis*
by *simp*
qed
with *assms*(2,4,5,8) **and** *buy-rcs*(3,6,8) **have** $?P_{SC} \leq ?P'_{SC}$
unfolding *stable-coin-actual-price-def* **by** (*fastforce dest*: *divide-strict-right-mono*)
show *?thesis*
proof (*cases* $?P_{SC} = R \ / \ N_{SC}$)
case *True*
from ‹$N'_{SC} > 0$› **and** ‹$X > 0$› **and** ‹$X = X'$› **and** *buy-rcs*(6) **obtain** γ **where** $?P'_{SC} = (R' \ / \ N_{SC}) * \gamma$ **and** $\gamma \in \{0<..1\}$
using *unpegged-stable-coin-actual-price-correction* **by** *blast*
have $E(X', R', N'_{SC}) \ / \ N'_{RC} = (R' - N'_{SC} * ?P'_{SC}) \ / \ N'_{RC}$
unfolding *equity-def* **and** *liabilities-def* **by** *simp*
also from ‹$?P'_{SC} = (R' \ / \ N_{SC}) * \gamma$› **and** *buy-rcs*(5−7) **have** $\ldots = (R + n * (1 + fee) * ?Pb_{RC} - N_{SC} * ((R + n * (1 + fee) * ?Pb_{RC}) \ / \ N_{SC}) * \gamma) \ / \ (N_{RC} + n)$
by *auto*
also from ‹$N_{SC} > 0$› **have** $\ldots = (R + n * (1 + fee) * ?Pb_{RC} - 1 * ((R + n * (1 + fee) * ?Pb_{RC}) * \gamma)) \ / \ (N_{RC} + n)$
by *auto*
also have $\ldots = (R + n * (1 + fee) * ?Pb_{RC} - R * \gamma - n * (1 + fee) * ?Pb_{RC} * \gamma) \ / \ (N_{RC} + n)$
by (*smt divide-cancel-right left-diff-distrib*)
also have $\ldots = (R * (1 - \gamma) + n * (1 + fee) * ?Pb_{RC} - n * (1 + fee) * ?Pb_{RC} * \gamma) \ / \ (N_{RC} + n)$
by (*simp add*: *right-diff-distrib'*)
also have $\ldots = (R * (1 - \gamma) + (n * (1 + fee) * ?Pb_{RC}) * (1 - \gamma)) \ / \ (N_{RC} + n)$
by (*simp add*: *left-diff-distrib' mult.commute*)
also have $\ldots = ((R + n * (1 + fee) * ?Pb_{RC}) * (1 - \gamma)) \ / \ (N_{RC} + n)$
by (*simp add*: *ring-class.ring-distribs*(2))
also have $\ldots \geq 0$
proof −
from ‹$\gamma \in \{0<..1\}$› **have** $1 - \gamma \geq 0$

by *auto*
moreover from ‹$R < R'$› **and** *buy-rcs*(*2*,*4*,*5*) **have** $R + n * (1 + fee) * ?Pb_{RC} > 0$
by *auto*
moreover from ‹$N_{RC} > 0$› **and** ‹$n > 0$› **have** $N_{RC} + n > 0$
by *auto*
ultimately show *?thesis*
by *auto*
qed
also from ‹$N_{SC} > 0$› **have** $0 = (R - N_{SC} * (R \mathbin{/} N_{SC})) \mathbin{/} N_{RC}$
by *auto*
also from *True* **have** $\ldots = E(X, R, N_{SC}) \mathbin{/} N_{RC}$
unfolding *equity-def* **and** *liabilities-def* **by** *presburger*
finally show *?thesis*
using ‹$X = X'$› **and** *assms*(*3*−*5*) **and** *buy-rcs*(*4*,*8*) **by** *blast*
next
case *False*
with ‹$N_{SC} > 0$› **have** $?P_{SC} < R \mathbin{/} N_{SC}$
unfolding *stable-coin-actual-price-def* **by** *auto*
then have $?P_{SC} = P^t{}_{SC}[X]$
unfolding *stable-coin-actual-price-def* **by** *fastforce*
with ‹$?P_{SC} \leq ?P'_{SC}$› **and** ‹$X = X'$› **have** $?P'_{SC} = P^t{}_{SC}[X]$
unfolding *stable-coin-actual-price-def* **by** *auto*
have $?Pb_{RC} \geq (R - N_{SC} * ?P_{SC}) \mathbin{/} N_{RC}$
proof −
from ‹$N_{RC} > 0$› **have** $?Pb_{RC} \geq P^t{}_{RC}(X, R, N_{SC}, N_{RC})$
using *reserve-coin-buying-price-def* **by** *auto*
then show *?thesis*
unfolding *equity-def* **and** *liabilities-def* **and** *reserve-coin-target-price-def* **by** *simp*
qed
moreover from ‹$N_{SC} > 0$› **and** ‹$?P_{SC} < R \mathbin{/} N_{SC}$› **have** $R - N_{SC} * ?P_{SC} > 0$
by (*metis diff-gt-0-iff-gt mult.commute pos-less-divide-eq*)
ultimately obtain α **where** $?Pb_{RC} = ((R - N_{SC} * ?P_{SC}) \mathbin{/} N_{RC}) * \alpha$
using ‹$N_{RC} > 0$› **by** (*metis divide-inverse less-irrefl mult.left-commute nonzero-mult-div-cancel-left*)
with ‹$N_{RC} > 0$› **and** ‹$?Pb_{RC} \geq (R - N_{SC} * ?P_{SC}) \mathbin{/} N_{RC}$› **and** ‹$R - N_{SC} * ?P_{SC} > 0$› **have** $\alpha \geq 1$
by (*metis divide-pos-pos mult-le-cancel-left1*)
let $?\mathcal{E} = (R - N_{SC} * ?P_{SC}) \mathbin{/} N_{RC}$
have $E(X',R',N'_{SC}) \mathbin{/} N'_{RC} = ((N_{RC} \mathbin{/} (N_{RC} + n)) + ((n * (1 + fee) * \alpha) \mathbin{/} (N_{RC} + n))) * ?\mathcal{E}$
proof −
have $E(X', R', N'_{SC}) \mathbin{/} N'_{RC} = (R' - N'_{SC} * ?P'_{SC}) \mathbin{/} N'_{RC}$
unfolding *equity-def* **and** *liabilities-def* **by** *simp*
also from ‹$?P_{SC} = P^t{}_{SC}[X]$› **and** ‹$?P'_{SC} = P^t{}_{SC}[X]$› **and** *buy-rcs*(*5*−*7*) **have** $\ldots = (R + n * (1 + fee) * ?Pb_{RC} - N_{SC} * ?P_{SC}) \mathbin{/} (N_{RC} + n)$
by *force*

```
      also have ... = ((R − N_SC * ?P_SC) / (N_RC + n)) + ((n * (1 +
fee) * ?Pb_RC) / (N_RC + n))
        by (metis add.commute add-diff-eq add-divide-distrib)
      also from ‹?Pb_RC = ((R − N_SC * ?P_SC) / N_RC) * α› have ... =
((R − N_SC * ?P_SC) / (N_RC + n)) + ((n * (1 + fee) * ?E * α) / (N_RC +
n))
        by simp
    also have ... = ((N_RC / (N_RC + n)) * ((R − N_SC * ?P_SC) / N_RC))
+ (((n * (1 + fee) * α) / (N_RC + n)) * ?E)
      proof −
        from ‹N_RC > 0› have N_RC ≠ 0
          by blast
        then show ?thesis
          by (simp add: mult.commute)
      qed
      finally show ?thesis
        by (metis (no-types) ring-class.ring-distribs(2))
    qed
    moreover have E(X, R, N_SC) / N_RC ≤ ((N_RC / (N_RC + n)) + ((n
* (1 + fee) * α) / (N_RC + n))) * ?E
    proof −
      have (R − N_SC * ?P_SC) / N_RC ≤ ((N_RC / (N_RC + n)) + ((n * (1
+ fee) * α) / (N_RC + n))) * ?E
      proof −
        have (N_RC + (n * (1 + fee) * α)) / (N_RC + n) ≥ 1
        by (smt ‹N_RC > 0› ‹α ≥ 1› ‹n > 0› fee-is-percentage greaterThanAtMost-iff
le-divide-eq-1 mult-less-cancel-left2)
        then have (N_RC / (N_RC + n)) + ((n * (1 + fee) * α) / (N_RC +
n)) ≥ 1
          by (simp add: add-divide-distrib)
        with ‹N_RC > 0› and ‹R − N_SC * ?P_SC > 0› show ?thesis
        by (metis divide-pos-pos less-eq-real-def mult.commute mult-less-cancel-left2
not-le)
      qed
      then show ?thesis
        unfolding equity-def and liabilities-def by simp
    qed
    ultimately show ?thesis
      using ‹X = X'› and assms(3−5) and buy-rcs(4,8) by auto
  qed
next
  case (sell-rcs n X R N_SC N_RC R' N'_SC N'_RC)
  let ?P_SC = P_SC(X, R, N_SC)
  let ?P'_SC = P_SC(X', R', N'_SC)
  let ?Pt_RC = P^t_RC(X, R, N_SC, N_RC)
  from assms(4,8) and sell-rcs(6,8) have N_SC > 0
    by blast
  from assms(2−7) and sell-rcs(3,4,8) have X = X' and N_RC > 0 and
N'_RC > 0
    by auto
```

from ‹$N_{SC} > 0$› **have** $R - N_{SC} * ?P_{SC} \geq 0$
unfolding *stable-coin-actual-price-def* **by** (*smt divide-strict-right-mono nonzero-mult-div-cancel-left*)
with ‹$N_{RC} > 0$› **have** $?Pt_{RC} \geq 0$
unfolding *equity-def* **and** *liabilities-def* **and** *reserve-coin-target-price-def* **by** *auto*
have $R \geq R'$
proof −
from *sell-rcs*(*1*,*3*,*4*) **have** $n * (1 - fee) * ?Pt_{RC} \leq R$
using *is-valid-transaction.simps*(*4*) **by** *blast*
with *assms*(*3*) **and** *sell-rcs*(*1*,*3*,*5*) **and** ‹$?Pt_{RC} \geq 0$› **show** *?thesis*
using *fee-is-percentage* **by** *auto*
qed
have $?P_{SC} = {P^t}_{SC}[X]$
proof −
from *sell-rcs*(*1*,*3*,*4*) **have** $r_{min} \leq r(X,\ R,\ N_{SC})$
using *is-valid-transaction.simps*(*4*) **by** *blast*
then show *?thesis*
by (*rule peg-when-reserve-ratio-within-lower-bound*)
qed
have $?P'_{SC} = {P^t}_{SC}[X]$
proof −
from *sell-rcs*(*1*,*3*,*4*,*5*) **have** $r_{min} \leq r(X,\ R',\ N_{SC})$
using *is-valid-transaction.simps*(*4*) **by** *blast*
with *sell-rcs*(*6*) **and** ‹$X = X'$› **show** *?thesis*
using *peg-when-reserve-ratio-within-lower-bound* **by** *simp*
qed
let $?\mathcal{E} = (R - N_{SC} * ?P_{SC})\ /\ N_{RC}$
have $?Pt_{RC} = ?\mathcal{E}$
unfolding *equity-def* **and** *liabilities-def* **and** *reserve-coin-target-price-def* **by** *simp*
have $E(X',R',N'_{SC})\ /\ N'_{RC} = ((N_{RC}\ /\ (N_{RC} - n)) - ((n * (1 - fee))\ /\ (N_{RC} - n))) * ?\mathcal{E}$
proof −
have $E(X',\ R',\ N'_{SC})\ /\ N'_{RC} = (R' - N'_{SC} * ?P'_{SC})\ /\ N'_{RC}$
unfolding *equity-def* **and** *liabilities-def* **by** *simp*
also from *sell-rcs*(*5*−*7*) **and** ‹$?P_{SC} = {P^t}_{SC}[X]$› **and** ‹$?P'_{SC} = {P^t}_{SC}[X]$› **have** $\ldots = (R - n * (1 - fee) * ?Pt_{RC} - N_{SC} * ?P_{SC})\ /\ (N_{RC} - n)$
by *force*
also from ‹$?Pt_{RC} = ?\mathcal{E}$› **have** $\ldots = ((R - N_{SC} * ?P_{SC})\ /\ (N_{RC} - n)) - ((n * (1 - fee) * ?\mathcal{E})\ /\ (N_{RC} - n))$
by (*simp add*: *diff-divide-distrib*)
also from ‹$N_{RC} > 0$› **have** $\ldots = ((N_{RC}\ /\ (N_{RC} - n)) * ?\mathcal{E}) - (((n * (1 - fee))\ /\ (N_{RC} - n)) * ?\mathcal{E})$
by *auto*
finally show *?thesis*
by (*metis left-diff-distrib'*)
qed

moreover have $E(X,\ R,\ N_{SC})\ /\ N_{RC} \leq ((N_{RC}\ /\ (N_{RC} - n)) - ((n * (1 - fee))\ /\ (N_{RC} - n))) * ?\mathcal{E}$
proof −
from *assms*(3) **and** *sell-rcs*(1,3) **have** $n * (1 - fee) \leq n$
using *fee-is-percentage* **by** *force*
with ‹$N'_{RC} > 0$› **and** *sell-rcs*(7) **have** $(N_{RC} - n * (1 - fee))\ /\ (N_{RC} - n) \geq 1$
by *auto*
then have $(N_{RC}\ /\ (N_{RC} - n)) - ((n * (1 - fee))\ /\ (N_{RC} - n)) \geq 1$
by (*simp add*: *diff-divide-distrib*)
with ‹$?Pt_{RC} \geq 0$› **and** ‹$?Pt_{RC} = ?\mathcal{E}$› **have** $((N_{RC}\ /\ (N_{RC} - n)) - ((n * (1 - fee))\ /\ (N_{RC} - n))) * ?\mathcal{E} \geq ?\mathcal{E}$
by (*metis mult-le-cancel-right1 not-le*)
then show *?thesis*
unfolding *equity-def* **and** *liabilities-def* **by** *simp*
qed
ultimately show *?thesis*
using *assms*(3−5) **and** *sell-rcs*(4,8) **and** ‹$X = X'$› **by** *auto*
qed
qed

lemma *sequence-monotonically-increasing-equity-per-reserve-coin*:
assumes $\mathcal{S} \rightarrow^* \{\!|txs|\!\}\ \mathcal{S}'$
and $\forall\, tx \in set\ txs.\ tx\text{-}rate\ tx = X$
and $\mathcal{S} = (R,\ N_{SC},\ N_{RC})$
and $\mathcal{S}' = (R',\ N'_{SC},\ N'_{RC})$
and $N_{RC} > 0$
and $N'_{RC} > 0$
and $N'_{SC} > 0$
shows $E(X,\ R',\ N'_{SC})\ /\ N'_{RC} \geq E(X,\ R,\ N_{SC})\ /\ N_{RC}$
using *assms*
proof (*induction txs arbitrary*: $\mathcal{S}'\ R'\ N'_{SC}\ N'_{RC}$ *rule*: *rev-induct*)
case *Nil*
then show *?case*
using *sequence-transition.simps* **by** *force*
next
case (*snoc tx txs*)
from *snoc.prems*(1) **obtain** $\mathcal{S}''\ R''\ N''_{SC}\ N''_{RC}$ **where** $\mathcal{S} \rightarrow^* \{\!|txs|\!\}\ \mathcal{S}''$ **and** $\mathcal{S}'' \rightarrow \{\!|tx|\!\}\ \mathcal{S}'$ **and** $\mathcal{S}'' = (R'',\ N''_{SC},\ N''_{RC})$ **and** $N''_{SC} > 0$ **and** $N''_{RC} > 0$
using *sequence-transition.simps* **by** (*metis Nil-is-append-conv append1-eq-conv list.simps*(3))
moreover from *snoc.prems*(2) **have** ∗: *tx-rate tx* $= X$ **if** $tx \in set\ txs$ **for** *tx*
by (*simp add*: *that*)
ultimately have $E(X,\ R'',\ N''_{SC})\ /\ N''_{RC} \geq E(X,\ R,\ N_{SC})\ /\ N_{RC}$
using ‹$\mathcal{S} \rightarrow^* \{\!|txs|\!\}\ \mathcal{S}''$› **and** *snoc.IH* **and** *snoc.prems*(3−6) **by** *simp*
moreover from *snoc.prems*(2,4,6,7) **and** ‹$\mathcal{S}'' \rightarrow \{\!|tx|\!\}\ \mathcal{S}'$› **and** ‹$\mathcal{S}'' = (R'',\ N''_{SC},\ N''_{RC})$› **and** ‹$N''_{RC} > 0$› **and** ‹$N''_{SC} > 0$› **and** ∗ **have** $E(X,\ R',\ N'_{SC})\ /\ N'_{RC} \geq E(X,\ R'',\ N''_{SC})\ /\ N''_{RC}$

using *monotonically-increasing-equity-per-reserve-coin* **by** (*metis in-set-conv-decomp-first*)
ultimately show *?case*
by *simp*
qed

lemma *monotonically-increasing-reserve-per-stable-coin*:
assumes $\mathcal{S} \rightarrow\!\!\{\!|tx|\!\}\!\!\rightarrow \mathcal{S}'$
and $\mathcal{S} = (R,\ N_{SC},\ N_{RC})$
and $N_{SC} > 0$
and $\mathcal{S}' = (R',\ N'_{SC},\ N'_{RC})$
and $X =$ *tx-rate tx*
and $E(X,\ R,\ N_{SC}) = 0$
and $N'_{SC} > 0$
shows $R' \ /\ N'_{SC} \geq R \ /\ N_{SC}$
proof −
from *assms*(*3*,*6*) **have** *P0*$_{SC}$*-def*: $P_{SC}(X,\ R,\ N_{SC}) = R \ /\ N_{SC}$
unfolding *equity-def* **and** *liabilities-def* **by** (*metis eq-iff-diff-eq-0 min.strict-order-iff nonzero-mult-div-cancel-left*)
with *assms*(*6*) **have** $r(X,\ R,\ N_{SC}) < r_{min}$
unfolding *equity-def* **and** *reserve-ratio-def* **using** r_{min}*-lower-bound* **and** *fee-is-percentage* **by** *auto*
from *assms* **show** *?thesis*
proof *cases*
case *buy-scs*
with *assms*(*2*,*5*) **and** ‹$r(X,\ R,\ N_{SC}) < r_{min}$› **show** *?thesis*
by *force*
next
case (*sell-scs n X R* N_{SC} N_{RC} R' N'_{SC} N'_{RC})
from *assms*(*2*,*3*) **and** *sell-scs*(*4*) **have** $N_{SC} > 0$
by *blast*
with *assms*(*4*,*7*) **and** *sell-scs*(*6*,*8*) **have** $N_{SC} - n > 0$
by *blast*
from *assms*(*2*) **and** *sell-scs*(*1*,*3*) **have** $n > 0$
using *is-valid-transaction.simps*(*2*) **by** *blast*
then have $n * (1 - fee) \leq n$
using *fee-is-percentage* **by** *auto*
then have $-\ n * (1 - fee) \geq -\ n$
by *simp*
then have $N_{SC} - n * (1 - fee) \geq N_{SC} - n$
by *simp*
with ‹$N_{SC} - n > 0$› **have** $(N_{SC} - n * (1 - fee)) \ /\ (N_{SC} - n) \geq 1$
using *le-divide-eq-1* **by** *blast*
then have $((R * (N_{SC} - n * (1 - fee))) \ /\ (N_{SC} - n)) * (1 \ /\ R) \geq R * (1 \ /\ R)$
by *simp*
with ‹$N_{SC} - n > 0$› **and** *sell-scs*(*2*,*4*) **have** $(R * (N_{SC} - n * (1 - fee))) \ /\ (N_{SC} - n) \geq R$
using *is-valid-bank-state.simps* **by** (*smt divide-nonneg-nonneg divide-pos-pos mult-not-zero mult-strict-right-mono*)

with ‹$N_{SC} > 0$› **have** $(1 \;/\; N_{SC}) * (R * (N_{SC} - n * (1 - fee))) \;/\; (N_{SC} - n) \geq (1 \;/\; N_{SC}) * R$
by (*smt divide-nonneg-pos mult-less-cancel-left times-divide-eq-right*)
then have $((R \;/\; N_{SC}) * (N_{SC} - n * (1 - fee))) \;/\; (N_{SC} - n) \geq R \;/\; N_{SC}$
by *simp*
then have $((R \;/\; N_{SC}) * N_{SC} - (R \;/\; N_{SC}) * n * (1 - fee)) \;/\; (N_{SC} - n) \geq R \;/\; N_{SC}$
proof −
have $(R \;/\; N_{SC}) * (N_{SC} - n * (1 - fee)) = (R \;/\; N_{SC}) * N_{SC} - (R \;/\; N_{SC}) * n * (1 - fee)$
by (*simp add*: *right-diff-distrib'*)
then show *?thesis*
using ‹$(R \;/\; N_{SC}) * (N_{SC} - n * (1 - fee)) \;/\; (N_{SC} - n) \geq R \;/\; N_{SC}$›
by *auto*
qed
with ‹$N_{SC} > 0$› **have** $(R - n * (1 - fee) * (R \;/\; N_{SC})) \;/\; (N_{SC} - n) \geq R \;/\; N_{SC}$
by (*metis* (*no-types*, *hide-lams*) *divide-inverse divide-self-if mult.commute mult.left-commute mult.left-neutral not-le order-refl*)
moreover from *assms*(*2*,*5*) **and** *sell-scs*(*3*−*6*) **and** *P0*$_{SC}$*-def* **have** $R' = R - n * (1 - fee) * (R \;/\; N_{SC})$ **and** $N'_{SC} = N_{SC} - n$
by *auto*
ultimately show *?thesis*
using *assms*(*2*,*4*) **and** *sell-scs*(*4*,*8*) **by** *blast*
next
case (*buy-rcs n X R* N_{SC} N_{RC} R' N'_{SC} N'_{RC})
have $R' > R$
proof −
from *buy-rcs*(*5*) **have** $R' = R + n * (1 + fee) * {P^b}_{RC}(X,\, R,\, N_{SC},\, N_{RC})$
by *auto*
moreover from *buy-rcs*(*1*,*3*,*4*) **have** $n > 0$
using *is-valid-transaction.simps*(*3*) **by** *blast*
moreover have $(1 + fee) * {P^b}_{RC}(X,\, R,\, N_{SC},\, N_{RC}) > 0$
unfolding *reserve-coin-buying-price-def* **using** *p-min-rc-positivity* **and** *fee-is-percentage* **by** *auto*
ultimately show *?thesis*
using *fee-is-percentage* **by** (*fastforce dest*: *zero-less-mult-pos*)
qed
with *assms*(*2*−*4*) **and** *buy-rcs*(*4*,*6*,*8*) **show** *?thesis*
by (*fastforce simp*: *divide-le-cancel*)
next
case *sell-rcs*
with *assms*(*2*,*5*) **and** ‹$r(X,\, R,\, N_{SC}) < r_{min}$› **show** *?thesis*
by *force*
qed
qed

lemma *constant-exchange-rate-non-negativity*:

```
  assumes txs ≠ []
    and 𝒮 →*⦇txs⦈ 𝒮'
    and ∀ tx ∈ set txs. tx-rate tx = X
  shows X ≥ 0
proof −
  from assms(2) and assms(1,3) show ?thesis
  proof cases
    case tx-seq-base
    with assms(1) show ?thesis
      by auto
  next
    case (txs-seq-ind txs' 𝒮'' tx R'' N''_SC N''_RC)
    from ‹𝒮'' →⦇tx⦈ 𝒮'› have tx-rate tx ≥ 0
      by (auto intro: transition.cases)
    with assms(3) and txs-seq-ind(1) show ?thesis
      by simp
  qed
qed

lemma bank-state-validity-invariancy:
  assumes 𝒮 →⦇tx⦈ 𝒮'
  shows is-valid-bank-state 𝒮'
  using assms
proof cases
  case buy-scs
  then show ?thesis
    using fee-is-percentage and peg-when-reserve-ratio-within-lower-bound by
auto
next
  case buy-rcs
  then show ?thesis
   unfolding reserve-coin-buying-price-def using fee-is-percentage and p-min-rc-positivity
by auto
qed auto

theorem no-reserve-draining-alt:
  assumes 𝒮 →*⦇txs⦈ 𝒮'
    and ∀ tx ∈ set txs. tx-rate tx = X
    and 𝒮 = (R, N_SC, N_RC)
    and 𝒮' = (R', N'_SC, N'_RC)
    and N_RC > 0
    and N'_RC > 0
    and N'_SC > 0
    and N_SC > 0
    and ∀ tx ∈ set txs. ∀ 𝒮 𝒮' R N_SC N_RC. 𝒮 →⦇tx⦈ 𝒮' ∧ 𝒮 = (R, N_SC,
N_RC) ⟶ N_SC > 0 ∧ N_RC > 0
    and txs ≠ []
  shows ¬ (R' < R ∧ N'_SC = N_SC ∧ N'_RC = N_RC)
proof (cases E(X, R, N_SC) = 0)
  case True
```

with *assms*(8) **have** *P0*$_{SC}$*-def*: $P_{SC}(X,\ R,\ N_{SC}) = R\ /\ N_{SC}$
unfolding *equity-def* **and** *liabilities-def* **by** (*metis eq-iff-diff-eq-0 min.strict-order-iff nonzero-mult-div-cancel-left*)
from *assms*(1,2,10) **have** $X \geq 0$
using *constant-exchange-rate-non-negativity* **by** *blast*
show *?thesis*
proof (*rule ccontr*)
assume $*$: $\neg$ *?thesis*
from $*$ **and** *assms*(7) **have** *R-N*$_{SC}$*-div*: $R'\ /\ N'_{SC} < R\ /\ N_{SC}$
by (*blast intro*: *divide-strict-right-mono*)
moreover from $\langle \mathcal{S} \rightarrow^* \{\!|txs|\!\} \mathcal{S}' \rangle$ **and** *assms*(2,9,5−8) **and** $\langle \mathcal{S} = (R,\ N_{SC},\ N_{RC}) \rangle$ **and** $\langle \mathcal{S}' = (R',\ N'_{SC},\ N'_{RC}) \rangle$ **and** $\langle E(X,\ R,\ N_{SC}) = 0 \rangle$
have $R'\ /\ N'_{SC} \geq R\ /\ N_{SC}$
proof (*induction txs arbitrary*: $\mathcal{S}\ R\ N_{SC}\ N_{RC}$)
case *Nil*
then show *?case*
by (*auto intro*: *sequence-transition.cases*)
next
case (*Cons tx txs*)
from *Cons.prems*(4,7) **and** $\langle \mathcal{S} \rightarrow^* \{\!|tx\ \#\ txs|\!\} \mathcal{S}' \rangle$ **obtain** $\mathcal{S}''$ **where** $\mathcal{S} \rightarrow \{\!|tx|\!\} \mathcal{S}''$ **and** $\mathcal{S}'' \rightarrow^* \{\!|txs|\!\} \mathcal{S}'$ **and** $N_{SC} > 0$ **and** $N_{RC} > 0$
using *sequence-transition-cons* **by** *blast*
moreover obtain $R''\ N''_{SC}\ N''_{RC}$ **where** $\mathcal{S}''$*-def*: $\mathcal{S}'' = (R'',\ N''_{SC},\ N''_{RC})$
by (*meson is-valid-bank-state.elims*(2,3))
moreover from *Cons.prems*(2) **have** *const-rate*: $\forall\, tx \in set\ txs.\ tx\text{-}rate\ tx = X$
by *simp*
moreover from *Cons.prems*(3) **have** *init-state-pos*: $\forall\, tx \in set\ txs.\ \forall \mathcal{S}\ \mathcal{S}'\ R\ N_{SC}\ N_{RC}.\ \mathcal{S} \rightarrow \{\!|tx|\!\} \mathcal{S}' \wedge \mathcal{S} = (R,\ N_{SC},\ N_{RC}) \longrightarrow 0 < N_{SC} \wedge 0 < N_{RC}$
by *auto*
moreover have $N''_{RC} > 0$ **and** $N''_{SC} > 0$
proof −
from $\langle \mathcal{S}'' \rightarrow^* \{\!|txs|\!\} \mathcal{S}' \rangle$ **obtain** Γ **where** *length* Γ = *length txs* + *1* **and** $\mathcal{S}'' = \Gamma\ !\ 0$ **and** $\mathcal{S}' = \Gamma\ !$ *length txs* **and** $\forall\, i \in \{0..<\text{length } txs\}.\ \Gamma\ !\ i \rightarrow \{\!|txs\ !\ i|\!\}\ \Gamma\ !\ (i + 1)$
using *sequence-transition-alt* **by** *blast*
then have $N''_{RC} > 0 \wedge N''_{SC} > 0$
proof (*cases txs* = [])
case *True* — $\mathcal{S}'' = \mathcal{S}'$
with *assms*(4,6,7) **and** $\mathcal{S}''$*-def* **and** $\langle \mathcal{S}'' = \Gamma\ !\ 0 \rangle$ **and** $\langle \mathcal{S}' = \Gamma\ !$ *length txs*$\rangle$ **show** *?thesis*
by *auto*
next
case *False*
with $\langle \mathcal{S}'' = \Gamma\ !\ 0 \rangle$ **and** $\langle \forall\, i \in \{0..<\text{length } txs\}.\ \Gamma\ !\ i \rightarrow \{\!|txs\ !\ i|\!\}\ \Gamma\ !\ (i + 1) \rangle$ **obtain** i **where** $i \in \{0..<\text{length } txs\}$ **and** $\mathcal{S}'' \rightarrow \{\!|txs\ !\ i|\!\}\ \Gamma\ !\ (i + 1)$
using *atLeastLessThan-iff* **by** *blast*

```
        moreover from ‹i ∈ {0..<length txs}› have txs ! i ∈ set txs
          by auto
        ultimately show ?thesis
          using 𝒮''-def and init-state-pos by blast
      qed
      then show N''_RC > 0 and N''_SC > 0
        by (rule conjunct1, rule conjunct2)
    qed
    ultimately have R'' / N''_SC ≥ R / N_SC
    using Cons.prems(2,8,10) and monotonically-increasing-reserve-per-stable-coin
by simp
    moreover have R'' / N''_SC ≤ R' / N'_SC
    proof −
      have E(X, R'', N''_SC) = 0
      proof (rule ccontr)
        assume E(X, R'', N''_SC) ≠ 0
        have E(X, R'', N''_SC) > 0
        proof −
          from ‹𝒮 →⦇tx⦈ 𝒮''› and 𝒮''-def have R'' ≥ 0
            using bank-state-validity-invariancy by fastforce
         with ‹E(X, R'', N''_SC) ≠ 0› and ‹X ≥ 0› and ‹N''_SC > 0› show
?thesis
            using no-insolvency unfolding less-eq-real-def by auto
        qed
       with assms(4,6,7) and ‹𝒮'' →*⦇txs⦈ 𝒮'› and const-rate and 𝒮''-def
and ‹N''_RC > 0›
        have E(X, R'', N''_SC) / N''_RC ≤ E(X, R', N'_SC) / N'_RC
          using sequence-monotonically-increasing-equity-per-reserve-coin by
blast
         moreover have E(X, R'', N''_SC) / N''_RC > E(X, R', N'_SC) /
N'_RC
        proof −
          from P0_SC-def and R-N_SC-div have R' / N'_SC < P^t_SC[X]
           unfolding stable-coin-actual-price-def by (metis min-less-iff-conj)
          then have E(X, R', N'_SC) = 0
          proof −
            have E(X, R', N'_SC) = R' − N'_SC * P_SC(X, R', N'_SC)
              unfolding equity-def and liabilities-def by simp
           also from ‹R' / N'_SC < P^t_SC[X]› have ... = R' − N'_SC * (R'
/ N'_SC)
            unfolding stable-coin-actual-price-def by (simp add: min.commute
min.strict-order-iff)
            finally show ?thesis
              using assms(7) by auto
          qed
          with ‹E(X, R'',N''_SC) > 0› and ‹N''_RC > 0› show ?thesis
            by simp
        qed
        ultimately show False
          by force
```

```
      qed
      with assms(6,7) and ⟨𝒮'' →* {|txs|} 𝒮'⟩ and ⟨N''_RC > 0⟩ and ⟨N''_SC
> 0⟩ and 𝒮''-def and ⟨𝒮' = (R', N'_SC, N'_RC)⟩ and init-state-pos and
const-rate show ?thesis
        using Cons.IH by blast
     qed
     ultimately show ?case
      by fastforce
   qed
   ultimately show False
    by fastforce
  qed
next
  case False
  then have P0_SC-def: P_SC(X, R, N_SC) = P^t_SC[X]
    unfolding equity-def and liabilities-def and stable-coin-actual-price-def
by (smt divide-eq-eq mult.commute)
  with assms(8) have P^t_SC[X] ≤ R / N_SC
   unfolding stable-coin-actual-price-def by auto
  have E(X, R, N_SC) > 0
  proof −
   from P0_SC-def have E(X, R, N_SC) = R − N_SC * P^t_SC[X]
    unfolding liabilities-def and equity-def by simp
   with False and ⟨N_SC > 0⟩ and ⟨P^t_SC[X] ≤ R / N_SC⟩ have P^t_SC[X]
< R / N_SC
    unfolding less-eq-real-def by force
   with ⟨N_SC > 0⟩ and ⟨E(X, R, N_SC) = R − N_SC * P^t_SC[X]⟩ show
?thesis
    by (simp add: mult.commute pos-less-divide-eq)
  qed
  show ?thesis
  proof (rule ccontr)
   assume *: ¬ ?thesis
   have E(X, R', N'_SC) < E(X, R, N_SC)
   proof −
    from * and assms(7) have N_SC > 0
     by blast
   from * have E'-def: E(X, R', N'_SC) = R' − N_SC * P_SC(X, R', N_SC)
     unfolding liabilities-def and equity-def by blast
    moreover from * have E-def: E(X, R, N_SC) = R − N_SC * P_SC(X,
R, N_SC)
     unfolding liabilities-def and equity-def by blast
    ultimately show ?thesis
    proof −
     consider (a) P^t_SC[X] ≤ R' / N_SC | (b) R' / N_SC < P^t_SC[X]
      using ⟨P^t_SC[X] ≤ R / N_SC⟩ by linarith
     then show ?thesis
     proof cases
      case a
      with E'-def have E(X, R', N'_SC) = R' − N_SC * P^t_SC[X]
```

```
          unfolding stable-coin-actual-price-def by auto
        also from * have ... < R − N_SC * P^t_SC[X]
          by auto
        also from ⟨P^t_SC[X] ≤ R / N_SC⟩ and E-def have ... = E(X, R,
N_SC)
          unfolding stable-coin-actual-price-def by auto
        finally show ?thesis .
      next
        case b
        with assms(8) have P_SC(X, R', N_SC) = R' / N_SC
          unfolding stable-coin-actual-price-def by auto
        with E'-def have E(X, R', N'_SC) = R' − N_SC * (R' / N_SC)
          unfolding stable-coin-actual-price-def by auto
        also from assms(8) have ... = 0
          by auto
        also from assms(8) and False and E-def and P0_SC-def and
⟨P^t_SC[X] ≤ R / N_SC⟩ have ... < E(X, R, N_SC)
          by (smt mult.commute pos-le-divide-eq)
        finally show ?thesis .
      qed
    qed
  qed
  moreover have E(X, R', N'_SC) ≥ E(X, R, N_SC)
  proof −
    from assms(1−7) and * have E(X, R', N'_SC) / N'_RC ≥ E(X, R,
N_SC) / N_RC
    using sequence-monotonically-increasing-equity-per-reserve-coin by simp
    with assms(6) and * show ?thesis
      using divide-le-cancel by blast
  qed
  ultimately show False
    by linarith
 qed
qed

corollary no-reserve-draining:
 assumes N_RC > 0
   and N'_RC > 0
   and N'_SC > 0
   and N_SC > 0
 shows ∄ txs. txs ≠ [] ∧ (R, N_SC, N_RC) →*⦃txs⦄ (R', N'_SC, N'_RC) ∧ (∀ tx
∈ set txs. ∀ S S' R N_SC N_RC. S →⦃tx⦄ S' ∧ S = (R, N_SC, N_RC) ⟶ N_SC
> 0 ∧ N_RC > 0) ∧ (∀ tx ∈ set txs. tx-rate tx = X) ∧ R' < R ∧ N'_SC =
N_SC ∧ N'_RC = N_RC
 using assms and no-reserve-draining-alt by blast

theorem bounded-dilution:
 assumes S →⦃tx⦄ S'
   and (BuyRCs n, X) = tx
   and S = (R, N_SC, N_RC)
```

and $\mathcal{S}' = (R',\ N'_{SC},\ N'_{RC})$
and $X = X'$
and $N'_{SC} = N_{SC}$
and $N_{RC} > 0$
and $N_{SC} > 0$
and $r(X',\ R',\ N'_{SC}) = r_{max}$
shows $n = (r_{max} * N_{SC} * P^t{}_{SC}[X] - R)\ /\ ((1 + fee) * P^b{}_{RC}(X,\ R,\ N_{SC},\ N_{RC}))$
proof −
let $?P_{SC} = P_{SC}(X,\ R,\ N_{SC})$
let $?P'_{SC} = P_{SC}(X',\ R',\ N'_{SC})$
have $r_{min} > 1$
using *fee-is-percentage* **and** r_{min}*-lower-bound* **by** *auto*
from *assms*(6) **have** $r(X',\ R',\ N'_{SC}) = R'\ /\ (N_{SC} * ?P'_{SC})$
unfolding *reserve-ratio-def* **and** *equity-def* **and** *liabilities-def* **by** *simp*
moreover from *assms*(9) **and** r_{min}*-upper-bound* **and** ‹$r_{min} > 1$› **have** $r(X',\ R',\ N'_{SC}) > 1$
by *linarith*
ultimately have $R'\ /\ N_{SC} > ?P'_{SC}$
unfolding *stable-coin-actual-price-def* **using** *assms*(6) **and** *less-divide-eq-1* **by** *fastforce*
with *assms*(5,9) **have** $?P'_{SC} = P^t{}_{SC}[X]$
using *peg-when-reserve-ratio-within-lower-bound* **and** r_{min}*-upper-bound* **by** *simp*
moreover from *assms*(1−4) **have** $R' = R + n * (1 + fee) * P^b{}_{RC}(X,\ R,\ N_{SC},\ N_{RC})$
by (*blast intro*: *transition.cases*)
ultimately have $(R + n * (1 + fee) * P^b{}_{RC}(X,\ R,\ N_{SC},\ N_{RC}))\ /\ (N_{SC} * P^t{}_{SC}[X]) = r_{max}$
using *assms*(9) **and** ‹$r(X',R',N'_{SC}) = R'\ /\ (N_{SC} * ?P'_{SC})$› **by** *auto*
moreover from *assms*(9) **and** ‹$r(X',\ R',\ N'_{SC}) > 1$› **have** $r_{max} > 1$
by *blast*
ultimately have $R + n * (1 + fee) * P^b{}_{RC}(X,\ R,\ N_{SC},\ N_{RC}) = r_{max} * N_{SC} * P^t{}_{SC}[X]$
by *fastforce*
then have $n * (1 + fee) * P^b{}_{RC}(X,\ R,\ N_{SC},\ N_{RC}) = r_{max} * N_{SC} * P^t{}_{SC}[X] - R$
by *argo*
moreover have $1 + fee \neq 0$
using *fee-is-percentage* **by** *auto*
ultimately have $n * P^b{}_{RC}(X,\ R,\ N_{SC},\ N_{RC}) = (r_{max} * N_{SC} * P^t{}_{SC}[X] - R)\ /\ (1 + fee)$
by (*metis mult.assoc nonzero-mult-div-cancel-left times-divide-eq-right*)
moreover have $P^b{}_{RC}(X,\ R,\ N_{SC},\ N_{RC}) \neq 0$
unfolding *reserve-coin-buying-price-def* **using** *p-min-rc-positivity* **by** *auto*
ultimately show *?thesis*
using *divide-divide-eq-left* **and** *eq-divide-imp* **by** *blast*
qed

end

end

F LUSTRE Formalization

Listing 5: Base.lus File

```
-- Basic Types
-- These types are to be defined for each specific implementation

-- type ReserveType = ...;
-- type N_Type = ...;

-- type RateType = ...;
-- type AmountType = ...;
-- type RatioType = ...;

-- Define Enumerations for Order and Reply
type Order = enum { MintSC, MintRC, NoOrder};
type Proceed = enum { MintedSC, MintedRC, RedeemedSC, RedeemedRC, Error, NoReply};
type ErrorInfo = enum { Min_Ratio_Violated, Max_Ratio_Violated, Invalid_Mint_Value, None };

-- Input Msg Type :Order Invocation
type InputMsg = struct { order: Order; qnt: AmountType };

-- Output Msg Type
type OutputMsg = struct { ack: Proceed; err: ErrorInfo; price: ReserveType };


-- Constant Definitions
-- ErrorMsg when transaction aborted
const ErrorCode1 = OutputMsg { ack = Error; err = Min_Ratio_Violated; price = 0 };
const ErrorCode2 = OutputMsg { ack = Error; err = Max_Ratio_Violated; price = 0 };
const ErrorCode3 = OutputMsg { ack = Error; err = Invalid_Mint_Value; price = 0 };

-- Null Reply Msg
const NullReply = OutputMsg { ack = NoReply; err = None; price = 0 };

-- StableCoin parameters type
type Parameters = struct { r_min: RatioType; r_max: RatioType; fee: ReserveType; n_sc_s: N_Type; p_min: ReserveType };


-- Function params is introduced to provide global access to the stablecoin parameters.
-- It can either be defined concretely with specific values or left abstract with
-- constraints characterizing the min value and max value allowed for each parameters.
-- For a concrete specification the following syntax should be used:
-- function params () returns (out: Parameters)
-- let
-- out = Parameters { r_min = < value> ; r_max = < value> ; fee = < value> ; n_sc_s = < value> ;
--    p_min = < value> };
-- tel

-- For an abstract specification the following syntax should be used:
-- function imported params () returns (out: Parameters);
--
-- with the constraints expressed on the stablecoin parameters to be specified in the Properties node.

function min (a: ReserveType; b: ReserveType )
        returns (out :ReserveType)
let
 out = if a < b then a else b;
tel

function max (a: ReserveType; b: ReserveType )
        returns (out: ReserveType)
let
 out = if a < b then b else a;
tel

function abs (a: ReserveType )
        returns (out: ReserveType)
let
 out = if a < 0 then -a else a;
tel

-- computeFee(t_price) computes transaction fee accroding to total price t_price.
-- In Lustre, Ecludien division is used where the type of rounding applied is
-- correlated depends on the sign of the divisor:
-- - Rounding towards negative infinity when divisor is positive
-- - Rounding towards positive infinity when divisor is negative
-- The computeFee function forces rounding towards infinity when computing
-- the fee percentage to guarantee accumulation of fees.
function computeFee(t_price: ReserveType)
        returns (out: ReserveType)
  var t_fee :ReserveType;
let
  t_fee = (abs(t_price) * params().fee + 99 ) div 100;
  out = t_price + t_fee;
tel
```

Listing 6: StableCoin.lus File

```
include "Base.lus"

-- Base Types Definition
type ReserveType = int;
type N_Type = int;

type RateType = int;
type AmountType = int;
type RatioType = int;


-- Function params left abstract to consider all possible values.
function imported params ()  returns (out: Parameters);


-- equity(reserve, n_sc, rate) computes the surplus of reserve w.r.t. liabilities s.t.:
-- - equity = reserve - min(reserve, n_sc * rate)
function equity(reserve: ReserveType; n_sc: N_Type; rate :RateType)
         returns (out :ReserveType)
let
  out = if reserve > n_sc * rate then reserve - (n_sc * rate) else 0;
tel


-- price_sc(reserve, n_sc, rate) computes price for a single stable coin s.t.:
-- price_sc = min(reserve / n_sc, rate) if n_sc > 0
-- = rate otherwise
function price_sc(reserve: ReserveType; n_sc: N_Type; rate: RateType)
         returns (out: ReserveType)
let
 out = if n_sc > 0 then
        if reserve >= n_sc * rate then rate else reserve div n_sc
       else
         rate;
tel

-- price_rc(d_rc, reserve, n_sc, n_rc, rate) computes price for a single reserve coin s.t.:
-- - price = p_min if n_rc = 0
-- - price = max(equity / n_rc, p_min) if d_rc >= 0
-- - price = equity / n_rc otherwise
function price_rc(d_rc: N_Type; reserve: ReserveType; n_sc: N_Type; n_rc: N_Type; rate: RateType)
         returns (out: ReserveType)
let
 out = if n_rc = 0 then
         params().p_min
       else
         if d_rc >= 0 then
           max(equity(reserve, n_sc, rate) div n_rc, params().p_min)
         else
           equity(reserve, n_sc, rate) div n_rc;
tel


-- mintSC(d_sc, rate, reserve, n_sc) performs buying and selling of stable coins s.t.:
-- - buying is performed when d_sc is positive, exchange rate > 0 and reserve ratio >= r_min
-- - selling is performed when d_sc is negative, | d_sc | <= n_sc and exchange rate > 0.
function mintSC(d_sc: N_Type; rate: RateType; reserve: ReserveType; n_sc: N_Type)
         returns (o_msg: OutputMsg)

 var s_price, t_price, t_reserve: ReserveType;

let
  s_price = price_sc(reserve, n_sc, rate) * d_sc;
  t_price = computeFee(s_price);
  t_reserve = reserve + t_price;

  o_msg = if d_sc >= 0 then
            -- buying stable coin: check min reserve ratio
           if rate > 0 and t_reserve >= (n_sc + d_sc) * rate * params().r_min
              -- buying not authorized when rate < 0
           then
            OutputMsg { ack = MintedSC; err = None; price = t_price }
           else
            ErrorCode1
         else if -d_sc <= n_sc and rate > 0 then
            -- selling stable coin
            -- selling not authorized when rate < 0
           OutputMsg { ack = RedeemedSC; err = None; price = t_price }
         else
           -- error
           ErrorCode3;
tel
```

```
-- mintRC(d_rc, rate, reserve, n_sc, n_rc) performs buying and selling of reserve coins s.t.:
-- - buying is performed when d_rc is positive and:
-- - n_sc < params().n_sc_s or
-- - reserve ratio <= r_max
-- - selling is performed when d_rc is negative and | d_rc | <= n_rc and:
-- - reserve ratio >= min ratio and exchange rate > 0 or
-- - N_SC = 0
function mintRC(d_rc: N_Type; rate: RateType; reserve: ReserveType; n_sc: N_Type; n_rc :N_Type)
         returns (o_msg: OutputMsg)

 var r_price, t_price, t_reserve: ReserveType;

let
  r_price = price_rc(d_rc, reserve, n_sc, n_rc, rate) * d_rc;
  t_price = computeFee(r_price);
  t_reserve = reserve + t_price;

  o_msg = if d_rc >= 0 then
           -- buying reserve coin: check max reserve ratio or n_sc <= params().n_sc_s
          if n_sc < params().n_sc_s or t_reserve <= n_sc * rate * params().r_max
          then
           -- buying not authorized when n_sc > 0 and rate <= 0
           OutputMsg { ack = MintedRC; err = None; price = t_price }
          else
           ErrorCode2
         else if -d_rc <= n_rc then
           -- selling reserve coin :check min reserve ratio
           if n_sc = 0 or (rate > 0 and t_reserve >= n_sc * rate * params().r_min)
             -- selling not authorized when rate <= 0
           then
             OutputMsg { ack = RedeemedRC; err = None; price = t_price }
           else
             ErrorCode1
         else
           -- error
           ErrorCode3;
tel


-- main node encoding the stablecoin logic and maintaining the bank state
node StableCoin_InitState(i_reserve: ReserveType; i_sc: N_Type; i_rc: N_Type; i_msg: InputMsg; rate: RateType)
     returns (p_reserve: ReserveType; p_sc: N_Type;
              p_rc: N_Type; reserve: ReserveType; n_sc: N_Type;
              n_rc: N_Type; o_msg: OutputMsg)

let

 p_reserve = i_reserve -> pre reserve;
 p_sc = i_sc -> pre n_sc;
 p_rc = i_rc -> pre n_rc;

 o_msg = if i_msg.order = NoOrder then
           NullReply
         else if i_msg.order = MintSC then
           mintSC(i_msg.qnt, rate, p_reserve, p_sc)
         else
           -- MintRC case
          mintRC(i_msg.qnt, rate, p_reserve, p_sc, p_rc);

 reserve = if o_msg.ack = Error
           then
             -- no modification
            p_reserve
           else
            p_reserve + o_msg.price;

 n_sc = if o_msg.ack = MintedSC or o_msg.ack = RedeemedSC
        then
          p_sc + i_msg.qnt
        else
          -- no modification
          p_sc;


 n_rc = if o_msg.ack = MintedRC or o_msg.ack = RedeemedRC
        then
          p_rc + i_msg.qnt
        else
          -- no modification
          p_rc;

tel
```

```
-- stablecoin node with intial bank state (R = 0, N_SC = 0 and N_RC = 0)
node StableCoin (i_msg: InputMsg; rate: RateType )
     returns (p_reserve: ReserveType; p_sc: N_Type;
              p_rc: N_Type; reserve: ReserveType; n_sc: N_Type;
              n_rc: N_Type; o_msg: OutputMsg )

let

 p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin_InitState(0, 0, 0, i_msg, rate);

tel
```

Listing 7: Theorem_Base.lus File

```
-- Node specifying the constraints to be satisfied by the stablecoin parameters.
node ParameterConstraints ()  returns (out :bool)
let
  -- Constraints on stablecoin parameters
  -- Parameters remain unchanged
  assert true ->
        (params().r_min = pre params().r_min and
          params().r_max = pre params().r_max and
          params().fee = pre params().fee and
          params().n_sc_s = pre params().n_sc_s and
          params().p_min = pre params().p_min );

  -- Parameters min and max value specification
  assert params().r_min > computeFee(1) and
        params().r_max >= params().r_min and
        params().fee > 0 and
        params().fee <= 100 and
        params().n_sc_s > 0 and
        params().p_min > 0;

  -- dummy output
  out = true;

tel
```

Listing 8: Theorem1_and_2.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

-- Types for modeling secondary market
type MarketAction = enum { BuyOffer, SellOffer, NoOffer};

type SecondaryMarket = struct { action :MarketAction; price :ReserveType; };


node Theorem1_and_2 (i_msg: InputMsg; rate: RateType; rational_user: bool;
                     s_market :SecondaryMarket)
     returns (o_msg: OutputMsg)

 var reserve: ReserveType;
    n_sc: N_Type;
    n_rc: N_Type;

    p_reserve :ReserveType;
    p_sc :N_Type;
    p_rc :N_Type;

    sufficient_reserve :bool;
 let

  --%MAIN;
 p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  -- Rational user behavioural constraints for buying SC from Bank
  assert (rational_user and s_market.action = SellOffer and
          computeFee(price_sc(p_reserve, p_sc, rate)) <= s_market.price ) => (i_msg.order = MintSC and i_msg.qnt > 0 );

  assert (rational_user and s_market.action = SellOffer and
           computeFee(price_sc(p_reserve, p_sc, rate)) > s_market.price ) => i_msg.order = NoOrder;


  -- Rational user behavioural constraints for Selling SC
  assert (rational_user and s_market.action = BuyOffer and
          - computeFee(price_sc(p_reserve, p_sc, rate) * -1) > s_market.price ) =>
            (i_msg.order = MintSC and i_msg.qnt < 0 );

  assert (rational_user and s_market.action = BuyOffer and
          - computeFee(price_sc(p_reserve, p_sc, rate) * -1) <= s_market.price ) => i_msg.order = NoOrder;


  sufficient_reserve = p_reserve + i_msg.qnt * rate >= (p_sc + i_msg.qnt ) * rate * params().r_min;

  -- THEOREM 1: Peg Maintenance - Upper Bound
  -- IF buying user is rational AND
  -- the bank reserve ratio is sufficient for a buy SC order to be accepted AND
  -- the exchange rate is greater than zero AND
  -- secondary market is selling SC for a price P > (1 + fee) * rate
  -- THEN
  -- The rational user will always buy SC from the bank (i.e., o_msg.ack = MintedSC)

  check "THEOREM_1"
   (rational_user and
     sufficient_reserve and -- epsilon sufficiently large for buy order to be accepted.
     rate > 0 and -- rate should be greater than zero for buy order to be accepted.
     s_market.action = SellOffer and -- secondary market is selling SC (buying for user)
     s_market.price > computeFee(price_sc(p_reserve, p_sc, rate)) ) => o_msg.ack = MintedSC;


  -- THEOREM 2: Peg Maintenance - Lower Bound
  -- IF selling user is rational AND
  -- bank reserve ratio >= 1 AND
  -- the exchange rate is greater than zero AND
  -- secondary market is buying SC for a price P < (1 - fee) * rate
  -- THEN
  -- The rational user will always sell SC to the bank (i.e., o_msg.ack = RedeemedSC)

  check "THEOREM_2"
   (rational_user and
     p_reserve >= p_sc * rate and -- reserve ratio >= 1
     rate > 0 and -- rate should be greater than zero for sell order to be accepted.
     -i_msg.qnt <= p_sc and -- number of SC to be sold <= number of stablecoin in circulation
     s_market.action = BuyOffer and -- secondary market is buying SC (selling for user)
     s_market.price < - computeFee(price_sc(p_reserve, p_sc, rate) * -1) ) => o_msg.ack = RedeemedSC;

tel
```

Listing 9: Theorem3.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

node Theorem3 (i_msg: InputMsg; rate: RateType)  returns (o_msg: OutputMsg)

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve :ReserveType;
     p_sc :N_Type;
     p_rc :N_Type;

     p_rate :RateType;

 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();


  p_rate = rate -> pre rate;

  -- THEOREM 3: Peg Robustness during Market Crashes
  -- IF N_SC > 0 AND
  -- p_rate and rate are the exchange rates before and after the crash s.t.,
  -- p_rate > 0 and rate > p_rate AND
  -- r is the current reserve ratio w.r.t. p_rate such that r > 1 AND
  -- crash price ratio is satisfied s.t., (rate - p_rate) / rate <= (r - 1) / r
  -- THEN
  -- The new stablecoin price shall still be "rate" (i.e., SC price still 1 PC in BC)

  check "THEOREM_3"
    (n_sc > 0 and
       p_rate > 0 and
       rate > p_rate and
       reserve > n_sc * p_rate and -- ratio > 1
       (rate - p_rate ) * reserve <= (reserve - n_sc * p_rate) * rate ) =>
          price_sc(reserve, n_sc, rate) = rate; -- SC price still 1 PC in BCs

  -- Lemmas
  check
    (n_sc > 0 and p_rate > 0 and
       rate > p_rate and
       reserve > n_sc * p_rate and
     (rate - p_rate ) * reserve <= (reserve - n_sc * p_rate) * rate ) => reserve >= n_sc * rate;

  check
    (n_sc > 0 and p_rate > 0 and
       rate > p_rate and reserve > n_sc * p_rate ) => (reserve - n_sc * p_rate) * rate > 0;

tel
```

Listing 10: Theorem4.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

node Theorem4 (i_msg: InputMsg; rate: RateType)  returns (o_msg: OutputMsg)

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve :ReserveType;
     p_sc :N_Type;
     p_rc :N_Type;

 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  -- THEOREM 4 :No Insolvency
  -- In all bank states and for any exchange rate, E(R, N_SC) >= 0

  check "THEOREM_4" equity(reserve, n_sc, rate) >= 0;

tel
```

Listing 11: Theorem5.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

type SCSellerType = struct { sold_once: bool; price_per_sc: ReserveType };

node update_SCSeller (p_reserve: ReserveType; p_sc: N_Type; rate: RateType; o_msg: OutputMsg; seller: SCSellerType )
     returns (out :SCSellerType)
let
 out = if o_msg.ack = RedeemedSC then
        SCSellerType { sold_once = true; price_per_sc = min(p_reserve div p_sc, rate) }
       else
         seller;
tel

const defaultSeller_SC = SCSellerType { sold_once = false; price_per_sc = 0 };


node Theorem5 (i_msg: InputMsg; rate: RateType )  returns (o_msg: OutputMsg);

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve :ReserveType;
     p_sc :N_Type;
     p_rc :N_Type;

     constant_rate :bool;
     p_seller :SCSellerType;
     c_seller :SCSellerType;

 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  constant_rate = true -> pre constant_rate and rate = pre rate;
  p_seller = defaultSeller_SC -> pre c_seller;
  c_seller = update_SCSeller(p_reserve, p_sc, rate, o_msg, p_seller);

  -- THEOREM 5: No Bank Runs for StableCoins
  -- IF the exchange rate remains constant AND
  -- at least one SC was already sold AND
  -- a selling SC order is accepted
  -- THEN
  -- Current selling price per SC >= previously applied price

  check "THEOREM_5"
   (constant_rate and p_seller.sold_once and o_msg.ack = RedeemedSC ) =>
       c_seller.price_per_sc >= p_seller.price_per_sc;

  -- Lemmas
  check (constant_rate and p_seller.sold_once ) => p_seller.price_per_sc <= rate;
  check (constant_rate and p_seller.sold_once and p_sc > 0 ) => p_seller.price_per_sc <= min(p_reserve div p_sc, rate);

tel
```

Listing 12: Theorem6.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

node Theorem6 (i_msg: InputMsg; rate: RateType)  returns (o_msg: OutputMsg);

 var reserve: ReserveType;
    n_sc: N_Type;
    n_rc: N_Type;

    p_reserve :ReserveType;
    p_sc :N_Type;
    p_rc :N_Type;

    constant_rate :bool;

 let

  --%MAIN;
 p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
 assert ParameterConstraints ();

 constant_rate = true -> pre constant_rate and rate = pre rate;

  -- THEOREM 6: Monotonically Increasing Equity per Reservecoin
  -- IF the exchange rate remains constant AND
  -- previous N_RC is greater than zero AND
  -- The bank state obtained after every action a is such that N_RC is still greater than zero
  -- THEN
  -- E(R, N_SC) / N_RC >= E(P_R, P_SC) / P_RC
  --
  -- where,
  -- - P_R, P_SC, P_RC are respectively the reserve, number of stablecoins and number of
       reservecoins before action a
  -- - R, N_SC, N_RC are respectively the reserve, number of stablecoins and number of
       reservecoins after action a

 check "THEOREM_6"
   (constant_rate and p_rc > 0 and n_rc > 0 ) =>
     equity(reserve, n_sc, rate) div n_rc >= equity(p_reserve, p_sc, rate) div p_rc;

  -- Lemmas
 check o_msg.ack = RedeemedRC => p_sc = n_sc;
 check o_msg.ack = RedeemedSC => p_rc = n_rc;
 check o_msg.ack = RedeemedRC => p_reserve >= reserve;

tel
```

Listing 13: Theorem7.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

node Theorem7 (i_msg: InputMsg; rate: RateType; i_reserve: ReserveType; i_sc: N_Type; i_rc: N_Type)
     returns (o_msg: OutputMsg)

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve: ReserveType;
     p_sc: N_Type;
     p_rc: N_Type;

     constant_rate: bool;

     reserve_0 :ReserveType;
     n_sc_0 :N_Type;
     n_rc_0 :N_Type;

     coins_positive :bool;

 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin_InitState(i_reserve, i_sc, i_rc, i_msg, rate);

  constant_rate = true -> pre constant_rate and rate = pre rate;

  -- Stores arbitrary initial bank state to check if a draining situation can happen.
  reserve_0 = i_reserve -> pre reserve_0;
  n_sc_0 = i_sc -> pre n_sc_0;
  n_rc_0 = i_rc -> pre n_rc_0;

  -- Checks whether N_SC and N_RC remain greater than zero.
  coins_positive = (i_sc > 0 and i_rc > 0 ) -> pre coins_positive and p_sc > 0 and p_rc > 0;

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  -- Assuming the validity of initial bank state
  assert i_reserve >= 0 and
         i_sc >= 0 and
         i_rc >= 0 and
         (i_sc > 0 => i_reserve div i_sc > 0 ) and
         (i_rc > 0 => i_reserve > 0 );


  -- THEOREM 7 :No Reserve Draining
  -- IF the exchange rate remains constant
  -- THEN
  -- For any initial bank state (R_0, N_SC_0, N_RC_0), there is no sequence of actions a1, a2,
       ..., a_n
  -- leading to a state (R_n, N_SC_n, N_RC_n) s.t.:
  -- - R_n < R_0 and N_SC_0 = N_SC_n and N_RC_0 = N_RC_n AND
  -- - For all i in [0, n - 1], N_SC_i > 0 and N_RC_i > 0

  check "THEOREM_7"
     (constant_rate and coins_positive ) => not (reserve < reserve_0 and n_sc = n_sc_0 and n_rc = n_rc_0 );

  -- Lemmas
  check (constant_rate and n_rc_0 > 0 and p_rc > 0 ) =>
      equity(p_reserve, p_sc, rate) div p_rc >= equity(reserve_0, n_sc_0, rate) div n_rc_0;

  check (constant_rate and coins_positive and equity(reserve_0, n_sc_0, rate) = 0 and n_sc > 0 ) =>
      reserve div n_sc >= reserve_0 div n_sc_0;

  check (constant_rate and coins_positive and p_sc <> n_sc_0 ) => rate > 0;

  check coins_positive => reserve_0 > 0;
  check coins_positive => p_reserve > 0;
  check coins_positive => (p_rc > 0 and p_sc > 0 );
  check coins_positive => (n_rc_0 > 0 and n_sc_0 > 0);

  check p_reserve >= 0 and p_sc >= 0 and p_rc >= 0;
  check p_rc > 0 => p_reserve > 0;
  check p_sc > 0 => p_reserve div p_sc > 0;

  check i_reserve >= 0 and i_rc >= 0 and i_sc >= 0;
  check i_rc > 0 => i_reserve > 0;
  check i_sc > 0 => i_reserve div i_sc > 0;

  check reserve_0 >= 0 and n_rc_0 >= 0 and n_sc_0 >= 0;
  check n_rc_0 > 0 => reserve_0 > 0;
  check n_sc_0 > 0 => reserve_0 div n_sc_0 > 0;

tel
```

Listing 14: Theorem8.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

function computeBound(reserve :ReserveType; n_sc: N_Type; n_rc: N_Type; rate: RateType)
         returns (out: ReserveType)

 var d_price :ReserveType;

let
  d_price = price_rc(1, reserve, n_sc, n_rc, rate) * (100 + params().fee);

  out = ((n_sc * rate * params().r_max - reserve ) * 100 + d_price - 1 ) div d_price;

tel


node Theorem8 (i_msg: InputMsg; rate: RateType; f_max: bool)  returns (o_msg: OutputMsg)

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve :ReserveType;
     p_sc :N_Type;
     p_rc :N_Type;

     constant_rate :bool;
     constant_sc :bool;
     max_bound :ReserveType;
     bound_defined :bool;
     no_rc_selling :bool;

     d_price :ReserveType;
     prev_d_price :ReserveType;

 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  -- max bound can be defined only when
  -- rate > 0, N_SC > 0, N_RC > 0 and
  -- R <= r_max * N_SC * rate and
  -- N_SC > params().n_sc_s
  max_bound = -1 -> if f_max and (pre max_bound = -1 ) and
                       rate > 0 and n_sc > 0 and n_rc > 0 and
                       n_sc >= params().n_sc_s and
                       reserve <= params().r_max * n_sc * rate and
                       o_msg.ack <> MintedRC
                    then
                      ((n_sc * rate * params().r_max - reserve ) * 100 + d_price - 1 ) div d_price
                    else
                      pre max_bound;

  -- detects rising edge on max bound definition
  bound_defined = false -> (pre max_bound = -1 and max_bound >= 0);

  -- N_SC remains constant once max bound is defined
  constant_sc = false -> (bound_defined or pre constant_sc ) and n_sc = p_sc;

  -- rate remains constant once max_bound is defined
  constant_rate = false -> (bound_defined or pre constant_rate ) and rate = pre rate;

  -- No Selling of RC once max bound is defined
  no_rc_selling = false -> (bound_defined or pre no_rc_selling ) and p_rc <= n_rc;


  d_price = price_rc(1, reserve, n_sc, n_rc, rate) * (100 + params().fee);
  prev_d_price = price_rc(1, p_reserve, n_sc, p_rc, rate) * (100 + params().fee);

  -- Stability Property
  check "THEOREM_8"
    (constant_rate and -- exchange rate remains constant
      constant_sc and -- N_SC > 0 and remains constant
      no_rc_selling and -- no selling of RC once max bound fixed
      o_msg.ack = MintedRC ) => i_msg.qnt <= max_bound;

  -- Lemmas
  check (constant_rate and constant_sc and no_rc_selling ) =>
    ((n_sc * rate * params().r_max - reserve ) * 100 + d_price - 1 ) div d_price <= max_bound;
```

```
  check (constant_rate and constant_sc and no_rc_selling ) => reserve <= n_sc * rate * params().r_max;
  check (constant_rate and constant_sc and no_rc_selling ) => n_sc >= params().n_sc_s;
  check (constant_rate and constant_sc and no_rc_selling ) => n_sc > 0;

  check (constant_rate and constant_sc and no_rc_selling ) => rate > 0;
  check (constant_rate and constant_sc and no_rc_selling ) => max_bound >= 0;
  check (constant_rate and constant_sc and no_rc_selling ) => (p_rc > 0 and n_rc > 0 );

  check p_reserve >= 0 and p_sc >= 0 and p_rc >= 0;

  check (constant_rate and constant_sc and no_rc_selling ) => reserve >= p_reserve;
  check (constant_rate and constant_sc and no_rc_selling ) => p_sc = n_sc;
  check (constant_rate and constant_sc and no_rc_selling ) => n_rc >= p_rc;


  check (constant_rate and constant_sc and no_rc_selling and reserve <> p_reserve ) => o_msg.ack = MintedRC;

  check (constant_rate and constant_sc and no_rc_selling ) =>
      n_sc * rate * params().r_max - reserve <= n_sc * rate * params().r_max - p_reserve;

  check (constant_rate and constant_sc and no_rc_selling ) => d_price div prev_d_price >= 1;
  check (constant_rate and constant_sc and no_rc_selling ) => d_price >= prev_d_price;
  check (constant_rate and constant_sc and no_rc_selling ) =>
     price_rc(1, reserve, n_sc, n_rc, rate) >= price_rc(1, p_reserve, n_sc, p_rc, rate);

  check prev_d_price = price_rc(1, p_reserve, n_sc, p_rc, rate) * (100 + params().fee);
  check d_price = price_rc(1, reserve, n_sc, n_rc, rate) * (100 + params().fee);

  check price_rc(1, reserve, n_sc, n_rc, rate) >= 0;
  check price_rc(1, p_reserve, p_sc, p_rc, rate) >= 0;
  check params().fee > 0;
  check params().r_max > 1;

tel
```

Listing 15: Theorem9.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

node Theorem9 (i_msg: InputMsg; rate: RateType)  returns (o_msg: OutputMsg)

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve :ReserveType;
     p_sc :N_Type;
     p_rc :N_Type;

     order_once: bool;
 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  -- Remains true once at least on order is successful.
  order_once = false -> pre order_once or (o_msg.ack <> Error and o_msg.ack <> NoReply and i_msg.qnt <> 0);

  -- Stability Property
  -- Reserve is always greater than zero once at least one buying/ selling order is successful
  check "THEOREM_9" order_once => reserve > 0;

  -- Lemmas
  check p_reserve >= 0 and p_sc >= 0 and p_rc >= 0;
  check p_rc > 0 => p_reserve > 0;
  check p_sc > 0 => p_reserve div p_sc > 0;


tel
```

Listing 16: Theorem10.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

node Theorem10 (i_msg: InputMsg; rate: RateType)  returns (o_msg: OutputMsg)

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve :ReserveType;
     p_sc :N_Type;
     p_rc :N_Type;

     rate_price :ReserveType;
     sc_price_pc :ReserveType;

 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  rate_price = i_msg.qnt * rate;
  sc_price_pc = computeFee(rate_price);

  -- THEOREM 10 :SCs are always bought for 1 PC in BCs from bank.
  check "THEOREM_10" o_msg.ack = MintedSC => o_msg.price = sc_price_pc;

  -- Lemmas
  check o_msg.ack = MintedSC => price_sc(p_reserve, p_sc, rate) = rate;

tel
```

Listing 17: Theorem11.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

node Theorem11 (i_msg: InputMsg; rate: RateType)  returns (o_msg: OutputMsg)

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve :ReserveType;
     p_sc :N_Type;
     p_rc :N_Type;

     rate_price :ReserveType;
     sc_price_pc :ReserveType;

 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  rate_price = i_msg.qnt * rate;
  sc_price_pc = computeFee(rate_price);

  -- THEOREM 11 :When reserve ratio >= 1, SCs are always bough for 1 PC in BCs.
  check "THEOREM_11" (o_msg.ack = RedeemedSC and p_reserve >= p_sc * rate ) => o_msg.price = sc_price_pc;

tel
```

Listing 18: Theorem12.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

node Theorem12 (i_msg: InputMsg; rate: RateType)  returns (o_msg: OutputMsg)

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve :ReserveType;
     p_sc :N_Type;
     p_rc :N_Type;

 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  -- THEOREM 12 :RCs can be sold only when E(R, N_SC) > 0.
  check "THEOREM_12" o_msg.ack = RedeemedRC => equity(p_reserve, p_sc, rate) > 0;

  -- Lemmas
  check p_reserve >= 0 and p_sc >= 0 and p_rc >= 0;
  check p_rc > 0 => p_reserve > 0;
  check p_sc > 0 => p_reserve div p_sc > 0;

tel
```

Listing 19: Theorem13_to_15.lus File

```
include "StableCoin.lus"
include "Theorem_Base.lus"

node Theorem13_to_15 (i_msg: InputMsg; rate: RateType)  returns (o_msg: OutputMsg)

 var reserve: ReserveType;
     n_sc: N_Type;
     n_rc: N_Type;

     p_reserve :ReserveType;
     p_sc :N_Type;
     p_rc :N_Type;

 let

  --%MAIN;
  p_reserve, p_sc, p_rc, reserve, n_sc, n_rc, o_msg = StableCoin(i_msg, rate);

  -- Assuming Parameter Constraints
  assert ParameterConstraints ();

  -- Stability Properties
  check "THEOREM_13" reserve >= 0 and n_sc >= 0 and n_rc >= 0;
  check "THEOREM_14" n_rc > 0 => reserve > 0;
  check "THEOREM_15" n_sc > 0 => reserve > 0;

  -- Lemmas
  check p_reserve >= 0 and p_sc >= 0 and p_rc >= 0;
  check p_rc > 0 => p_reserve > 0;
  check p_sc > 0 => p_reserve div p_sc > 0;

  check o_msg.ack = RedeemedSC => rate > 0;
  check o_msg.ack = RedeemedSC => reserve > (p_reserve div p_sc) * (p_sc + i_msg.qnt);
  check o_msg.ack = RedeemedSC => p_sc + i_msg.qnt >= 0;
  check o_msg.ack = RedeemedSC => p_sc > 0;
  check o_msg.ack = RedeemedSC => (p_reserve div p_sc) > 0;

  check o_msg.ack = RedeemedRC => p_sc * rate >= 0;
  check o_msg.ack = RedeemedRC => p_reserve > 0;
  check o_msg.ack = RedeemedRC => reserve > ((p_reserve - p_sc * rate) div p_rc ) * (p_rc + i_msg.qnt);
  check o_msg.ack = RedeemedRC => p_rc + i_msg.qnt >= 0;
  check o_msg.ack = RedeemedRC => p_rc > 0;

tel
```

-----BEGIN PGP MESSAGE-----

```
hQEMA3mYtjIcCbbOAQf/ci71Krwdlkd3ZzsoAkZdMKQYseQxI1YAxqEeshvnqcDJ
aCbKf5YXMdxEjumXW9EvTzjG8PnkLfviN9tpPWxmTujXOJcoiBkZ/CGSeO2msuhd
tC+W9xn5z0+Z7p4HZPRFfiZ4ZmlBow77JFRMcqqf/OG/eZ+7kkigECZzs9bBE4b+
kG3EF1268D69JVu6q1eiyybF6U9D8OmDK4kRA3LuPyzqrE8sfdUDW8U0AuSUIsfY
YW8Jy6TdgikEDq6NzSS6X/jcjBqO3XM3HJvA7812XIooNSFKyoXWcJHTe9J+zXHe
krtnsI8Q/5US/C3FK/fX77k2gIKNgO81YZQrQztBLNLA7AGYfZvDVc7WGliw8UJn
wPlJR6DUuOLvo1laej7OhYHvLqFQFoLjSOKzM77uaftXWpEIc+ETY4eD6mEthBnz
kLraN9ZRhu3u8yunuIpBrSIpX5OEXKwOA/aUhTvuR8ZH9GEMVyL3tpqxNbUFbSTp
lXElcCeE7TkcORLpV8E/NzPFk2TlazRl8iHGepLR7vw1DT66H3xN4+qcI2FwtEs8
5rDKq6NDkhPKNC8IsYcdtmNzdy1CK2KbD3lBrEB3io1DvU2sD2Oa+NEs18MeMknY
YW5GZ75y8W3YfL5/SOpVB6RqanpyZIHcq2P6wMYojpPhnPc50OchOYGdLHeKwlJL
uEC3dEOMg3LBtkBWQTiNXF/Nk02WS4YmTZJROI7oZnlGuoIZYDkj3r/WaJHTZd7K
7A7BhxJSiDbj5uC2F/cpBduHV6mtdpByv87wXWF1JMNFWcVow0x99gg1j/aCCToE
fjO1/sqpsxoo4jNqr9kREqpVYghoayxNyOfQYfew/aaPOp382gfKEbDeEhNfMV+f
Gq3MDJfGuUmfFyDnGLvr/fZu8o83elb446mwtBj+
=cj7k
-----END PGP MESSAGE-----
```