Soroban

nano

April 1, 2024

Contents

```
theory SorobanAMM
  imports Complex-Main HOL−Statespace.StateSpaceSyntax
begin

sledgehammer-params [timeout=300]

definition deposit-amounts where
  deposit-amounts desired-a min-a desired-b min-b reserve-a reserve-b ≡
    if (reserve-a = 0 ∧ reserve-b = 0)
    then Some (desired-a, desired-b)
    else let amount-b = (desired-a∗reserve-b)/reserve-a in
      if (amount-b ≤ desired-b)
      then
        if (amount-b < min-b)
        then None
        else Some (desired-a, amount-b)
      else let amount-a = (desired-b∗reserve-a)/reserve-b in
        if (amount-a > desired-a ∨ amount-a < min-a)
        then None
        else Some (amount-a, desired-b)

lemma deposit-amounts-property:
  assumes da ≥ (0::real) and db ≥ 0 and ma ≥ 0 and mb ≥ 0 and ra ≥ a and
rb ≥ 0
    and ma ≤ da and mb ≤ db
    and deposit-amounts da ma db mb ra rb = Some (a, b)
  shows ma ≤ a and mb ≤ b and a ≤ da and b ≤ db and (ra ≠ 0 ∨ rb ≠ 0)
⟶ a∗rb = b∗ra
    and (ra = 0 ∧ rb = 0) ⟶ a = da ∧ b = db and (ra ≠ 0 ∧ rb ≠ 0) ⟶ a/ra
= b/rb
    and (ra+a)∗rb = (rb+b)∗ra
proof −
  show ma ≤ a
    using assms unfolding deposit-amounts-def
    by (simp split:if-splits add:Let-def; force)
  show mb ≤ b
```

using *assms* **unfolding** *deposit-amounts-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show $a \leq da$
using *assms* **unfolding** *deposit-amounts-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show $b \leq db$
using *assms* **unfolding** *deposit-amounts-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show $(ra \neq 0 \vee rb \neq 0) \longrightarrow a*rb = b*ra$
using *assms* **unfolding** *deposit-amounts-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show $(ra = 0 \wedge rb = 0) \longrightarrow a = da \wedge b = db$
using *assms* **unfolding** *deposit-amounts-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show $(ra \neq 0 \wedge rb \neq 0) \longrightarrow a/ra = b/rb$
using *assms* **unfolding** *deposit-amounts-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show $(ra+a)*rb = (rb+b)*ra$
by (*metis* ‹$ra \neq 0 \vee rb \neq 0 \longrightarrow a * rb = b * ra$› *add-cancel-left-right mult.commute ring-class.ring-distribs*(*1*))
qed

definition *deposit-amounts-2*
— Now we round to integers
where
deposit-amounts-2 desired-a min-a desired-b min-b reserve-a reserve-b ≡
if (*reserve-a* = *0* ∧ *reserve-b* = *0*)
then Some (*desired-a*, *desired-b*)
else let amount-b = ⌊(*desired-a*∗*reserve-b*)/*reserve-a*⌋ *in*
if (*amount-b* ≤ *desired-b*)
then
if (*amount-b* < *min-b*)
then None
else Some (*desired-a*, *amount-b*)
else let amount-a = ⌊(*desired-b*∗*reserve-a*)/*reserve-b*⌋ *in*
if (*amount-a* > *desired-a* ∨ *amount-a* < *min-a*)
then None
else Some (*amount-a*, *desired-b*)

lemma *deposit-amounts-2-property*:
assumes $da \geq (0::real)$ **and** $db \geq 0$ **and** $ma \geq 0$ **and** $mb \geq 0$ **and** $ra \geq a$ **and** $rb \geq 0$
and $ma \leq da$ **and** $mb \leq db$
and *deposit-amounts-2 da ma db mb ra rb* = *Some* (*a*, *b*)
shows $ma \leq a$ **and** $mb \leq b$ **and** $a \leq da$ **and** $b \leq db$
and $(ra = 0 \wedge rb = 0) \longrightarrow a = da \wedge b = db$
and $(ra \neq 0 \wedge rb \neq 0) \longrightarrow ((b/rb \leq a/ra \wedge a/ra \leq b/rb + 1) \vee (a/ra \leq b/rb \wedge b/rb \leq a/ra + 1))$
proof −

show *ma* $\leq$ *a*
using *assms* **unfolding** *deposit-amounts-2-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show *mb* $\leq$ *b*
using *assms* **unfolding** *deposit-amounts-2-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show *a* $\leq$ *da*
using *assms* **unfolding** *deposit-amounts-2-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show *b* $\leq$ *db*
using *assms* **unfolding** *deposit-amounts-2-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show $(ra = 0 \wedge rb = 0) \longrightarrow a = da \wedge b = db$
using *assms* **unfolding** *deposit-amounts-2-def*
by (*simp split*:*if-splits add*:*Let-def*; *force*)
show $(ra \neq 0 \wedge rb \neq 0) \longrightarrow ((b/rb \leq a/ra \wedge a/ra \leq b/rb + 1) \vee (a/ra \leq b/rb \wedge b/rb \leq a/ra + 1))$
using *assms* **unfolding** *deposit-amounts-2-def*
apply (*simp split*:*if-splits add*:*Let-def*)
apply (*smt* (*verit*, *del-insts*) *floor-divide-real-eq-div floor-of-int nonzero-mult-div-cancel-right of-int-floor-le of-int-pos real-of-int-div3 times-divide-eq-left*)
apply (*smt* (*verit*) *eq-divide-imp floor-divide-of-int-eq floor-divide-real-eq-div floor-le-zero floor-less-zero le-divide-eq-1-pos of-int-floor-le of-int-pos real-of-int-floor-add-one-ge times-divide-eq-left*)
done
qed

definition *new-total-shares* **where**
new-total-shares old-a new-a old-b new-b old-shares ≡
if (*old-a* > *0* ∧ *old-b* > *0*)
then
let shares-a = (*new-a*∗*old-shares*)/*old-a*;
shares-b = (*new-b*∗*old-shares*)/*old-b in*
min shares-a shares-b
else sqrt (*new-a*∗*new-b*)

lemma *deposit-lemma*:
assumes $da \geq (0::real)$ **and** $db \geq 0$ **and** $ma \geq 0$ **and** $mb \geq 0$ **and** $ra \geq a$ **and** $rb \geq 0$
and $ma \leq da$ **and** $mb \leq db$ **and** $s \geq 0$ **and** $(ra = 0) \longleftrightarrow (rb = 0)$ — note we need this invariant
and *deposit-amounts da ma db mb ra rb* = *Some* (*a*, *b*)
and *new-total-shares ra* (*ra*+*a*) *rb* (*rb*+*b*) *s* = *ns*
shows *ra*∗*ns* = (*ra*+*a*)∗*s*
using *assms* **unfolding** *new-total-shares-def*
apply (*simp split*:*if-splits option.splits add*:*Let-def split-def*)
apply (*smt* (*verit*, *best*) *deposit-amounts-property*(*8*) *mult.commute nonzero-eq-divide-eq times-divide-eq-left*)
apply (*smt* (*verit*, *best*) *deposit-amounts-property*(*1*) *mult-not-zero*)

done

The attacker buys shares and then sells them back in two steps. We want to check that no money is made by the attacker.

```
statespace 'addr system =
  reserve-a :: real
  reserve-b :: real
  total-shares :: real
  attacker-shares :: real
  attacker-a :: real
  attacker-b :: real

definition (in system) init where
  init s ≡
      s·reserve-a = 0
    ∧ s·reserve-b = 0
    ∧ s·total-shares = 0
    ∧ s·attacker-shares = 0
    ∧ s·attacker-a ≥ 0
    ∧ s·attacker-b ≥ 0

definition (in system) deposit where
  deposit a b ma mb s s' ≡
    a ≥ 0 ∧ b ≥ 0 ∧ ma ≤ a ∧ mb ≤ b
    ∧ (s·attacker-a) ≥ a
    ∧ (s·attacker-b) ≥ b
    ∧ (s'·attacker-a) = (s·attacker-a) − a
    ∧ (s'·attacker-b) = (s·attacker-b) − b
    ∧ (let amounts = deposit-amounts a ma b mb (s·reserve-a) (s·reserve-b) in
       (case amounts of
         None ⇒ False
       | Some (amount-a, amount-b) ⇒
            (s'·attacker-a) = (s·attacker-a) − amount-a
          ∧ (s'·attacker-b) = (s·attacker-b) − amount-b
          ∧ (let new-a = (s·reserve-a)+amount-a;
                 new-b = (s·reserve-b)+amount-b;
               new-total-shares = new-total-shares (s·reserve-a) new-a (s·reserve-b)
new-b (s·total-shares)
             in
               (s'·reserve-a) = new-a
             ∧ (s'·reserve-b) = new-b
             ∧ (s'·total-shares) = new-total-shares
          ∧ (s'·attacker-shares) = (s·attacker-shares)+new-total-shares−(s·total-shares))))

definition (in system) withdraw where
  withdraw shrs min-a min-b s s' ≡
      (s·attacker-shares) ≥ shrs
    ∧ (s'·attacker-shares) = (s·attacker-shares)−shrs
    ∧ (s'·total-shares) = (s·total-shares)−shrs — We burn the shares
```

$\wedge$ (*let* *out-a* = (*shrs*∗(*s*·*reserve-a*))/(*s*·*total-shares*);
out-b = (*shrs*∗(*s*·*reserve-b*))/(*s*·*total-shares*) *in*
out-a ≥ *min-a* ∧ *out-b* ≥ *min-b*
∧ (*s*′·*attacker-a*) = (*s*·*attacker-a*)+*out-a*
∧ (*s*′·*attacker-b*) = (*s*·*attacker-b*)+*out-b*
∧ (*s*′·*reserve-a*) = (*s*·*reserve-a*)−*out-a*
∧ (*s*′·*reserve-b*) = (*s*·*reserve-b*)−*out-b*)

lemma (**in** *system*) *deposit-withdraw*:
assumes *deposit a b ma mb s s*′ **and** *withdraw* (*s*′·*attacker-shares*) *min-a min-b* *s*′ *s*′′
and *s*·*attacker-shares* = *0* **and** *s*′′·*attacker-shares* = *0*
shows *s*′′·*attacker-a* ≤ *s*·*attacker-a*
using *assms*
unfolding *deposit-def withdraw-def*
apply (*simp split*:*if-splits option.splits add*:*Let-def split-def*)
apply *auto*
oops — We are going to need more lemmas for this

end