AMM-Zhang

nano

October 3, 2023

Contents

theory *AMM-Zhang*
imports *HOL.Real*
begin

This is an attempt at translating the AMM formalization of Zhang et al. https://github.com/runtimeverification/verified-smart-contracts/blob/uniswap/uniswap/x-y-k.pdf in Isabelle/HOL

We formalize a basic constant-product liquidity pool. We define an add-liquidity and a remove-liquidity operation on a pool whose token amounts are reals. We also define the code versions of those operations, where amounts are rounded to obtain integers. Rounding the wrong way can allow an attacker to drain the pool. We would like to prove that this cannot happen.

1 Pool state

record *pool-state* =
— A liquidity pool between tokens x and y; l is the token representing pool shares
x :: *real*
y :: *real*
l :: *real*

definition k **where** $k\ p \equiv (x\ p){*}(y\ p)$

2 Adding liquidity

definition *add-liquidity-spec* **where**
  add-liquidity-spec $p\ \Delta x \equiv$ *let* $\alpha = \Delta x\ /\ x\ p$
    in $(\!|x = (1{+}\alpha){*}(x\ p),\ y = (1{+}\alpha){*}(y\ p),\ l = (1{+}\alpha){*}(l\ p)|\!)$

definition *add-liquidity-code-spec* **where**
  add-liquidity-code-spec $p\ \Delta x \equiv$
    $(\!|x = x\ p + \Delta x,$
     $y = y\ p + \lfloor \Delta x{*}(y\ p)\ /\ x\ p \rfloor + 1,$ — note we need to round in favor of the pool or the pool can be drained
     $l = l\ p + \lfloor \Delta x{*}(l\ p)\ /\ x\ p \rfloor|\!)$

lemma *l1*:$(1 + \Delta x\ /\ x\ p) * a = a + \Delta x * a\ /\ x\ p$ **for** a
  — this simple lemma helps automated provers a lot
  by *algebra*

lemma *add-liquidity-properties*:
  — properties of add-liquidity
  fixes $p\ \Delta x$
  assumes $x\ p > 0$ **and** $y\ p > 0$ **and** $l\ p > 0$
    and $\Delta x \geq 0$
  defines $p' \equiv$ *add-liquidity-spec* $p\ \Delta x$
    and $p'' \equiv$ *add-liquidity-code-spec* $p\ \Delta x$
  shows $x\ p \leq x\ p'$ **and** $x\ p' = x\ p''$
    and $y\ p \leq y\ p'$ **and** $y\ p' < y\ p''$ **and** $y\ p'' \leq y\ p' + 1$
    and $l\ p' - 1 < l\ p''$ **and** $l\ p'' \leq l\ p'$ **and** $l\ p \leq l\ p''$ **and** $l\ p \leq l\ p'$
    and $k\ p \leq k\ p'$ **and** $k\ p' < k\ p''$
    and $k\ p'/k\ p = (l\ p'/l\ p)^2$
    and $(l\ p''/l\ p)^2 < k\ p''/k\ p$
    and $x\ p\ /\ l\ p = x\ p'\ /\ l\ p'$
    and $y\ p\ /\ l\ p = y\ p'\ /\ l\ p'$
proof $-$
  show $x\ p \leq x\ p'$
     by (*smt* (*verit*) *add-liquidity-spec-def assms*(*1*) *assms*(*4*) *divide-nonneg-pos eq-divide-imp ext-inject le-divide-eq-1 p'-def surjective*)
  show $x\ p' = x\ p''$
   using *assms*(*1*−*4*) **unfolding** *p'-def p''-def add-liquidity-spec-def add-liquidity-code-spec-def Let-def*
   by (*auto*; *metis add-divide-distrib eq-divide-eq-1 nonzero-eq-divide-eq order-less-irrefl*)
  show $y\ p \leq y\ p'$
    by (*smt* (*verit*, *del-insts*) *add-liquidity-spec-def assms*(*1*) *assms*(*2*) *assms*(*4*) *divide-nonneg-pos l1 p'-def select-convs*(*2*) *split-mult-pos-le*)
  show $y\ p' < y\ p''$
   by (*simp add*: *add-liquidity-code-spec-def add-liquidity-spec-def l1 p''-def p'-def*)
  show $y\ p'' \leq y\ p' + 1$
   by (*simp add*: *add-liquidity-code-spec-def add-liquidity-spec-def l1 p''-def p'-def*)
  show $l\ p' - 1 < l\ p''$
   by (*simp add*: *add-liquidity-code-spec-def add-liquidity-spec-def l1 p''-def p'-def*)

show $l\ p'' \leq l\ p'$
by (*simp add*: *add-liquidity-code-spec-def add-liquidity-spec-def l1 p''-def p'-def*)
show $l\ p \leq l\ p''$
by (*smt* (*verit*) *add-liquidity-code-spec-def assms*(*1*) *assms*(*3*) *assms*(*4*) *divide-nonneg-pos of-int-0-le-iff p''-def select-convs*(*3*) *split-mult-pos-le zero-le-floor*)
show $l\ p \leq l\ p'$
using ‹$l\ p \leq l\ p''$› ‹$l\ p'' \leq l\ p'$› **by** *force*
show $k\ p \leq k\ p'$
by (*smt* (*verit*, *ccfv-SIG*) ‹$x\ p \leq x\ p'$› ‹$y\ p \leq y\ p'$› *add-liquidity-spec-def assms*(*1*) *assms*(*2*) *k-def mult.left-commute mult-le-less-imp-less nonzero-mult-div-cancel-right p'-def select-convs*(*1*) *select-convs*(*2*) *times-divide-eq-right*)
show $k\ p' < k\ p''$
by (*metis* ‹$x\ p \leq x\ p'$› ‹$x\ p' = x\ p''$› ‹$y\ p' < y\ p''$› *assms*(*1*) *dual-order.strict-trans1 k-def mult-less-cancel-left-pos*)
show $k\ p'/k\ p = (l\ p'/l\ p)^2$
using *assms*(*1*−*4*) **unfolding** *p'-def add-liquidity-spec-def Let-def k-def*
by (*simp*; *algebra*)
show $(l\ p''/l\ p)^2 < k\ p''/k\ p$
proof −
have $((l\ p + \lfloor \Delta x * l\ p\ /\ x\ p \rfloor)\ /\ l\ p)^2 \leq ((l\ p + \Delta x * l\ p\ /\ x\ p)\ /\ l\ p)^2$
proof −
have $((l\ p + \lfloor \Delta x * l\ p\ /\ x\ p \rfloor)\ /\ l\ p) \leq ((l\ p + \Delta x * l\ p\ /\ x\ p)\ /\ l\ p)$
by (*simp add*: *assms*(*3*) *pos-le-divide-eq*)
thus *?thesis*
using *assms*(*1*,*3*,*4*) **by** *fastforce*
qed
moreover
have $(x\ p + \Delta x) * (y\ p + \lfloor \Delta x * y\ p\ /\ x\ p \rfloor + 1)\ /\ (x\ p * y\ p)$
$> (x\ p + \Delta x) * (y\ p + \Delta x * y\ p\ /\ x\ p)\ /\ (x\ p * y\ p)$
by (*smt* (*verit*) *assms*(*1*) *assms*(*2*) *assms*(*4*) *divide-strict-right-mono less-floor-iff mult-pos-pos mult-strict-left-mono*)
ultimately
show *?thesis*
by (*smt* (*verit*, *best*) ‹$k\ p'\ /\ k\ p = (l\ p'\ /\ l\ p)^2$› ‹$k\ p' < k\ p''$› *add-liquidity-code-spec-def add-liquidity-spec-def assms*(*1*,*2*) *divide-le-cancel k-def l1 p''-def p'-def select-convs*(*3*) *split-mult-pos-le*)
qed
show $x\ p\ /\ l\ p = x\ p'\ /\ l\ p'$
by (*smt* (*verit*, *del-insts*) ‹$x\ p \leq x\ p'$› *add-liquidity-spec-def assms*(*1*) *divide-pos-pos nonzero-eq-divide-eq nonzero-mult-divide-mult-cancel-left p'-def select-convs*(*1*) *select-convs*(*3*))
show $y\ p\ /\ l\ p = y\ p'\ /\ l\ p'$
by (*smt* (*verit*, *ccfv-threshold*) ‹$y\ p \leq y\ p'$› *add-liquidity-spec-def assms*(*2*) *divide-eq-0-iff nonzero-eq-divide-eq nonzero-mult-divide-mult-cancel-left p'-def select-convs*(*2*) *select-convs*(*3*))
qed

3 Removing liquidity

definition *remove-liquidity-spec* **where**
  $\textit{remove-liquidity-spec}\ p\ \Delta l \equiv \textit{let}\ \alpha = \Delta l\ /\ l\ p$
    $\textit{in}\ (\!|x = (1{-}\alpha){*}(x\ p),\ y = (1{-}\alpha){*}(y\ p),\ l = (1{-}\alpha){*}(l\ p)|\!)$

definition *remove-liquidity-code-spec* **where**
  $\textit{remove-liquidity-code-spec}\ p\ \Delta l \equiv$
    $(\!|x = x\ p - \lfloor \Delta l{*}(x\ p)\ /\ l\ p\rfloor,$
     $y = y\ p - \lfloor \Delta l{*}(y\ p)\ /\ l\ p\rfloor,$
     $l = l\ p - \Delta l|\!)$

lemma $\textit{l2}{:}(1 - \Delta l\ /\ l\ p) * a = a - \Delta l * a\ /\ l\ p$ **for** a
  — this simple lemma helps automated provers a lot
  by *algebra*

lemma *remove-liquidity-properties*:
  — properties of remove-liquidity
  fixes $p\ \Delta l$
  assumes $x\ p > 0$ **and** $y\ p > 0$ **and** $l\ p > 0$
    and $\Delta l < l\ p$ **and** $0 \le \Delta l$
  defines $p' \equiv \textit{remove-liquidity-spec}\ p\ \Delta l$
    and $p'' \equiv \textit{remove-liquidity-code-spec}\ p\ \Delta l$
  shows $x\ p' \le x\ p$ **and** $x\ p' \le x\ p''$
    and $y\ p' \le y\ p$ **and** $y\ p' \le y\ p''$
    and $l\ p' = l\ p''$ **and** $l\ p' \le l\ p$
    and $k\ p' \le k\ p''$ **and** $k\ p' \le k\ p$
    and $k\ p'/k\ p = (l\ p'/l\ p)^2$
    and $(l\ p''/l\ p)^2 \le k\ p''/k\ p$
    and $x\ p\ /\ l\ p = x\ p'\ /\ l\ p'$
    and $y\ p\ /\ l\ p = y\ p'\ /\ l\ p'$
proof −
  show $x\ p' \le x\ p$
    by (*smt* (*verit*, *del-insts*) *assms*(*1*) *assms*(*3*) *assms*(*5*) *divide-divide-eq-right divide-eq-0-iff divide-nonneg-pos l2 p′-def remove-liquidity-spec-def select-convs*(*1*))
  show $x\ p' \le x\ p''$
    by (*simp add*: *assms*(*7*) *l2 p′-def remove-liquidity-code-spec-def remove-liquidity-spec-def*)
  show $y\ p' \le y\ p$
    by (*smt* (*verit*, *ccfv-SIG*) *assms*(*2*) *assms*(*3*) *assms*(*5*) *divide-nonneg-pos l2 p′-def remove-liquidity-spec-def select-convs*(*2*) *split-mult-pos-le*)
  show $y\ p' \le y\ p''$
    by (*simp add*: *l2 p′′-def p′-def remove-liquidity-code-spec-def remove-liquidity-spec-def*)
  show $l\ p' = l\ p''$
    by (*metis assms*(*3*) *l2 nonzero-mult-div-cancel-right not-less-iff-gr-or-eq p′′-def p′-def pool-state.select-convs*(*3*) *remove-liquidity-code-spec-def remove-liquidity-spec-def*)
  show $l\ p' \le l\ p$
    by (*simp add*: ‹$l\ p' = l\ p''$› *assms*(*5*) *p′′-def remove-liquidity-code-spec-def*)
  show $k\ p' \le k\ p$
    by (*smt* (*verit*) ‹$x\ p' \le x\ p$› ‹$y\ p' \le y\ p$› *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*)

divide-less-eq-1-pos k-def mult-less-cancel-left-pos mult-less-iff1 mult-pos-pos p'*-def remove-liquidity-spec-def select-convs*(2))
  show $k\ p' \leq k\ p''$
    by (*smt* (*verit*) ‹$x\ p' \leq x\ p''$› ‹$y\ p' \leq y\ p''$› *assms*(1) *assms*(2) *assms*(3) *assms*(4) *divide-less-eq-1-pos k-def mult.commute mult-less-cancel-left-pos mult-pos-pos* p'*-def pool-state.select-convs*(1) *pool-state.select-convs*(2) *remove-liquidity-spec-def*)
  show $k\ p'/k\ p = (l\ p'/l\ p)^2$
   by (*smt* (*verit*, *ccfv-SIG*) *ab-semigroup-mult-class.mult-ac*(1) *assms*(1) *assms*(2) *assms*(3) *k-def mult.left-commute mult-pos-pos nonzero-eq-divide-eq* p'*-def power2-eq-square remove-liquidity-spec-def select-convs*(1) *select-convs*(2) *select-convs*(3))
  show $(l\ p''/l\ p)^2 \leq k\ p''/k\ p$
    by (*metis* ‹$k\ p'\ /\ k\ p = (l\ p'\ /\ l\ p)^2$› ‹$k\ p' \leq k\ p''$› ‹$l\ p' = l\ p''$› *assms*(1) *assms*(2) *divide-right-mono k-def less-le-not-le split-mult-pos-le*)
  show $x\ p\ /\ l\ p = x\ p'\ /\ l\ p'$
    by (*smt* (*verit*) *assms*(4) *divide-divide-eq-right eq-divide-eq-1 mult.commute nonzero-mult-div-cancel-left* p'*-def remove-liquidity-spec-def select-convs*(1) *select-convs*(3))
  show $y\ p\ /\ l\ p = y\ p'\ /\ l\ p'$
   by (*smt* (*verit*, *best*) ‹$x\ p\ /\ l\ p = x\ p'\ /\ l\ p'$› *assms*(1) *divide-divide-eq-left divide-eq-0-iff nonzero-mult-div-cancel-left* p'*-def remove-liquidity-spec-def select-convs*(2) *select-convs*(3))
qed

4 No free money

Here we want to prove that, no matter what sequence of operations one applies, withdrawing all the liquidity obtained leaves the pool with at least the same amount of tokens it started from. We could formalize executions as lists, inductive invariants, etc.

definition *inv* **where**
  $inv\ p_0\ p \equiv l\ p \geq l\ p_0 \wedge ($
    let $p' = remove\text{-}liquidity\text{-}spec\ p\ (l\ p - l\ p_0)$
    in $x\ p' = x\ p_0 \wedge y\ p' = y\ p_0)$

definition *pool-ne* **where**
  — non-empty pool
  $pool\text{-}ne\ p \equiv x\ p > 0 \wedge y\ p > 0 \wedge l\ p > 0$

lemma *l3*:
  — if the ratio x to l is the same as x' to l', then removing liquidity l' minus l results in balance x
  fixes $x\ l\ x'\ l'$ :: *real*
  assumes $x/l = x'/l'$ **and** $l>0$ **and** $x>0$
  shows $x'*(1-(l'-l)/l') = x$
 by (*metis* (*no-types*, *opaque-lifting*) *add.right-neutral add-diff-cancel assms diff-diff-eq diff-divide-distrib divide-divide-eq-right divide-eq-0-iff eq-divide-eq-1 minus-diff-eq mult-1 nonzero-eq-divide-eq order-less-irrefl times-divide-eq-left*)

lemma *l4*:

```
  fixes p0 p
  assumes inv p0 p and pool-ne p0 and pool-ne p
  shows x p / l p = x p0 / l p0 and y p / l p = y p0 / l p0
proof −
  define Δl where Δl ≡ l p − l p0
  define p' where p' ≡ remove-liquidity-spec p Δl
  have 1:l p0 = l p'
  proof −
    have (1 − (x' − x)/x')*x' = x if x' ≠ 0 for x x' :: real
      by (simp add: diff-divide-distrib that)
    thus ?thesis
      unfolding p'-def remove-liquidity-spec-def Let-def
      by (metis Δl-def assms(3) order-less-irrefl pool-ne-def select-convs(3))
  qed
  have 2:Δl ≥ 0
    using AMM-Zhang.inv-def Δl-def assms(1) by auto
  have 3:Δl < l p
    using Δl-def assms(2) pool-ne-def by auto
  show x p / l p = x p0 / l p0
  proof −
    have x p / l p = x p' / l p'
     using 2 3 assms(3) p'-def pool-ne-def remove-liquidity-properties(11) by blast
    thus ?thesis using 1
      by (metis AMM-Zhang.inv-def Δl-def assms(1) p'-def)
  qed
  show y p / l p = y p0 / l p0
  proof −
    have y p / l p = y p' / l p'
     using 2 3 assms(3) p'-def pool-ne-def remove-liquidity-properties(12) by blast
    thus ?thesis using 1
      by (metis AMM-Zhang.inv-def Δl-def assms(1) p'-def)
  qed
qed

lemma l5:
  fixes p0 p p' Δx
  assumes inv p0 p and l p' ≥ l p0 and pool-ne p0 and pool-ne p
    and x p' / l p' = x p / l p and y p' / l p' = y p / l p
  shows inv p0 p'
proof −
  have l p ≥ l p0
    using inv-def assms(1)
    by blast
  define p'' where p'' ≡ remove-liquidity-spec p' (l p' − l p0)
  have x p0 / l p0 =  x p' / l p'
    using ‹l p0 ≤ l p› assms(1) assms(3) assms(4) assms(5) l4(1) by fastforce
  hence x p0 = x p'' using l3 unfolding p''-def remove-liquidity-spec-def Let-def
    by (metis assms(3) mult.commute pool-ne-def select-convs(1))
  have y p0 / l p0 =  y p' / l p'
```

```
    using ‹l p0 ≤ l p› assms(1) assms(3) assms(4) assms(6) l4(2) by fastforce
  hence y p0 = y p'' using l3 unfolding p''-def remove-liquidity-spec-def Let-def
    by (metis assms(3) mult.commute pool-ne-def select-convs(2))
  show ?thesis
    using AMM-Zhang.inv-def ‹x p0 = x p''› ‹y p0 = y p''› assms(2) p''-def by
fastforce
qed

lemma inv-add-okay:
  fixes p0 p p' Δx
  assumes inv p0 p and 0 ≤ Δx and pool-ne p0 and pool-ne p
  defines p' ≡ add-liquidity-spec p Δx
  shows inv p0 p'
proof −
  have x p' / l p' = x p / l p and y p' / l p' = y p / l p
    by (metis add-liquidity-properties(14) assms(2) assms(4) p'-def pool-ne-def
    , metis add-liquidity-properties(15) assms(2) assms(4) p'-def pool-ne-def)
  moreover
  have l p' ≥ l p
   using add-liquidity-properties(9) assms(2) assms(4) p'-def pool-ne-def by blast
  moreover
  have l p ≥ l p0
    using AMM-Zhang.inv-def assms(1) by blast
  ultimately show ?thesis using l5
    using assms(1) assms(3) assms(4) order-trans by blast
qed

lemma inv-rem-okay:
  fixes p0 p p' Δl
  defines p' ≡ remove-liquidity-spec p Δl
  assumes inv p0 p and 0 ≤ Δl and pool-ne p0 and pool-ne p and l p0 < l p'
  shows inv p0 p'
proof −
  have x p' / l p' = x p / l p and y p' / l p' = y p / l p
   apply (smt (verit, best) assms(4) assms(6) nonzero-mult-divide-mult-cancel-left
p'-def pool-ne-def remove-liquidity-spec-def select-convs(1) select-convs(3) zero-less-mult-iff)
   apply (smt (verit) assms(4) assms(6) divide-divide-eq-left divide-eq-0-iff nonzero-mult-div-cancel-left
p'-def pool-ne-def remove-liquidity-spec-def select-convs(2) select-convs(3))
    done
  thus ?thesis using l5
    by (metis assms(2) assms(4) assms(5) assms(6) less-le-not-le)
qed

end
```