Formalization of the Ledger Rules in Isabelle/HOL

Javier Díaz

javier.diaz@iohk.io

github.com/input-output-hk/fm-ouroboros

October 8, 2019

Contents

1 No Double-Spending Property

theory *Ledger-Rules*
  imports *Main*
begin

1.1 Non-standard map operators

definition *dom-exc* $:: 'a\ set \Rightarrow ('a \rightharpoonup 'b) \Rightarrow ('a \rightharpoonup 'b)$ (- $\lhd$'/ - [*61*, *61*] *60*) **where**
  $s \lhd/\ m = m\ |`\ (-\ s)$

lemma *dom-exc-distr*: $(s_1 \cup s_2) \lhd/\ m = s_1 \lhd/\ (s_2 \lhd/\ m)$
  by (*simp add*: *dom-exc-def inf-commute*)

lemma *dom-exc-assoc*:
  assumes $dom\ m_1 \cap dom\ m_2 = \{\}$
  and $s \cap dom\ m_2 = \{\}$
  shows $(s \lhd/\ m_1) ++ m_2 = s \lhd/\ (m_1 ++ m_2)$
  using *assms*

proof −
  have *dom* $(s \lhd/ m_1) \cap$ *dom* $m_2 = \{\}$
    by (*simp add*: *assms*(*1*) *dom-exc-def inf-commute inf-left-commute*)
  then have *rtl*: $(s \lhd/ m_1)$ ++ $m_2 \subseteq_m s \lhd/ (m_1$ ++ $m_2)$
   by (*smt assms*(*2*) *disjoint-eq-subset-Compl disjoint-iff-not-equal dom-exc-def map-add-dom-app-simps*(*1*) *map-add-dom-app-simps*(*3*) *map-le-def restrict-map-def subsetCE*)
  moreover have $s \lhd/ (m_1$ ++ $m_2) \subseteq_m (s \lhd/ m_1)$ ++ m_2
    by (*smt rtl domIff dom-exc-def map-add-None map-le-def restrict-map-def*)
  ultimately show *?thesis*
    using *map-le-antisym* **by** *blast*
qed

1.2 Abstract types

typedecl *tx-id*
typedecl *ix*
typedecl *addr*
typedecl *tx*

1.3 Derived types

type-synonym *coin* = *int*
type-synonym *tx-in* = *tx-id* × *ix*
type-synonym *tx-out* = *addr* × *coin*
type-synonym *utxo* = *tx-in* ⇀ *tx-out*

1.4 Transaction Types

type-synonym *tx-body* = *tx-in set* × (*ix* ⇀ *tx-out*)

1.5 Abstract functions

fun *txid* :: *tx* ⇒ *tx-id* **where**
  txid - = *undefined*

fun *txbody* :: *tx* ⇒ *tx-body* **where**
  txbody - = *undefined*

1.6 Accessor functions

fun *txins* :: *tx* ⇒ *tx-in set* **where**
  txins tx = (*let* (*inputs*, -) = *txbody tx in inputs*)

fun *txouts* :: *tx* ⇒ *utxo* **where**
  txouts tx = (
    let (-, *outputs*) = *txbody tx in* (
      λ(*id*, *ix*). *if id* ≠ *txid tx then None else case outputs ix of None* ⇒ *None* | *Some txout* ⇒ *Some txout*))

lemma *dom-txouts-is-txid*:

shows $\bigwedge i\ ix.\ (i,\ ix) \in dom\ (txouts\ tx) \Longrightarrow i = txid\ tx$
by (*smt case-prod-conv domIff surj-pair txouts.simps*)

1.7 UTxO transition-system types

— UTxO environment
typedecl *utxo-env* — Abstract, don't care for now

— UTxO states
type-synonym *utxo-state* = *utxo* — Simplified

1.8 UTxO inference rules

inductive *utxo-sts* :: *utxo-env* ⇒ *utxo-state* ⇒ *tx* ⇒ *utxo-state* ⇒ *bool*
$(\text{-} \vdash \text{-} \rightarrow_{UTXO}\{\text{-}\}\ \text{-}\ [51,\ 0,\ 51]\ 50)$
for Γ
where
utxo-inductive:
⟦
$txins\ tx \subseteq dom\ utxo\text{-}st$;
$txins\ tx \neq \{\}$;
$txouts\ tx \neq Map.empty$;
$\forall (\text{-},\ c) \in ran\ (txouts\ tx).\ c > 0$
⟧
$\Longrightarrow$
$\Gamma \vdash utxo\text{-}st \rightarrow_{UTXO}\{tx\}\ (txins\ tx \lhd/\ utxo\text{-}st) +\!+\ txouts\ tx$

1.9 Transaction sequences

inductive *utxows* :: *utxo-env* ⇒ *utxo-state* ⇒ *tx list* ⇒ *utxo-state* ⇒ *bool*
$(\text{-} \vdash \text{-} \rightarrow_{UTXOWS}\{\text{-}\}\ \text{-}\ [51,\ 0,\ 51]\ 50)$
for Γ
where
empty: $\Gamma \vdash s \rightarrow_{UTXOWS}\{[]\}\ s$ |
step: $\Gamma \vdash s \rightarrow_{UTXOWS}\{txs\ @\ [tx]\}\ s''$ **if** $\Gamma \vdash s \rightarrow_{UTXOWS}\{txs\}\ s'$ **and** $\Gamma \vdash s' \rightarrow_{UTXO}\{tx\}$ s''

1.10 Auxiliary lemmas and main theorem

abbreviation *txid-injectivity* :: *bool* **where**
$txid\text{-}injectivity \equiv \forall\ tx\ tx'.\ txid\ tx = txid\ tx' \longrightarrow tx = tx'$

lemma *lemma-1*:
assumes $\Gamma \vdash utxo_0 \rightarrow_{UTXOWS}\{T\}\ utxo$
and $\forall\ T_i \in set\ T.\ txins\ T_i \cap txins\ tx = \{\}$
and $dom\ (txouts\ tx) \cap dom\ utxo_0 = \{\}$
and *txid-injectivity*
shows $(\bigcup\ T_i \in set\ T.\ txins\ T_i) \cap dom\ (txouts\ tx) = \{\}$
and $dom\ (txouts\ tx) \cap dom\ utxo = \{\}$
using *assms*

proof (*induction rule*: *utxows.induct*)
  case (*empty s*)
  { case *1*
    then show *?case*
      by *force*
  next
    case *2*
    then show *?case*
      by *blast*
  }
next
  case (*step* $utxo_0$ *T utxo tx′ utxo′*)
  { case *1*
    then have $txins\ tx' \cap dom\ (txouts\ tx) = \{\}$
    proof −
      have $\forall\ T_i \in set\ T.\ txins\ T_i \cap txins\ tx = \{\}$
        using *1.prems*(*1*) **and** *assms*(*4*) **by** *auto*
      then have $(\bigcup T_i \in set\ T.\ txins\ T_i) \cap dom\ (txouts\ tx) = \{\}$
        using *step.IH*(*1*) **and** *1.prems*(*2*) **and** *assms*(*4*) **by** *simp*
      moreover have $txins\ tx' \subseteq dom\ utxo$
        using *step.hyps*(*2*) **and** *utxo-sts.simps* **by** *blast*
      ultimately show *?thesis*
      by (*smt 1.prems*(*2*) *assms*(*4*) ‹$\forall\ T_i{\in}set\ T.\ txins\ T_i \cap txins\ tx = \{\}$› *inf.orderE inf-bot-right inf-sup-aci*(*1*) *inf-sup-aci*(*2*) *step.IH*(*2*))
    qed
    moreover have $(\bigcup T_i \in set\ T.\ txins\ T_i) \cap dom\ (txouts\ tx) = \{\}$
      using *1.prems* **and** *step.IH*(*1*) **by** *simp*
    ultimately show *?case*
    by (*smt Int-empty-right SUP-empty UN-Un UN-insert empty-set inf-commute inf-sup-distrib1 list.simps*(*15*) *set-append*)
  next
    case *2*
    then have $dom\ (txouts\ tx) \cap dom\ (txins\ tx' \lhd/\ utxo) = \{\}$
      by (*smt Int-iff butlast-snoc disjoint-iff-not-equal dom-exc-def dom-restrict in-set-butlastD step.IH*(*2*))
    moreover have $dom\ (txouts\ tx) \cap dom\ (txouts\ tx') = \{\}$
    proof −
      have $txins\ tx' \cap txins\ tx = \{\}$
        using *2.prems*(*1*) **by** (*meson in-set-conv-decomp*)
      then have $txins\ tx' \neq txins\ tx$
        using *inf.idem step.hyps*(*2*) *utxo-sts.cases* **by** *auto*
      then have $tx' \neq tx$
        by *blast*
      then have $txid\ tx' \neq txid\ tx$
        using *assms*(*4*) **by** *blast*
      then show *?thesis*
        using *dom-txouts-is-txid* **by** (*simp add*: *ComplI disjoint-eq-subset-Compl subrelI*)
    qed
    ultimately have $dom\ (txouts\ tx) \cap dom\ ((txins\ tx' \lhd/\ utxo)\ {+}{+}\ txouts\ tx') = \{\}$

by *blast*
then show *?case*
using *utxo-sts.simps* **and** *step.hyps*(*2*) **by** *simp*
}
qed

lemma *aux-lemma*:
assumes $\Gamma \vdash utxo_0 \rightarrow_{UTXOWS}\{T\}\ utxo$
and $\Gamma \vdash utxo \rightarrow_{UTXO}\{tx\}\ utxo'$
and $\forall\, T_i \in set\ (T\ @\ [tx]).\ dom\ (txouts\ T_i) \cap dom\ utxo_0 = \{\}$
and $\forall\, T_i \in set\ T.\ dom\ (txouts\ T_i) \cap dom\ utxo_0 = \{\} \Longrightarrow \forall\, T_i \in set\ T.\ txins\ T_i \cap dom\ utxo = \{\}$
and *txid-injectivity*
shows $txins\ tx \cap dom\ (txouts\ tx) = \{\}$
using *assms*
proof −
have $txins\ tx \subseteq dom\ utxo$
using *assms*(*2*) *utxo-sts.simps* **by** *blast*
then have $\forall\, T_i \in set\ T.\ txins\ T_i \cap txins\ tx = \{\}$
by (*smt assms*(*3*) *assms*(*4*) *butlast-snoc in-set-butlastD inf.orderE inf-bot-right inf-left-commute*)

then have $(\bigcup\, T_i \in set\ T.\ txins\ T_i) \cap dom\ (txouts\ tx) = \{\}$ **and** $dom\ (txouts\ tx) \cap dom\ utxo = \{\}$
using *lemma-1* **and** *assms*(*1*−*3*,*5*) **by** *auto*
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *disjoint-iff-not-equal assms*(*2*) *subsetCE utxo-sts.simps*)
qed

lemma *lemma-3*:
assumes $\Gamma \vdash utxo_0 \rightarrow_{UTXOWS}\{T\}\ utxo$
and $\forall\, T_i \in set\ T.\ dom\ (txouts\ T_i) \cap dom\ utxo_0 = \{\}$
and *txid-injectivity*
shows $\forall\, T_i \in set\ T.\ txins\ T_i \cap dom\ utxo = \{\}$
using *assms*
proof (*induction rule*: *utxows.induct*)
case (*empty s*)
then show *?case*
by *simp*
next
case (*step* $utxo_0$ *T utxo tx* $utxo'$)
then have $\bigwedge T_i.\ T_i \in set\ T \Longrightarrow txins\ T_i \cap dom\ utxo' = \{\}$
proof −
fix T_i
assume $T_i \in set\ T$
then have $txins\ T_i \cap dom\ (txins\ tx \lhd/\ utxo) = \{\}$
proof −
from *step.IH* **and** $\langle T_i \in set\ T \rangle$ **and** *step.prems* **have** $txins\ T_i \cap dom\ utxo = \{\}$
by (*metis butlast-snoc in-set-butlastD*)
then show *?thesis*

```
      by (simp add: disjoint-iff-not-equal dom-exc-def)
    qed
    moreover have txins T_i ∩ dom (txouts tx) = {}
    proof −
      have txins tx ⊆ dom utxo
        using step.hyps(2) utxo-sts.simps by blast
      then have ∀ T_i ∈ set T. txins T_i ∩ txins tx = {}
        using step.IH and ‹T_i ∈ set T› and step.prems
        by (smt butlast-snoc in-set-butlastD inf.orderE inf-bot-right inf-left-commute)
      then have (⋃ T_i ∈ set T. txins T_i) ∩ dom (txouts tx) = {}
        using lemma-1 in-set-conv-decomp step.hyps(1) step.prems by auto
      then show ?thesis
        using ‹T_i ∈ set T› by blast
    qed
    ultimately have txins T_i ∩ dom ((txins tx ◁/ utxo) ++ txouts tx) = {}
      by blast
    then show txins T_i ∩ dom utxo' = {}
      using step.hyps(2) utxo-sts.simps by auto
  qed
  moreover have txins tx ∩ dom utxo' = {}
  proof −
    have txins tx ∩ dom (txins tx ◁/ utxo) = {}
      by (simp add: dom-exc-def)
    moreover have txins tx ∩ dom (txouts tx) = {}
      using aux-lemma step.IH step.hyps(1−2) step.prems by blast
    ultimately have txins tx ∩ dom ((txins tx ◁/ utxo) ++ txouts tx) = {}
      by blast
    then show ?thesis
      using utxo-sts.simps and step.hyps(2) by auto
  qed
  ultimately show ?case
    by simp
qed

theorem no-double-spending:
  assumes Γ ⊢ utxo_0 →_UTXOWS{T} utxo
  and ∀ T_i ∈ set T. dom (txouts T_i) ∩ dom utxo_0 = {}
  and txid-injectivity
  shows ∀ i ≥ 0. ∀ j < length T. i < j ⟶ txins (T ! i) ∩ txins (T ! j) = {}
  using assms
proof (induction arbitrary: utxo rule: utxows.induct)
  case (empty s)
  then show ?case
    by simp
next
  case (step utxo_0 T utxo tx utxo')
  then show ?case
  proof (intro allI impI)
    fix i j
```

assume $i \geq 0$ **and** $j <$ *length* $(T$ @ $[tx])$ **and** $i < j$
then consider
(a) $j <$ *length* T |
(b) $j =$ *length* T
by *fastforce*
then show *txins* $((T$ @ $[tx])$! $i) \cap$ *txins* $((T$ @ $[tx])$! $j) = \{\}$
proof (*cases*)
case *a*
with ‹$i \geq 0$› **and** ‹$i < j$› **and** *step.prems* **and** *step.IH* **show** *?thesis*
by (*smt butlast-snoc in-set-conv-nth length-append-singleton less-Suc-eq less-trans nth-butlast*)
next
case *b*
with ‹$i < j$› **have** $(T$ @ $[tx])$! $i = T$! i
by (*simp add*: *nth-append*)
moreover with ‹$j =$ *length* T› **have** $(T$ @ $[tx])$! $j = tx$
by *simp*
ultimately have *txins* $(T$! $i) \cap$ *txins* $tx = \{\}$
proof −
have *txins* $(T$! $i) \cap$ *dom utxo* $= \{\}$
using *lemma-3* ‹$\Gamma \vdash utxo_0 \rightarrow_{UTXOWS}\{T\}$ *utxo*› ‹$i < j$› *b* *step.prems*
by (*metis UnCI nth-mem set-append*)
moreover from ‹$\Gamma \vdash utxo \rightarrow_{UTXO}\{tx\}$ *utxo'*› **and** *utxo-sts.simps* **have** *txins tx* $\subseteq$ *dom utxo*
by *simp*
ultimately show *?thesis* **by** *blast*
qed
with ‹$(T$ @ $[tx])$! $j = tx$› **and** ‹$(T$ @ $[tx])$! $i = T$! i› **show** *?thesis*
by *simp*
qed
qed
qed

end