Isabelle/Solidity

A deep Embedding of Solidity in Isabelle/HOL

Diego Marmsoler and Achim D. Brucker

August 12, 2022

Department of Computer Science, University of Exeter, Exeter, UK
{d.marmsoler,a.brucker}@exeter.ac.uk

Abstract

Smart contracts are automatically executed programs, usually representing legal agreements such as financial transactions. Thus, bugs in smart contracts can lead to large financial losses. For example, an incorrectly initialized contract was the root cause of the Parity Wallet bug that saw $280M worth of Ether destroyed. Ether is the cryptocurrency of the Ethereum blockchain that uses Solidity for expressing smart contracts.

We address this problem by formalizing an executable denotational semantics for Solidity in the interactive theorem prover Isabelle/HOL. This formal semantics builds the foundation of an interactive program verification environment for Solidity programs and allows for inspecting them by (symbolic) execution. We combine the latter with grammar based fuzzing to ensure that our formal semantics complies to the Solidity implementation on the Ethereum Blockchain. Finally, we demonstrate the formal verification of Solidity programs by two examples: constant folding and a simple verified token.

Keywords: Solidity, Denotational Semantics, Isabelle/HOL, Gas

Contents

1 Introduction

An increasing number of businesses is adopting blockchain-based solutions. Most notably, the market value of Bitcoin, most likely the first and most well-known blockchain-based cryptocurrency, passed USD 1 trillion in February 2021 [1]. While Bitcoin might be the most well-known application of a blockchain, it lacks features that applications outside cryptocurrencies require and that make blockchain solutions attractive to businesses.

For example, the Ethereum blockchain [6] is a feature-rich distributed computing platform that provides not only a cryptocurrency, called *Ether*: Ethereum also provides an immutable distributed data structure (the *blockchain*) on which distributed programs, called *smart contracts*, can be executed. Essentially, smart contracts are automatically executed programs, usually representing a legal agreement, e.g., financial transactions. To support those applications, Ethereum provides a dedicated account data structure on its blockchain that smart contracts can modify, i.e., transferring Ether between accounts. Thus, bugs in smart contracts can lead to large financial losses. For example, an incorrectly initialized contract was the root cause of the Parity Wallet bug that saw $280M worth of Ether destroyed [5]. This risk of bugs being costly is already a big motivation for using formal verification techniques to minimize this risk. The fact that smart contracts are deployed on the blockchain immutably, i.e., they cannot be updated or removed easily, makes it even more important to "get smart contracts" right, before they are deployed on a blockchain for the very first time.

For implementing smart contracts, Ethereum provides *Solidity* [4], a Turing-complete, statically typed programming language that has been designed to look familiar to people knowing Java, C, or JavaScript. Notably, the type system provides, e.g., numerous integer types of different sizes (e.g., `uint256`) and Solidity also relies on different types of stores. While Solidity is Turing-complete, the execution of Solidity programs is guaranteed to terminate. The reason for this is that executing Solidity operations costs *gas*, a tradable commodity on the Ethereum blockchain. Gas does cost Ether and hence, programmers of smart contracts have an incentive to write highly optimized contracts whose execution consumes as little gas as possible. For example, the size of the integer types used can impact the amount of gas required for executing a contract. This desire for highly optimized contracts can conflict with the desire to write correct contracts.

In this paper, we address the problem of developing smart contracts in Solidity that are correct: we present an executable denotational semantics for Solidity in the interactive theorem prover Isabelle/HOL.

In particular, our semantics supports the following features of Solidity:

- *Fixed-size integer types* of various lengths and corresponding arithmetic.
- *Domain-specific primitives*, such as money transfer or balance queries.
- *Different types of stores*, such as storage, memory, and stack.
- *Complex data types*, such as hash-maps and arrays.
- *Assignments with different semantics*, depending on data types.
- An extendable *gas model.*
- *Internal and external method calls.*

A more abstract description of the semantics is given in [2] and the conformance testing approach for ensuring that our semantics conforms to the actual implementation is described in [3].

The rest of this document is automatically generated from the formalization in Isabelle/HOL, i.e., all content is checked by Isabelle. The structure follows the theory dependencies (see Figure 1.1).

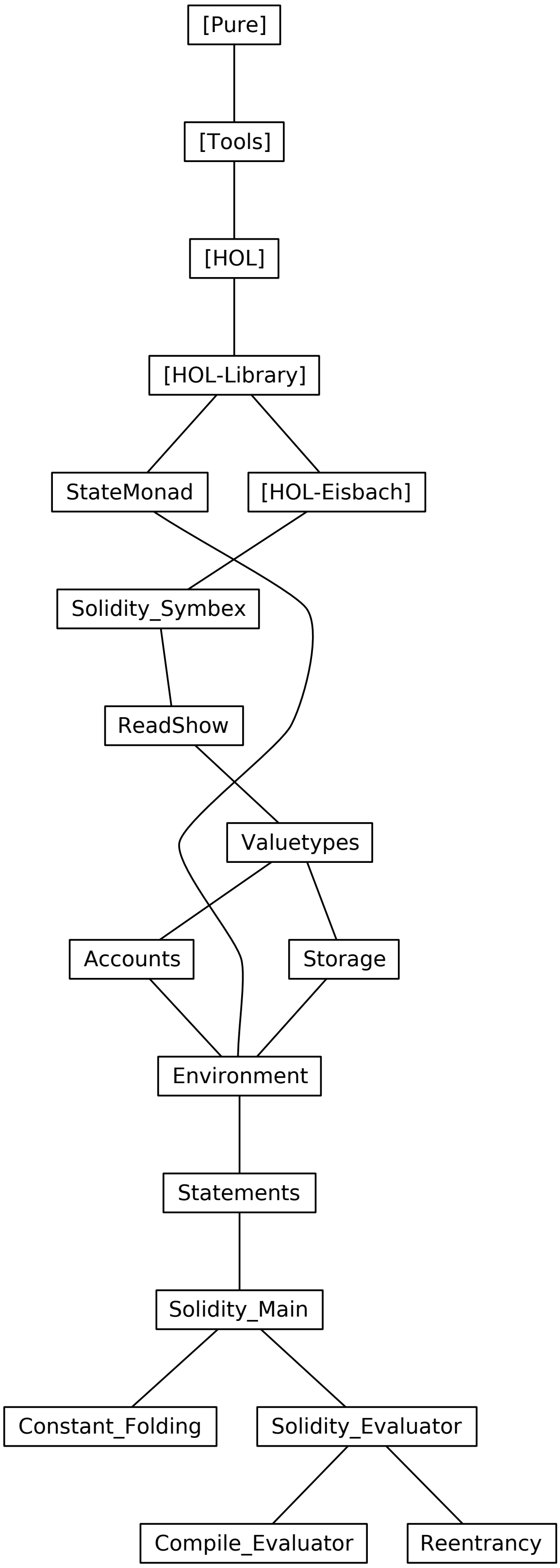

Figure 1.1: The Dependency Graph of the Isabelle Theories.

2 Preliminaries

In this chapter, we discuss auxiliary formalizations and functions that are used in our Solidity semantics but are more generic, i.e., not specific to Solidity. This includes, for example, functions to convert values of basic types to/from strings.

2.1 Converting Types to Strings and Back Again (ReadShow)

```
theory ReadShow
  imports
    Solidity_Symbex
begin
```

In the following, we formalize a family of projection (and injection) functions for injecting (projecting) basic types (i.e., *nat*, *int*, and *bool* in (out) of the domains of strings. We provide variants for the two string representations of Isabelle/HOL, namely *string* and *String.literal*.

Bool

```
definition
 ‹Read_{bool} s = (if s = ''True'' then True else False)›
definition
 ‹Show_{bool} b = (if b then ''True'' else ''False'')›
definition
 ‹STR_is_bool s = (Show_{bool} (Read_{bool} s) = s)›

declare Read_{bool}_def [solidity_symbex]
        Show_{bool}_def [solidity_symbex]

lemma Show_Read_bool_id: ‹STR_is_bool s ⟹ (Show_{bool} (Read_{bool} s) = s)›
  using STR_is_bool_def by fastforce

lemma STR_is_bool_split: ‹STR_is_bool s ⟹ s = ''False'' ∨ s = ''True''›
  by(auto simp: STR_is_bool_def Read_{bool}_def Show_{bool}_def)

lemma Read_Show_bool_id: ‹Read_{bool} (Show_{bool} b) = b›
  by(auto simp: Read_{bool}_def Show_{bool}_def)

definition ReadL_{bool}::‹String.literal ⇒ bool› (‹⌊_⌋›) where
 ‹ReadL_{bool} s = (if s = STR ''True'' then True else False)›
definition ShowL_{bool}:: ‹bool ⇒ String.literal› (‹⌈_⌉›) where
 ‹ShowL_{bool} b = (if b then STR ''True'' else STR ''False'')›
definition
 ‹strL_is_bool' s = (ShowL_{bool} (ReadL_{bool} s) = s)›

declare ReadL_{bool}_def [solidity_symbex]
        ShowL_{bool}_def [solidity_symbex]


lemma Show_Read_bool'_id: ‹strL_is_bool' s ⟹ (ShowL_{bool} (ReadL_{bool} s) = s)›
  using strL_is_bool'_def by fastforce

lemma strL_is_bool'_split:  ‹strL_is_bool' s ⟹ s = STR ''False'' ∨ s = STR ''True''›
  by(auto simp: strL_is_bool'_def ReadL_{bool}_def ShowL_{bool}_def)

lemma Read_Show_bool'_id: ‹ReadL_{bool} (ShowL_{bool} b) = b›
  by(auto simp: ReadL_{bool}_def ShowL_{bool}_def)
```

Natural Numbers

```
definition  nat_of_digit ::  ‹char ⇒ nat› where
  ‹nat_of_digit c =
    (if c = CHR ''0'' then 0
    else if c = CHR ''1'' then 1
    else if c = CHR ''2'' then 2
    else if c = CHR ''3'' then 3
    else if c = CHR ''4'' then 4
    else if c = CHR ''5'' then 5
    else if c = CHR ''6'' then 6
    else if c = CHR ''7'' then 7
    else if c = CHR ''8'' then 8
    else if c = CHR ''9'' then 9
    else undefined)›

declare nat_of_digit_def [solidity_symbex]

definition  digit_of_nat ::  ‹nat ⇒ char› where
  ‹digit_of_nat x =
    (if x = 0 then CHR ''0''
    else if x = 1 then CHR ''1''
    else if x = 2 then CHR ''2''
    else if x = 3 then CHR ''3''
    else if x = 4 then CHR ''4''
    else if x = 5 then CHR ''5''
    else if x = 6 then CHR ''6''
    else if x = 7 then CHR ''7''
    else if x = 8 then CHR ''8''
    else if x = 9 then CHR ''9''
    else undefined)›

declare digit_of_nat_def [solidity_symbex]

lemma nat_of_digit_digit_of_nat_id:
    ‹x < 10 ⟹ nat_of_digit (digit_of_nat x) = x›
  by(simp add:nat_of_digit_def digit_of_nat_def)

lemma img_digit_of_nat:
‹n < 10 ⟹ digit_of_nat n ∈ {CHR ''0'', CHR ''1'', CHR ''2'', CHR ''3'', CHR ''4'',
                            CHR ''5'', CHR ''6'', CHR ''7'', CHR ''8'', CHR ''9''}›
  by(simp add:nat_of_digit_def digit_of_nat_def)

lemma digit_of_nat_nat_of_digit_id:
    ‹c ∈ {CHR ''0'', CHR ''1'', CHR ''2'', CHR ''3'', CHR ''4'',
          CHR ''5'', CHR ''6'', CHR ''7'', CHR ''8'', CHR ''9''}
      ⟹ digit_of_nat (nat_of_digit c) = c›
  by(code_simp, auto)

definition
  nat_implode :: ‹'a::{numeral,power,zero} list ⇒ 'a› where
 ‹nat_implode n = foldr (+) (map (λ (p,d) ⇒ 10 ^ p * d) (enumerate 0 (rev n))) 0›

declare nat_implode_def [solidity_symbex]

fun nat_explode' :: ‹nat ⇒ nat list› where
   ‹nat_explode' x = (case  x < 10 of True ⇒ [x mod 10]
                                   | _ ⇒ (x mod 10 )#(nat_explode' (x div 10)))›

definition
  nat_explode :: ‹nat ⇒ nat list› where
 ‹nat_explode x = (rev (nat_explode' x))›

declare nat_explode_def [solidity_symbex]
```

```
lemma nat_explode'_not_empty: ‹nat_explode' n ≠ []›
  by (smt (z3) list.simps(3) nat_explode'.simps)

lemma nat_explode_not_empty: ‹nat_explode n ≠ []›
  using nat_explode'_not_empty nat_explode_def by auto

lemma nat_explode'_ne_suc: ‹∃ n. nat_explode' (Suc n) ≠ nat_explode' n›
  by(rule exI [of _ ‹0::nat›], simp)

lemma nat_explode'_digit: ‹hd (nat_explode' n ) < 10›
proof(induct ‹n›)
  case 0
  then show ?case by simp
next
  case (Suc n)
  then show ?case proof (cases ‹n < 9›)
    case True
    then show ?thesis by simp
  next
    case False
    then show ?thesis
      by simp
  qed
qed

lemma div_ten_less: ‹n ≠ 0 ⟹ ((n::nat) div 10) < n›
  by simp

lemma unroll_nat_explode':
 ‹¬ n < 10 ⟹ (case n < 10 of True ⇒ [n mod 10] | False ⇒ n mod 10 # nat_explode' (n div 10)) =
       (n mod 10 # nat_explode' (n div 10))›
  by simp

lemma nat_explode_mod_10_ident: ‹map (λ x. x mod 10) (nat_explode' n) = nat_explode' n›
proof (cases ‹n < 10›)
  case True
  then show ?thesis by simp
next
  case False
  then show ?thesis
  proof (induct ‹n› rule: nat_less_induct)
  case (1 n) note * = this
  then show ?case
    using div_ten_less apply(simp (no_asm))
    using unroll_nat_explode'[of n] *
    by (smt (z3) list.simps(8) list.simps(9) mod_div_trivial mod_eq_self_iff_div_eq_0
                nat_explode'.simps zero_less_numeral)
  qed
qed

lemma nat_explode'_digits: ‹∀ d ∈ set (nat_explode' n). d < 10›
  using image_set[of ‹(λ x. x mod 10)› ‹(nat_explode' n)›]
        nat_explode_mod_10_ident[symmetric]
  by (metis (no_types, lifting) Euclidean_Division.div_eq_0_iff
                              imageE mod_div_trivial zero_neq_numeral)

lemma nat_explode_digits: ‹∀ d ∈ set (nat_explode n). d < 10›
  using nat_explode_def set_rev
  by (metis nat_explode'_digits)

value ‹nat_implode(nat_explode 42) = 42›
value ‹nat_explode (Suc 21)›
```

```
lemma nat_implode_append:
 ‹nat_implode (a@[b]) = (1*b + foldr (+) (map (λ(p, y). 10 ^ p * y) (enumerate (Suc 0) (rev a))) 0 )›
  by(simp add:nat_implode_def)

lemma enumerate_suc: ‹enumerate (Suc n) l = map (λ (a,b). (a+1::nat,b)) (enumerate n l)›
proof (induction ‹l›)
  case Nil
  then show ?case by simp
next
  case (Cons a x) note * = this
  then show ?case apply(simp only:enumerate_simps)

    apply(simp only:‹enumerate (Suc n) x = map (λa. case a of (a, b) ⇒ (a + 1, b)) (enumerate n
x)›[symmetric])
    apply(simp)
    using *
    by (metis apfst_conv cond_case_prod_eta enumerate_Suc_eq)
qed

lemma mult_assoc_aux1:
 ‹(λ(p, y). 10 ^ p * y) ∘ (λ(a, y). (Suc a, y)) = (λ(p, y). (10::nat) * (10 ^ p) * y)›
  by(auto simp:o_def)

lemma fold_map_transfer:
 ‹(foldr (+) (map (λ(x,y). 10 * (f (x,y))) l) (0::nat)) = 10 * (foldr (+) (map (λx. (f x)) l)
(0::nat))›
proof(induct ‹l›)
  case Nil
  then show ?case by(simp)
next
  case (Cons a l)
  then show ?case  by(simp)
qed

lemma mult_assoc_aux2: ‹(λ(p, y). 10 * 10 ^ p * (y::nat)) = (λ(p, y). 10 * (10 ^ p * y))›
  by(auto)

lemma nat_implode_explode_id: ‹nat_implode (nat_explode n) = n›
proof (cases ‹n=0›)
  case True note * = this
  then show ?thesis
    by (simp add: nat_explode_def nat_implode_def)
next
  case False
  then show ?thesis
  proof (induct ‹n› rule: nat_less_induct)
    case (1 n) note * = this
    then  have
      **: ‹nat_implode (nat_explode (n div 10)) = n div 10›
    proof (cases ‹n div 10 = 0›)
      case True
      then show ?thesis by(simp add: nat_explode_def nat_implode_def)
    next
      case False
      then show ?thesis
      using div_ten_less[of ‹n›] *
      by(simp)
  qed
    then show ?case
    proof (cases ‹n < 10›)
      case True
      then show ?thesis  by(simp add: nat_explode_def nat_implode_def)
    next
```

```
      case False note *** = this
      then show ?thesis
        apply(simp (no_asm)  add:nat_explode_def del:rev_rev_ident)
        apply(simp only: bool.case(2))
        apply(simp del:nat_explode'.simps rev_rev_ident)
        apply(fold nat_explode_def)
        apply(simp only:nat_implode_append)
        apply(simp add:enumerate_suc)
        apply(simp only:mult_assoc_aux1)
        using mult_assoc_aux2 apply(simp)
        using fold_map_transfer[of ‹λ(p, y). 10 ^ p * y›
                                 ‹(enumerate 0 (rev (nat_explode (n div 10))))›, simplified]
        apply(simp) apply(fold nat_implode_def) using **
        by simp
      qed
  qed
qed

definition
  Read_nat :: ‹string ⇒ nat› where
 ‹Read_nat s = nat_implode (map nat_of_digit s)›

definition
  Show_nat::"nat ⇒ string" where
 ‹Show_nat n = map digit_of_nat (nat_explode n)›

declare Read_nat_def [solidity_symbex]
        Show_nat_def [solidity_symbex]

definition
  ‹STR_is_nat s = (Show_nat (Read_nat s) = s)›

value ‹Read_nat ''10''›
value ‹Show_nat 10›
value ‹Read_nat (Show_nat (10)) = 10›
value ‹Show_nat (Read_nat (''10'')) = ''10''›

lemma Show_nat_not_neg:
  ‹set (Show_nat n) ⊆{CHR ''0'', CHR ''1'', CHR ''2'', CHR ''3'', CHR ''4'',
                     CHR ''5'', CHR ''6'', CHR ''7'', CHR ''8'', CHR ''9''}›
  unfolding Show_nat_def
  using  nat_explode_digits[of n]  img_digit_of_nat imageE image_set subsetI
  by (smt (verit) imageE image_set subsetI)

lemma Show_nat_not_empty: ‹(Show_nat n) ≠ []›
  by (simp add: Show_nat_def nat_explode_not_empty)

lemma not_hd: ‹L ≠ [] ⟹ e ∉ set(L) ⟹ hd L ≠ e›
  by auto

lemma Show_nat_not_neg'': ‹hd (Show_nat n) ≠ (CHR ''-'')›
  using Show_nat_not_neg[of ‹n›]
  Show_nat_not_empty[of ‹n›] not_hd[of ‹Show_nat n›]
  by auto

lemma Show_Read_nat_id: ‹STR_is_nat s ⟹ (Show_nat (Read_nat s) = s)›
  by(simp add:STR_is_nat_def)

lemma bar': ‹∀ d ∈ set l . d < 10 ⟹ map nat_of_digit (map digit_of_nat l) = l›
  using  nat_of_digit_digit_of_nat_id
  by (simp add: map_idI)

lemma Read_Show_nat_id: ‹Read_nat(Show_nat n) = n›
  apply(unfold Read_nat_def Show_nat_def)
```

```
  using bar' nat_of_digit_digit_of_nat_id nat_explode_digits
  using nat_implode_explode_id
  by presburger

definition
  ReadL_{nat} :: ‹String.literal ⇒ nat› (‹⌈_⌉›) where
 ‹ReadL_{nat} = Read_{nat} ∘ String.explode›

definition
  ShowL_{nat}::‹nat ⇒ String.literal› (‹⌊_⌋›)where
 ‹ShowL_{nat} = String.implode ∘ Show_{nat}›

declare ReadL_{nat}_def [solidity_symbex]
        ShowL_{nat}_def [solidity_symbex]


definition
  ‹strL_is_nat' s = (ShowL_{nat} (ReadL_{nat} s) = s)›

value ‹⌈STR ''10''⌉::nat›
value ‹ReadL_{nat} (STR ''10'')›
value ‹⌊10::nat⌋›
value ‹ShowL_{nat} 10›
value ‹ReadL_{nat} (ShowL_{nat} (10)) = 10›
value ‹ShowL_{nat} (ReadL_{nat} (STR ''10'')) = STR ''10''›

lemma Show_Read_nat'_id: ‹strL_is_nat' s ⟹ (ShowL_{nat} (ReadL_{nat} s) = s)›
  by(simp add:strL_is_nat'_def)

lemma digits_are_ascii:
  ‹c ∈ {CHR ''0'', CHR ''1'', CHR ''2'', CHR ''3'', CHR ''4'',
        CHR ''5'', CHR ''6'', CHR ''7'', CHR ''8'', CHR ''9''}
   ⟹ String.ascii_of c = c›
  by auto

lemma Show_{nat}_ascii: ‹map String.ascii_of (Show_{nat} n) = Show_{nat} n›
  using Show_nat_not_neg digits_are_ascii
  by (meson map_idI subsetD)


lemma Read_Show_nat'_id: ‹ReadL_{nat}(ShowL_{nat} n) = n›
  apply(unfold ReadL_{nat}_def ShowL_{nat}_def, simp)
  by (simp add: Show_{nat}_ascii  Read_Show_nat_id)
```

Integer

```
definition
  Read_{int} :: ‹string ⇒ int› where
 ‹Read_{int} x = (if hd x = (CHR ''-'') then  -(int (Read_{nat} (tl x))) else int (Read_{nat} x))›

definition
  Show_{int}::‹int ⇒ string› where
 ‹Show_{int} i = (if i < 0 then (CHR ''-'')#(Show_{nat} (nat (-i)))
                      else Show_{nat} (nat i))›

definition
 ‹STR_is_int s = (Show_{int} (Read_{int} s) = s)›


declare Read_{int}_def [solidity_symbex]
        Show_{int}_def [solidity_symbex]

value ‹Read_{int} (Show_{int}   10)  =  10›
value ‹Read_{int} (Show_{int} (-10)) = -10›
```

```
value ‹Show_{int} (Read_{int} (''10''))  =  ''10''›
value ‹Show_{int} (Read_{int} (''-10'')) = ''-10''›

lemma Show_Read_id: ‹STR_is_int s ⟹ (Show_{int} (Read_{int} s) = s)›
  by(simp add:STR_is_int_def)

lemma Read_Show_id: ‹Read_{int}(Show_{int}(x)) = x›
  apply(code_simp)
  apply(auto simp:Show_nat_not_neg Read_Show_nat_id)[1]
  apply(thin_tac ‹¬ x < 0 ›)
  using Show_nat_not_neg''
  by blast

lemma STR_is_int_Show: ‹STR_is_int (Show_{int} n)›
  by(simp add:STR_is_int_def Read_Show_id)

definition
  ReadL_{int} :: ‹String.literal ⇒ int› (‹⌈_⌉›) where
 ‹ReadL_{int}  = Read_{int} ∘ String.explode›

definition
  ShowL_{int}::‹int ⇒ String.literal› (‹⌊_⌋›) where
 ‹ShowL_{int} =String.implode ∘ Show_{int}›

definition
 ‹strL_is_int' s = (ShowL_{int} (ReadL_{int} s) = s)›

declare ReadL_{int}_def [solidity_symbex]
        ShowL_{int}_def [solidity_symbex]

value ‹ReadL_{int} (ShowL_{int}   10)  =  10›
value ‹ReadL_{int} (ShowL_{int} (-10)) = -10›

value ‹ShowL_{int} (ReadL_{int} (STR ''10''))  =  STR ''10''›
value ‹ShowL_{int} (ReadL_{int} (STR ''-10'')) = STR ''-10''›

lemma Show_ReadL_id: ‹strL_is_int' s ⟹ (ShowL_{int} (ReadL_{int} s) = s)›
  by(simp add:strL_is_int'_def)

lemma Read_ShowL_id: ‹ReadL_{int} (ShowL_{int} x) = x›
proof(cases ‹x < 0›)
  case True
  then show ?thesis using ShowL_{int}_def ReadL_{int}_def Show_{int}_def Show_{nat}_ascii
    by (metis (no_types, lifting) Read_Show_id String.ascii_of_Char comp_def implode.rep_eq
list.simps(9))
next
  case False
  then show ?thesis using ShowL_{int}_def ReadL_{int}_def Show_{int}_def Show_{nat}_ascii
    by (metis Read_Show_id String.explode_implode_eq comp_apply)
qed

lemma STR_is_int_ShowL: ‹strL_is_int' (ShowL_{int} n)›
  by(simp add:strL_is_int'_def Read_ShowL_id)

lemma String_Cancel: "a + (c::String.literal) = b + c ⟹ a = b"
using String.plus_literal.rep_eq
by (metis append_same_eq literal.explode_inject)

end
theory StateMonad
imports Main "HOL-Library.Monad_Syntax"
begin
```

2.2 State Monad with Exceptions (StateMonad)

```
datatype ('n, 'e) result = Normal 'n | Exception 'e

type_synonym ('a, 'e, 's) state_monad = "'s ⇒ ('a × 's, 'e) result"

lemma result_cases[cases type: result]:
  fixes x :: "('a × 's, 'e) result"
  obtains (n) a s where "x = Normal (a, s)"
        | (e) e where "x = Exception e"
proof (cases x)
    case (Normal n)
    then show ?thesis
      by (metis n prod.swap_def swap_swap)
  next
    case (Exception e)
    then show ?thesis ..
qed
```

2.2.1 Fundamental Definitions

```
fun
  return :: "'a ⇒ ('a, 'e, 's) state_monad" where
  "return a s = Normal (a, s)"

fun
  throw :: "'e ⇒ ('a, 'e, 's) state_monad" where
  "throw e s = Exception e"

fun
  bind :: "('a, 'e, 's) state_monad ⇒ ('a ⇒ ('b, 'e, 's) state_monad) ⇒
           ('b, 'e, 's) state_monad" (infixl ">>=" 60)
  where
    "bind f g s = (case f s of
                     Normal (a, s') ⇒ g a s'
                   | Exception e ⇒ Exception e)"

adhoc_overloading Monad_Syntax.bind bind

lemma throw_left[simp]: "throw x ⤜ y = throw x" by auto
```

2.2.2 The Monad Laws

return is absorbed at the left of a *(⤜)*, applying the return value directly:

```
lemma return_bind [simp]: "(return x ⤜ f) = f x"
  by auto
```

return is absorbed on the right of a *(⤜)*

```
lemma bind_return [simp]: "(m ⤜ return) = m"
proof -
  have "∀s. bind m return s = m s"
  proof
    fix s
    show "bind m return s = m s"
    proof (cases "m s" rule: result_cases)
      case (n a s)
      then show ?thesis by auto
    next
      case (e e)
      then show ?thesis by auto
    qed
  qed
  thus ?thesis by auto
qed
```

(≫=) is associative

```
lemma bind_assoc:
  fixes m :: "('a,'e,'s) state_monad"
  fixes f :: "'a ⇒ ('b,'e,'s) state_monad"
  fixes g :: "'b ⇒ ('c,'e,'s) state_monad"
  shows "(m ≫= f) ≫= g  =  m ≫= (λx. f x≫= g)"
proof
  fix s
  show "bind (bind m f) g s = bind m (λx. bind (f x) g) s"
    by (cases "m s" rule: result_cases, simp+)
qed
```

2.2.3 Basic Conguruence Rules

```
lemma monad_cong[fundef_cong]:
  fixes m1 m2 m3 m4
  assumes "m1 s = m2 s"
      and "⋀v s'. m2 s = Normal (v, s') ⟹ m3 v s' = m4 v s'"
    shows "(bind m1 m3) s = (bind m2 m4) s"
  apply(insert assms, cases "m1 s")
  apply (metis StateMonad.bind.simps old.prod.case result.simps(5))
  by (metis bind.elims result.simps(6))


lemma bind_case_nat_cong [fundef_cong]:
  assumes "x = x'" and "⋀a. x = Suc a ⟹ f a h = f' a h"
  shows "(case x of Suc a ⇒ f a | 0 ⇒ g) h = (case x' of Suc a ⇒ f' a | 0 ⇒ g) h"
  by (metis assms(1) assms(2) old.nat.exhaust old.nat.simps(4) old.nat.simps(5))

lemma if_cong[fundef_cong]:
  assumes "b = b'"
    and "b' ⟹ m1 s = m1' s"
    and "¬ b' ⟹ m2 s = m2' s"
  shows "(if b then m1 else m2) s = (if b' then m1' else m2') s"
  using assms(1) assms(2) assms(3) by auto

lemma bind_case_pair_cong [fundef_cong]:
  assumes "x = x'" and "⋀a b. x = (a,b) ⟹ f a b s = f' a b s"
  shows "(case x of (a,b) ⇒ f a b) s = (case x' of (a,b) ⇒ f' a b) s"
  by (simp add: assms(1) assms(2) prod.case_eq_if)

lemma bind_case_let_cong [fundef_cong]:
  assumes "M = N"
      and "(⋀x. x = N ⟹ f x s = g x s)"
    shows "(Let M f) s = (Let N g) s"
  by (simp add: assms(1) assms(2))

lemma bind_case_some_cong [fundef_cong]:
  assumes "x = x'" and "⋀a. x = Some a ⟹ f a s = f' a s" and "x = None ⟹ g s = g' s"
  shows "(case x of Some a ⇒ f a | None ⇒ g) s = (case x' of Some a ⇒ f' a | None ⇒ g') s"
  by (simp add: assms(1) assms(2) assms(3) option.case_eq_if)
```

2.2.4 Other functions

The basic accessor functions of the state monad. *get* returns the current state as result, does not fail, and does not change the state. *put s* returns unit, changes the current state to *s* and does not fail.

```
fun get :: "('s, 'e, 's) state_monad" where
  "get s = Normal (s, s)"

fun put :: "'s ⇒ (unit, 'e, 's) state_monad" where
  "put s _ = Normal ((), s)"
```

Apply a function to the current state and return the result without changing the state.

```
fun
  applyf :: "('s ⇒ 'a) ⇒ ('a, 'e, 's) state_monad" where
 "applyf f = get ≫= (λs. return (f s))"
```

Modify the current state using the function passed in.

```
fun
  modify :: "('s ⇒ 's) ⇒ (unit, 'e, 's) state_monad"
where "modify f = get ≫= (λs::'s. put (f s))"
```

Perform a test on the current state, performing the left monad if the result is true or the right monad if the result is false.

```
fun
  condition :: "('s ⇒ bool) ⇒ ('a, 'e, 's) state_monad ⇒ ('a, 'e, 's) state_monad ⇒ ('a, 'e, 's)
state_monad"
where
  "condition P L R s = (if (P s) then (L s) else (R s))"

notation (output)
  condition  ("(condition (_)//  (_)//  (_))" [1000,1000,1000] 1000)

lemma condition_cong[fundef_cong]:
  assumes "b s = b' s"
    and "b' s ⟹ m1 s = m1' s"
    and "⋀s'. s' = s ⟹ ¬ b' s' ⟹ m2 s' = m2' s'"
  shows "(condition b m1 m2) s = (condition b' m1' m2') s"
  by (simp add: assms(1) assms(2) assms(3))

fun
  assert :: "'e ⇒ ('s ⇒ bool) ⇒ ('a, 'e, 's) state_monad ⇒ ('a, 'e, 's) state_monad" where
 "assert x t m = condition t (throw x) m"

notation (output)
  assert  ("(assert (_)//  (_)//  (_))" [1000,1000,1000] 1000)

lemma assert_cong[fundef_cong]:
  assumes "b s = b' s"
    and "¬ b' s ⟹ m s = m' s"
  shows "(assert x b m) s = (assert x b' m') s"
  by (simp add: assms(1) assms(2))
```

2.2.5 Some basic examples

```
lemma "do {
        x ← return 1;
        return (2::nat);
        return x
       } =
       return 1 ≫= (λx. return (2::nat) ≫= (λ_. (return x)))" ..

lemma "do {
        x ← return 1;
          return 2;
          return x
       } = return 1"
  by auto

fun sub1 :: "(unit, nat, nat) state_monad" where
    "sub1 0 = put 0 0"
  | "sub1 (Suc n) = (do {
                      x ← get;
                      put x;
                      sub1
                     }) n"
```

```
fun sub2 :: "(unit, nat, nat) state_monad" where
  "sub2 s =
     (do {
        n ← get;
        (case n of
          0 ⇒ put 0
        | Suc n' ⇒ (do {
                  put n';
                  sub2
                }))
     }) s"

fun sub3 :: "(unit, nat, nat) state_monad" where
  "sub3 s =
     condition (λn. n=0)
       (return ())
       (do {
         n ← get;
         put (n - 1);
         sub3
        }) s"

fun sub4 :: "(unit, nat, nat) state_monad" where
  "sub4 s =
     assert (0) (λn. n=0)
       (do {
         n ← get;
         put (n - 1);
         sub4
        }) s"

fun sub5 :: "(unit, nat, (nat*nat)) state_monad" where
  "sub5 s =
     assert (0) (λn. fst n=0)
       (do {
         (n,m) ← get;
          put (n - 1,m);
          sub5
        }) s"
end
```

3 Types and Accounts

In this chapter, we discuss the basic data types of Solidity and the representations of accounts.

3.1 Value Types (Valuetypes)

```
theory Valuetypes
imports ReadShow
begin

fun iter :: "(int ⇒ 'b ⇒ 'b) ⇒ 'b ⇒ int ⇒ 'b"
where
  "iter f v x = (if x ≤ 0 then v
                else f (x-1) (iter f v (x-1)))"

fun iter' :: "(int ⇒ 'b ⇒ 'b option) ⇒ 'b ⇒ int ⇒ 'b option"
where
  "iter' f v x = (if x ≤ 0 then Some v
                 else case iter' f v (x-1) of
                         Some v' ⇒ f (x-1) v'
                       | None ⇒ None)"

type_synonym Address = String.literal
type_synonym Location = String.literal
type_synonym Valuetype = String.literal


datatype Types = TSInt nat
               | TUInt nat
               | TBool
               | TAddr


fun createSInt :: "nat ⇒ int ⇒ Valuetype"
where
  "createSInt b v =
    (if v ≥ 0
      then ShowL_int (-(2^(b-1)) + (v+2^(b-1)) mod (2^b))
      else ShowL_int (2^(b-1) - (-v+2^(b-1)-1) mod (2^b) - 1))"

lemma upper_bound:
  fixes b::nat
    and c::int
  assumes "b > 0"
      and "c < 2^(b-1)"
    shows "c + 2^(b-1) < 2^b"
proof -
  have a1: "⋀P. (∀b::nat. P b) ⟹ (∀b>0. P ((b-1)::nat))" by simp
  have b2: "∀b::nat. (∀(c::int)<2^b. (c + 2^b) < 2^(Suc b))" by simp
  show ?thesis using a1[OF b2] assms by simp
qed

lemma upper_bound2:
  fixes b::nat
      and c::int
    assumes "b > 0"
      and "c < 2^b"
      and "c ≥ 0"
```

```
    shows "c - (2^(b-1)) < 2^(b-1)"
proof -
  have a1: "⋀P. (∀b::nat. P b) ⟹ (∀b>0. P ((b-1)::nat))" by simp
  have b2: "∀b::nat. (∀(c::int)<2^(Suc b). c≥0 ⟶ (c - 2^b) < 2^b)" by simp
  show ?thesis using a1[OF b2] assms by simp
qed

lemma upper_bound3:
  fixes b::nat
    and v::int
      defines "x ≡ - (2 ^ (b - 1)) + (v + 2 ^ (b - 1)) mod 2 ^ b"
    assumes "b>0"
    shows "x < 2^(b-1)"
  using upper_bound2 assms by auto

lemma lower_bound:
    fixes b::nat
  assumes "b>0"
    shows "∀(c::int) ≥ -(2^(b-1)). (-c + 2^(b-1) - 1 < 2^b)"
proof -
  have a1: "⋀P. (∀b::nat. P b) ⟹ (∀b>0. P ((b-1)::nat))" by simp
  have b2: "∀b::nat. ∀(c::int) ≥ -(2^b). (-c + (2^b) - 1) < 2^(Suc b)" by simp
  show ?thesis using a1[OF b2] assms by simp
qed

lemma lower_bound2:
  fixes b::nat
    and v::int
      defines "x ≡ 2^(b - 1) - (-v+2^(b-1)-1) mod 2^b - 1"
    assumes "b>0"
    shows "x ≥ - (2 ^ (b - 1))"
  using upper_bound2 assms by auto

lemma createSInt_id_g0:
    fixes b::nat
      and v::int
  assumes "v ≥ 0"
      and "v < 2^(b-1)"
      and "b > 0"
    shows "createSInt b v = ShowL_int v"
proof -
  from assms have "v + 2^(b-1) ≥ 0" by simp
  moreover from assms have "v + (2^(b-1)) < 2^b" using upper_bound[of b] by auto
  ultimately have "(v + 2^(b-1)) mod (2^b) = v + 2^(b-1)" by simp
  moreover from assms have "createSInt b v=ShowL_int (-(2^(b-1)) + (v+2^(b-1)) mod (2^b))" by simp
  ultimately show ?thesis by simp
qed

lemma createSInt_id_l0:
    fixes b::nat
      and v::int
  assumes "v < 0"
      and "v ≥ -(2^(b-1))"
      and "b > 0"
    shows "createSInt b v = ShowL_int v"
proof -
  from assms have "-v + 2^(b-1) - 1 ≥ 0" by simp
  moreover from assms have "-v + 2^(b-1) - 1 < 2^b" using lower_bound[of b] by auto
  ultimately have "(-v + 2^(b-1) - 1) mod (2^b) = (-v + 2^(b-1) - 1)" by simp
  moreover from assms have "createSInt b v= ShowL_int (2^(b-1) - (-v+2^(b-1)-1) mod (2^b) - 1)" by
simp
  ultimately show ?thesis by simp
qed
```

```
lemma createSInt_id:
    fixes b::nat
      and v::int
  assumes "v < 2^(b-1)"
      and "v ≥ -(2^(b-1))"
      and "b > 0"
    shows "createSInt b v = ShowL_int v" using createSInt_id_g0 createSInt_id_l0 assms by simp


fun createUInt :: "nat ⇒ int ⇒ Valuetype"
  where "createUInt b v = ShowL_int (v mod (2^b))"

lemma createUInt_id:
  assumes "v ≥ 0"
      and "v < 2^b"
    shows "createUInt b v =  ShowL_int v"
by (simp add: assms(1) assms(2))

fun createBool :: "bool ⇒ Valuetype"
where
  "createBool b = ShowL_bool b"

fun createAddress :: "Address ⇒ Valuetype"
where
  "createAddress ad = ad"

fun convert :: "Types ⇒ Types ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "convert (TSInt b1) (TSInt b2) v =
    (if b1 ≤ b2
      then Some (v, TSInt b2)
      else None)"
| "convert (TUInt b1) (TUInt b2) v =
    (if b1 ≤ b2
      then Some (v, TUInt b2)
      else None)"
| "convert (TUInt b1) (TSInt b2) v =
    (if b1 < b2
      then Some (v, TSInt b2)
      else None)"
| "convert TBool TBool v = Some (v, TBool)"
| "convert TAddr TAddr v = Some (v, TAddr)"
| "convert _ _ _ = None"

lemma convert_id[simp]:
  "convert tp tp kv = Some (kv, tp)"
    by (metis Types.exhaust convert.simps(1) convert.simps(2) convert.simps(4) convert.simps(5)
order_refl)


fun olift ::
  "(int ⇒ int ⇒ int) ⇒ Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "olift op (TSInt b1) (TSInt b2) v1 v2 =
    Some (createSInt (max b1 b2) (op ⌈v1⌉ ⌈v2⌉), TSInt (max b1 b2))"
| "olift op (TUInt b1) (TUInt b2) v1 v2 =
    Some (createUInt (max b1 b2) (op ⌈v1⌉ ⌈v2⌉), TUInt (max b1 b2))"
| "olift op (TSInt b1) (TUInt b2) v1 v2 =
    (if b2 < b1
      then Some (createSInt b1 (op ⌈v1⌉ ⌈v2⌉), TSInt b1)
      else None)"
| "olift op (TUInt b1) (TSInt b2) v1 v2 =
    (if b1 < b2
      then Some (createSInt b2 (op ⌈v1⌉ ⌈v2⌉), TSInt b2)
```

```
      else None)"
| "olift _ _ _ _ _ = None"
```

fun *plift ::*

```
  "(int ⇒ int ⇒ bool) ⇒ Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "plift op (TSInt b1) (TSInt b2) v1 v2 = Some (createBool (op ⌈v1⌉ ⌈v2⌉), TBool)"
| "plift op (TUInt b1) (TUInt b2) v1 v2 = Some (createBool (op ⌈v1⌉ ⌈v2⌉), TBool)"
| "plift op (TSInt b1) (TUInt b2) v1 v2 =
    (if b2 < b1
      then Some (createBool (op ⌈v1⌉ ⌈v2⌉), TBool)
      else None)"
| "plift op (TUInt b1) (TSInt b2) v1 v2 =
    (if b1 < b2
      then Some (createBool (op ⌈v1⌉ ⌈v2⌉), TBool)
      else None)"
| "plift _ _ _ _ _ = None"
```

```
definition add :: "Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "add = olift (+)"
```

```
definition sub :: "Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "sub = olift (-)"
```

```
definition equal :: "Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "equal = plift (=)"
```

```
definition less :: "Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "less = plift (<)"

declare less_def [solidity_symbex]
```

```
definition leq :: "Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "leq = plift (≤)"
```

```
fun vtand :: "Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "vtand TBool TBool a b =
    (if a = ShowL_bool True ∧ b = ShowL_bool True then Some (ShowL_bool True, TBool)
    else Some (ShowL_bool False, TBool))"
| "vtand _ _ _ _ = None"
```

```
fun vtor :: "Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option"
where
  "vtor TBool TBool a b =
    (if a = ShowL_bool False ∧ b = ShowL_bool False
      then Some (ShowL_bool False, TBool)
      else Some (ShowL_bool True, TBool))"
| "vtor _ _ _ _ = None"
```

```
fun ival :: "Types ⇒ Valuetype"
where
  "ival (TSInt x) = ShowL_int 0"
| "ival (TUInt x) = ShowL_int 0"
| "ival TBool = ShowL_bool False"
| "ival TAddr = STR ''0x0000000000000000000000000000000000000000''"

end
```

4 Stores and Environment

In this chapter, we focus on a particular aspect of Solidity that is different to most programming languages: the handling of memory in general and, in particular, the different between store and storage.

4.1 Storage (Storage)

```
theory Storage
imports Valuetypes "HOL-Library.Finite_Map"

begin


fun hash :: "Location ⇒ String.literal ⇒ Location"
where "hash loc ix = ix + (STR ''.'' + loc)"
```

4.1.1 General Store

```
record 'v Store =
  mapping :: "(Location,'v) fmap"
  toploc :: nat

fun accessStore :: "Location ⇒ 'v Store ⇒ 'v option"
where "accessStore loc st = fmlookup (mapping st) loc"

definition emptyStore :: "'v Store"
where "emptyStore = (| mapping=fmempty, toploc=0 |)"

declare emptyStore_def [solidity_symbex]

fun allocate :: "'v Store ⇒ Location * ('v Store)"
where "allocate s = (let ntop = Suc(toploc s) in (ShowL_nat ntop, s (|toploc := ntop|)))"

fun updateStore :: "Location ⇒ 'v ⇒ 'v Store ⇒ 'v Store"
where "updateStore loc val s = s (| mapping := fmupd loc val (mapping s)|)"

fun push :: "'v ⇒ 'v Store ⇒ 'v Store"
  where "push val sto = (let s = updateStore (ShowL_nat (toploc sto)) val sto in snd (allocate s))"
```

4.1.2 Stack

```
datatype Stackvalue = KValue Valuetype
                    | KCDptr Location
                    | KMemptr Location
                    | KStoptr Location


type_synonym Stack = "Stackvalue Store"
```

4.1.3 Storage

Definition

```
type_synonym Storagevalue = Valuetype


type_synonym StorageT = "(Location,Storagevalue) fmap"
```

```
datatype STypes = STArray int STypes
                | STMap Types STypes
                | STValue Types
```

Example

```
abbreviation mystorage::StorageT
where "mystorage ≡ (fmap_of_list
  [(STR ''0.0.1'', STR ''True''),
   (STR ''1.0.1'', STR ''False''),
   (STR ''0.1.1'', STR ''True''),
   (STR ''1.1.1'', STR ''False'')])"
```

Access storage

```
fun accessStorage :: "Types ⇒ Location ⇒ StorageT ⇒ Storagevalue"
where
  "accessStorage t loc sto =
    (case fmlookup sto loc of
      Some v ⇒ v
    | None ⇒ ival t)"
```

Copy from storage to storage

```
fun copyRec :: "Location ⇒ Location ⇒ STypes ⇒ StorageT ⇒ StorageT option"
where
  "copyRec loc loc' (STArray x t) sto =
    iter' (λi s'. copyRec (hash loc (ShowL_int i)) (hash loc' (ShowL_int i)) t s') sto x"
| "copyRec loc loc' (STValue t) sto =
    (let e = accessStorage t loc sto in Some (fmupd loc' e sto))"
| "copyRec _ _ (STMap _ _) _ = None"

fun copy :: "Location ⇒ Location ⇒ int ⇒ STypes ⇒ StorageT ⇒ StorageT option"
where
  "copy loc loc' x t sto =
    iter' (λi s'. copyRec (hash loc (ShowL_int i)) (hash loc' (ShowL_int i)) t s') sto x"
```

4.1.4 Memory and Calldata

Definition

```
datatype Memoryvalue =
  MValue Valuetype
  | MPointer Location

type_synonym MemoryT = "Memoryvalue Store"

type_synonym CalldataT = MemoryT

datatype MTypes = MTArray int MTypes
  | MTValue Types
```

Example

```
abbreviation mymemory::MemoryT
  where "mymemory ≡
    ⦇mapping = fmap_of_list
      [(STR ''1.1.0'', MValue STR ''False''),
       (STR ''0.1.0'', MValue STR ''True''),
       (STR ''1.0'', MPointer STR ''1.0''),
       (STR ''1.0.0'', MValue STR ''False''),
       (STR ''0.0.0'', MValue STR ''True''),
       (STR ''0.0'', MPointer STR ''0.0'')],
     toploc = 1⦈"
```

Initialization

Definition

```
fun minitRec :: "Location ⇒ MTypes ⇒ MemoryT ⇒ MemoryT"
where
  "minitRec loc (MTArray x t) = (λmem.
    let m = updateStore loc (MPointer loc) mem
    in iter (λi m' . minitRec (hash loc (ShowL_int i)) t m') m x)"
| "minitRec loc (MTValue t) = updateStore loc (MValue (ival t))"
```

```
fun minit :: "int ⇒ MTypes ⇒ MemoryT ⇒ MemoryT"
where
  "minit x t mem =
    (let l = ShowL_nat (toploc mem);
         m = iter (λi m' . minitRec (hash l (ShowL_int i)) t m') mem x
     in snd (allocate m))"
```

Example

```
lemma "minit 2 (MTArray 2 (MTValue TBool)) emptyStore =
⦇mapping = fmap_of_list
  [(STR ''0.0'', MPointer STR ''0.0''), (STR ''0.0.0'', MValue STR ''False''),
   (STR ''1.0.0'', MValue STR ''False''), (STR ''1.0'', MPointer STR ''1.0''),
   (STR ''0.1.0'', MValue STR ''False''), (STR ''1.1.0'', MValue STR ''False'')],
  toploc = 1⦈" by eval
```

Copy from memory to memory

Definition

```
fun cpm2mrec :: "Location ⇒ Location ⇒ MTypes ⇒ MemoryT ⇒ MemoryT ⇒ MemoryT option"
where
  "cpm2mrec l_s l_d (MTArray x t) m_s m_d =
    (case accessStore l_s m_s of
      Some e ⇒
        (case e of
          MPointer l ⇒ (let m = updateStore l_d (MPointer l_d) m_d
             in iter' (λi m'. cpm2mrec (hash l_s (ShowL_int i)) (hash l_d (ShowL_int i)) t m_s m') m x)
        | _ ⇒ None)
    | None ⇒ None)"
| "cpm2mrec l_s l_d (MTValue t) m_s m_d =
    (case accessStore l_s m_s of
      Some e ⇒ (case e of
          MValue v ⇒ Some (updateStore l_d (MValue v) m_d)
        | _ ⇒ None)
    | None ⇒ None)"
```

```
fun cpm2m :: "Location ⇒ Location ⇒ int ⇒ MTypes ⇒ MemoryT ⇒ MemoryT ⇒ MemoryT option"
where
  "cpm2m l_s l_d x t m_s m_d = iter' (λi m. cpm2mrec (hash l_s (ShowL_int i)) (hash l_d (ShowL_int i)) t m_s m)
m_d x"
```

Example

```
lemma "cpm2m (STR ''0'') (STR ''0'') 2 (MTArray 2 (MTValue TBool)) mymemory (snd (allocate
emptyStore)) = Some mymemory"
  by eval
```

4.1.5 Copy from storage to memory

Definition

```
fun cps2mrec :: "Location ⇒ Location ⇒ STypes ⇒ StorageT ⇒ MemoryT ⇒ MemoryT option"
where
```

```
  "cps2mrec locs locm (STArray x t) sto mem =
    (let m = updateStore locm (MPointer locm) mem
    in iter' (λi m'. cps2mrec (hash locs (ShowL_int i)) (hash locm (ShowL_int i)) t sto m') m x)"
| "cps2mrec locs locm (STValue t) sto mem =
    (let v = accessStorage t locs sto
    in Some (updateStore locm (MValue v) mem))"
| "cps2mrec _ _ (STMap _ _) _ _ = None"
```

```
fun cps2m :: "Location ⇒ Location ⇒ int ⇒ STypes ⇒ StorageT ⇒ MemoryT ⇒ MemoryT option"
where
  "cps2m locs locm x t sto mem =
    iter' (λi m'. cps2mrec (hash locs (ShowL_int i)) (hash locm (ShowL_int i)) t sto m') mem x"
```

Example

```
lemma "cps2m (STR ''1'') (STR ''0'') 2 (STArray 2 (STValue TBool)) mystorage (snd (allocate
emptyStore)) = Some mymemory"
  by eval
```

4.1.6 Copy from memory to storage

Definition

```
fun cpm2srec :: "Location ⇒ Location ⇒ MTypes ⇒ MemoryT ⇒ StorageT ⇒ StorageT option"
where
  "cpm2srec locm locs (MTArray x t) mem sto =
    (case accessStore locm mem of
      Some e ⇒
        (case e of
          MPointer l ⇒ iter' (λi s'. cpm2srec (hash locm (ShowL_int i)) (hash locs (ShowL_int i)) t mem
s') sto x
        | _ ⇒ None)
    | None ⇒ None)"
| "cpm2srec locm locs (MTValue t) mem sto =
    (case accessStore locm mem of
      Some e ⇒ (case e of
          MValue v ⇒ Some (fmupd locs v sto)
        | _ ⇒ None)
    | None ⇒ None)"
```

```
fun cpm2s :: "Location ⇒ Location ⇒ int ⇒ MTypes ⇒ MemoryT ⇒ StorageT ⇒ StorageT option"
where
  "cpm2s locm locs x t mem sto =
    iter' (λi s'. cpm2srec (hash locm (ShowL_int i)) (hash locs (ShowL_int i)) t mem s') sto x"
```

Example

```
lemma "cpm2s (STR ''0'') (STR ''1'') 2 (MTArray 2 (MTValue TBool)) mymemory fmempty = Some mystorage"
  by eval

end
```

4.2 Environment and State (Environment)

```
theory Environment
imports Accounts Storage StateMonad
begin
```

4.2.1 Environment

```
datatype Type = Value Types
              | Calldata MTypes
```

```
              | Memory MTypes
              | Storage STypes

datatype Denvalue = Stackloc Location
                  | Storeloc Location

type_synonym Identifier = String.literal

record Environment =
  address :: Address
  sender :: Address
  svalue :: Valuetype
  denvalue :: "(Identifier, Type × Denvalue) fmap"

fun identifiers :: "Environment ⇒ Identifier fset"
  where "identifiers e = fmdom (denvalue e)"

fun emptyEnv :: "Address ⇒ Address ⇒ Valuetype ⇒ Environment"
  where "emptyEnv a s v = ⦇address = a, sender = s, svalue =v, denvalue = fmempty⦈"

definition eempty :: "Environment"
  where "eempty = emptyEnv (STR '''') (STR '''') (STR '''')"

declare eempty_def [solidity_symbex]

fun updateEnv :: "Identifier ⇒ Type ⇒ Denvalue ⇒ Environment ⇒ Environment"
  where "updateEnv i t v e = e ⦇ denvalue := fmupd i (t,v) (denvalue e) ⦈"

fun updateEnvOption :: "Identifier ⇒ Type ⇒ Denvalue ⇒ Environment ⇒ Environment option"
  where "updateEnvOption i t v e = (case fmlookup (denvalue e) i of
            Some _ ⇒ None
          | None ⇒ Some (updateEnv i t v e))"

lemma updateEnvOption_address: "updateEnvOption i t v e = Some e' ⟹ address e = address e'"
by (auto split:option.split_asm)

fun updateEnvDup :: "Identifier ⇒ Type ⇒ Denvalue ⇒ Environment ⇒ Environment"
  where "updateEnvDup i t v e = (case fmlookup (denvalue e) i of
            Some _ ⇒ e
          | None ⇒ updateEnv i t v e)"

lemma updateEnvDup_address[simp]: "address (updateEnvDup i t v e) = address e"
  by (auto split:option.split)

lemma updateEnvDup_sender[simp]: "sender (updateEnvDup i t v e) = sender e"
  by (auto split:option.split)

lemma updateEnvDup_svalue[simp]: "svalue (updateEnvDup i t v e) = svalue e"
  by (auto split:option.split)

lemma updateEnvDup_dup:
  assumes "i≠i'" shows "fmlookup (denvalue (updateEnvDup i t v e)) i' = fmlookup (denvalue e) i'"
proof (cases "fmlookup (denvalue e) i = None")
  case True
  then show ?thesis using assms by simp
next
  case False
  then obtain e' where "fmlookup (denvalue e) i = Some e'" by auto
  then show ?thesis using assms by simp
qed

lemma env_reorder_neq:
  assumes "x≠y"
  shows "updateEnv x t1 v1 (updateEnv y t2 v2 e) = updateEnv y t2 v2 (updateEnv x t1 v1 e)"
```

```
proof -
  have "address (updateEnv x t1 v1 (updateEnv y t2 v2 e)) = address (updateEnv y t2 v2 (updateEnv x t1
v1 e))" by simp
  moreover from assms have "denvalue (updateEnv x t1 v1 (updateEnv y t2 v2 e)) = denvalue (updateEnv
y t2 v2 (updateEnv x t1 v1 e))" using Finite_Map.fmupd_reorder_neq[of x y "(t1,v1)" "(t2,v2)"] by simp
  ultimately show ?thesis by simp
qed

lemma uEO_in:
  assumes "i |∈| fmdom (denvalue e)"
  shows "updateEnvOption i t v e = None"
  using assms by auto

lemma uEO_n_In:
  assumes "¬ i |∈| fmdom (denvalue e)"
  shows "updateEnvOption i t v e = Some (updateEnv i t v e)"
  using assms by auto

fun astack :: "Identifier ⇒ Type ⇒ Stackvalue ⇒ Stack * Environment ⇒ Stack * Environment"
  where "astack i t v (s, e) = (push v s, (updateEnv i t (Stackloc (ShowL_nat (toploc s))) e))"
```

4.2.2 State

```
type_synonym Gas = nat

record State =
  accounts :: Accounts
  stack :: Stack
  memory :: MemoryT
  storage :: "(Address,StorageT) fmap"
  gas :: Gas

datatype Ex = Gas | Err

fun append :: "Identifier ⇒ Type ⇒ Stackvalue
  ⇒ CalldataT ⇒ Environment ⇒ (CalldataT × Environment, Ex, State) state_monad"
where
  "append id0 tp v cd e st =
    (let (k, e') = astack id0 tp v (stack st, e)
    in do {
      modify (λst. st ⦇stack := k⦈);
      return (cd, e')
    }) st"
```

4.2.3 Declarations

This function is used to declare a new variable: decl id tp val copy cd mem cd' env st

id is the name of the variable

tp is the type of the variable

val is an optional initialization parameter. If it is None, the types default value is taken.

copy is a flag to indicate whether memory should be copied (from mem parameter) or not (copying is required for example for external method calls).

cd is the original calldata which is used as a source

mem is the original memory which is used as a source

cd' is the new calldata

env is the new environment

st is the new state

```
fun decl :: "Identifier ⇒ Type ⇒ (Stackvalue * Type) option ⇒ bool ⇒ CalldataT ⇒ MemoryT
    ⇒ CalldataT ⇒ Environment ⇒ (CalldataT × Environment, Ex, State) state_monad"
  where

  "decl i (Value t) None _ _ _ c env st = append i (Value t) (KValue (ival t)) c env st"
| "decl i (Value t) (Some (KValue v, Value t')) _ _ _ c env st =
    (case convert t' t v of
      Some (v', t'') ⇒ append i (Value t'') (KValue v') c env
    | None ⇒ throw Err) st"
| "decl _ (Value _) (Some _) _ _ _ _ _ st = throw Err st"


| "decl i (Calldata (MTArray x t)) (Some (KCDptr p, _)) True cd _ c env st =
    (let l = ShowL_nat (toploc c);
         (_, c') = allocate c
     in (case cpm2m p l x t cd c' of
          Some c'' ⇒ append i (Calldata (MTArray x t)) (KCDptr l) c'' env
        | None ⇒ throw Err)) st"
| "decl i (Calldata (MTArray x t)) (Some (KMemptr p, _)) True _ mem c env st =
    (let l = ShowL_nat (toploc c);
         (_, c') = allocate c
     in (case cpm2m p l x t mem c' of
          Some c'' ⇒ append i (Calldata (MTArray x t)) (KCDptr l) c'' env
        | None ⇒ throw Err)) st"
| "decl i (Calldata _) _ _ _ _ _ _ st = throw Err st"


| "decl i (Memory (MTArray x t)) None _ _ _ c env st =
    (do {
      m ← applyf (λst. memory st);
      modify (λst. st ⦇memory := minit x t m⦈);
      append i (Memory (MTArray x t)) (KMemptr (ShowL_nat (toploc m))) c env
    }) st"
| "decl i (Memory (MTArray x t)) (Some (KMemptr p, _)) True _ mem c env st =
    (do {
      m ← (applyf (λst. memory st));
      (case cpm2m p (ShowL_nat (toploc m)) x t mem (snd (allocate m)) of
         Some m' ⇒
          do {
           modify (λst. st ⦇memory := m'⦈);
           append i (Memory (MTArray x t)) (KMemptr (ShowL_nat (toploc m))) c env
          }
       | None ⇒ throw Err)
    }) st"
| "decl i (Memory (MTArray x t)) (Some (KMemptr p, _)) False _ _ c env st =
   append i (Memory (MTArray x t)) (KMemptr p) c env st"
| "decl i (Memory (MTArray x t)) (Some (KCDptr p, _)) _ cd _ c env st =
    (do {
      m ← (applyf (λst. memory st));
      (case cpm2m p (ShowL_nat (toploc m)) x t cd (snd (allocate m)) of
        Some m' ⇒
          do {
          modify (λst. st ⦇memory := m'⦈);
          append i (Memory (MTArray x t)) (KMemptr (ShowL_nat (toploc m))) c env
          }
      | None ⇒ throw Err)
    }) st"
| "decl i (Memory (MTArray x t)) (Some (KStoptr p, Storage (STArray x' t'))) _ _ _ c env st =
    (do {
      s ← (applyf (λst. storage st));
      (case fmlookup s (address env) of
        Some s' ⇒
        (do {
          m ← (applyf (λst. memory st));
```

```
          (case cps2m p (ShowL_nat (toploc m)) x' t' s' (snd (allocate m)) of
              Some m'' ⇒
              do {
                modify (λst. st (|memory := m''|));
              append i (Memory (MTArray x t)) (KMemptr (ShowL_nat (toploc m))) c env
              }
            | None ⇒ throw Err)
        })
      | None ⇒ throw Err)
    }) st"
| "decl _ (Memory (MTArray _ _)) (Some _) _ _ _ _ _ st = throw Err st"
| "decl _ (Memory (MTValue _)) _ _ _ _ _ _ st = throw Err st"


| "decl _ (Storage (STArray _ _)) None _ _ _ _ _ st = throw Err st"
| "decl i (Storage (STArray x t)) (Some (KStoptr p, _)) _ _ _ c env st =
    append i (Storage (STArray x t)) (KStoptr p) c env st"
| "decl _ (Storage (STArray _ _)) (Some _) _ _ _ _ _ st = throw Err st"
| "decl _ (Storage (STMap _ _)) None _ _ _ _ _ st = throw Err st"
| "decl i (Storage (STMap t t')) (Some (KStoptr p, _)) _ _ _ c env st =
  append i (Storage (STMap t t')) (KStoptr p) c env st"
| "decl _ (Storage (STMap _ _)) (Some _) _ _ _ _ _ st = throw Err st"
| "decl _ (Storage (STValue _)) _ _ _ _ _ _ st = throw Err st"


lemma decl_gas_address:
  assumes "decl a1 a2 a3 cp cd mem c env st = Normal ((l1', t1'), st1')"
    shows "gas st1' = gas st ∧ address env = address t1' ∧ sender env = sender t1' ∧ svalue env =
svalue t1'"
proof (cases a2)
  case (Value t)
  then show ?thesis
  proof (cases a3)
    case None
    with Value show ?thesis using assms by auto
  next
    case (Some a)
    show ?thesis
    proof (cases a)
      case (Pair a b)
      then show ?thesis
      proof (cases a)
        case (KValue v)
        then show ?thesis
        proof (cases b)
          case v2: (Value t')
          show ?thesis
          proof (cases "convert t' t v")
            case None
            with Some Pair KValue Value v2 show ?thesis using assms by simp
          next
            case s2: (Some a)
            show ?thesis
            proof (cases a)
              case p2: (Pair a b)
              with Some Pair KValue Value v2 s2 show ?thesis using assms by auto
            qed
          qed
        next
          case (Calldata x2)
          with Some Pair KValue Value show ?thesis using assms by simp
        next
          case (Memory x3)
          with Some Pair KValue Value show ?thesis using assms by simp
```

```
        next
          case (Storage x4)
          with Some Pair KValue Value show ?thesis using assms by simp
        qed
      next
        case (KCDptr x2)
        with Some Pair Value show ?thesis using assms by simp
      next
        case (KMemptr x3)
        with Some Pair Value show ?thesis using assms by simp
      next
        case (KStoptr x4)
        with Some Pair Value show ?thesis using assms by simp
      qed
    qed
  qed
next
  case (Calldata x2)
  then show ?thesis
  proof (cases cp)
    case True
    then show ?thesis
    proof (cases x2)
      case (MTArray x t)
      then show ?thesis
      proof (cases a3)
        case None
        with Calldata show ?thesis using assms by simp
      next
        case (Some a)
        show ?thesis
        proof (cases a)
          case (Pair a b)
          then show ?thesis
          proof (cases a)
            case (KValue x1)
            with Calldata Some Pair show ?thesis using assms by simp
          next
            case (KCDptr p)
            define l where "l = ShowL_nat (toploc c)"
            obtain c' where c_def: "∃x. (x, c') = allocate c" by simp
            show ?thesis
            proof (cases "cpm2m p l x t cd c'")
              case None
              with Calldata MTArray Some Pair KCDptr l_def c_def True show ?thesis using assms by
simp
            next
              case s2: (Some a)
              with Calldata MTArray Some Pair KCDptr l_def c_def True show ?thesis using assms by
auto
            qed
          next
            case (KMemptr p)
            define l where "l = ShowL_nat (toploc c)"
            obtain c' where c_def: "∃x. (x, c') = allocate c" by simp
            show ?thesis
            proof (cases "cpm2m p l x t mem c'")
              case None
              with Calldata MTArray Some Pair KMemptr l_def c_def True show ?thesis using assms by
simp
            next
              case s2: (Some a)
              with Calldata MTArray Some Pair KMemptr l_def c_def True show ?thesis using assms by
auto
```

```
          qed
        next
          case (KStoptr x4)
          with Calldata Some Pair show ?thesis using assms by simp
        qed
      qed
    qed
  next
    case (MTValue x2)
    with Calldata show ?thesis using assms by simp
  qed
next
  case False
  with Calldata show ?thesis using assms by simp
qed
next
  case (Memory x3)
  then show ?thesis
  proof (cases x3)
    case (MTArray x t)
    then show ?thesis
    proof (cases a3)
      case None
      with Memory MTArray None show ?thesis using assms by (auto simp add:Let_def)
    next
      case (Some a)
      then show ?thesis
      proof (cases a)
        case (Pair a b)
        then show ?thesis
        proof (cases a)
          case (KValue x1)
          with Memory MTArray Some Pair show ?thesis using assms by simp
        next
          case (KCDptr p)
          define m l where "m = memory st" and "l = ShowL_nat (toploc m)"
          obtain m' where m'_def: "∃x. (x, m') = allocate m" by simp
          then show ?thesis
          proof (cases "cpm2m p l x t cd m'")
            case None
            with Memory MTArray Some Pair KCDptr m_def l_def m'_def show ?thesis using assms by simp
          next
            case s2: (Some a)
            with Memory MTArray Some Pair KCDptr m_def l_def m'_def show ?thesis using assms by auto
          qed
        next
          case (KMemptr p)
          then show ?thesis
          proof (cases cp)
            case True
            define m l where "m = memory st" and "l = ShowL_nat (toploc m)"
            obtain m' where m'_def: "∃x. (x, m') = allocate m" by simp
            then show ?thesis
            proof (cases "cpm2m p l x t mem m'")
              case None
              with Memory MTArray Some Pair KMemptr True m_def l_def m'_def show ?thesis using assms
by simp
            next
              case s2: (Some a)
              with Memory MTArray Some Pair KMemptr True m_def l_def m'_def show ?thesis using assms
by auto
            qed
          next
            case False
```

```
          with Memory MTArray Some Pair KMemptr show ?thesis using assms by auto
        qed
      next
        case (KStoptr p)
        then show ?thesis
        proof (cases b)
          case (Value x1)
          with Memory MTArray Some Pair KStoptr show ?thesis using assms by simp
        next
          case (Calldata x2)
          with Memory MTArray Some Pair KStoptr show ?thesis using assms by simp
        next
          case m2: (Memory x3)
          with Memory MTArray Some Pair KStoptr show ?thesis using assms by simp
        next
          case (Storage x4)
          then show ?thesis
          proof (cases x4)
            case (STArray x' t')
            define m l where "m = memory st" and "l = ShowL_nat (toploc m)"
            obtain m' where m'_def: "∃x. (x, m') = allocate m" by simp
            from assms(1) Memory MTArray Some Pair KStoptr Storage STArray m_def l_def m'_def
            obtain s where *: "fmlookup (storage st) (address env) = Some s" using Let_def by (auto
simp add: Let_def split:option.split_asm)
            then show ?thesis
            proof (cases "cps2m p l x' t' s m'")
              case None
              with Memory MTArray Some Pair KStoptr Storage STArray m_def l_def m'_def * show
?thesis using assms by simp
            next
              case s2: (Some a)
              with Memory MTArray Some Pair KStoptr Storage STArray m_def l_def m'_def * show
?thesis using assms by auto
            qed
          next
            case (STMap x21 x22)
            with Memory MTArray Some Pair KStoptr Storage show ?thesis using assms by simp
          next
            case (STValue x3)
            with Memory MTArray Some Pair KStoptr Storage show ?thesis using assms by simp
          qed
        qed
      qed
    qed
  qed
next
  case (MTValue x2)
  with Memory show ?thesis using assms by simp
qed
next
case (Storage x4)
then show ?thesis
proof (cases x4)
  case (STArray x t)
  then show ?thesis
  proof (cases a3)
    case None
    with Storage STArray show ?thesis using assms by simp
  next
    case (Some a)
    then show ?thesis
    proof (cases a)
      case (Pair a b)
      then show ?thesis
```

```
        proof (cases a)
          case (KValue x1)
          with Storage STArray Some Pair show ?thesis using assms by simp
        next
          case (KCDptr x2)
          with Storage STArray Some Pair show ?thesis using assms by simp
        next
          case (KMemptr x3)
          with Storage STArray Some Pair show ?thesis using assms by simp
        next
          case (KStoptr x4)
          with Storage STArray Some Pair show ?thesis using assms by auto
        qed
      qed
    qed
  next
    case (STMap t t')
    then show ?thesis
    proof (cases a3)
      case None
      with Storage STMap show ?thesis using assms by simp
    next
      case (Some a)
      then show ?thesis
      proof (cases a)
        case (Pair a b)
        then show ?thesis
        proof (cases a)
          case (KValue x1)
          with Storage STMap Some Pair show ?thesis using assms by simp
        next
          case (KCDptr x2)
          with Storage STMap Some Pair show ?thesis using assms by simp
        next
          case (KMemptr x3)
          with Storage STMap Some Pair show ?thesis using assms by simp
        next
          case (KStoptr x4)
          with Storage STMap Some Pair show ?thesis using assms by auto
        qed
      qed
    qed
  next
    case (STValue x3)
    with Storage show ?thesis using assms by simp
  qed
qed

end
```

5 Expressions and Statements

In this chapter, we formalize expressions, declarations, and statements. The results up to here form the core of our Solidity semantics.

5.1 Statements (Statements)

```
theory Statements
  imports Environment StateMonad
begin
```

5.1.1 Syntax

Expressions

```
datatype L = Id Identifier
           | Ref Identifier "E list"
and        E = INT nat int
             | UINT nat int
             | ADDRESS String.literal
             | BALANCE E
             | THIS
             | SENDER
             | VALUE
             | TRUE
             | FALSE
             | LVAL L
             | PLUS E E
             | MINUS E E
             | EQUAL E E
             | LESS E E
             | AND E E
             | OR E E
             | NOT E
             | CALL Identifier "E list"
             | ECALL E Identifier "E list" E
```

Statements

```
datatype S = SKIP
             | BLOCK "(Identifier × Type) × (E option)" S
             | ASSIGN L E
             | TRANSFER E E
             | COMP S S
             | ITE E S S
             | WHILE E S
             | INVOKE Identifier "E list"
             | EXTERNAL E Identifier "E list" E
```

```
abbreviation
  "vbits≡{8,16,24,32,40,48,56,64,72,80,88,96,104,112,120,128,
          136,144,152,160,168,176,184,192,200,208,216,224,232,240,248,256}"
```

```
lemma vbits_max[simp]:
  assumes "b1 ∈ vbits"
    and "b2 ∈ vbits"
  shows "(max b1 b2) ∈ vbits"
proof -
```

```
  consider (b1) "max b1 b2 = b1" | (b2) "max b1 b2 = b2" by (metis max_def)
  then show ?thesis
  proof cases
    case b1
    then show ?thesis using assms(1) by simp
  next
    case b2
    then show ?thesis using assms(2) by simp
  qed
qed

lemma vbits_ge_0: "(x::nat)∈vbits ⟹ x>0" by auto
```

5.1.2 Contracts

A contract consists of methods or storage variables. A method is a triple consisting of

- A list of formal parameters
- A statement
- An optional return value

```
datatype Member = Method "(Identifier × Type) list × S × E option"
                | Var STypes
```

A procedure environment assigns a contract to an address. A contract consists of

- An assignment of members to identifiers
- An optional fallback statement which is executed after money is beeing transferred to the contract.

https://docs.soliditylang.org/en/v0.8.6/contracts.html#fallback-function

```
type_synonym Environment_P = "(Address, (Identifier, Member) fmap × S) fmap"

definition init::"(Identifier, Member) fmap ⇒ Identifier ⇒ Environment ⇒ Environment"
  where "init ct i e = (case fmlookup ct i of
                          Some (Var tp) ⇒ updateEnvDup i (Storage tp) (Storeloc i) e
                        | _ ⇒ e)"

lemma init_s11[simp]:
  assumes "fmlookup ct i = Some (Var tp)"
  shows "init ct i e = updateEnvDup i (Storage tp) (Storeloc i) e"
  using assms init_def by simp

lemma init_s12[simp]:
  assumes "i |∈| fmdom (denvalue e)"
  shows "init ct i e = e"
proof (cases "fmlookup ct i")
  case None
  then show ?thesis using init_def by simp
next
  case (Some a)
  then show ?thesis
  proof (cases a)
    case (Method x1)
    with Some show ?thesis using init_def by simp
  next
    case (Var tp)
    with Some have "init ct i e = updateEnvDup i (Storage tp) (Storeloc i) e" using init_def by simp
    moreover from assms have "updateEnvDup i (Storage tp) (Storeloc i) e = e" by auto
    ultimately show ?thesis by simp
  qed
qed
```

```
lemma init_s13[simp]:
  assumes "fmlookup ct i = Some (Var tp)"
      and "¬ i |∈| fmdom (denvalue e)"
  shows "init ct i e = updateEnv i (Storage tp) (Storeloc i) e"
  using assms init_def by auto

lemma init_s21[simp]:
  assumes "fmlookup ct i = None"
  shows "init ct i e = e"
  using assms init_def by auto

lemma init_s22[simp]:
  assumes "fmlookup ct i = Some (Method m)"
  shows "init ct i e = e"
  using assms init_def by auto

lemma init_commte: "comp_fun_commute (init ct)"
proof
  fix x y
  show "init ct y ∘ init ct x = init ct x ∘ init ct y"
  proof
    fix e
    show "(init ct y ∘ init ct x) e = (init ct x ∘ init ct y) e"
    proof (cases "fmlookup ct x")
      case None
      then show ?thesis by simp
    next
      case s1: (Some a)
      then show ?thesis
      proof (cases a)
        case (Method x1)
        with s1 show ?thesis by simp
      next
        case v1: (Var tp)
        then show ?thesis
        proof (cases "x |∈| fmdom (denvalue e)")
          case True
          with s1 v1 have *: "init ct x e = e" by auto
          then show ?thesis
          proof (cases "fmlookup ct y")
            case None
            then show ?thesis by simp
          next
            case s2: (Some a)
            then show ?thesis
            proof (cases a)
              case (Method x1)
              with s2 show ?thesis by simp
            next
              case v2: (Var tp')
              then show ?thesis
              proof (cases "y |∈| fmdom (denvalue e)")
                case t1: True
                with s1 v1 True s2 v2 show ?thesis by fastforce
              next
                define e' where "e' = updateEnv y (Storage tp') (Storeloc y) e"
                case False
                with s2 v2 have "init ct y e = e'" using e'_def by auto
                with s1 v1 True e'_def * show ?thesis by auto
              qed
            qed
          qed
        next
          define e' where "e' = updateEnv x (Storage tp) (Storeloc x) e"
```

```
          case f1: False
          with s1 v1 have *: "init ct x e = e'" using e'_def by auto
          then show ?thesis
          proof (cases "fmlookup ct y")
            case None
            then show ?thesis by simp
          next
            case s3: (Some a)
            then show ?thesis
            proof (cases a)
              case (Method x1)
              with s3 show ?thesis by simp
            next
              case v2: (Var tp')
              then show ?thesis
              proof (cases "y |∈| fmdom (denvalue e)")
                case t1: True
                with e'_def have "y |∈| fmdom (denvalue e')" by simp
                with s1 s3 v1 f1 v2 show ?thesis using e'_def by fastforce
              next
                define f' where "f' = updateEnv y (Storage tp') (Storeloc y) e"
                define e'' where "e'' = updateEnv y (Storage tp') (Storeloc y) e'"
                case f2: False
                with s3 v2 have **: "init ct y e = f'" using f'_def by auto
                show ?thesis
                proof (cases "y = x")
                  case True
                  with s3 v2 e'_def have "init ct y e' = e'" by simp
                  moreover from s3 v2 True f'_def have "init ct x f' = f'" by simp
                  ultimately show ?thesis using True by simp
                next
                  define f'' where "f'' = updateEnv x (Storage tp) (Storeloc x) f'"
                  case f3: False
                  with f2 have "¬ y |∈| fmdom (denvalue e')" using e'_def by simp
                  with s3 v2 e''_def have "init ct y e' = e''" by auto
                  with * have "(init ct y ∘ init ct x) e = e''" by simp
                  moreover have "init ct x f' = f''"
                  proof -
                    from s1 v1 have "init ct x f' = updateEnvDup x (Storage tp) (Storeloc x) f'" by
simp
                    moreover from f1 f3 have "x |∉| fmdom (denvalue f')" using f'_def by simp
                    ultimately show ?thesis using f''_def by auto
                  qed
                  moreover from f''_def e''_def f'_def e'_def f3 have "Some f'' = Some e''" using
env_reorder_neq by simp
                  ultimately show ?thesis using ** by simp
                qed
              qed
            qed
          qed
        qed
      qed
    qed
  qed
qed

lemma init_address[simp]:
  "address (init ct i e) = address e ∧ sender (init ct i e) = sender e"
proof (cases "fmlookup ct i")
  case None
  then show ?thesis by simp
next
  case (Some a)
  show ?thesis
```

```
  proof (cases a)
    case (Method x1)
    with Some show ?thesis by simp
  next
    case (Var tp)
    with Some show ?thesis using updateEnvDup_address updateEnvDup_sender by simp
  qed
qed

lemma init_sender[simp]:
"sender (init ct i e) = sender e"
proof (cases "fmlookup ct i")
  case None
  then show ?thesis by simp
next
  case (Some a)
  show ?thesis
  proof (cases a)
    case (Method x1)
    with Some show ?thesis by simp
  next
    case (Var tp)
    with Some show ?thesis using updateEnvDup_sender by simp
  qed
qed

lemma init_svalue[simp]:
"svalue (init ct i e) = svalue e"
proof (cases "fmlookup ct i")
  case None
  then show ?thesis by simp
next
  case (Some a)
  show ?thesis
  proof (cases a)
    case (Method x1)
    with Some show ?thesis by simp
  next
    case (Var tp)
    with Some show ?thesis using updateEnvDup_svalue by simp
  qed
qed

lemma ffold_init_ad_same[rule_format]: "∀e'. ffold (init ct) e xs = e' ⟶ address e' = address e ∧
sender e' = sender e ∧ svalue e' = svalue e"
proof (induct xs)
  case empty
  then show ?case by (simp add: ffold_def)
next
  case (insert x xs)
  then have *: "ffold (init ct) e (finsert x xs) =
    init ct x (ffold (init ct) e xs)" using FSet.comp_fun_commute.ffold_finsert[OF init_commte] by simp
  show ?case
  proof (rule allI[OF impI])
    fix e' assume **: "ffold (init ct) e (finsert x xs) = e'"
    with * obtain e'' where ***: "ffold (init ct) e xs = e''" by simp
    with insert have "address e'' = address e ∧ sender e'' = sender e ∧ svalue e'' = svalue e" by
blast
    with * ** *** show "address e' = address e ∧ sender e' = sender e ∧ svalue e' = svalue e" using
init_address init_sender init_svalue by metis
  qed
qed

lemma ffold_init_dom:
```

```
  "fmdom (denvalue (ffold (init ct) e xs)) |⊆| fmdom (denvalue e) |∪| xs"
proof (induct "xs")
  case empty
  then show ?case
  proof
    fix x
    assume "x |∈| fmdom (denvalue (ffold (init ct) e {||}))"
    moreover have "ffold (init ct) e {||} = e" using FSet.comp_fun_commute.ffold_empty[OF init_commte,
of "init ct" e] by simp
    ultimately show "x |∈| fmdom (denvalue e) |∪| {||}" by simp
  qed
next
  case (insert x xs)
  then have *: "ffold (init ct) e (finsert x xs) =
    init ct x (ffold (init ct) e xs)" using FSet.comp_fun_commute.ffold_finsert[OF init_commte] by simp

  show ?case
  proof
    fix x' assume "x' |∈| fmdom (denvalue (ffold (init ct) e (finsert x xs)))"
    with * have **: "x' |∈| fmdom (denvalue (init ct x (ffold (init ct) e xs)))" by simp
    then consider "x' |∈| fmdom (denvalue (ffold (init ct) e xs))" | "x'=x"
    proof (cases "fmlookup ct x")
      case None
      then show ?thesis using that ** by simp
    next
      case (Some a)
      then show ?thesis
      proof (cases a)
        case (Method x1)
        then show ?thesis using Some ** that by simp
      next
        case (Var x2)
        show ?thesis
        proof (cases "x=x'")
          case True
          then show ?thesis using that by simp
        next
          case False
          then have "fmlookup (denvalue (updateEnvDup x (Storage x2) (Storeloc x) (ffold (init ct) e
xs))) x' = fmlookup (denvalue (ffold (init ct) e xs)) x'" using updateEnvDup_dup by simp
          moreover from ** Some Var have ***:"x' |∈| fmdom (denvalue (updateEnvDup x (Storage x2)
(Storeloc x) (ffold (init ct) e xs)))" by simp
          ultimately have "x' |∈| fmdom (denvalue (ffold (init ct) e xs))" by (simp add:
fmlookup_dom_iff)
          then show ?thesis using that by simp
        qed
      qed
    qed
    then show "x' |∈| fmdom (denvalue e) |∪| finsert x xs"
    proof cases
      case 1
      then show ?thesis using insert.hyps by auto
    next
      case 2
      then show ?thesis by simp
    qed
  qed
qed

lemma ffold_init_fmap:
  assumes "fmlookup ct i = Some (Var tp)"
      and "i |∉| fmdom (denvalue e)"
  shows "i|∈|xs ⟹ fmlookup (denvalue (ffold (init ct) e xs)) i = Some (Storage tp, Storeloc i)"
proof (induct "xs")
```

```
  case empty
  then show ?case by simp
next
  case (insert x xs)
  then have *: "ffold (init ct) e (finsert x xs) =
    init ct x (ffold (init ct) e xs)" using FSet.comp_fun_commute.ffold_finsert[OF init_commte] by simp

  from insert.prems consider (a) "i |∈| xs" | (b) "¬ i |∈| xs ∧ i = x" by auto
  then show "fmlookup (denvalue (ffold (init ct) e (finsert x xs))) i = Some (Storage tp, Storeloc i)"
  proof cases
    case a
    with insert.hyps(2) have "fmlookup (denvalue (ffold (init ct) e xs)) i = Some (Storage tp, Storeloc
i)" by simp
    moreover have "fmlookup (denvalue (init ct x (ffold (init ct) e xs))) i = fmlookup (denvalue
(ffold (init ct) e xs)) i"
    proof (cases "fmlookup ct x")
      case None
      then show ?thesis by simp
    next
      case (Some a)
      then show ?thesis
      proof (cases a)
        case (Method x1)
        with Some show ?thesis by simp
      next
        case (Var x2)
        with Some have "init ct x (ffold (init ct) e xs) = updateEnvDup x (Storage x2) (Storeloc x)
(ffold (init ct) e xs)" using init_def[of ct x "(ffold (init ct) e xs)"] by simp
        moreover from insert a have "i≠x" by auto
        then have "fmlookup (denvalue (updateEnvDup x (Storage x2) (Storeloc x) (ffold (init ct) e
xs))) i = fmlookup (denvalue (ffold (init ct) e xs)) i" using updateEnvDup_dup[of x i] by simp
        ultimately show ?thesis by simp
      qed
    qed
    ultimately show ?thesis using * by simp
  next
    case b
    with assms(1) have "fmlookup ct x = Some (Var tp)" by simp
    moreover from b assms(2) have "¬ x |∈| fmdom (denvalue (ffold (init ct) e xs))" using
ffold_init_dom by auto
    ultimately have "init ct x (ffold (init ct) e xs) = updateEnv x (Storage tp) (Storeloc x) (ffold
(init ct) e xs)" by auto
    with b * show ?thesis by simp
  qed
qed
```

The following definition allows for a more fine-grained configuration of the code generator.

```
definition ffold_init::"(String.literal, Member) fmap ⇒ Environment ⇒ String.literal fset ⇒
Environment" where
        ‹ffold_init ct a c = ffold (init ct) a c›
declare ffold_init_def [simp]

lemma ffold_init_code [code]:
     ‹ffold_init ct a c = fold (init ct) (remdups (sorted_list_of_set (fset c))) a›
  using  comp_fun_commute_on.fold_set_fold_remdups ffold.rep_eq
           ffold_init_def init_commte sorted_list_of_fset.rep_eq
           sorted_list_of_fset_simps(1)
  by (metis comp_fun_commute.comp_fun_commute comp_fun_commute_on.intro order_refl)

lemma bind_case_stackvalue_cong [fundef_cong]:
  assumes "x = x'"
      and "⋀v. x = KValue v ⟹ f v s = f' v s"
      and "⋀p. x = KCDptr p ⟹ g p s = g' p s"
      and "⋀p. x = KMemptr p ⟹ h p s = h' p s"
```

```
    and "⋀p. x = KStoptr p ⟹ i p s = i' p s"
  shows "(case x of KValue v ⇒ f v | KCDptr p ⇒ g p | KMemptr p ⇒ h p | KStoptr p ⇒ i p) s
       = (case x' of KValue v ⇒ f' v | KCDptr p ⇒ g' p | KMemptr p ⇒ h' p | KStoptr p ⇒ i' p) s"
  using assms by (cases x, auto)

lemma bind_case_type_cong [fundef_cong]:
  assumes "x = x'"
    and "⋀t. x = Value t ⟹ f t s = f' t s"
    and "⋀t. x = Calldata t ⟹ g t s = g' t s"
    and "⋀t. x = Memory t ⟹ h t s = h' t s"
    and "⋀t. x = Storage t ⟹ i t s = i' t s"
  shows "(case x of Value t ⇒ f t | Calldata t ⇒ g t | Memory t ⇒ h t | Storage t ⇒ i t ) s
       = (case x' of Value t ⇒ f' t | Calldata t ⇒ g' t | Memory t ⇒ h' t | Storage t ⇒ i' t) s"
  using assms by (cases x, auto)

lemma bind_case_denvalue_cong [fundef_cong]:
  assumes "x = x'"
    and "⋀a. x = (Stackloc a) ⟹ f a s = f' a s"
    and "⋀a. x = (Storeloc a) ⟹ g a s = g' a s"
  shows "(case x of (Stackloc a) ⇒ f a | (Storeloc a) ⇒ g a) s
       = (case x' of (Stackloc a) ⇒ f' a | (Storeloc a) ⇒ g' a) s"
  using assms by (cases x, auto)

lemma bind_case_mtypes_cong [fundef_cong]:
  assumes "x = x'"
    and "⋀a t. x = (MTArray a t) ⟹ f a t s = f' a t s"
    and "⋀p. x = (MTValue p) ⟹ g p s = g' p s"
  shows "(case x of (MTArray a t) ⇒ f a t | (MTValue p) ⇒ g p) s
       = (case x' of (MTArray a t) ⇒ f' a t | (MTValue p) ⇒ g' p) s"
  using assms by (cases x, auto)

lemma bind_case_stypes_cong [fundef_cong]:
  assumes "x = x'"
    and "⋀a t. x = (STArray a t) ⟹ f a t s = f' a t s"
    and "⋀a t. x = (STMap a t) ⟹ g a t s = g' a t s"
    and "⋀p. x = (STValue p) ⟹ h p s = h' p s"
  shows "(case x of (STArray a t) ⇒ f a t | (STMap a t) ⇒ g a t | (STValue p) ⇒ h p) s
       = (case x' of (STArray a t) ⇒ f' a t | (STMap a t) ⇒ g' a t | (STValue p) ⇒ h' p) s"
  using assms by (cases x, auto)

lemma bind_case_types_cong [fundef_cong]:
  assumes "x = x'"
    and "⋀a. x = (TSInt a) ⟹ f a s = f' a s"
    and "⋀a. x = (TUInt a) ⟹ g a s = g' a s"
    and "x = TBool ⟹ h s = h' s"
    and "x = TAddr ⟹ i s = i' s"
  shows "(case x of (TSInt a) ⇒ f a | (TUInt a) ⇒ g a | TBool ⇒ h | TAddr ⇒ i) s
       = (case x' of (TSInt a) ⇒ f' a | (TUInt a) ⇒ g' a | TBool ⇒ h' | TAddr ⇒ i') s"
  using assms by (cases x, auto)

lemma bind_case_contract_cong [fundef_cong]:
  assumes "x = x'"
    and "⋀a. x = Method a ⟹ f a s = f' a s"
    and "⋀a. x = Var a ⟹ g a s = g' a s"
  shows "(case x of (Method a) ⇒ f a | (Var a) ⇒ g a) s
       = (case x' of (Method a) ⇒ f' a | (Var a) ⇒ g' a) s"
  using assms by (cases x, auto)

lemma bind_case_memoryvalue_cong [fundef_cong]:
  assumes "x = x'"
    and "⋀a. x = MValue a ⟹ f a s = f' a s"
    and "⋀a. x = MPointer a ⟹ g a s = g' a s"
  shows "(case x of (MValue a) ⇒ f a | (MPointer a) ⇒ g a) s
       = (case x' of (MValue a) ⇒ f' a | (MPointer a) ⇒ g' a) s"
```

```
  using assms by (cases x, auto)

abbreviation lift ::
  "(E ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ (Stackvalue * Type, Ex, State) state_monad)
  ⇒ (Types ⇒ Types ⇒ Valuetype ⇒ Valuetype ⇒ (Valuetype * Types) option)
  ⇒ E ⇒ E ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ (Stackvalue * Type, Ex, State) state_monad"
where
  "lift expr f e1 e2 e_p e cd ≡
    (do {
      kv1 ← expr e1 e_p e cd;
      (case kv1 of
        (KValue v1, Value t1) ⇒ (do
          {
            kv2 ← expr e2 e_p e cd;
            (case kv2 of
              (KValue v2, Value t2) ⇒
                (case f t1 t2 v1 v2 of
                  Some (v, t) ⇒ return (KValue v, Value t)
                | None ⇒ throw Err)
            | _ ⇒ (throw Err::(Stackvalue * Type, Ex, State) state_monad))
          })
      | _ ⇒ (throw Err::(Stackvalue * Type, Ex, State) state_monad))
    })"

abbreviation gascheck ::
  "(State ⇒ Gas) ⇒ (unit, Ex, State) state_monad"
where
  "gascheck check ≡
  do {
    g ← (applyf check::(Gas, Ex, State) state_monad);
    (assert Gas (λst. gas st ≤ g) (modify (λst. st (|gas:=gas st - g|))::(unit, Ex, State) state_monad))
  }"
```

5.1.3 Semantics

```
datatype LType = LStackloc Location
               | LMemloc Location
               | LStoreloc Location


locale statement_with_gas =
  fixes costs :: "S⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ State ⇒ Gas"
    and costs_e :: "E⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ State ⇒ Gas"
  assumes while_not_zero[termination_simp]: "⋀e e_p cd st ex s0. 0 < (costs (WHILE ex s0) e_p e cd st) "
      and call_not_zero[termination_simp]: "⋀e e_p cd st i ix. 0 < (costs_e (CALL i ix) e_p e cd st)"
      and ecall_not_zero[termination_simp]: "⋀e e_p cd st a i ix val. 0 < (costs_e (ECALL a i ix val) e_p
e cd st)"
      and invoke_not_zero[termination_simp]: "⋀e e_p cd st i xe. 0 < (costs (INVOKE i xe) e_p e cd st)"
      and external_not_zero[termination_simp]: "⋀e e_p cd st ad i xe val. 0 < (costs (EXTERNAL ad i xe
val) e_p e cd st)"
      and transfer_not_zero[termination_simp]: "⋀e e_p cd st ex ad. 0 < (costs (TRANSFER ad ex) e_p e cd
st)"
begin

function msel::"bool ⇒ MTypes ⇒ Location ⇒ E list ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒
(Location * MTypes, Ex, State) state_monad"
     and ssel::"STypes ⇒ Location ⇒ E list ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ (Location
* STypes, Ex, State) state_monad"
     and lexp :: "L ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ (LType * Type, Ex, State)
state_monad"
     and expr::"E ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ (Stackvalue * Type, Ex, State)
state_monad"
     and load :: "bool ⇒ (Identifier × Type) list ⇒ E list ⇒ Environment_P ⇒ Environment ⇒
CalldataT ⇒ State ⇒ Environment ⇒ CalldataT ⇒ (Environment × CalldataT × State, Ex, State)
```

```
state_monad"
     and rexp::"L ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ (Stackvalue * Type, Ex, State)
state_monad"
     and stmt :: "S ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ (unit, Ex, State) state_monad"
where
  "msel _ _ _ [] _ _ _ st = throw Err st"
| "msel _ (MTValue _) _ _ _ _ _ st = throw Err st"
| "msel _ (MTArray al t) loc [x] e_p env cd st =
    (do {
      kv ← expr x e_p env cd;
      (case kv of
        (KValue v, Value t') ⇒
          (if less t' (TUInt 256) v (ShowL_int al) = Some (ShowL_bool True, TBool)
            then return (hash loc v, t)
            else throw Err)
      | _ ⇒ throw Err)
    }) st"

| "msel mm (MTArray al t) loc (x # y # ys) e_p env cd st =
    (do {
      kv ← expr x e_p env cd;
      (case kv of
        (KValue v, Value t') ⇒
          (if less t' (TUInt 256) v (ShowL_int al) = Some (ShowL_bool True, TBool)
            then do {
              s ← applyf (λst. if mm then memory st else cd);
              (case accessStore (hash loc v) s of
                Some (MPointer l) ⇒ msel mm t l (y#ys) e_p env cd
              | _ ⇒ throw Err)
            } else throw Err)
      | _ ⇒ throw Err)
    }) st"
| "ssel tp loc Nil _ _ _ st = return (loc, tp) st"
| "ssel (STValue _) _ (_ # _) _ _ _ st = throw Err st"
| "ssel (STArray al t) loc (x # xs) e_p env cd st =
    (do {
      kv ← expr x e_p env cd;
      (case kv of
        (KValue v, Value t') ⇒
          (if less t' (TUInt 256) v (ShowL_int al) = Some (ShowL_bool True, TBool)
            then ssel t (hash loc v) xs e_p env cd
            else throw Err)
      | _ ⇒ throw Err)
    }) st"
| "ssel (STMap _ t) loc (x # xs) e_p env cd st =
    (do {
      kv ← expr x e_p env cd;
      (case kv of
        (KValue v, _) ⇒ ssel t (hash loc v) xs e_p env cd
      | _ ⇒ throw Err)
    }) st"
| "lexp (Id i) _ e _ st =
    (case fmlookup (denvalue e) i of
      Some (tp, (Stackloc l)) ⇒ return (LStackloc l, tp)
    | Some (tp, (Storeloc l)) ⇒ return (LStoreloc l, tp)
    | _ ⇒ throw Err) st"
| "lexp (Ref i r) e_p e cd st =
    (case fmlookup (denvalue e) i of
      Some (tp, Stackloc l) ⇒
        do {
          k ← applyf (λst. accessStore l (stack st));
          (case k of
            Some (KCDptr _) ⇒ throw Err
          | Some (KMemptr l') ⇒
```

```
          (case tp of
            Memory t ⇒
              do {
                (l'', t') ← msel True t l' r e_p e cd;
                return (LMemloc l'', Memory t')
              }
          | _ ⇒ throw Err)
        | Some (KStoptr l') ⇒
          (case tp of
            Storage t ⇒
              do {
                (l'', t') ← ssel t l' r e_p e cd;
                return (LStoreloc l'', Storage t')
              }
          | _ ⇒ throw Err)
        | Some (KValue _) ⇒ throw Err
        | None ⇒ throw Err)
      }
    | Some (tp, Storeloc l) ⇒
        (case tp of
          Storage t ⇒
            do {
              (l', t') ← ssel t l r e_p e cd;
              return (LStoreloc l', Storage t')
            }
        | _ ⇒ throw Err)
    | None ⇒ throw Err) st"
| "expr (E.INT b x) e_p e cd st =
    (do {
      gascheck (costs_e (E.INT b x) e_p e cd);
      (if (b ∈ vbits)
        then (return (KValue (createSInt b x), Value (TSInt b)))
        else (throw Err))
    }) st"
| "expr (UINT b x) e_p e cd st =
    (do {
      gascheck (costs_e (UINT b x) e_p e cd);
      (if (b ∈ vbits)
        then (return (KValue (createUInt b x), Value (TUInt b)))
        else (throw Err))
  }) st"
| "expr (ADDRESS ad) e_p e cd st =
    (do {
      gascheck (costs_e (ADDRESS ad) e_p e cd);
      return (KValue ad, Value TAddr)
    }) st"
| "expr (BALANCE ad) e_p e cd st =
    (do {
      gascheck (costs_e (BALANCE ad) e_p e cd);
      kv ← expr ad e_p e cd;
      (case kv of
        (KValue adv, Value TAddr) ⇒
          return (KValue (accessBalance (accounts st) adv), Value (TUInt 256))
      | _ ⇒ throw Err)
    }) st"
| "expr THIS e_p e cd st =
    (do {
      gascheck (costs_e THIS e_p e cd);
      return (KValue (address e), Value TAddr)
    }) st"
| "expr SENDER e_p e cd st =
    (do {
      gascheck (costs_e SENDER e_p e cd);
      return (KValue (sender e), Value TAddr)
```

```
    }) st"
| "expr VALUE eₚ e cd st =
    (do {
      gascheck (costsₑ VALUE eₚ e cd);
      return (KValue (svalue e), Value (TUInt 256))
    }) st"
| "expr TRUE eₚ e cd st =
    (do {
      gascheck (costsₑ TRUE eₚ e cd);
      return (KValue (ShowL_bool True), Value TBool)
    }) st"
| "expr FALSE eₚ e cd st =
    (do {
      gascheck (costsₑ FALSE eₚ e cd);
      return (KValue (ShowL_bool False), Value TBool)
    }) st"
| "expr (NOT x) eₚ e cd st =
    (do {
      gascheck (costsₑ (NOT x) eₚ e cd);
      kv ← expr x eₚ e cd;
      (case kv of
        (KValue v, Value t) ⇒
          (if v = ShowL_bool True
            then expr FALSE eₚ e cd
            else (if v = ShowL_bool False
              then expr TRUE eₚ e cd
              else throw Err))
      | _ ⇒ throw Err)
    }) st"
| "expr (PLUS e1 e2) eₚ e cd st = (gascheck (costsₑ (PLUS e1 e2) eₚ e cd) ≫= (λ_. lift expr add e1 e2
eₚ e cd)) st"
| "expr (MINUS e1 e2) eₚ e cd st = (gascheck (costsₑ (MINUS e1 e2) eₚ e cd) ≫= (λ_. lift expr sub e1
e2 eₚ e cd)) st"
| "expr (LESS e1 e2) eₚ e cd st = (gascheck (costsₑ (LESS e1 e2) eₚ e cd) ≫= (λ_. lift expr less e1 e2
eₚ e cd)) st"
| "expr (EQUAL e1 e2) eₚ e cd st = (gascheck (costsₑ (EQUAL e1 e2) eₚ e cd) ≫= (λ_. lift expr equal e1
e2 eₚ e cd)) st"
| "expr (AND e1 e2) eₚ e cd st = (gascheck (costsₑ (AND e1 e2) eₚ e cd) ≫= (λ_. lift expr vtand e1 e2
eₚ e cd)) st"
| "expr (OR e1 e2) eₚ e cd st = (gascheck (costsₑ (OR e1 e2) eₚ e cd) ≫= (λ_. lift expr vtor e1 e2 eₚ
e cd)) st"
| "expr (LVAL i) eₚ e cd st =
    (do {
      gascheck (costsₑ (LVAL i) eₚ e cd);
      rexp i eₚ e cd
    }) st"

| "expr (CALL i xe) eₚ e cd st =
    (do {
      gascheck (costsₑ (CALL i xe) eₚ e cd);
      (case fmlookup eₚ (address e) of
         Some (ct, _) ⇒
           (case fmlookup ct i of
             Some (Method (fp, f, Some x)) ⇒
               let e' = ffold_init ct (emptyEnv (address e) (sender e) (svalue e)) (fmdom ct)
               in (do {
                 st' ← applyf (λst. st⦇stack:=emptyStore⦈);
                 (e'', cd', st'') ← load False fp xe eₚ e' emptyStore st' e cd;
                 st''' ← get;
                 put st'';
                 stmt f eₚ e'' cd';
                 rv ← expr x eₚ e'' cd';
                 modify (λst. st⦇stack:=stack st''', memory := memory st'''⦈);
                 return rv
```

```
          })
        | _ ⇒ throw Err)
      | None ⇒ throw Err)
   }) st"
| "expr (ECALL ad i xe val) e_p e cd st =
   (do {
     gascheck (costs_e (ECALL ad i xe val) e_p e cd);
     kad ← expr ad e_p e cd;
     (case kad of
       (KValue adv, Value TAddr) ⇒
       (case fmlookup e_p adv of
          Some (ct, _) ⇒
            (case fmlookup ct i of
              Some (Method (fp, f, Some x)) ⇒
              (do {
                kv ← expr val e_p e cd;
                (case kv of
                  (KValue v, Value t) ⇒
                    let e' = ffold_init ct (emptyEnv adv (address e) v) (fmdom ct)
                    in (do {
                      st' ← applyf (λst. st⦇stack:=emptyStore, memory:=emptyStore⦈);
                      (e'', cd', st'') ← load True fp xe e_p e' emptyStore st' e cd;
                      st''' ← get;
                      (case transfer (address e) adv v (accounts st'') of
                        Some acc ⇒
                          do {
                            put (st''⦇accounts := acc⦈);
                            stmt f e_p e'' cd';
                            rv ← expr x e_p e'' cd';
                            modify (λst. st⦇stack:=stack st''', memory := memory st'''⦈);
                            return rv
                          }
                      | None ⇒ throw Err)
                    })
                | _ ⇒ throw Err)
              })
            | _ ⇒ throw Err)
       | None ⇒ throw Err)
     | _ ⇒ throw Err)
   }) st"
| "load cp ((i_p, t_p)#pl) (e#el) e_p e_v' cd' st' e_v cd st =
   (do {
     (v, t) ← expr e e_p e_v cd;
     st'' ← get;
     put st';
     (cd'', e_v'') ← decl i_p t_p (Some (v,t)) cp cd (memory st'') cd' e_v';
     st''' ← get;
     put st'';
     load cp pl el e_p e_v'' cd'' st''' e_v cd
   }) st"
| "load _ [] (_#_) _ _ _ _ _ _ _ st = throw Err st"
| "load _ (_#_) [] _ _ _ _ _ _ _ st = throw Err st"
| "load _ [] [] _ e_v' cd' st' e_v cd st = return (e_v', cd', st') st"

| "rexp (Id i) e_p e cd st =
   (case fmlookup (denvalue e) i of
     Some (tp, Stackloc l) ⇒
       do {
         s ← applyf (λst. accessStore l (stack st));
         (case s of
           Some (KValue v) ⇒ return (KValue v, tp)
         | Some (KCDptr p) ⇒ return (KCDptr p, tp)
         | Some (KMemptr p) ⇒ return (KMemptr p, tp)
         | Some (KStoptr p) ⇒ return (KStoptr p, tp)
```

```
          | _ ⇒ throw Err)
        }
    | Some (Storage (STValue t), Storeloc l) ⇒
      do {
        so ← applyf (λst. fmlookup (storage st) (address e));
        (case so of
          Some s ⇒ return (KValue (accessStorage t l s), Value t)
        | None ⇒ throw Err)
      }
    | Some (Storage (STArray x t), Storeloc l) ⇒ return (KStoptr l, Storage (STArray x t))
    | _ ⇒ throw Err) st"
| "rexp (Ref i r) e_p e cd st =
    (case fmlookup (denvalue e) i of
      Some (tp, (Stackloc l)) ⇒
        do {
          kv ← applyf (λst. accessStore l (stack st));
          (case kv of
            Some (KCDptr l') ⇒
              (case tp of
                Calldata t ⇒
                  do {
                    (l'', t') ← msel False t l' r e_p e cd;
                    (case t' of
                      MTValue t'' ⇒
                        (case accessStore l'' cd of
                          Some (MValue v) ⇒ return (KValue v, Value t'')
                        | _ ⇒ throw Err)
                    | MTArray x t'' ⇒
                        (case accessStore l'' cd of
                          Some (MPointer p) ⇒ return (KCDptr p, Calldata (MTArray x t''))
                        | _ ⇒ throw Err))
                  }
              | _ ⇒ throw Err)
          | Some (KMemptr l') ⇒
              (case tp of
                Memory t ⇒
                do {
                  (l'', t') ← msel True t l' r e_p e cd;
                  (case t' of
                    MTValue t'' ⇒
                    do {
                      mv ← applyf (λst. accessStore l'' (memory st));
                      (case mv of
                        Some (MValue v) ⇒ return (KValue v, Value t'')
                      | _ ⇒ throw Err)
                    }
                  | MTArray x t'' ⇒
                    do {
                      mv ← applyf (λst. accessStore l'' (memory st));
                      (case mv of
                        Some (MPointer p) ⇒ return (KMemptr p, Memory (MTArray x t''))
                      | _ ⇒ throw Err)
                    }
                  )
                }
              | _ ⇒ throw Err)
          | Some (KStoptr l') ⇒
              (case tp of
                Storage t ⇒
                do {
                  (l'', t') ← ssel t l' r e_p e cd;
                  (case t' of
                    STValue t'' ⇒
                      do {
```

```
                so ← applyf (λst. fmlookup (storage st) (address e));
                (case so of
                  Some s ⇒ return (KValue (accessStorage t'' l'' s), Value t'')
                | None ⇒ throw Err)
              }
            | STArray _ _ ⇒ return (KStoptr l'', Storage t')
            | STMap _ _ ⇒ return (KStoptr l'', Storage t'))
          }
        | _ ⇒ throw Err)
      | _ ⇒ throw Err)
    }
  | Some (tp, (Storeloc l)) ⇒
      (case tp of
        Storage t ⇒
        do {
          (l', t') ← ssel t l r e_p e cd;
          (case t' of
            STValue t'' ⇒
              do {
                so ← applyf (λst. fmlookup (storage st) (address e));
                (case so of
                  Some s ⇒ return (KValue (accessStorage t'' l' s), Value t'')
                | None ⇒ throw Err)
              }
          | STArray _ _ ⇒ return (KStoptr l', Storage t')
          | STMap _ _ ⇒ return (KStoptr l', Storage t'))
        }
      | _ ⇒ throw Err)
  | None ⇒ throw Err) st"
| "stmt SKIP e_p e cd st = gascheck (costs SKIP e_p e cd) st"
| "stmt (ASSIGN lv ex) e_p env cd st =
  (do {
    gascheck (costs (ASSIGN lv ex) e_p env cd);
    re ← expr ex e_p env cd;
    (case re of
      (KValue v, Value t) ⇒
        do {
          rl ← lexp lv e_p env cd;
          (case rl of
            (LStackloc l, Value t') ⇒
              (case convert t t' v of
                 Some (v', _) ⇒ modify (λst. st (|stack := updateStore l (KValue v') (stack st)|))
               | None ⇒ throw Err)
          | (LStoreloc l, Storage (STValue t')) ⇒
              (case convert t t' v of
                Some (v', _) ⇒
                  do {
                    so ← applyf (λst. fmlookup (storage st) (address env));
                    (case so of
                      Some s ⇒ modify (λst. st(|storage := fmupd (address env) (fmupd l v' s)
(storage st)|))
                    | None ⇒ throw Err)
                  }
              | None ⇒ throw Err)
          | (LMemloc l, Memory (MTValue t')) ⇒
              (case convert t t' v of
                Some (v', _) ⇒ modify (λst. st(|memory := updateStore l (MValue v') (memory st)|))
              | None ⇒ throw Err)
          | _ ⇒ throw Err)
        }
      | (KCDptr p, Calldata (MTArray x t)) ⇒
        do {
          rl ← lexp lv e_p env cd;
          (case rl of
```

```
              (LStackloc l, Memory _) ⇒ modify (λst. st (|stack := updateStore l (KCDptr p) (stack
st)|))
            | (LStackloc l, Storage _) ⇒
                do {
                  sv ← applyf (λst. accessStore l (stack st));
                  (case sv of
                    Some (KStoptr p') ⇒
                      do {
                        so ← applyf (λst. fmlookup (storage st) (address env));
                        (case so of
                          Some s ⇒
                            (case cpm2s p p' x t cd s of
                              Some s' ⇒ modify (λst. st (|storage := fmupd (address env) s' (storage
st)|))

                            | None ⇒ throw Err)
                        | None ⇒ throw Err)
                      }
                  | _ ⇒ throw Err)
                }
             | (LStoreloc l, _) ⇒
                do {
                  so ← applyf (λst. fmlookup (storage st) (address env));
                  (case so of
                    Some s ⇒
                      (case cpm2s p l x t cd s of
                        Some s' ⇒ modify (λst. st (|storage := fmupd (address env) s' (storage st)|))
                       | None ⇒ throw Err)
                  | None ⇒ throw Err)
                }
             | (LMemloc l, _) ⇒
                do {
                  cs ← applyf (λst. cpm2m p l x t cd (memory st));
                  (case cs of
                    Some m ⇒ modify (λst. st (|memory := m|))
                  | None ⇒ throw Err)
                }
             | _ ⇒ throw Err)
          }
        | (KMemptr p, Memory (MTArray x t)) ⇒
            do {
              rl ← lexp lv e_p env cd;
              (case rl of
                (LStackloc l, Memory _) ⇒ modify (λst. st(|stack := updateStore l (KMemptr p) (stack
st)|))
              | (LStackloc l, Storage _) ⇒
                  do {
                    sv ← applyf (λst. accessStore l (stack st));
                    (case sv of
                      Some (KStoptr p') ⇒
                      do {
                        so ← applyf (λst. fmlookup (storage st) (address env));
                        (case so of
                          Some s ⇒
                            do {
                              cs ← applyf (λst. cpm2s p p' x t (memory st) s);
                              (case cs of
                              Some s' ⇒ modify (λst. st (|storage := fmupd (address env) s' (storage
st)|))

                            | None ⇒ throw Err)
                            }
                        | None ⇒ throw Err)
                      }
                    | _ ⇒ throw Err)
                  }
```

```
              | (LStoreloc l, _) ⇒
                  do {
                    so ← applyf (λst. fmlookup (storage st) (address env));
                    (case so of
                      Some s ⇒
                        do {
                          cs ← applyf (λst. cpm2s p l x t (memory st) s);
                          (case cs of
                            Some s' ⇒ modify (λst. st (|storage := fmupd (address env) s' (storage
st)|))
                          | None ⇒ throw Err)
                        }
                    | None ⇒ throw Err)
                  }
              | (LMemloc l, _) ⇒ modify (λst. st (|memory := updateStore l (MPointer p) (memory st)|))
              | _ ⇒ throw Err)
            }
        | (KStoptr p, Storage (STArray x t)) ⇒
            do {
              rl ← lexp lv e_p env cd;
              (case rl of
                (LStackloc l, Memory _) ⇒
                  do {
                    sv ← applyf (λst. accessStore l (stack st));
                    (case sv of
                      Some (KMemptr p') ⇒
                        do {
                          so ← applyf (λst. fmlookup (storage st) (address env));
                          (case so of
                            Some s ⇒
                              do {
                                cs ← applyf (λst. cps2m p p' x t s (memory st));
                                (case cs of
                                  Some m ⇒ modify (λst. st(|memory := m|))
                                | None ⇒ throw Err)
                              }
                          | None ⇒ throw Err)
                        }
                    | _ ⇒ throw Err)
                  }
              | (LStackloc l, Storage _) ⇒ modify (λst. st(|stack := updateStore l (KStoptr p) (stack
st)|))
              | (LStoreloc l, _) ⇒
                  do {
                    so ← applyf (λst. fmlookup (storage st) (address env));
                    (case so of
                      Some s ⇒
                        (case copy p l x t s of
                          Some s' ⇒ modify (λst. st (|storage := fmupd (address env) s' (storage st)|))
                        | None ⇒ throw Err)
                     | None ⇒ throw Err)
                  }
              | (LMemloc l, _) ⇒
                  do {
                    so ← applyf (λst. fmlookup (storage st) (address env));
                    (case so of
                      Some s ⇒
                        do {
                          cs ← applyf (λst. cps2m p l x t s (memory st));
                          (case cs of
                            Some m ⇒ modify (λst. st(|memory := m|))
                          | None ⇒ throw Err)
                        }
                    | None ⇒ throw Err)
```

```
                  }
              | _ ⇒ throw Err)
            }
        | (KStoptr p, Storage (STMap t t')) ⇒
            do {
              rl ← lexp lv e_p env cd;
              (case rl of
                (LStackloc l, _) ⇒ modify (λst. st⦇stack := updateStore l (KStoptr p) (stack st)⦈)
              | _ ⇒ throw Err)
            }
        | _ ⇒ throw Err)
    }) st"
| "stmt (COMP s1 s2) e_p e cd st =
    (do {
      gascheck (costs (COMP s1 s2) e_p e cd);
      stmt s1 e_p e cd;
      stmt s2 e_p e cd
    }) st"
| "stmt (ITE ex s1 s2) e_p e cd st =
    (do {
      gascheck (costs (ITE ex s1 s2)  e_p e cd);
      v ← expr ex e_p e cd;
      (case v of
        (KValue b, Value TBool) ⇒
            (if b = ShowL_bool True
              then stmt s1 e_p e cd
              else stmt s2 e_p e cd)
      | _ ⇒ throw Err)
    }) st"
| "stmt (WHILE ex s0) e_p e cd st =
    (do {
      gascheck (costs (WHILE ex s0) e_p e cd);
      v ← expr ex e_p e cd;
      (case v of
        (KValue b, Value TBool) ⇒
          (if b = ShowL_bool True
            then do {
              stmt s0 e_p e cd;
              stmt (WHILE ex s0) e_p e cd
            }
            else return ())
      | _ ⇒ throw Err)
    }) st"
| "stmt (INVOKE i xe) e_p e cd st =
    (do {
      gascheck (costs (INVOKE i xe) e_p e cd);
      (case fmlookup e_p (address e) of
          Some (ct, _) ⇒
            (case fmlookup ct i of
              Some (Method (fp, f, None)) ⇒
                 (let e' = ffold_init ct (emptyEnv (address e) (sender e) (svalue e)) (fmdom ct)
                 in (do {
                    st' ← applyf (λst. (st⦇stack:=emptyStore⦈));
                    (e'', cd', st'') ← load False fp xe e_p e' emptyStore st' e cd;
                    st''' ← get;
                    put st'';
                    stmt f e_p e'' cd';
                    modify (λst. st⦇stack:=stack st''', memory := memory st'''⦈)
                  }))
            | _ ⇒ throw Err)
        | None  ⇒ throw Err)
    }) st"
```

```
| "stmt (EXTERNAL ad i xe val) e_p e cd st =
    (do {
      gascheck (costs (EXTERNAL ad i xe val) e_p e cd);
      kad ← expr ad e_p e cd;
      (case kad of
        (KValue adv, Value TAddr) ⇒
          (case fmlookup e_p adv of
            Some (ct, fb) ⇒
              (do {
                kv ← expr val e_p e cd;
                (case kv of
                  (KValue v, Value t) ⇒
                    (case fmlookup ct i of
                      Some (Method (fp, f, None)) ⇒
                      let e' = ffold_init ct (emptyEnv adv (address e) v) (fmdom ct)
                      in (do {
                        st' ← applyf (λst. st(|stack:=emptyStore, memory:=emptyStore|));
                        (e'', cd', st'') ← load True fp xe e_p e' emptyStore st' e cd;
                        st''' ← get;
                        (case transfer (address e) adv v (accounts st'') of
                          Some acc ⇒
                            do {
                              put (st''(|accounts := acc|));
                              stmt f e_p e'' cd';
                              modify (λst. st(|stack:=stack st''', memory := memory st'''|))
                            }
                          | None ⇒ throw Err)
                       })
                    | None ⇒
                      do {
                        st' ← get;
                        (case transfer (address e) adv v (accounts st') of
                          Some acc ⇒
                            do {
                              st'' ← get;
                              modify (λst. st(|accounts := acc,stack:=emptyStore, memory:=emptyStore|));
                              stmt fb e_p (emptyEnv adv (address e) v) cd;
                              modify (λst. st(|stack:=stack st'', memory := memory st''|))
                            }
                        | None ⇒ throw Err)
                      }
                    | _ ⇒ throw Err)
                | _ ⇒ throw Err)
              })
          | None ⇒ throw Err)
      | _ ⇒ throw Err)
    }) st"
| "stmt (TRANSFER ad ex) e_p e cd st =
    (do {
      gascheck (costs (TRANSFER ad ex) e_p e cd);
      kv ← expr ex e_p e cd;
      (case kv of
        (KValue v, Value t) ⇒
          (do {
            kv' ← expr ad e_p e cd;
            (case kv' of
              (KValue adv, Value TAddr) ⇒
                (do {
                  acs ← applyf accounts;
                  (case transfer (address e) adv v acs of
                    Some acc ⇒ (case fmlookup e_p adv of
                                  Some (ct, f) ⇒
                                    let e' = ffold_init ct (emptyEnv adv (address e) v) (fmdom ct)
                                    in (do {
```

```
                                      st' ← get;
                                      modify (λst. (st(|accounts := acc, stack:=emptyStore,
memory:=emptyStore|)));
                                      stmt f ep e' emptyStore;
                                      modify (λst. st(|stack:=stack st', memory := memory st'|))
                                    })
                                | None ⇒ modify (λst. (st(|accounts := acc|)))
                  | None ⇒ throw Err)
                })
            | _ ⇒ throw Err)
          })
      | _ ⇒ throw Err)
    }) st"
| "stmt (BLOCK ((id0, tp), ex) s) ep ev cd st =
    (do {
      gascheck (costs (BLOCK ((id0, tp), ex) s) ep ev cd);
      (case ex of
         None ⇒ (do {
           mem ← applyf memory;
           (cd', e') ← decl id0 tp None False cd mem cd ev;
           stmt s ep e' cd'
         })
       | Some ex' ⇒ (do {
           (v, t) ← expr ex' ep ev cd;
           mem ← applyf memory;
           (cd', e') ← decl id0 tp (Some (v, t)) False cd mem cd ev;
           stmt s ep e' cd'
         }))
    }) st"
  by pat_completeness auto
```

5.1.4 Gas Consumption

```
lemma lift_gas:
  assumes "lift expr f e1 e2 ep e cd st = Normal ((v, t), st4')"
      and "⋀st4' v4 t4. expr e1 ep e cd st = Normal ((v4, t4), st4') ⟹ gas st4' ≤ gas st"
      and "⋀x1 x y xa ya x1a x1b st4' v4 t4. expr e1 ep e cd st = Normal (x, y)
            ⟹ (xa, ya) = x
            ⟹ xa = KValue x1a
            ⟹ ya = Value x1b
            ⟹ expr e2 ep e cd y = Normal ((v4, t4), st4')
          ⟹ gas st4' ≤ gas y"
      shows "gas st4' ≤ gas st"
proof (cases "expr e1 ep e cd st")
  case (n a st')
  then show ?thesis
  proof (cases a)
    case (Pair b c)
    then show ?thesis
    proof (cases b)
      case (KValue v1)
      then show ?thesis
      proof (cases c)
        case (Value t1)
        then show ?thesis
        proof (cases "expr e2 ep e cd st'")
          case r2: (n a' st'')
          then show ?thesis
          proof (cases a')
            case p2: (Pair b c)
            then show ?thesis
            proof (cases b)
              case v2: (KValue v2)
              then show ?thesis
```

```
          proof (cases c)
            case t2: (Value t2)
            then show ?thesis
            proof (cases "f t1 t2 v1 v2")
              case None
              with assms n Pair KValue Value r2 p2 v2 t2 show ?thesis by simp
            next
              case (Some a'')
              then show ?thesis
              proof (cases a'')
                case p3: (Pair v t)
                with assms n Pair KValue Value r2 p2 v2 t2 Some have "gas st4'≤gas st''" by simp
                moreover from assms n Pair KValue Value r2 p2 v2 t2 Some have "gas st''≤gas st'"
by simp
                moreover from assms n Pair KValue Value r2 p2 v2 t2 Some have "gas st'≤gas st"
by simp
                ultimately show ?thesis by arith
              qed
            qed
          next
            case (Calldata x2)
            with assms n Pair KValue Value r2 p2 v2 show ?thesis by simp
          next
            case (Memory x3)
            with assms n Pair KValue Value r2 p2 v2 show ?thesis by simp
          next
            case (Storage x4)
            with assms n Pair KValue Value r2 p2 v2 show ?thesis by simp
          qed
        next
          case (KCDptr x2)
          with assms n Pair KValue Value r2 p2 show ?thesis by simp
        next
          case (KMemptr x3)
          with assms n Pair KValue Value r2 p2 show ?thesis by simp
        next
          case (KStoptr x4)
          with assms n Pair KValue Value r2 p2 show ?thesis by simp
        qed
      qed
    next
      case (e x)
      with assms n Pair KValue Value show ?thesis by simp
    qed
  next
    case (Calldata x2)
    with assms n Pair KValue show ?thesis by simp
  next
    case (Memory x3)
    with assms n Pair KValue show ?thesis by simp
  next
    case (Storage x4)
    with assms n Pair KValue show ?thesis by simp
  qed
next
  case (KCDptr x2)
  with assms n Pair show ?thesis by simp
next
  case (KMemptr x3)
  with assms n Pair show ?thesis by simp
next
  case (KStoptr x4)
  with assms n Pair show ?thesis by simp
qed
```

```
  qed
next
  case (e x)
  with assms show ?thesis by simp
qed

lemma msel_ssel_lexp_expr_load_rexp_stmt_dom_gas:
    "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inl (Inl (c1, t1, l1, xe1, ep1, ev1, cd1, st1)))
      ⟹ (∀l1' t1' st1'. msel c1 t1 l1 xe1 ep1 ev1 cd1 st1 = Normal ((l1', t1'), st1') ⟶ gas st1' ≤
gas st1)"
    "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inl (Inr (Inl (t2, l2, xe2, ep2, ev2, cd2, st2))))
      ⟹ (∀l2' t2' st2'. ssel t2 l2 xe2 ep2 ev2 cd2 st2 = Normal ((l2', t2'), st2') ⟶ gas st2' ≤
gas st2)"
    "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inl (Inr (Inr (l5, ep5, ev5, cd5, st5))))
      ⟹ (∀l5' t5' st5'. lexp l5 ep5 ev5 cd5 st5 = Normal ((l5', t5'), st5') ⟶ gas st5' ≤ gas st5)"
    "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (e4, ep4, ev4, cd4, st4))))
      ⟹ (∀st4' v4 t4. expr e4 ep4 ev4 cd4 st4 = Normal ((v4, t4), st4') ⟶ gas st4' ≤ gas st4)"
    "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inr (lcp, lis, lxs, lep, lev0, lcd0, lst0, lev,
lcd, lst))))
      ⟹ (∀ev cd st st'. load lcp lis lxs lep lev0 lcd0 lst0 lev lcd lst = Normal ((ev, cd, st), st')
⟶ gas st ≤ gas lst0 ∧ gas st' ≤ gas lst ∧ address ev = address lev0)"
    "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inr (Inl (l3, ep3, ev3, cd3, st3))))
      ⟹ (∀l3' t3' st3'. rexp l3 ep3 ev3 cd3 st3 = Normal ((l3', t3'), st3') ⟶ gas st3' ≤ gas st3)"
    "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inr (Inr (s6, ep6, ev6, cd6, st6))))
      ⟹ (∀st6'. stmt s6 ep6 ev6 cd6 st6 = Normal((), st6') ⟶ gas st6' ≤ gas st6)"
proof (induct rule: msel_ssel_lexp_expr_load_rexp_stmt.pinduct
[where ?P1.0="λc1 t1 l1 xe1 ep1 ev1 cd1 st1. (∀l1' t1' st1'. msel c1 t1 l1 xe1 ep1 ev1 cd1 st1 =
Normal ((l1', t1'), st1') ⟶ gas st1' ≤ gas st1)"
   and ?P2.0="λt2 l2 xe2 ep2 ev2 cd2 st2. (∀l2' t2' st2'. ssel t2 l2 xe2 ep2 ev2 cd2 st2 = Normal
((l2', t2'), st2') ⟶ gas st2' ≤ gas st2)"
   and ?P3.0="λl5 ep5 ev5 cd5 st5. (∀l5' t5' st5'. lexp l5 ep5 ev5 cd5 st5 = Normal ((l5', t5'), st5')
⟶ gas st5' ≤ gas st5)"
   and ?P4.0="λe4 ep4 ev4 cd4 st4. (∀st4' v4 t4. expr e4 ep4 ev4 cd4 st4 = Normal ((v4, t4), st4') ⟶
gas st4' ≤ gas st4)"
   and ?P5.0="λlcp lis lxs lep lev0 lcd0 lst0 lev lcd lst. (∀ev cd st st'. load lcp lis lxs lep
lev0 lcd0 lst0 lev lcd lst = Normal ((ev, cd, st), st') ⟶ gas st ≤ gas lst0 ∧ gas st' ≤ gas lst
∧ address ev = address lev0)"
   and ?P6.0="λl3 ep3 ev3 cd3 st3. (∀l3' t3' st3'. rexp l3 ep3 ev3 cd3 st3 = Normal ((l3', t3'), st3')
⟶ gas st3' ≤ gas st3)"
   and ?P7.0="λs6 ep6 ev6 cd6 st6. (∀st6'. stmt s6 ep6 ev6 cd6 st6 = Normal ((), st6') ⟶ gas st6' ≤
gas st6)"
])
  case (1 uu uv uw ux uy uz va)
  then show ?case using msel.psimps(1) by auto
next
  case (2 vb vc vd ve vf vg vh vi)
  then show ?case using msel.psimps(2) by auto
next
  case (3 vj al t loc x e_p env cd st)
  then show ?case using msel.psimps(3) by (auto split: if_split_asm Type.split_asm
Stackvalue.split_asm prod.split_asm StateMonad.result.split_asm)
next
  case (4 mm al t loc x y ys e_p env cd st)
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix l1' t1' st1' assume a1: "msel mm (MTArray al t) loc (x # y # ys) e_p env cd st = Normal ((l1',
t1'), st1')"
    show "gas st1' ≤ gas st"
    proof (cases "expr x e_p env cd st")
      case (n a st')
      then show ?thesis
      proof (cases a)
        case (Pair b c)
        then show ?thesis
```

```
        proof (cases b)
          case (KValue v)
          then show ?thesis
          proof (cases c)
            case (Value t')
            then show ?thesis
            proof (cases)
              assume l: "less t' (TUInt 256) v (ShowL_int a1) = Some (ShowL_bool True, TBool)"
              then show ?thesis
              proof (cases "accessStore (hash loc v) (if mm then memory st' else cd)")
                case None
                with 4 a1 n Pair KValue Value l show ?thesis using msel.psimps(4) by simp
              next
                case (Some a)
                then show ?thesis
                proof (cases a)
                  case (MValue x1)
                  with 4 a1 n Pair KValue Value Some l show ?thesis using msel.psimps(4) by simp
                next
                  case (MPointer l)
                  with n Pair KValue Value l Some
                  have "msel mm (MTArray al t) loc (x # y # ys) e_p env cd st = msel mm t l (y # ys) e_p
env cd st'"
                    using msel.psimps(4) 4(1) by simp
                  moreover from n Pair have "gas st' ≤ gas st" using 4(2) by simp
                  moreover from a1 MPointer n Pair KValue Value l Some
                  have "gas st1' ≤ gas st'" using msel.psimps(4) 4(3) 4(1) by simp
                  ultimately show ?thesis by simp
                qed
              qed
            next
              assume "¬ less t' (TUInt 256) v (ShowL_int a1) = Some (ShowL_bool True, TBool)"
              with 4 a1 n Pair KValue Value show ?thesis using msel.psimps(4) by simp
            qed
          next
            case (Calldata x2)
            with 4 a1 n Pair KValue show ?thesis using msel.psimps(4) by simp
          next
            case (Memory x3)
            with 4 a1 n Pair KValue show ?thesis using msel.psimps(4) by simp
          next
            case (Storage x4)
            with 4 a1 n Pair KValue show ?thesis using msel.psimps(4) by simp
          qed
        next
          case (KCDptr x2)
          with 4 a1 n Pair show ?thesis using msel.psimps(4) by simp
        next
          case (KMemptr x3)
          with 4 a1 n Pair show ?thesis using msel.psimps(4) by simp
        next
          case (KStoptr x4)
          with 4 a1 n Pair show ?thesis using msel.psimps(4) by simp
        qed
      qed
    next
      case (e x)
      with 4 a1 show ?thesis using msel.psimps(4) by simp
    qed
  qed
next
  case (5 tp loc vk vl vm st)
  then show ?case using ssel.psimps(1) by auto
next
```

```
  case (6 vn vo vp vq vr vs vt vu)
  then show ?case using ssel.psimps(2) by auto
next
  case (7 al t loc x xs e_p env cd st)
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix l2' t2' st2' assume a1: "ssel (STArray al t) loc (x # xs) e_p env cd st = Normal ((l2', t2'),
st2')"
    show "gas st2' ≤ gas st"
    proof (cases "expr x e_p env cd st")
      case (n a st'')
      then show ?thesis
      proof (cases a)
        case (Pair b c)
        then show ?thesis
        proof (cases b)
          case (KValue v)
          then show ?thesis
          proof (cases c)
            case (Value t')
            then show ?thesis
            proof (cases)
              assume l: "less t' (TUInt 256) v (ShowL_int al) = Some (ShowL_bool True, TBool)"
              with n Pair KValue Value l
              have "ssel (STArray al t) loc (x # xs) e_p env cd st = ssel t (hash loc v) xs e_p env cd
st''"
              using ssel.psimps(3) 7(1) by simp
              moreover from n Pair have "gas st'' ≤ gas st" using 7(2) by simp
              moreover from a1 n Pair KValue Value l
              have "gas st2' ≤ gas st''" using ssel.psimps(3) 7(3) 7(1) by simp
              ultimately show ?thesis by simp
            next
              assume "¬ less t' (TUInt 256) v (ShowL_int al) = Some (ShowL_bool True, TBool)"
              with 7 a1 n Pair KValue Value show ?thesis using ssel.psimps(3) by simp
            qed
          next
            case (Calldata x2)
            with 7 a1 n Pair KValue show ?thesis using ssel.psimps(3) by simp
          next
            case (Memory x3)
            with 7 a1 n Pair KValue show ?thesis using ssel.psimps(3) by simp
          next
            case (Storage x4)
            with 7 a1 n Pair KValue show ?thesis using ssel.psimps(3) by simp
          qed
        next
          case (KCDptr x2)
          with 7 a1 n Pair show ?thesis using ssel.psimps(3) by simp
        next
          case (KMemptr x3)
          with 7 a1 n Pair show ?thesis using ssel.psimps(3) by simp
        next
          case (KStoptr x4)
          with 7 a1 n Pair show ?thesis using ssel.psimps(3) by simp
        qed
      qed
    next
      case (e e)
      with 7 a1 show ?thesis using ssel.psimps(3) by simp
    qed
  qed
next
  case (8 vv t loc x xs e_p env cd st)
  show ?case
```

```
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix l2' t2' st2' assume a1: "ssel (STMap vv t) loc (x # xs) e_p env cd st = Normal ((l2', t2'),
st2')"
    show "gas st2' ≤ gas st"
    proof (cases "expr x e_p env cd st")
      case (n a st')
      then show ?thesis
      proof (cases a)
        case (Pair b c)
        then show ?thesis
        proof (cases b)
          case (KValue v)
          with 8 n Pair have "ssel (STMap vv t) loc (x # xs) e_p env cd st = ssel t (hash loc v) xs e_p
env cd st'" using ssel.psimps(4) by simp
          moreover from n Pair have "gas st' ≤ gas st" using 8(2) by simp
          moreover from a1 n Pair KValue
          have "gas st2' ≤ gas st'" using ssel.psimps(4) 8(3) 8(1) by simp
          ultimately show ?thesis by simp
        next
          case (KCDptr x2)
          with 8 a1 n Pair show ?thesis using ssel.psimps(4) by simp
        next
          case (KMemptr x3)
          with 8 a1 n Pair show ?thesis using ssel.psimps(4) by simp
        next
          case (KStoptr x4)
          with 8 a1 n Pair show ?thesis using ssel.psimps(4) by simp
        qed
      qed
    next
      case (e x)
      with 8 a1 show ?thesis using ssel.psimps(4) by simp
    qed
  qed
next
  case (9 i vw e vx st)
  then show ?case using lexp.psimps(1)[of i vw e vx st] by (simp split: option.split_asm
Denvalue.split_asm prod.split_asm)
next
  case (10 i r e_p e cd st)
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix st5' xa xaa
    assume a1: "lexp (Ref i r) e_p e cd st = Normal ((st5', xa), xaa)"
    then show "gas xaa ≤ gas st"
    proof (cases "fmlookup (denvalue e) i")
      case None
      with 10 a1 show ?thesis using lexp.psimps(2) by simp
    next
      case (Some a)
      then show ?thesis
      proof (cases a)
        case (Pair tp b)
        then show ?thesis
        proof (cases b)
          case (Stackloc l)
          then show ?thesis
          proof (cases "accessStore l (stack st)")
            case None
            with 10 a1 Some Pair Stackloc show ?thesis using lexp.psimps(2) by simp
          next
            case s2: (Some a)
            then show ?thesis
            proof (cases a)
```

```
          case (KValue x1)
          with 10 a1 Some Pair Stackloc s2 show ?thesis using lexp.psimps(2) by simp
        next
          case (KCDptr x2)
          with 10 a1 Some Pair Stackloc s2 show ?thesis using lexp.psimps(2) by simp
        next
          case (KMemptr l')
          then show ?thesis
          proof (cases tp)
            case (Value x1)
            with 10 a1 Some Pair Stackloc s2 KMemptr show ?thesis using lexp.psimps(2) by simp
          next
            case (Calldata x2)
            with 10 a1 Some Pair Stackloc s2 KMemptr show ?thesis using lexp.psimps(2) by simp
          next
            case (Memory t)
            then show ?thesis
            proof (cases "msel True t l' r e_p e cd st")
              case (n a s)
              with 10 a1 Some Pair Stackloc s2 KMemptr Memory show ?thesis using lexp.psimps(2)
by (simp split: prod.split_asm)
            next
              case (e e)
              with 10 a1 Some Pair Stackloc s2 KMemptr Memory show ?thesis using lexp.psimps(2)
by simp
            qed
          next
            case (Storage x4)
            with 10 a1 Some Pair Stackloc s2 KMemptr show ?thesis using lexp.psimps(2) by simp
          qed
        next
          case (KStoptr l')
          then show ?thesis
          proof (cases tp)
            case (Value x1)
            with 10 a1 Some Pair Stackloc s2 KStoptr show ?thesis using lexp.psimps(2) by simp
          next
            case (Calldata x2)
            with 10 a1 Some Pair Stackloc s2 KStoptr show ?thesis using lexp.psimps(2) by simp
          next
            case (Memory t)
            with 10 a1 Some Pair Stackloc s2 KStoptr show ?thesis using lexp.psimps(2) by simp
          next
            case (Storage t)
            then show ?thesis
            proof (cases "ssel t l' r e_p e cd st")
              case (n a s)
              with 10 a1 Some Pair Stackloc s2 KStoptr Storage show ?thesis using lexp.psimps(2)
by (auto split: prod.split_asm)
            next
              case (e x)
              with 10 a1 Some Pair Stackloc s2 KStoptr Storage show ?thesis using lexp.psimps(2)
by simp
            qed
          qed
        qed
      qed
    next
      case (Storeloc l)
      then show ?thesis
      proof (cases tp)
        case (Value x1)
        with 10 a1 Some Pair Storeloc show ?thesis using lexp.psimps(2) by simp
      next
```

```
            case (Calldata x2)
            with 10 a1 Some Pair Storeloc show ?thesis using lexp.psimps(2) by simp
          next
            case (Memory t)
            with 10 a1 Some Pair Storeloc show ?thesis using lexp.psimps(2) by simp
          next
            case (Storage t)
            then show ?thesis
            proof (cases "ssel t l r e_p e cd st")
              case (n a s)
              with 10 a1 Some Pair Storeloc Storage show ?thesis using lexp.psimps(2) by (auto split:
prod.split_asm)
            next
              case (e x)
              with 10 a1 Some Pair Storeloc Storage show ?thesis using lexp.psimps(2) by simp
            qed
          qed
        qed
      qed
    qed
  qed
next
  case (11 b x e_p e vy st)
  then show ?case using expr.psimps(1) by (simp split:if_split_asm)
next
  case (12 b x e_p e vz st)
  then show ?case using expr.psimps(2) by (simp split:if_split_asm)
next
  case (13 ad e_p e wa st)
  then show ?case using expr.psimps(3) by simp
next
  case (14 ad e_p e wb st)
  define g where "g = costs_e (BALANCE ad) e_p e wb st"
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix t4 xa xaa
    assume *: "expr (BALANCE ad) e_p e wb st = Normal ((xa, xaa), t4)"
    show "gas t4 ≤ gas st"
    proof (cases)
      assume "gas st ≤ g"
      with 14 g_def * show ?thesis using expr.psimps(4) by simp
    next
      assume gcost: "¬ gas st ≤ g"
      then show ?thesis
      proof (cases "expr ad e_p e wb (st⦇gas := gas st - g⦈)")
        case (n a s)
        show ?thesis
        proof (cases a)
          case (Pair b c)
          then show ?thesis
          proof (cases b)
            case (KValue x1)
            then show ?thesis
            proof (cases c)
              case (Value x1)
              then show ?thesis
              proof (cases x1)
                case (TSInt x1)
                with 14 g_def * gcost n Pair KValue Value show ?thesis using expr.psimps(4)[of ad e_p e
wb st] by simp
              next
                case (TUInt x2)
                with 14 g_def * gcost n Pair KValue Value show ?thesis using expr.psimps(4)[of ad e_p e
wb st] by simp
```

```
            next
              case TBool
              with 14 g_def * gcost n Pair KValue Value show ?thesis using expr.psimps(4)[of ad e_p e
wb st] by simp
            next
              case TAddr
              with 14 g_def * gcost n Pair KValue Value show "gas t4 ≤ gas st" using
expr.psimps(4)[of ad e_p e wb st] by simp
            qed
          next
            case (Calldata x2)
            with 14 g_def * gcost n Pair KValue show ?thesis using expr.psimps(4)[of ad e_p e wb st]
by simp
          next
            case (Memory x3)
            with 14 g_def * gcost n Pair KValue show ?thesis using expr.psimps(4)[of ad e_p e wb st]
by simp
          next
            case (Storage x4)
            with 14 g_def * gcost n Pair KValue show ?thesis using expr.psimps(4)[of ad e_p e wb st]
by simp
          qed
        next
          case (KCDptr x2)
          with 14 g_def * gcost n Pair show ?thesis using expr.psimps(4)[of ad e_p e wb st] by simp
        next
          case (KMemptr x3)
          with 14 g_def * gcost n Pair show ?thesis using expr.psimps(4)[of ad e_p e wb st] by simp
        next
          case (KStoptr x4)
          with 14 g_def * gcost n Pair show ?thesis using expr.psimps(4)[of ad e_p e wb st] by simp
        qed
      qed
    next
      case (e _)
      with 14 g_def * gcost show ?thesis using expr.psimps(4)[of ad e_p e wb st] by simp
    qed
  qed
 qed
next
 case (15 e_p e wc st)
 then show ?case using expr.psimps(5) by simp
next
 case (16 e_p e wd st)
 then show ?case using expr.psimps(6) by simp
next
 case (17 e_p e wd st)
 then show ?case using expr.psimps(7) by simp
next
 case (18 e_p e wd st)
 then show ?case using expr.psimps(8) by simp
next
 case (19 e_p e wd st)
 then show ?case using expr.psimps(9) by simp
next
 case (20 x e_p e cd st)
 define g where "g = costs_e (NOT x) e_p e cd st"
 show ?case
 proof (rule allI[THEN allI, THEN allI, OF impI])
   fix st4' v4 t4 assume a1: "expr (NOT x) e_p e cd st = Normal ((v4, t4), st4')"
   show "gas st4' ≤ gas st"
   proof (cases)
     assume "gas st ≤ g"
     with 20 g_def a1 show ?thesis using expr.psimps by simp
```

```
    next
      assume gcost: "¬ gas st ≤ g"
      then show ?thesis
      proof (cases "expr x e_p e cd (st(|gas := gas st - g|))")
        case (n a st')
        then show ?thesis
        proof (cases a)
          case (Pair b c)
          then show ?thesis
          proof (cases b)
            case (KValue v)
            then show ?thesis
            proof (cases c)
              case (Value t)
              then show ?thesis
              proof (cases)
                assume v_def: "v = ShowL_bool True"
                with 20(1) g_def gcost n Pair KValue Value have "expr (NOT x) e_p e cd st = expr FALSE
e_p e cd st'" using expr.psimps(10) by simp
                moreover from 20(2) g_def gcost n Pair have "gas st' ≤ gas st" by simp
                moreover from 20(1) 20(3) a1 g_def gcost n Pair KValue Value v_def have "gas st4' ≤
gas st'" using expr.psimps(10) by simp
                ultimately show ?thesis by arith
              next
                assume v_def: "¬ v = ShowL_bool True"
                then show ?thesis
                proof (cases)
                  assume v_def2: "v = ShowL_bool False"
                  with 20(1) g_def gcost n Pair KValue Value v_def have "expr (NOT x) e_p e cd st =
expr TRUE e_p e cd st'" using expr.psimps(10) by simp
                  moreover from 20(2) g_def gcost n Pair have "gas st' ≤ gas st" by simp
                  moreover from 20 a1 g_def gcost n Pair KValue Value v_def v_def2 have "gas st4' ≤
gas st'" using expr.psimps(10) by simp
                  ultimately show ?thesis by arith
                next
                  assume "¬ v = ShowL_bool False"
                  with 20 a1 g_def gcost n Pair KValue Value v_def show ?thesis using expr.psimps(10)
by simp
                qed
              qed
            next
              case (Calldata x2)
              with 20 a1 g_def gcost n Pair KValue show ?thesis using expr.psimps(10) by simp
            next
              case (Memory x3)
              with 20 a1 g_def gcost n Pair KValue show ?thesis using expr.psimps(10) by simp
            next
              case (Storage x4)
              with 20 a1 g_def gcost n Pair KValue show ?thesis using expr.psimps(10) by simp
            qed
          next
            case (KCDptr x2)
            with 20 a1 g_def gcost n Pair show ?thesis using expr.psimps(10) by simp
          next
            case (KMemptr x3)
            with 20 a1 g_def gcost n Pair show ?thesis using expr.psimps(10) by simp
          next
            case (KStoptr x4)
            with 20 a1 g_def gcost n Pair show ?thesis using expr.psimps(10) by simp
          qed
        qed
      next
        case (e e)
        with 20 a1 g_def gcost show ?thesis using expr.psimps(10) by simp
```

```
      qed
    qed
  qed
next
  case (21 e1 e2 ep e cd st)
  define g where "g = costse (PLUS e1 e2) ep e cd st"
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix t4 xa xaa assume e_def: "expr (PLUS e1 e2) ep e cd st = Normal ((xa, xaa), t4)"
    then show "gas t4 ≤ gas st"
    proof (cases)
      assume "gas st ≤ g"
      with 21(1) e_def show ?thesis using expr.psimps(11) g_def by simp
    next
      assume "¬ gas st ≤ g"
      with 21(1) e_def g_def have "lift expr add e1 e2 ep e cd (st(|gas := gas st - g|)) = Normal ((xa,
xaa), t4)" using expr.psimps(11)[of e1 e2 ep e cd st] by simp
      moreover from 21(2) ‘¬ gas st ≤ g‘ g_def have "(⋀st4' v4 t4. expr e1 ep e cd (st(|gas := gas
st - g|)) = Normal ((v4, t4), st4') ⟹ gas st4' ≤ gas (st(|gas := gas st - g|)))" by simp
      moreover from 21(3) ‘¬ gas st ≤ g‘ g_def have "(⋀x1 x y xa ya x1a x1b st4' v4 t4.
          expr e1 ep e cd (st(|gas := gas st - g|)) = Normal (x, y) ⟹
          (xa, ya) = x ⟹
          xa = KValue x1a ⟹
          ya = Value x1b ⟹ expr e2 ep e cd y = Normal ((v4, t4), st4') ⟹ gas st4' ≤ gas y)" by
auto
      ultimately show "gas t4 ≤ gas st" using lift_gas[of e1 ep e cd e2 "add" "st(|gas := gas st - g|)"
xa xaa t4] by simp
    qed
  qed
next
  case (22 e1 e2 ep e cd st)
  define g where "g = costse (MINUS e1 e2) ep e cd st"
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix t4 xa xaa assume e_def: "expr (MINUS e1 e2) ep e cd st = Normal ((xa, xaa), t4)"
    then show "gas t4 ≤ gas st"
    proof (cases)
      assume "gas st ≤ g"
      with 22(1) e_def show ?thesis using expr.psimps(12) g_def by simp
    next
      assume "¬ gas st ≤ g"
      with 22(1) e_def g_def have "lift expr sub e1 e2 ep e cd (st(|gas := gas st - g|)) = Normal ((xa,
xaa), t4)" using expr.psimps(12)[of e1 e2 ep e cd st] by simp
      moreover from 22(2) ‘¬ gas st ≤ g‘ g_def have "(⋀st4' v4 t4. expr e1 ep e cd (st(|gas := gas
st - g|)) = Normal ((v4, t4), st4') ⟹ gas st4' ≤ gas (st(|gas := gas st - g|)))" by simp
      moreover from 22(3) ‘¬ gas st ≤ g‘ g_def have "(⋀x1 x y xa ya x1a x1b st4' v4 t4.
          expr e1 ep e cd (st(|gas := gas st - g|)) = Normal (x, y) ⟹
          (xa, ya) = x ⟹
          xa = KValue x1a ⟹
          ya = Value x1b ⟹ expr e2 ep e cd y = Normal ((v4, t4), st4') ⟹ gas st4' ≤ gas y)" by
auto
      ultimately show "gas t4 ≤ gas st" using lift_gas[of e1 ep e cd e2 "sub" "st(|gas := gas st - g|)"
xa xaa t4] by simp
    qed
  qed
next
  case (23 e1 e2 ep e cd st)
  define g where "g = costse (LESS e1 e2) ep e cd st"
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix t4 xa xaa assume e_def: "expr (LESS e1 e2) ep e cd st = Normal ((xa, xaa), t4)"
    then show "gas t4 ≤ gas st"
    proof (cases)
      assume "gas st ≤ g"
```

```
      with 23(1) e_def show ?thesis using expr.psimps(13) g_def by simp
    next
      assume "¬ gas st ≤ g"
      with 23(1) e_def g_def have "lift expr less e1 e2 e_p e cd (st(|gas := gas st - g|)) = Normal ((xa,
xaa), t4)" using expr.psimps(13)[of e1 e2 e_p e cd st] by simp
      moreover from 23(2) ‘¬ gas st ≤ g‘ g_def have "(⋀st4’ v4 t4. expr e1 e_p e cd (st(|gas := gas
st - g|)) = Normal ((v4, t4), st4’) ⟹ gas st4’ ≤ gas (st(|gas := gas st - g|)))" by simp
      moreover from 23(3) ‘¬ gas st ≤ g‘ g_def have "(⋀x1 x y xa ya x1a x1b st4’ v4 t4.
          expr e1 e_p e cd (st(|gas := gas st - g|)) = Normal (x, y) ⟹
          (xa, ya) = x ⟹
          xa = KValue x1a ⟹
          ya = Value x1b ⟹ expr e2 e_p e cd y = Normal ((v4, t4), st4’) ⟹ gas st4’ ≤ gas y)" by
auto
      ultimately show "gas t4 ≤ gas st" using lift_gas[of e1 e_p e cd e2 "less" "st(|gas := gas st -
g|)" xa xaa t4] by simp
    qed
  qed
next
  case (24 e1 e2 e_p e cd st)
  define g where "g = costs_e (EQUAL e1 e2) e_p e cd st"
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix t4 xa xaa assume e_def: "expr (EQUAL e1 e2) e_p e cd st = Normal ((xa, xaa), t4)"
    then show "gas t4 ≤ gas st"
    proof (cases)
      assume "gas st ≤ g"
      with 24(1) e_def show ?thesis using expr.psimps(14) g_def by simp
    next
      assume "¬ gas st ≤ g"
      with 24(1) e_def g_def have "lift expr equal e1 e2 e_p e cd (st(|gas := gas st - g|)) = Normal
((xa, xaa), t4)" using expr.psimps(14)[of e1 e2 e_p e cd st] by simp
      moreover from 24(2) ‘¬ gas st ≤ g‘ g_def have "(⋀st4’ v4 t4. expr e1 e_p e cd (st(|gas := gas
st - g|)) = Normal ((v4, t4), st4’) ⟹ gas st4’ ≤ gas (st(|gas := gas st - g|)))" by simp
      moreover from 24(3) ‘¬ gas st ≤ g‘ g_def have "(⋀x1 x y xa ya x1a x1b st4’ v4 t4.
          expr e1 e_p e cd (st(|gas := gas st - g|)) = Normal (x, y) ⟹
          (xa, ya) = x ⟹
          xa = KValue x1a ⟹
          ya = Value x1b ⟹ expr e2 e_p e cd y = Normal ((v4, t4), st4’) ⟹ gas st4’ ≤ gas y)" by
auto
      ultimately show "gas t4 ≤ gas st" using lift_gas[of e1 e_p e cd e2 "equal" "st(|gas := gas st -
g|)" xa xaa t4] by simp
    qed
  qed
next
  case (25 e1 e2 e_p e cd st)
  define g where "g = costs_e (AND e1 e2) e_p e cd st"
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix t4 xa xaa assume e_def: "expr (AND e1 e2) e_p e cd st = Normal ((xa, xaa), t4)"
    then show "gas t4 ≤ gas st"
    proof (cases)
      assume "gas st ≤ g"
      with 25(1) e_def show ?thesis using expr.psimps(15) g_def by simp
    next
      assume "¬ gas st ≤ g"
      with 25(1) e_def g_def have "lift expr vtand e1 e2 e_p e cd (st(|gas := gas st - g|)) = Normal
((xa, xaa), t4)" using expr.psimps(15)[of e1 e2 e_p e cd st] by simp
      moreover from 25(2) ‘¬ gas st ≤ g‘ g_def have "(⋀st4’ v4 t4. expr e1 e_p e cd (st(|gas := gas
st - g|)) = Normal ((v4, t4), st4’) ⟹ gas st4’ ≤ gas (st(|gas := gas st - g|)))" by simp
      moreover from 25(3) ‘¬ gas st ≤ g‘ g_def have "(⋀x1 x y xa ya x1a x1b st4’ v4 t4.
          expr e1 e_p e cd (st(|gas := gas st - g|)) = Normal (x, y) ⟹
          (xa, ya) = x ⟹
          xa = KValue x1a ⟹
          ya = Value x1b ⟹ expr e2 e_p e cd y = Normal ((v4, t4), st4’) ⟹ gas st4’ ≤ gas y)" by
```

```
auto
      ultimately show "gas t4 ≤ gas st" using lift_gas[of e1 e_p e cd e2 "vtand" "st(|gas := gas st -
g|)" xa xaa t4] by simp
    qed
  qed
next
  case (26 e1 e2 e_p e cd st)
  define g where "g = costs_e (OR e1 e2) e_p e cd st"
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix t4 xa xaa assume e_def: "expr (OR e1 e2) e_p e cd st = Normal ((xa, xaa), t4)"
    then show "gas t4 ≤ gas st"
    proof (cases)
      assume "gas st ≤ g"
      with 26(1) e_def show ?thesis using expr.psimps(16) g_def by simp
    next
      assume "¬ gas st ≤ g"
      with 26(1) e_def g_def have "lift expr vtor e1 e2 e_p e cd (st(|gas := gas st - g|)) = Normal ((xa,
xaa), t4)" using expr.psimps(16)[of e1 e2 e_p e cd st] by simp
      moreover from 26(2) ‘¬ gas st ≤ g‘ g_def have "(⋀st4’ v4 t4. expr e1 e_p e cd (st(|gas := gas
st - g|)) = Normal ((v4, t4), st4’) ⟹ gas st4’ ≤ gas (st(|gas := gas st - g|)))" by simp
      moreover from 26(3) ‘¬ gas st ≤ g‘ g_def have "(⋀x1 x y xa ya x1a x1b st4’ v4 t4.
          expr e1 e_p e cd (st(|gas := gas st - g|)) = Normal (x, y) ⟹
          (xa, ya) = x ⟹
          xa = KValue x1a ⟹
          ya = Value x1b ⟹ expr e2 e_p e cd y = Normal ((v4, t4), st4’) ⟹ gas st4’ ≤ gas y)" by
auto
      ultimately show "gas t4 ≤ gas st" using lift_gas[of e1 e_p e cd e2 "vtor" "st(|gas := gas st -
g|)" xa xaa t4] by simp
    qed
  qed
next
  case (27 i e_p e cd st)
  then show ?case using expr.psimps(17) by (auto split: prod.split_asm option.split_asm
StateMonad.result.split_asm)
next
  case (28 i xe e_p e cd st)
  define g where "g = costs_e (CALL i xe) e_p e cd st"
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix st4’ v4 t4 assume a1: "expr (CALL i xe) e_p e cd st = Normal ((v4, t4), st4’)"
    show "gas st4’ ≤ gas st"
    proof (cases)
      assume "gas st ≤ g"
      with 28 g_def a1 show ?thesis using expr.psimps by simp
    next
      assume gcost: "¬ gas st ≤ g"
      then show ?thesis
      proof (cases "fmlookup e_p (address e)")
        case None
        with 28(1) a1 g_def gcost show ?thesis using expr.psimps(18) by simp
      next
        case (Some a)
        then show ?thesis
        proof (cases a)
          case (Pair ct _)
          then show ?thesis
          proof (cases "fmlookup ct i")
            case None
            with 28(1) a1 g_def gcost Some Pair show ?thesis using expr.psimps(18) by simp
          next
            case s1: (Some a)
            then show ?thesis
            proof (cases a)
```

```
              case (Method x1)
              then show ?thesis
              proof (cases x1)
                case (fields fp f c)
                then show ?thesis
                proof (cases c)
                  case None
                  with 28(1) a1 g_def gcost Some Pair s1 Method fields show ?thesis using
expr.psimps(18) by simp
                next
                  case s2: (Some x)
                  define st' e'
                    where "st' = st(|gas := gas st - g|)(|stack:=emptyStore|)"
                      and "e' = ffold (init ct) (emptyEnv (address e) (sender e) (svalue e)) (fmdom
ct)"
                  then show ?thesis
                  proof (cases "load False fp xe e_p e' emptyStore st' e cd (st(|gas := gas st - g|))")
                    case s4: (n a st''')
                    then show ?thesis
                    proof (cases a)
                      case f2: (fields e'' cd' st'')
                      then show ?thesis
                      proof (cases "stmt f e_p e'' cd' st''")
                        case n2: (n a st'''')
                        then show ?thesis
                        proof (cases "expr x e_p e'' cd' st''''")
                          case n3: (n a st''''')
                          then show ?thesis
                          proof (cases a)
                            case p1: (Pair sv tp)
                            with 28(1) a1 g_def gcost Some Pair s1 Method fields s2 st'_def e'_def s4
f2 n2 n3
                            have "expr (CALL i xe) e_p e cd st = Normal ((sv, tp), st'''''(|stack:=stack
st''', memory := memory st'''|))" and *: "gas st' ≤ gas (st(|gas := gas st - g|))"
                                using expr.psimps(18)[of i xe e_p e cd st] by (auto simp add: Let_def
split: unit.split_asm)
                            with a1 have "gas st4' ≤ gas st'''''" by auto
                            also from 28(4)[of "()" "st(|gas := gas st - g|)" _ ct] g_def gcost Some Pair
s1 Method fields s2 st'_def e'_def s4 f2 n2 n3
                              have "... ≤ gas st''''" by auto
                            also from 28(3)[of "()" "st(|gas := gas st - g|)" _ ct _ _ x1 fp _ f c x e'
st' "st(|gas := gas st - g|)" _ st''' e'' _ cd' st'' st''' st''' "()" st''] a1 g_def gcost Some Pair s1
Method fields s2 st'_def e'_def s4 f2 n2
                              have "... ≤ gas st''" by auto
                            also have "... ≤ gas st - g"
                            proof -
                              from g_def gcost have "(applyf (costs_e (CALL i xe) e_p e cd) ≫= (λg.
assert Gas (λst. gas st ≤ g) (modify (λst. st(|gas := gas st - g|)))) st = Normal ((), st(|gas := gas st
- g|))" by simp
                              moreover from e'_def have "e' = ffold_init ct (emptyEnv (address e)
(sender e) (svalue e)) (fmdom ct)" by simp
                              moreover from st'_def have "applyf (λst. st(|stack := emptyStore|))
(st(|gas := gas st - g|)) =  Normal (st', st(|gas := gas st - g|))" by simp
                              ultimately have "∀ev cda sta st'a. load False fp xe e_p e' emptyStore st'
e cd (st(|gas := gas st - g|)) = Normal ((ev, cda, sta), st'a) ⟶ gas sta ≤ gas st' ∧ gas st'a ≤ gas
(st(|gas := gas st - g|)) ∧ address ev = address e'" using 28(2)[of "()" "st(|gas := gas st - g|)" _ ct _
_ x1 fp "(f,c)" f c x e' st' "st(|gas := gas st - g|)"] using Some Pair s1 Method fields s2 by blast
                              thus ?thesis using st'_def s4 f2 by auto
                            qed
                            finally show ?thesis by simp
                          qed
                        next
                          case (e x)
                          with 28(1) a1 g_def gcost Some Pair s1 Method fields s2 st'_def e'_def
```

```
s4 f2 n2 show ?thesis using expr.psimps(18)[of i xe e_p e cd st] by (auto simp add:Let_def
split:unit.split_asm)
                        qed
                      next
                        case (e x)
                        with 28(1) a1 g_def gcost Some Pair s1 Method fields s2 st'_def e'_def s4 f2
show ?thesis using expr.psimps(18)[of i xe e_p e cd st] by (auto split:unit.split_asm)
                      qed
                    qed
                  next
                    case (e x)
                    with 28(1) a1 g_def gcost Some Pair s1 Method fields s2 st'_def e'_def show
?thesis using expr.psimps(18)[of i xe e_p e cd st] by auto
                  qed
                qed
              qed
            next
              case (Var x2)
              with 28(1) a1 g_def gcost Some Pair s1 show ?thesis using expr.psimps(18) by simp
            qed
          qed
        qed
      qed
    qed
  qed
next
  case (29 ad i xe val e_p e cd st)
  define g where "g = costs_e (ECALL ad i xe val) e_p e cd st"
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix st4' v4 t4 assume a1: "expr (ECALL ad i xe val) e_p e cd st = Normal ((v4, t4), st4')"
    show "gas st4' ≤ gas st"
    proof (cases)
      assume "gas st ≤ g"
      with 29 g_def a1 show ?thesis using expr.psimps by simp
    next
      assume gcost: "¬ gas st ≤ g"
      then show ?thesis
      proof (cases "expr ad e_p e cd (st(|gas := gas st - g|))")
        case (n a st')
        then show ?thesis
        proof (cases a)
          case (Pair a b)
          then show ?thesis
          proof (cases a)
            case (KValue adv)
            then show ?thesis
            proof (cases b)
              case (Value x1)
              then show ?thesis
              proof (cases x1)
                case (TSInt x1)
                with 29(1) a1 g_def gcost n Pair KValue Value show ?thesis using expr.psimps(19)[of ad
i xe val e_p e cd st] by simp
              next
                case (TUInt x2)
                with 29(1) a1 g_def gcost n Pair KValue Value show ?thesis using expr.psimps(19)[of ad
i xe val e_p e cd st] by simp
              next
                case TBool
                with 29(1) a1 g_def gcost n Pair KValue Value show ?thesis using expr.psimps(19)[of ad
i xe val e_p e cd st] by simp
              next
                case TAddr
```

```
          then show ?thesis
          proof (cases "fmlookup e_p adv")
            case None
            with 29(1) a1 g_def gcost n Pair KValue Value TAddr show ?thesis using
expr.psimps(19)[of ad i xe val e_p e cd st] by simp
          next
            case (Some a)
            then show ?thesis
            proof (cases a)
              case p2: (Pair ct _)
              then show ?thesis
              proof (cases "fmlookup ct i")
                case None
                with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 show ?thesis using
expr.psimps(19) by simp
              next
                case s1: (Some a)
                then show ?thesis
                proof (cases a)
                  case (Method x1)
                  then show ?thesis
                  proof (cases x1)
                    case (fields fp f c)
                    then show ?thesis
                    proof (cases c)
                      case None
                      with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1 Method
fields show ?thesis using expr.psimps(19) by simp
                    next
                      case s2: (Some x)
                      then show ?thesis
                      proof (cases "expr val e_p e cd st'")
                        case n1: (n kv st'')
                        then show ?thesis
                        proof (cases kv)
                          case p3: (Pair a b)
                          then show ?thesis
                          proof (cases a)
                            case k1: (KValue v)
                            then show ?thesis
                            proof (cases b)
                              case v1: (Value t)
                              define stl e'
                              where "stl = st''(|stack:=emptyStore, memory:=emptyStore|)"
                                and "e' = ffold (init ct) (emptyEnv adv (address e) v) (fmdom
ct)"

                              then show ?thesis
                              proof (cases "load True fp xe e_p e' emptyStore stl e cd st''")
                                case s4: (n a st''')
                                then show ?thesis
                                proof (cases a)
                                  case f2: (fields e'' cd' st'''')
                                  then show ?thesis
                                  proof (cases "transfer (address e) adv v (accounts st'''')")
                                    case n2: None
                                    with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2
s1 Method fields s2 n1 p3 k1 v1 stl_def e'_def s4 f2 show ?thesis using expr.psimps(19) by simp
                                  next
                                    case s3: (Some acc)
                                    show ?thesis
                                    proof (cases "stmt f e_p e'' cd' (st''''(|accounts:=acc|))")
                                      case n2: (n a st''''')
                                      then show ?thesis
                                      proof (cases "expr x e_p e'' cd' st'''''")
```

```
                                        case n3: (n a st'''''')
                                        then show ?thesis
                                        proof (cases a)
                                          case p1: (Pair sv tp)
                                          with 29(1) a1 g_def gcost n Pair KValue Value TAddr
Some p2 s1 Method fields s2 n1 p3 k1 v1 s3 stl_def e'_def s4 f2 n2 n3
                                          have "expr (ECALL ad i xe val) e_p e cd st = Normal
((sv, tp), st''''''⦇stack:=stack st''', memory := memory st'''⦈)"
                                              using expr.psimps(19)[of ad i xe val e_p e cd st] by
(auto simp add: Let_def split: unit.split_asm)
                                          with a1 have "gas st4' ≤ gas st''''''" by auto
                                          also from 29(6)[of "()" "st⦇gas := gas st - g⦈" _ st' _
_ adv _ _ ct _ _ x1 fp "(f,c)" f c x kv st'' _ b v t] a1 g_def gcost n Pair KValue Value TAddr Some p2
s1 Method fields s2 n1 p3 k1 v1 s3 stl_def e'_def s4 f2 n2 n3
                                            have "... ≤ gas st'''''" by auto
                                          also from 29(5)[OF _ n Pair KValue Value TAddr Some p2
s1 Method fields _ s2 n1 p3 k1 v1 _ _ s4 f2 _ _ _, of "()" f cd' st'''' st''' st''' acc]  f2 s3 stl_def
e'_def   n2 n3 a1 g_def gcost
                                            have "... ≤ gas (st''''⦇accounts:=acc⦈)" by auto
                                          also have "... ≤ gas stl"
                                          proof -
                                            from g_def gcost have "(applyf (costs_e (ECALL ad i
xe val) e_p e cd) ≫= (λg. assert Gas (λst. gas st ≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st =
Normal ((), st⦇gas := gas st - g⦈)" by simp
                                            moreover from e'_def have "e' = ffold_init ct
(emptyEnv adv (address e) v) (fmdom ct)" by simp
                                            moreover from n1 have "expr val e_p e cd st' =
Normal (kv, st'')" by simp
                                            moreover from stl_def have "applyf (λst. st⦇stack
:= emptyStore, memory := emptyStore⦈) st'' = Normal (stl, st'')" by simp
                                            moreover have "applyf accounts st'' = Normal
((accounts st''), st'')" by simp
                                            ultimately have "∀ev cda sta st'a. load True fp xe
e_p e' emptyStore stl e cd st'' = Normal ((ev, cda, sta), st'a) ⟶ gas sta ≤ gas stl ∧ gas st'a ≤ gas
st'' ∧ address ev = address e'" using 29(4)[of "()" "st⦇gas := gas st - g⦈" _ st' _ _ adv _ _ ct _ _ x1
fp "(f,c)" f c x kv st'' _ b v t] a1 g_def gcost n Pair KValue Value TAddr Some p2 s1 Method fields s2
n1 p3 k1 v1 s3 stl_def e'_def s4 f2 n2 n3 by blast
                                            thus ?thesis using stl_def s4 f2 by auto
                                          qed
                                          also from stl_def have "... ≤ gas st''" by simp
                                          also from 29(3)[of "()" "st⦇gas := gas st - g⦈" _ st' _
_ adv _ _ ct _ _ x1 fp "(f,c)" f c x] a1 g_def gcost n Pair KValue Value TAddr Some p2 s1 Method fields
s2 n1 p3 k1 v1 s3 stl_def e'_def s4 f2 n2 n3
                                            have "... ≤ gas st'" by (auto split:unit.split_asm)
                                          also from 29(2)[of "()" "st⦇gas := gas st - g⦈"] a1
g_def gcost n Pair KValue Value TAddr Some p2 s1 Method fields s2 n1 p3 k1 v1 s3 stl_def e'_def s4 f2 n2
n3
                                            have "... ≤ gas (st⦇gas := gas st - g⦈)" by simp
                                          finally show ?thesis by simp
                                        qed
                                      next
                                        case (e x)
                                        with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some
p2 s1 Method fields s2 n1 p3 k1 v1 s3 stl_def e'_def s4 f2 n2 show ?thesis using expr.psimps(19)[of ad
i xe val e_p e cd st] by simp
                                      qed
                                    next
                                      case (e x)
                                      with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2
s1 Method fields s2 n1 p3 k1 v1 s3 stl_def e'_def s4 f2 show ?thesis using expr.psimps(19)[of ad i xe
val e_p e cd st] by simp
                                    qed
                                  qed
                                qed
```

```
                                next
                                  case (e x)
                                  with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1
Method fields s2 n1 p3 k1 v1 stl_def e'_def show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd
st] by simp
                                qed
                              next
                                case (Calldata x2)
                                with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1
Method fields s2 n1 p3 k1 show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd st] by simp
                              next
                                case (Memory x3)
                                with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1
Method fields s2 n1 p3 k1 show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd st] by simp
                              next
                                case (Storage x4)
                                with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1
Method fields s2 n1 p3 k1 show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd st] by simp
                              qed
                            next
                              case (KCDptr x2)
                              with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1 Method
fields s2 n1 p3 show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd st] by simp
                            next
                              case (KMemptr x3)
                              with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1 Method
fields s2 n1 p3 show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd st] by simp
                            next
                              case (KStoptr x4)
                              with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1 Method
fields s2 n1 p3 show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd st] by simp
                            qed
                          qed
                        next
                          case n2: (e x)
                          with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1 Method
fields s2 show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd st] by simp
                        qed
                      qed
                    qed
                  next
                    case (Var x2)
                    with 29(1) a1 g_def gcost n Pair KValue Value TAddr Some p2 s1 show ?thesis
using expr.psimps(19)[of ad i xe val e_p e cd st] by simp
                  qed
                qed
              qed
            qed
          qed
        next
          case (Calldata x2)
          with 29(1) a1 g_def gcost n Pair KValue show ?thesis using expr.psimps(19)[of ad i xe
val e_p e cd st] by simp
        next
          case (Memory x3)
          with 29(1) a1 g_def gcost n Pair KValue show ?thesis using expr.psimps(19)[of ad i xe
val e_p e cd st] by simp
        next
          case (Storage x4)
          with 29(1) a1 g_def gcost n Pair KValue show ?thesis using expr.psimps(19)[of ad i xe
val e_p e cd st] by simp
        qed
      next
        case (KCDptr x2)
```

```
            with 29(1) a1 g_def gcost n Pair show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd
st] by simp
          next
            case (KMemptr x3)
            with 29(1) a1 g_def gcost n Pair show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd
st] by simp
          next
            case (KStoptr x4)
            with 29(1) a1 g_def gcost n Pair show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd
st] by simp
          qed
        qed
      next
        case (e _)
        with 29(1) a1 g_def gcost show ?thesis using expr.psimps(19)[of ad i xe val e_p e cd st] by
simp
      qed
    qed
  qed
next
  case (30 cp i_p t_p pl e el e_p e_v' cd' st' e_v cd st)
  then show ?case
  proof (cases "expr e e_p e_v cd st")
    case (n a st'')
    then show ?thesis
    proof (cases a)
      case (Pair v t)
      then show ?thesis
      proof (cases "decl i_p t_p (Some (v,t)) cp cd (memory st'') cd' e_v' st'")
        case n2: (n a' st''')
        then show ?thesis
        proof (cases a')
          case f2: (Pair cd'' e_v'')
          show ?thesis
          proof (rule allI[THEN allI, THEN allI, THEN allI, OF impI])
            fix ev xa xaa xaaa assume load_def: "load cp ((i_p, t_p) # pl) (e # el) e_p e_v' cd' st' e_v cd
st = Normal ((ev, xa, xaa), xaaa)"
            with 30(1) n Pair n2 f2 have "load cp ((i_p, t_p) # pl) (e # el) e_p e_v' cd' st' e_v cd st =
load cp pl el e_p e_v'' cd'' st''' e_v cd st''" using load.psimps(1)[of cp i_p t_p pl e el e_p e_v' cd' st'
e_v cd st] by simp
            with load_def have "load cp pl el e_p e_v'' cd'' st''' e_v cd st'' = Normal ((ev, xa, xaa),
xaaa)" by simp
            with n Pair n2 f2 have "gas xaa ≤ gas st''' ∧ gas xaaa ≤ gas st'' ∧ address ev = address
e_v''" using 30(3)[of a st'' v t st'' st'' "()" st' a' st''' cd'' e_v'' st''' st''' "()" st''] by simp
            moreover from n Pair have "gas st'' ≤ gas st" using 30(2) by simp
            moreover from n2 f2 have " address e_v'' =  address e_v'" and "gas st''' ≤ gas st'" using
decl_gas_address by auto
            ultimately show "gas xaa ≤ gas st' ∧ gas xaaa ≤ gas st ∧ address ev = address e_v'" by
simp
          qed
        qed
      next
        case (e x)
        with 30(1) n Pair show ?thesis using load.psimps(1) by simp
      qed
    qed
  next
    case (e x)
    with 30(1) show ?thesis using load.psimps(1) by simp
  qed
next
  case (31 we wf wg wh wi wj wk st)
  then show ?case using load.psimps(2) by auto
next
```

```
  case (32 wl wm wn wo wp wq wr st)
  then show ?case using load.psimps(3)[of wl wm wn wo wp wq wr] by auto
next
  case (33 ws wt wu wv cd e_v s st)
  then show ?case using load.psimps(4)[of ws wt wu wv cd e_v s st] by auto
next
  case (34 i e_p e cd st)
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix st3' xa xaa assume "rexp (L.Id i) e_p e cd st = Normal ((st3', xa), xaa)"
    then show "gas xaa ≤ gas st" using 34(1) rexp.psimps(1) by (simp split: option.split_asm
Denvalue.split_asm Stackvalue.split_asm prod.split_asm Type.split_asm STypes.split_asm)
  qed
next
  case (35 i r e_p e cd st)
  show ?case
  proof (rule allI[THEN allI, THEN allI, OF impI])
    fix st3' xa xaa assume rexp_def: "rexp (Ref i r) e_p e cd st = Normal ((st3', xa), xaa)"
    show "gas xaa ≤ gas st"
    proof (cases "fmlookup (denvalue e) i")
      case None
      with 35(1) show ?thesis using rexp.psimps rexp_def by simp
    next
      case (Some a)
      then show ?thesis
      proof (cases a)
        case (Pair tp b)
        then show ?thesis
        proof (cases b)
          case (Stackloc l)
          then show ?thesis
          proof (cases "accessStore l (stack st)")
            case None
            with 35(1) Some Pair Stackloc show ?thesis using rexp.psimps(2) rexp_def by simp
          next
            case s1: (Some a)
            then show ?thesis
            proof (cases a)
              case (KValue x1)
              with 35(1) Some Pair Stackloc s1 show ?thesis using rexp.psimps(2) rexp_def by simp
            next
              case (KCDptr l')
              with 35 Some Pair Stackloc s1 show ?thesis using rexp.psimps(2)[of i r e_p e cd st]
rexp_def by (simp split: option.split_asm Memoryvalue.split_asm MTypes.split_asm prod.split_asm
Type.split_asm StateMonad.result.split_asm)
            next
              case (KMemptr x3)
              with 35 Some Pair Stackloc s1 show ?thesis using rexp.psimps(2)[of i r e_p e cd st]
rexp_def by (simp split: option.split_asm Memoryvalue.split_asm MTypes.split_asm prod.split_asm
Type.split_asm StateMonad.result.split_asm)
            next
              case (KStoptr x4)
              with 35 Some Pair Stackloc s1 show ?thesis using rexp.psimps(2)[of i r e_p e cd
st] rexp_def by (simp split: option.split_asm STypes.split_asm prod.split_asm Type.split_asm
StateMonad.result.split_asm)
            qed
          qed
        next
          case (Storeloc x2)
          with 35 Some Pair show ?thesis using rexp.psimps rexp_def by (simp split: option.split_asm
STypes.split_asm prod.split_asm Type.split_asm  StateMonad.result.split_asm)
        qed
      qed
    qed
```

```
  qed
next
  case (36 e_p e cd st)
  then show ?case using stmt.psimps(1) by simp
next
  case (37 lv ex e_p env cd st)
  define g where "g = costs (ASSIGN lv ex) e_p env cd st"
  show ?case
  proof (rule allI[OF impI])
    fix st6'
    assume stmt_def: "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st6')"
    then show "gas st6' ≤ gas st"
    proof cases
      assume "gas st ≤ g"
      with 37 stmt_def show ?thesis using stmt.psimps(2) g_def by simp
    next
      assume "¬ gas st ≤ g"
      show ?thesis
      proof (cases "expr ex e_p env cd (st⦇gas := gas st - g⦈)")
        case (n a st')
        then show ?thesis
        proof (cases a)
          case (Pair b c)
          then show ?thesis
          proof (cases b)
            case (KValue v)
            then show ?thesis
            proof (cases c)
              case (Value t)
              then show ?thesis
              proof (cases "lexp lv e_p env cd st'")
                case n2: (n a st'')
                then show ?thesis
                proof (cases a)
                  case p1: (Pair a b)
                  then show ?thesis
                  proof (cases a)
                    case (LStackloc l)
                    then show ?thesis
                    proof (cases b)
                      case v2: (Value t')
                      then show ?thesis
                      proof (cases "convert t t' v ")
                        case None
                        with 37(1) stmt_def `¬ gas st ≤ g` n Pair KValue Value n2 p1 LStackloc v2
show ?thesis using stmt.psimps(2) g_def by simp
                      next
                        case s3: (Some a)
                        then show ?thesis
                        proof (cases a)
                          case p2: (Pair v' b)
                          with 37(1) `¬ gas st ≤ g` n Pair KValue Value n2 p1 LStackloc v2 s3
                          have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st'' ⦇stack :=
updateStore l (KValue v') (stack st'')⦈)"
                            using stmt.psimps(2) g_def by simp
                          with stmt_def have "st6'= (st''⦇stack := updateStore l (KValue v') (stack
st'')⦈)" by simp
                          moreover from 37(3) `¬ gas st ≤ g` n Pair KValue Value n2 p1 have "gas
st'' ≤ gas st'" using g_def by simp
                          moreover from 37(2) `¬ gas st ≤ g` n Pair KValue Value n2 p2 have "gas st'
≤ gas st" using g_def by simp
                          ultimately show ?thesis by simp
                        qed
                      qed
```

```
          next
            case (Calldata x2)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStackloc show ?thesis using stmt.psimps(2) g_def by simp
          next
            case (Memory x3)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStackloc show ?thesis using stmt.psimps(2) g_def by simp
          next
            case (Storage x4)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStackloc show ?thesis using stmt.psimps(2) g_def by simp
          qed
        next
          case (LMemloc l)
          then show ?thesis
          proof (cases b)
            case v2: (Value t’)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LMemloc show ?thesis using stmt.psimps(2) g_def by simp
          next
            case (Calldata x2)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LMemloc show ?thesis using stmt.psimps(2) g_def by simp
          next
            case (Memory x3)
            then show ?thesis
            proof (cases x3)
              case (MTArray x11 x12)
              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LMemloc Memory show ?thesis using stmt.psimps(2) g_def by simp
            next
              case (MTValue t’)
              then show ?thesis
              proof (cases "convert t t’ v ")
                case None
                with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LMemloc Memory MTValue show ?thesis using stmt.psimps(2) g_def by simp
              next
                case s3: (Some a)
                then show ?thesis
                proof (cases a)
                  case p2: (Pair v’ b)
                  with 37(1) ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LMemloc Memory MTValue s3
                  have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st’’ ⦇memory := updateStore l (MValue v’) (memory st’’)⦈)"
                    using stmt.psimps(2) g_def by simp
                  with stmt_def have "st6’= (st’’⦇memory := updateStore l (MValue v’) (memory st’’)⦈)" by simp
                  moreover from 37(3) ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 have "gas st’’ ≤ gas st’" using g_def by simp
                  moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 have "gas st’ ≤ gas st" using g_def by simp
                  ultimately show ?thesis by simp
                qed
              qed
            qed
          next
            case (Storage x4)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LMemloc show ?thesis using stmt.psimps(2) g_def by simp
          qed
        next
```

```
          case (LStoreloc l)
          then show ?thesis
          proof (cases b)
            case v2: (Value t')
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStoreloc show ?thesis using stmt.psimps(2) g_def by simp
          next
            case (Calldata x2)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStoreloc show ?thesis using stmt.psimps(2) g_def by simp
          next
            case (Memory x3)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStoreloc show ?thesis using stmt.psimps(2) g_def by simp
          next
            case (Storage x4)
            then show ?thesis
            proof (cases x4)
              case (STArray x11 x12)
              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStoreloc Storage show ?thesis using stmt.psimps(2) g_def by simp
            next
              case (STMap x21 x22)
              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStoreloc Storage show ?thesis using stmt.psimps(2) g_def by simp
            next
              case (STValue t')
              then show ?thesis
              proof (cases "convert t t' v ")
                case None
                with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStoreloc Storage STValue show ?thesis using stmt.psimps(2) g_def by simp
              next
                case s3: (Some a)
                then show ?thesis
                proof (cases a)
                  case p2: (Pair v' b)
                  then show ?thesis
                  proof (cases "fmlookup (storage st'') (address env)")
                    case None
                    with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStoreloc Storage STValue s3 p2 show ?thesis using stmt.psimps(2) g_def by simp
                  next
                    case s4: (Some s)
                    with 37(1) ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 LStoreloc Storage STValue s3 p2
                    have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st'' ⦇storage := fmupd (address env) (fmupd l v' s) (storage st'')⦈)"
                      using stmt.psimps(2) g_def by simp
                    with stmt_def have "st6'= st'' ⦇storage := fmupd (address env) (fmupd l v' s) (storage st'')⦈" by simp
                    moreover from 37(3) ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 have "gas st'' ≤ gas st'" using g_def by simp
                    moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p1 have "gas st' ≤ gas st" using g_def by simp
                    ultimately show ?thesis by simp
                  qed
                qed
              qed
            qed
          qed
        qed
      qed
    next
```

```
        case (e x)
        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value show ?thesis using
stmt.psimps(2) g_def by simp
       qed
      next
       case (Calldata x2)
       with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue show ?thesis using stmt.psimps(2)
g_def by simp
      next
       case (Memory x3)
       with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue show ?thesis using stmt.psimps(2)
g_def by simp
      next
       case (Storage x4)
       with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue show ?thesis using stmt.psimps(2)
g_def by simp
      qed
     next
      case (KCDptr p)
      then show ?thesis
      proof (cases c)
       case (Value x1)
       with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr show ?thesis using stmt.psimps(2)
g_def by simp
      next
       case (Calldata x2)
       then show ?thesis
       proof (cases x2)
        case (MTArray x t)
        then show ?thesis
        proof (cases "lexp lv e_p env cd st’")
         case n2: (n a st’’)
         then show ?thesis
         proof (cases a)
          case p2: (Pair a b)
          then show ?thesis
          proof (cases a)
           case (LStackloc l)
           then show ?thesis
           proof (cases b)
            case v2: (Value t’)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStackloc show ?thesis using stmt.psimps(2) g_def by simp
           next
            case c2: (Calldata x2)
            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStackloc show ?thesis using stmt.psimps(2) g_def by simp
           next
            case (Memory x3)
            with 37(1) ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2 LStackloc
            have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st’’ ⦇stack := updateStore
l (KCDptr p) (stack st’’)⦈)"
             using stmt.psimps(2) g_def by simp
            with stmt_def have "st6’= (st’’⦇stack := updateStore l (KCDptr p) (stack
st’’)⦈)" by simp
            moreover from 37(4) ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2 have
"gas st’’ ≤ gas st’" using g_def by simp
            moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st’ ≤ gas st" using
g_def by simp
            ultimately show ?thesis by simp
           next
            case (Storage x4)
            then show ?thesis
            proof (cases "accessStore l (stack st’’)")
```

```
                        case None
                        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStackloc Storage show ?thesis using stmt.psimps(2) g_def by simp
                      next
                        case s3: (Some a)
                        then show ?thesis
                        proof (cases a)
                          case (KValue x1)
                          with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStackloc Storage s3 show ?thesis using stmt.psimps(2) g_def by simp
                        next
                          case c3: (KCDptr x2)
                          with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStackloc Storage s3 show ?thesis using stmt.psimps(2) g_def by simp
                        next
                          case (KMemptr x3)
                          with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStackloc Storage s3 show ?thesis using stmt.psimps(2) g_def by simp
                        next
                          case (KStoptr p’)
                          then show ?thesis
                          proof (cases "fmlookup (storage st’’) (address env)")
                            case None
                            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStackloc Storage s3 KStoptr show ?thesis using stmt.psimps(2) g_def by simp
                          next
                            case s4: (Some s)
                            then show ?thesis
                            proof (cases "cpm2s p p’ x t cd s")
                              case None
                              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2
p2 LStackloc Storage s3 KStoptr s4 show ?thesis using stmt.psimps(2) g_def by simp
                            next
                              case (Some s’)
                              with 37(1) ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStackloc Storage s3 KStoptr s4
                              have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st’’ (|storage :=
fmupd (address env) s’ (storage st’’)|))"
                                using stmt.psimps(2) g_def by simp
                              with stmt_def have "st6’= st’’ (|storage := fmupd (address env) s’
(storage st’’)|)" by simp
                              moreover from 37(4) ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2
p2 have "gas st’’ ≤ gas st’" using g_def by simp
                              moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st’ ≤ gas st"
using g_def by simp
                              ultimately show ?thesis by simp
                            qed
                          qed
                        qed
                      qed
                    qed
                  next
                    case (LMemloc l)
                    then show ?thesis
                    proof (cases "cpm2m p l x t cd (memory st’’)")
                      case None
                      with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LMemloc show ?thesis using stmt.psimps(2) g_def by simp
                    next
                      case (Some m)
                      with 37(1) ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2 LMemloc
                      have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st’’ (|memory := m|))"
                        using stmt.psimps(2) g_def by simp
                      with stmt_def have "st6’= (st’’(|memory := m|))" by simp
```

```
              moreover from 37(4) ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2 have
"gas st'' ≤ gas st'" using g_def by simp
              moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st' ≤ gas st" using
g_def by simp
              ultimately show ?thesis by simp
            qed
          next
            case (LStoreloc l)
            then show ?thesis
            proof (cases "fmlookup (storage st'') (address env)")
              case None
              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStoreloc show ?thesis using stmt.psimps(2) g_def by simp
            next
              case s4: (Some s)
              then show ?thesis
              proof (cases "cpm2s p l x t cd s")
                case None
                with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
LStoreloc s4 show ?thesis using stmt.psimps(2) g_def by simp
              next
                case (Some s')
                with 37(1) ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2 LStoreloc s4
                have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st'' (|storage := fmupd
(address env) s' (storage st'')|))"
                  using stmt.psimps(2) g_def by simp
                with stmt_def have "st6'= (st''(|storage := fmupd (address env) s' (storage
st'')|))" by simp
                moreover from 37(4) ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray n2 p2
have "gas st'' ≤ gas st'" using g_def by simp
                moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st' ≤ gas st" using
g_def by simp
                ultimately show ?thesis by simp
              qed
            qed
          qed
        qed
      next
        case (e x)
        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata MTArray show ?thesis
using stmt.psimps(2) g_def by simp
      qed
    next
      case (MTValue x2)
      with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr Calldata show ?thesis using
stmt.psimps(2) g_def by simp
    qed
  next
    case (Memory x3)
    with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr show ?thesis using stmt.psimps(2)
g_def by simp
  next
    case (Storage x4)
    with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KCDptr show ?thesis using stmt.psimps(2)
g_def by simp
  qed
next
  case (KMemptr p)
  then show ?thesis
  proof (cases c)
    case (Value x1)
    with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr show ?thesis using stmt.psimps(2)
g_def by simp
  next
```

```
              case (Calldata x2)
              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr show ?thesis using stmt.psimps(2)
g_def by simp
            next
              case (Memory x3)
              then show ?thesis
              proof (cases x3)
                case (MTArray x t)
                then show ?thesis
                proof (cases "lexp lv e_p env cd st’")
                  case n2: (n a st’’)
                  then show ?thesis
                  proof (cases a)
                    case p2: (Pair a b)
                    then show ?thesis
                    proof (cases a)
                      case (LStackloc l)
                      then show ?thesis
                      proof (cases b)
                        case v2: (Value t’)
                        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStackloc show ?thesis using stmt.psimps(2) g_def by simp
                      next
                        case c2: (Calldata x2)
                        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStackloc show ?thesis using stmt.psimps(2) g_def by simp
                      next
                        case m2: (Memory x3)
                        with 37(1) ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2 LStackloc
                        have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st’’ ⦇stack := updateStore
l (KMemptr p) (stack st’’)⦈)"
                          using stmt.psimps(2) g_def by simp
                        with stmt_def have "st6’= (st’’⦇stack := updateStore l (KMemptr p) (stack
st’’)⦈)" by simp
                        moreover from 37(5) ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2 have
"gas st’’ ≤ gas st’" using g_def by simp
                        moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st’ ≤ gas st" using
g_def by simp
                        ultimately show ?thesis by simp
                      next
                        case (Storage x4)
                        then show ?thesis
                        proof (cases "accessStore l (stack st’’)")
                          case None
                          with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStackloc Storage show ?thesis using stmt.psimps(2) g_def by simp
                        next
                          case s3: (Some a)
                          then show ?thesis
                          proof (cases a)
                            case (KValue x1)
                            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStackloc Storage s3 show ?thesis using stmt.psimps(2) g_def by simp
                          next
                            case c3: (KCDptr x2)
                            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStackloc Storage s3 show ?thesis using stmt.psimps(2) g_def by simp
                          next
                            case m3: (KMemptr x3)
                            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStackloc Storage s3 show ?thesis using stmt.psimps(2) g_def by simp
                          next
                            case (KStoptr p’)
                            then show ?thesis
```

```
                        proof (cases "fmlookup (storage st'') (address env)")
                          case None
                          with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStackloc Storage s3 KStoptr show ?thesis using stmt.psimps(2) g_def by simp
                        next
                          case s4: (Some s)
                          then show ?thesis
                          proof (cases "cpm2s p p' x t (memory st'') s")
                            case None
                            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStackloc Storage s3 KStoptr s4 show ?thesis using stmt.psimps(2) g_def by simp
                          next
                            case (Some s')
                            with 37(1) ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStackloc Storage s3 KStoptr s4
                            have "stmt (ASSIGN lv ex) e_p env cd st =  Normal ((), st'' ⦇storage :=
fmupd (address env) s' (storage st'')⦈)"
                              using stmt.psimps(2) g_def by simp
                            with stmt_def have "st6'= (st''⦇storage := fmupd (address env) s'
(storage st'')⦈)" by simp
                            moreover from 37(5) ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2
p2 have "gas st'' ≤ gas st'" using g_def by simp
                            moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st' ≤ gas st"
using g_def by simp
                            ultimately show ?thesis by simp
                          qed
                        qed
                      qed
                    qed
                  qed
                next
                  case (LMemloc l)
                  with 37(1) ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2 LMemloc
                  have "stmt (ASSIGN lv ex) e_p env cd st =  Normal ((), st'' ⦇memory := updateStore
l (MPointer p) (memory st'')⦈)"
                    using stmt.psimps(2) g_def by simp
                  with stmt_def have "st6'= st'' ⦇memory := updateStore l (MPointer p) (memory
st'')⦈" by simp
                  moreover from 37(5) ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2 have
"gas st'' ≤ gas st'" using g_def by simp
                  moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st' ≤ gas st" using g_def
by simp
                  ultimately show ?thesis by simp
                next
                  case (LStoreloc l)
                  then show ?thesis
                  proof (cases "fmlookup (storage st'') (address env)")
                    case None
                    with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStoreloc show ?thesis using stmt.psimps(2) g_def by simp
                  next
                    case s3: (Some s)
                    then show ?thesis
                    proof (cases "cpm2s p l x t (memory st'') s")
                      case None
                      with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
LStoreloc s3 show ?thesis using stmt.psimps(2) g_def by simp
                    next
                      case (Some s')
                      with 37(1) ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2 LStoreloc s3
                      have "stmt (ASSIGN lv ex) e_p env cd st =  Normal ((), st'' ⦇storage := fmupd
(address env) s' (storage st'')⦈)"
                        using stmt.psimps(2) g_def by simp
                      with stmt_def have "st6'= st''⦇storage := fmupd (address env) s' (storage
```

```
st'')⟧" by simp
                          moreover from 37(5) ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray n2 p2
have "gas st'' ≤ gas st'" using g_def by simp
                          moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st' ≤ gas st" using
g_def by simp
                          ultimately show ?thesis by simp
                        qed
                      qed
                    qed
                  qed
                next
                  case (e x)
                  with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory MTArray show ?thesis using
stmt.psimps(2) g_def by simp
                qed
              next
                case (MTValue x2)
                with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr Memory show ?thesis using
stmt.psimps(2) g_def by simp
              qed
            next
              case (Storage x4)
              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KMemptr show ?thesis using stmt.psimps(2)
g_def by simp
            qed
          next
            case (KStoptr p)
            then show ?thesis
            proof (cases c)
              case (Value x1)
              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr show ?thesis using stmt.psimps(2)
g_def by simp
            next
              case (Calldata x2)
              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr show ?thesis using stmt.psimps(2)
g_def by simp
            next
              case (Memory x3)
              with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr show ?thesis using stmt.psimps(2)
g_def by simp
            next
              case (Storage x4)
              then show ?thesis
              proof (cases x4)
                case (STArray x t)
                then show ?thesis
                proof (cases "lexp lv e_p env cd st'")
                  case n2: (n a st'')
                  then show ?thesis
                  proof (cases a)
                    case p2: (Pair a b)
                    then show ?thesis
                    proof (cases a)
                      case (LStackloc l)
                      then show ?thesis
                      proof (cases b)
                        case v2: (Value t')
                        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2
LStackloc show ?thesis using stmt.psimps(2) g_def by simp
                      next
                        case c2: (Calldata x2)
                        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2
LStackloc show ?thesis using stmt.psimps(2) g_def by simp
                      next
```

```
                    case (Memory x3)
                    then show ?thesis
                    proof (cases "accessStore l (stack st'')")
                      case None
                      with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2
LStackloc Memory show ?thesis using stmt.psimps(2) g_def by simp
                    next
                      case s3: (Some a)
                      then show ?thesis
                      proof (cases a)
                        case (KValue x1)
                        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2
LStackloc Memory s3 show ?thesis using stmt.psimps(2) g_def by simp
                      next
                        case c3: (KCDptr x2)
                        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2
LStackloc Memory s3 show ?thesis using stmt.psimps(2) g_def by simp
                      next
                        case (KMemptr p')
                        then show ?thesis
                        proof (cases "fmlookup (storage st'') (address env)")
                          case None
                          with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2
LStackloc Memory s3 KMemptr show ?thesis using stmt.psimps(2) g_def by simp
                        next
                          case s4: (Some s)
                          then show ?thesis
                          proof (cases "cps2m p p' x t s (memory st'')")
                            case None
                            with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2
p2 LStackloc Memory s3 KMemptr s4 show ?thesis using stmt.psimps(2) g_def by simp
                          next
                            case (Some m)
                            with 37(1) ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2
LStackloc Memory s3 KMemptr s4
                            have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st'' ⦇memory :=
m⦈)"
                              using stmt.psimps(2) g_def by simp
                            with stmt_def have "st6'= (st''⦇memory := m⦈)" by simp
                            moreover from 37(6) ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2
p2 have "gas st'' ≤ gas st'" using g_def by simp
                            moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st' ≤ gas st"
using g_def by simp
                            ultimately show ?thesis by simp
                          qed
                        qed
                      next
                        case sp2: (KStoptr p')
                        with 37(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2
LStackloc Memory s3 show ?thesis using stmt.psimps(2) g_def by simp
                      qed
                    qed
                  next
                    case st2: (Storage x4)
                    with 37(1) ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2 LStackloc
                    have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st'' ⦇stack :=
updateStore l (KStoptr p) (stack st'')⦈)"
                      using stmt.psimps(2) g_def by simp
                    with stmt_def have "st6'= (st''⦇stack := updateStore l (KStoptr p) (stack
st'')⦈)" by simp
                    moreover from 37(6) ‘¬ gas st ≤ g‘ n Pair KStoptr Storage STArray n2 p2 have
"gas st'' ≤ gas st'" using g_def by simp
                    moreover from 37(2) ‘¬ gas st ≤ g‘ n Pair have "gas st' ≤ gas st" using
g_def by simp
```

```
                  ultimately show ?thesis by simp
                qed
              next
                case (LMemloc l)
                then show ?thesis
                proof (cases "fmlookup (storage st'') (address env)")
                  case None
                  with 37(1) stmt_def `¬ gas st ≤ g` n Pair KStoptr Storage STArray n2 p2
LMemloc show ?thesis using stmt.psimps(2) g_def by simp
                next
                  case s4: (Some s)
                  then show ?thesis
                  proof (cases "cps2m p l x t s (memory st'')")
                    case None
                    with 37(1) stmt_def `¬ gas st ≤ g` n Pair KStoptr Storage STArray n2 p2
LMemloc s4 show ?thesis using stmt.psimps(2) g_def by simp
                  next
                    case (Some m)
                    with 37(1) `¬ gas st ≤ g` n Pair KStoptr Storage STArray n2 p2 LMemloc s4
                    have "stmt (ASSIGN lv ex) e_p env cd st =  Normal ((), st'' ⦇memory := m⦈)"
                      using stmt.psimps(2) g_def by simp
                    with stmt_def have "st6'= (st''⦇memory := m⦈)" by simp
                    moreover from 37(6) `¬ gas st ≤ g` n Pair KStoptr Storage STArray n2 p2
have "gas st'' ≤ gas st'" using g_def by simp
                    moreover from 37(2) `¬ gas st ≤ g` n Pair have "gas st' ≤ gas st" using
g_def by simp
                    ultimately show ?thesis by simp
                  qed
                qed
              next
                case (LStoreloc l)
                then show ?thesis
                proof (cases "fmlookup (storage st'') (address env)")
                  case None
                  with 37(1) stmt_def `¬ gas st ≤ g` n Pair KStoptr Storage STArray n2 p2
LStoreloc show ?thesis using stmt.psimps(2) g_def by simp
                next
                  case s4: (Some s)
                  then show ?thesis
                  proof (cases "copy p l x t s")
                    case None
                    with 37(1) stmt_def `¬ gas st ≤ g` n Pair KStoptr Storage STArray n2 p2
LStoreloc s4 show ?thesis using stmt.psimps(2) g_def by simp
                  next
                    case (Some s')
                    with 37(1) `¬ gas st ≤ g` n Pair KStoptr Storage STArray n2 p2 LStoreloc s4
                    have "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st'' ⦇storage := fmupd
(address env) s' (storage st'')⦈)"
                      using stmt.psimps(2) g_def by simp
                    with stmt_def have "st6'= st''⦇storage := fmupd (address env) s' (storage
st'')⦈" by simp
                    moreover from 37(6) `¬ gas st ≤ g` n Pair KStoptr Storage STArray n2 p2
have "gas st'' ≤ gas st'" using g_def by simp
                    moreover from 37(2) `¬ gas st ≤ g` n Pair have "gas st' ≤ gas st" using
g_def by simp
                    ultimately show ?thesis by simp
                  qed
                qed
              qed
            qed
          next
            case (e x)
            with 37(1) stmt_def `¬ gas st ≤ g` n Pair KStoptr Storage STArray show ?thesis
using stmt.psimps(2) g_def by simp
```

```
              qed
            next
              case (STMap t t')
              then show ?thesis
              proof (cases "lexp lv ep env cd st'")
                case n2: (n a st'')
                then show ?thesis
                proof (cases a)
                  case p2: (Pair a b)
                  then show ?thesis
                  proof (cases a)
                    case (LStackloc l)
                    with 37(1) `¬ gas st ≤ g` n Pair KStoptr Storage STMap n2 p2
                    have "stmt (ASSIGN lv ex) ep env cd st = Normal ((), st'' (|stack := updateStore l
(KStoptr p) (stack st'')|))"
                      using stmt.psimps(2) g_def by simp
                    with stmt_def have "st6'= st''(|stack := updateStore l (KStoptr p) (stack st'')|)"
by simp
                    moreover from 37(7) `¬ gas st ≤ g` n Pair KStoptr Storage STMap n2 p2 have
"gas st'' ≤ gas st'" using g_def by simp
                    moreover from 37(2) `¬ gas st ≤ g` n Pair have "gas st' ≤ gas st" using g_def
by simp
                    ultimately show ?thesis by simp
                  next
                    case (LMemloc x2)
                    with 37(1) stmt_def `¬ gas st ≤ g` n Pair KStoptr Storage STMap n2 p2 show
?thesis using stmt.psimps(2) g_def by simp
                  next
                    case (LStoreloc x3)
                    with 37(1) stmt_def `¬ gas st ≤ g` n Pair KStoptr Storage STMap n2 p2 show
?thesis using stmt.psimps(2) g_def by simp
                  qed
                qed
              next
                case (e x)
                with 37(1) stmt_def `¬ gas st ≤ g` n Pair KStoptr Storage STMap show ?thesis using
stmt.psimps(2) g_def by simp
              qed
            next
              case (STValue x3)
              with 37(1) stmt_def `¬ gas st ≤ g` n Pair KStoptr Storage show ?thesis using
stmt.psimps(2) g_def by simp
            qed
          qed
        qed
      qed
    next
      case (e x)
      with 37(1) stmt_def `¬ gas st ≤ g` show ?thesis using stmt.psimps(2) g_def by (simp split:
Ex.split_asm)
    qed
  qed
 qed
next
  case (38 s1 s2 ep e cd st)
  define g where "g = costs (COMP s1 s2) ep e cd st"
  show ?case
  proof (rule allI[OF impI])
    fix st6'
    assume stmt_def: "stmt (COMP s1 s2) ep e cd st = Normal ((), st6')"
    then show "gas st6' ≤ gas st"
    proof cases
      assume "gas st ≤ g"
      with 38 stmt_def g_def show ?thesis using stmt.psimps(3) by simp
```

```
    next
      assume "¬ gas st ≤ g"
      show ?thesis
      proof (cases "stmt s1 e_p e cd (st⦇gas := gas st - g⦈)")
        case (n a st')
        with 38(1) stmt_def ‘¬ gas st ≤ g‘ have "stmt (COMP s1 s2) e_p e cd st = stmt s2 e_p e cd st'" using stmt.psimps(3)[of s1 s2 e_p e cd st] g_def by (simp add:Let_def split:unit.split_asm)
        with 38(3)[of _ "(st⦇gas := gas st - g⦈)" _ st'] stmt_def ‹¬ gas st ≤ g› n have "gas st6' ≤ gas st'" using g_def by fastforce
        moreover from 38(2) ‹¬ gas st ≤ g› n have "gas st' ≤ gas st" using g_def by simp
        ultimately show ?thesis by simp
      next
        case (e x)
        with 38 stmt_def ‘¬ gas st ≤ g‘ show ?thesis using stmt.psimps(3)[of s1 s2 e_p e cd st] g_def by (simp split: Ex.split_asm)
      qed
    qed
  qed
next
  case (39 ex s1 s2 e_p e cd st)
  define g where "g = costs (ITE ex s1 s2) e_p e cd st"
  show ?case
  proof (rule allI[OF impI])
    fix st6'
    assume stmt_def: "stmt (ITE ex s1 s2) e_p e cd st = Normal ((), st6')"
    then show "gas st6' ≤ gas st"
    proof cases
      assume "gas st ≤ g"
      with 39 stmt_def show ?thesis using stmt.psimps(4) g_def by simp
    next
      assume "¬ gas st ≤ g"
      show ?thesis
      proof (cases "expr ex e_p e cd (st⦇gas := gas st - g⦈)")
        case (n a st')
        then show ?thesis
        proof (cases a)
          case (Pair b c)
          then show ?thesis
          proof (cases b)
            case (KValue b)
            then show ?thesis
            proof (cases c)
              case (Value x1)
              then show ?thesis
              proof (cases x1)
                case (TSInt x1)
                with 39(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value show ?thesis using stmt.psimps(4) g_def by simp
              next
                case (TUInt x2)
                with 39(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value show ?thesis using stmt.psimps(4) g_def by simp
              next
                case TBool
                then show ?thesis
                proof cases
                  assume "b = ShowL_bool True"
                  with 39(1) ‘¬ gas st ≤ g‘ n Pair KValue Value TBool have "stmt (ITE ex s1 s2) e_p e cd st = stmt s1 e_p e cd st'" using stmt.psimps(4) g_def by simp
                  with 39(3) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value TBool ‘b = ShowL_bool True‘ have "gas st6' ≤ gas st'" using g_def by simp
                  moreover from 39(2) ‘¬ gas st ≤ g‘ n Pair have "gas st' ≤ gas st" using g_def by simp
                  ultimately show ?thesis by arith
```

```
              next
                assume "¬ b = ShowL_bool True"
                with 39(1) `¬ gas st ≤ g` n Pair KValue Value TBool have "stmt (ITE ex s1 s2) e_p e cd st = stmt s2 e_p e cd st'" using stmt.psimps(4) g_def by simp
                with 39(4) stmt_def `¬ gas st ≤ g` n Pair KValue Value TBool `¬ b = ShowL_bool True` have "gas st6' ≤ gas st'" using g_def by simp
                moreover from 39(2) `¬ gas st ≤ g` n Pair have "gas st' ≤ gas st" using g_def by simp
                ultimately show ?thesis by arith
              qed
            next
              case TAddr
              with 39(1) stmt_def `¬ gas st ≤ g` n Pair KValue Value show ?thesis using stmt.psimps(4) g_def by simp
            qed
          next
            case (Calldata x2)
            with 39(1) stmt_def `¬ gas st ≤ g` n Pair KValue show ?thesis using stmt.psimps(4) g_def by simp
          next
            case (Memory x3)
            with 39(1) stmt_def `¬ gas st ≤ g` n Pair KValue show ?thesis using stmt.psimps(4) g_def by simp
          next
            case (Storage x4)
            with 39(1) stmt_def `¬ gas st ≤ g` n Pair KValue show ?thesis using stmt.psimps(4) g_def by simp
          qed
        next
          case (KCDptr x2)
          with 39(1) stmt_def `¬ gas st ≤ g` n Pair show ?thesis using stmt.psimps(4) g_def by simp
        next
          case (KMemptr x3)
          with 39(1) stmt_def `¬ gas st ≤ g` n Pair show ?thesis using stmt.psimps(4) g_def by simp
        next
          case (KStoptr x4)
          with 39(1) stmt_def `¬ gas st ≤ g` n Pair show ?thesis using stmt.psimps(4) g_def by simp
        qed
      qed
    next
      case (e e)
      with 39(1) stmt_def `¬ gas st ≤ g` show ?thesis using stmt.psimps(4) g_def by simp
    qed
  qed
qed
next
  case (40 ex s0 e_p e cd st)
  define g where "g = costs (WHILE ex s0) e_p e cd st"
  show ?case
  proof (rule allI[OF impI])
    fix st6'
    assume stmt_def: "stmt (WHILE ex s0) e_p e cd st = Normal ((), st6')"
    then show "gas st6' ≤ gas st"
    proof cases
      assume "gas st ≤ costs (WHILE ex s0) e_p e cd st"
      with 40(1) stmt_def show ?thesis using stmt.psimps(5) by simp
    next
      assume gcost: "¬ gas st ≤ costs (WHILE ex s0) e_p e cd st"
      show ?thesis
      proof (cases "expr ex e_p e cd (st(|gas := gas st - g|))")
        case (n a st')
```

```
        then show ?thesis
        proof (cases a)
          case (Pair b c)
          then show ?thesis
          proof (cases b)
            case (KValue b)
            then show ?thesis
            proof (cases c)
              case (Value x1)
              then show ?thesis
              proof (cases x1)
                case (TSInt x1)
                with 40(1) stmt_def gcost n Pair KValue Value show ?thesis using stmt.psimps(5) g_def by simp
              next
                case (TUInt x2)
                with 40(1) stmt_def gcost n Pair KValue Value show ?thesis using stmt.psimps(5) g_def by simp
              next
                case TBool
                then show ?thesis
                proof cases
                  assume "b = ShowL_bool True"
                  then show ?thesis
                  proof (cases "stmt s0 e_p e cd st'")
                    case n2: (n a st'')
                    with 40(1) gcost n Pair KValue Value TBool ‘b = ShowL_bool True‘ have "stmt (WHILE ex s0) e_p e cd st = stmt (WHILE ex s0) e_p e cd st''" using stmt.psimps(5)[of ex s0 e_p e cd st] g_def by (simp add: Let_def split:unit.split_asm)
                    with 40(4) stmt_def gcost n2 Pair KValue Value TBool ‘b = ShowL_bool True‘ n have "gas st6' ≤ gas st''" using g_def by simp
                    moreover from 40(3) gcost n2 Pair KValue Value TBool ‘b = ShowL_bool True‘ n have "gas st'' ≤ gas st'" using g_def by simp
                    moreover from 40(2)[of _ "st⦇gas := gas st - g⦈"] gcost n Pair have "gas st' ≤ gas st" using g_def by simp
                    ultimately show ?thesis by simp
                  next
                    case (e x)
                    with 40(1) stmt_def gcost n Pair KValue Value TBool ‘b = ShowL_bool True‘ show ?thesis using stmt.psimps(5) g_def by (simp split: Ex.split_asm)
                  qed
                next
                  assume "¬ b = ShowL_bool True"
                  with 40(1) gcost n Pair KValue Value TBool have "stmt (WHILE ex s0) e_p e cd st = return () st'" using stmt.psimps(5) g_def by simp
                  with stmt_def have "gas st6' ≤ gas st'" by simp
                  moreover from 40(2)[of _ "st⦇gas := gas st - g⦈"] gcost n have "gas st' ≤ gas st" using g_def by simp
                  ultimately show ?thesis by simp
                qed
              next
                case TAddr
                with 40(1) stmt_def gcost n Pair KValue Value show ?thesis using stmt.psimps(5) g_def by simp
              qed
            next
              case (Calldata x2)
              with 40(1) stmt_def gcost n Pair KValue show ?thesis using stmt.psimps(5) g_def by simp
            next
              case (Memory x3)
              with 40(1) stmt_def gcost n Pair KValue show ?thesis using stmt.psimps(5) g_def by simp
            next
              case (Storage x4)
              with 40(1) stmt_def gcost n Pair KValue show ?thesis using stmt.psimps(5) g_def by simp
```

```
          qed
        next
          case (KCDptr x2)
          with 40(1) stmt_def gcost n Pair show ?thesis using stmt.psimps(5) g_def by simp
        next
          case (KMemptr x3)
          with 40(1) stmt_def gcost n Pair show ?thesis using stmt.psimps(5) g_def by simp
        next
          case (KStoptr x4)
          with 40(1) stmt_def gcost n Pair show ?thesis using stmt.psimps(5) g_def by simp
        qed
      qed
    next
      case (e e)
      with 40(1) stmt_def gcost show ?thesis using stmt.psimps(5) g_def by simp
    qed
  qed
 qed
next
 case (41 i xe e_p e cd st)
 define g where "g = costs (INVOKE i xe) e_p e cd st"
 show ?case
 proof (rule allI[OF impI])
  fix st6' assume a1: "stmt (INVOKE i xe) e_p e cd st = Normal ((), st6')"
  show "gas st6' ≤ gas st"
  proof (cases)
   assume "gas st ≤ costs (INVOKE i xe) e_p e cd st"
   with 41(1) a1 show ?thesis using stmt.psimps(6) by simp
  next
   assume gcost: "¬ gas st ≤ costs (INVOKE i xe) e_p e cd st"
   then show ?thesis
   proof (cases "fmlookup e_p (address e)")
    case None
    with 41(1) a1 gcost show ?thesis using stmt.psimps(6) by simp
   next
    case (Some x)
    then show ?thesis
    proof (cases x)
     case (Pair ct _)
     then show ?thesis
     proof (cases "fmlookup ct i")
      case None
      with 41(1) g_def a1 gcost Some Pair show ?thesis using stmt.psimps(6) by simp
     next
      case s1: (Some a)
      then show ?thesis
      proof (cases a)
       case (Method x1)
       then show ?thesis
       proof (cases x1)
        case (fields fp f c)
        then show ?thesis
        proof (cases c)
         case None
         define st' e'
          where "st' = st(|gas := gas st - g|)(|stack:=emptyStore|)"
           and "e' = ffold (init ct) (emptyEnv (address e) (sender e) (svalue e)) (fmdom
ct)"
         then show ?thesis
         proof (cases "load False fp xe e_p e' emptyStore st' e cd (st(|gas := gas st - g|))")
          case s3: (n a st''')
          then show ?thesis
          proof (cases a)
           case f1: (fields e'' cd' st'')
```

```
                  then show ?thesis
                  proof (cases "stmt f e_p e'' cd' st''")
                    case n2: (n a st'''')
                    with 41(1) g_def a1 gcost Some Pair s1 Method fields None st'_def e'_def s3 f1
                    have "stmt (INVOKE i xe) e_p e cd st = Normal ((), st''''⦇stack:=stack st''',
memory := memory st'''⦈)" and *: "gas st' ≤ gas (st⦇gas := gas st - g⦈)"
                      using stmt.psimps(6)[of i xe e_p e cd st] by (auto simp add:Let_def
split:unit.split_asm)
                    with a1 have "gas st6' ≤ gas st''''" by auto
                    also from 41(3) gcost g_def Some Pair s1 Method fields None st'_def e'_def s3
f1 n2
                      have "... ≤ gas st''" by (auto split:unit.split_asm)
                    also have "... ≤ gas st'"
                      proof -
                        from g_def gcost have "(applyf (costs (INVOKE i xe) e_p e cd) ≫= (λg.
assert Gas (λst. gas st ≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st = Normal ((), st⦇gas := gas st
- g⦈)" by simp
                        moreover from e'_def have "e' = ffold_init ct (emptyEnv (address e)
(sender e) (svalue e)) (fmdom ct)" by simp
                        moreover from st'_def have "applyf (λst. st⦇stack := emptyStore⦈) (st⦇gas
:= gas st - g⦈) = Normal (st', st⦇gas := gas st - g⦈)" by simp
                        ultimately have "∀ev cda sta st'a. load False fp xe e_p e' emptyStore st'
e cd (st⦇gas := gas st - g⦈) = Normal ((ev, cda, sta), st'a) ⟶ gas sta ≤ gas st' ∧ gas st'a ≤ gas
(st⦇gas := gas st - g⦈) ∧ address ev = address e'" using a1 g_def gcost Some Pair s1 Method fields None
st'_def e'_def s3 f1 41(2)[of _ "st⦇gas := gas st - g⦈" x ct _ _ x1 fp _ f c e' st' "st⦇gas := gas st -
g⦈"] by blast
                        then show ?thesis using s3 f1 by auto
                      qed
                    also from * have "... ≤ gas (st⦇gas := gas st - g⦈)" .
                    finally show ?thesis by simp
                  next
                    case (e x)
                    with 41(1) g_def a1 gcost Some Pair s1 Method fields None st'_def e'_def s3 f1
show ?thesis using stmt.psimps(6)[of i xe e_p e cd st] by auto
                  qed
                qed
              next
                case n2: (e x)
                with 41(1) g_def a1 gcost Some Pair s1 Method fields None st'_def e'_def show
?thesis using stmt.psimps(6) by auto
              qed
            next
              case s2: (Some a)
              with 41(1) g_def a1 gcost Some Pair s1 Method fields show ?thesis using
stmt.psimps(6) by simp
            qed
          qed
        next
          case (Var x2)
          with 41(1) g_def a1 gcost Some Pair s1 show ?thesis using stmt.psimps(6) by simp
        qed
      qed
    qed
  qed
  qed
 qed
next
  case (42 ad i xe val e_p e cd st)
  define g where "g = costs (EXTERNAL ad i xe val) e_p e cd st"
  show ?case
  proof (rule allI[OF impI])
    fix st6' assume a1: "stmt (EXTERNAL ad i xe val) e_p e cd st = Normal ((), st6')"
    show "gas st6' ≤ gas st"
    proof (cases)
```

```
      assume "gas st ≤ costs (EXTERNAL ad i xe val) e_p e cd st"
      with 42(1) a1 show ?thesis using stmt.psimps(7) by simp
    next
      assume gcost: "¬ gas st ≤ costs (EXTERNAL ad i xe val) e_p e cd st"
      then show ?thesis
      proof (cases "expr ad e_p e cd (st(|gas := gas st - g|))")
        case (n a st')
        then show ?thesis
        proof (cases a)
          case (Pair b c)
          then show ?thesis
          proof (cases b)
            case (KValue adv)
            then show ?thesis
            proof (cases c)
              case (Value x1)
              then show ?thesis
              proof (cases x1)
                case (TSInt x1)
                with 42(1) g_def a1 gcost n Pair KValue Value show ?thesis using stmt.psimps(7) by
auto
              next
                case (TUInt x2)
                with 42(1) g_def a1 gcost n Pair KValue Value show ?thesis using stmt.psimps(7) by
simp
              next
                case TBool
                with 42(1) g_def a1 gcost n Pair KValue Value show ?thesis using stmt.psimps(7) by
simp
              next
                case TAddr
                then show ?thesis
                proof (cases "fmlookup e_p adv")
                  case None
                  with 42(1) g_def a1 gcost n Pair KValue Value TAddr show ?thesis using
stmt.psimps(7) by simp
                next
                  case (Some x)
                  then show ?thesis
                  proof (cases x)
                    case p2: (Pair ct fb)
                    then show ?thesis
                    proof (cases "expr val e_p e cd st'")
                      case n1: (n kv st'')
                      then show ?thesis
                      proof (cases kv)
                        case p3: (Pair a b)
                        then show ?thesis
                        proof (cases a)
                          case k2: (KValue v)
                          then show ?thesis
                          proof (cases b)
                            case v: (Value t)
                            show ?thesis
                            proof (cases "fmlookup ct i")
                              case None
                              show ?thesis
                              proof (cases "transfer (address e) adv v (accounts st'')")
                                case n2: None
                                with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 None n1 p3
k2 v show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                              next
                                case s4: (Some acc)
                                show ?thesis
```

```
                      proof (cases "stmt fb e_p (emptyEnv adv (address e) v) cd (st''⦇accounts := acc,stack:=emptyStore, memory:=emptyStore⦈)")
                        case n2: (n a st''')
                        with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 None n1 p3 k2 v s4
                        have "stmt (EXTERNAL ad i xe val) e_p e cd st = Normal ((), st'''⦇stack:=stack st'', memory := memory st''⦈)"
                          using stmt.psimps(7)[of ad i xe val e_p e cd st] by (auto simp add:Let_def split:unit.split_asm)
                        with a1 have "gas st6' ≤ gas st'''" by auto
                        also from 42(6)[OF _ n Pair KValue Value TAddr Some p2 n1 p3 k2 v None _ s4, of _ st'' st'' st'' "()"] n2 g_def gcost
                          have "... ≤ gas (st''⦇accounts := acc,stack:=emptyStore, memory:=emptyStore⦈)" by auto
                        also from 42(3)[of _ "st⦇gas := gas st - g⦈" _ st' _ _ adv _ x ct] g_def a1 gcost n Pair KValue Value TAddr Some p2 None n1 p3 k2 v s4 n2
                          have "... ≤ gas st'" by auto
                        also from 42(2)[of _ "st⦇gas := gas st - g⦈"] g_def a1 gcost n Pair KValue Value TAddr Some p2 None n1 p3 k2 v s4 n2
                          have "... ≤ gas (st⦇gas := gas st - g⦈)" by auto
                        finally show ?thesis by simp
                      next
                        case (e x)
                        with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 None n1 p3 k2 v s4 show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                      qed
                    qed
                  next
                    case s1: (Some a)
                    then show ?thesis
                    proof (cases a)
                      case (Method x1)
                      then show ?thesis
                      proof (cases x1)
                        case (fields fp f c)
                        then show ?thesis
                        proof (cases c)
                          case None
                          define stl e'
                          where "stl = st''⦇stack:=emptyStore, memory:=emptyStore⦈"
                            and "e' = ffold (init ct) (emptyEnv adv (address e) v) (fmdom ct)"

                          then show ?thesis
                          proof (cases "load True fp xe e_p e' emptyStore stl e cd st''")
                            case s3: (n a st''')
                            then show ?thesis
                            proof (cases a)
                              case f1: (fields e'' cd' st'''')
                              show ?thesis
                              proof (cases "transfer (address e) adv v (accounts st'''')")
                                case n2: None
                                with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 s1 Method fields None n1 p3 k2 v s3 f1 stl_def e'_def show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                              next
                                case s4: (Some acc)
                                show ?thesis
                                proof (cases "stmt f e_p e'' cd' (st''''⦇accounts := acc⦈)")
                                  case n2: (n a st''''')
                                  with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 s1 Method fields None n1 p3 k2 v stl_def e'_def s3 f1 s4
                                  have "stmt (EXTERNAL ad i xe val) e_p e cd st = Normal ((), st'''''⦇stack:=stack st''', memory := memory st'''⦈)"
                                    using stmt.psimps(7)[of ad i xe val e_p e cd st] by (auto
```

```
simp add:Let_def split:unit.split_asm)
                                        with a1 have "gas st6' ≤ gas (st''''')" by auto
                                        also from 42(5)[OF _ n Pair KValue Value TAddr Some p2 n1
p3 k2 v s1 Method fields _ None _ _ s3 _ _ _ _, of "()" f e'' "(cd', st'''')" cd' st'''' st''' st''' acc
"()" "st''''⦇accounts := acc⦈"] s4 stl_def e'_def f1 n2 g_def gcost
                                          have "... ≤ gas (st''''⦇accounts := acc⦈)" by auto
                                        also have "... ≤ gas stl"
                                        proof -
                                          from g_def gcost have "(applyf (costs (EXTERNAL ad i
xe val) e_p e cd) ≫= (λg. assert Gas (λst. gas st ≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st =
Normal ((), st⦇gas := gas st - g⦈)" by simp
                                          moreover from e'_def have "e' = ffold_init ct (emptyEnv
adv (address e) v) (fmdom ct)" by simp
                                          moreover from n1 have "expr val e_p e cd st' = Normal
(kv, st'')" by simp
                                          moreover from stl_def have "applyf (λst. st⦇stack :=
emptyStore, memory := emptyStore⦈) st'' = Normal (stl, st'')" by simp
                                          moreover have "applyf accounts st'' = Normal ((accounts
st''), st'')" by simp
                                          ultimately have "∀ev cda sta st'a. load True fp xe e_p e'
emptyStore stl e cd st'' = Normal ((ev, cda, sta), st'a) ⟶ gas sta ≤ gas stl ∧ gas st'a ≤ gas st''
∧ address ev = address e'" using 42(4)[of _ "st⦇gas := gas st - g⦈" _ st' _ _ adv _ x ct _ _ st'' _ b v
t _ x1 fp "(f,c)" f c e'] g_def a1 gcost n Pair KValue Value TAddr Some p2 s1 Method fields None n1 p3
k2 v stl_def e'_def s3 f1 s4 n2 by blast
                                          then show ?thesis using s3 f1 by auto
                                        qed
                                        also from stl_def have "... ≤ gas st''" by simp
                                        also from 42(3)[of _ "st⦇gas := gas st - g⦈" _ st' _ _ adv
_ x ct] g_def a1 gcost n Pair KValue Value TAddr Some p2 s1 Method fields None n1 p3 k2 v stl_def e'_def
s3 f1 s4 n2
                                          have "... ≤ gas st'" by auto
                                        also from 42(2)[of _ "st⦇gas := gas st - g⦈"] g_def a1
gcost n Pair KValue Value TAddr Some p2 s1 Method fields None n1 p3 k2 v stl_def e'_def s3 f1 s4 n2
                                          have "... ≤ gas (st⦇gas := gas st - g⦈)" by auto
                                        finally show ?thesis by simp
                                      next
                                        case (e x)
                                        with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2
s1 Method fields None n1 p3 k2 v stl_def e'_def s3 f1 s4 show ?thesis using stmt.psimps(7)[of ad i xe
val e_p e cd st] by simp
                                      qed
                                    qed
                                  qed
                                next
                                  case (e x)
                                  with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 s1
Method fields None n1 p3 k2 v stl_def e'_def show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd
st] by simp
                                qed
                              next
                                case s2: (Some a)
                                with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 s1
Method fields n1 p3 k2 v show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                              qed
                            qed
                          next
                            case (Var x2)
                            with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 s1 n1 p3 k2
v show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                          qed
                        qed
                      next
                        case (Calldata x2)
                        with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 n1 p3 k2 show
```

```
?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                          next
                            case (Memory x3)
                            with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 n1 p3 k2 show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                          next
                            case (Storage x4)
                            with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 n1 p3 k2 show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                          qed
                        next
                          case (KCDptr x2)
                          with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 n1 p3 show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                        next
                          case (KMemptr x3)
                          with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 n1 p3 show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                        next
                          case (KStoptr x4)
                          with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 n1 p3 show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                        qed
                      qed
                    next
                      case n2: (e x)
                      with 42(1) g_def a1 gcost n Pair KValue Value TAddr Some p2 show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
                    qed
                  qed
                qed
              qed
            next
              case (Calldata x2)
              with 42(1) g_def a1 gcost n Pair KValue show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
            next
              case (Memory x3)
              with 42(1) g_def a1 gcost n Pair KValue show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
            next
              case (Storage x4)
              with 42(1) g_def a1 gcost n Pair KValue show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
            qed
          next
            case (KCDptr x2)
            with 42(1) g_def a1 gcost n Pair show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
          next
            case (KMemptr x3)
            with 42(1) g_def a1 gcost n Pair show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
          next
            case (KStoptr x4)
            with 42(1) g_def a1 gcost n Pair show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
          qed
        qed
      next
        case (e _)
        with 42(1) g_def a1 gcost show ?thesis using stmt.psimps(7)[of ad i xe val e_p e cd st] by simp
      qed
```

```
    qed
  qed
next
  case (43 ad ex e_p e cd st)
  define g where "g = costs (TRANSFER ad ex) e_p e cd st"
  show ?case
  proof (rule allI[OF impI])
    fix st6' assume stmt_def: "stmt (TRANSFER ad ex) e_p e cd st = Normal ((), st6')"
    show "gas st6' ≤ gas st"
    proof cases
      assume "gas st ≤ g"
      with 43 stmt_def g_def show ?thesis using stmt.psimps(8)[of ad ex e_p e cd st] by simp
    next
      assume "¬ gas st ≤ g"
      show ?thesis
      proof (cases "expr ex e_p e cd (st(|gas := gas st - g|))")
        case (n a st')
        then show ?thesis
        proof (cases a)
          case (Pair b c)
          then show ?thesis
          proof (cases b)
            case (KValue v)
            then show ?thesis
            proof (cases c)
              case (Value t)
              then show ?thesis
              proof (cases "expr ad e_p e cd st'")
                case n2: (n a st'')
                then show ?thesis
                proof (cases a)
                  case p2: (Pair b c)
                  then show ?thesis
                  proof (cases b)
                    case k2: (KValue adv)
                    then show ?thesis
                    proof (cases c)
                      case v2: (Value x1)
                      then show ?thesis
                      proof (cases x1)
                        case (TSInt x1)
                        with 43(1) stmt_def `¬ gas st ≤ g` n Pair KValue Value n2 p2 k2 v2 g_def show
?thesis using stmt.psimps(8) by simp
                      next
                        case (TUInt x2)
                        with 43(1) stmt_def `¬ gas st ≤ g` n Pair KValue Value n2 p2 k2 v2 g_def show
?thesis using stmt.psimps(8) by simp
                      next
                        case TBool
                        with 43(1) stmt_def `¬ gas st ≤ g` n Pair KValue Value n2 p2 k2 v2 g_def show
?thesis using stmt.psimps(8) by simp
                      next
                        case TAddr
                        then show ?thesis
                        proof (cases "transfer (address e) adv v (accounts st'')")
                          case None
                          with 43(1) stmt_def g_def `¬ gas st ≤ g` n Pair KValue Value n2 p2 k2 v2
TAddr show ?thesis using stmt.psimps(8) by simp
                        next
                          case (Some acc)
                          then show ?thesis
                          proof (cases "fmlookup e_p adv")
                            case None
                            with 43(1) `¬ gas st ≤ g` n Pair KValue Value n2 p2 k2 v2 TAddr Some g_def
```

```
                        have "stmt (TRANSFER ad ex) e_p e cd st = Normal ((),st''⦇accounts:=acc⦈)" using stmt.psimps(8)[of ad ex e_p e cd st] by simp
                        with stmt_def have "gas st6' ≤ gas st''" by auto
                        also from 43(3)[of "()" "(st⦇gas := gas st - g⦈)" _ st'] ‘¬ gas st ≤ g‘ n Pair KValue Value n2 g_def have "... ≤ gas st'" by simp
                        also from 43(2)[of "()" "(st⦇gas := gas st - g⦈)"] ‘¬ gas st ≤ g‘ n g_def have "... ≤ gas st" by simp
                        finally show ?thesis .
                      next
                        case s2: (Some a)
                        then show ?thesis
                        proof (cases a)
                          case p3: (Pair ct f)
                          define e' where "e' = ffold_init ct (emptyEnv adv (address e) v) (fmdom ct)"
                          show ?thesis
                          proof (cases "stmt f e_p e' emptyStore (st''⦇accounts := acc, stack:=emptyStore, memory:=emptyStore⦈)")
                            case n3: (n a st''')
                            with 43(1) ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p2 k2 v2 TAddr Some s2 p3 g_def
                            have "stmt (TRANSFER ad ex) e_p e cd st = Normal ((),st'''⦇stack:=stack st'', memory := memory st''⦈)" using e'_def stmt.psimps(8)[of ad ex e_p e cd st] by simp
                            with stmt_def have "gas st6' ≤ gas st'''" by auto
                            also from 43(4)[of "()" "st⦇gas := gas st - g⦈" _ st' _ _ v t _ st'' _ _ adv x1 "accounts st''" st'' acc _ ct f e' _ st'' "()" "st''⦇accounts := acc, stack:=emptyStore, memory:=emptyStore⦈"] ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p2 k2 v2 TAddr Some s2 p3 g_def n2 e'_def n3
                              have "... ≤ gas (st''⦇accounts := acc, stack := emptyStore, memory := emptyStore⦈)" by simp
                            also from 43(3)[of "()" "(st⦇gas := gas st - g⦈)" _ st'] ‘¬ gas st ≤ g‘ n Pair KValue Value n2 g_def have "... ≤ gas st'" by simp
                            also from 43(2)[of "()" "(st⦇gas := gas st - g⦈)"] ‘¬ gas st ≤ g‘ n g_def have "... ≤ gas st" by simp
                            finally show ?thesis .
                          next
                            case (e x)
                            with 43(1) ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p2 k2 v2 TAddr Some s2 p3 g_def e'_def stmt_def show ?thesis using stmt.psimps(8)[of ad ex e_p e cd st] by simp
                          qed
                        qed
                      qed
                    qed
                  qed
                next
                  case (Calldata x2)
                  with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p2 k2 g_def show ?thesis using stmt.psimps(8) by simp
                next
                  case (Memory x3)
                  with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p2 k2 g_def show ?thesis using stmt.psimps(8) by simp
                next
                  case (Storage x4)
                  with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p2 k2 g_def show ?thesis using stmt.psimps(8) by simp
                qed
              next
                case (KCDptr x2)
                with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p2 g_def show ?thesis using stmt.psimps(8) by simp
              next
                case (KMemptr x3)
                with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p2 g_def show ?thesis
```

```
using stmt.psimps(8) by simp
                  next
                    case (KStoptr x4)
                    with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value n2 p2 g_def show ?thesis
using stmt.psimps(8) by simp
                  qed
                qed
              next
                case (e e)
                with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue Value g_def show ?thesis using
stmt.psimps(8) by simp
              qed
            next
              case (Calldata x2)
              with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue g_def show ?thesis using
stmt.psimps(8) by simp
            next
              case (Memory x3)
              with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue g_def show ?thesis using
stmt.psimps(8) by simp
            next
              case (Storage x4)
              with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair KValue g_def show ?thesis using
stmt.psimps(8) by simp
            qed
          next
            case (KCDptr x2)
            with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair g_def show ?thesis using stmt.psimps(8) by
simp
          next
            case (KMemptr x3)
            with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair g_def show ?thesis using stmt.psimps(8) by
simp
          next
            case (KStoptr x4)
            with 43(1) stmt_def ‘¬ gas st ≤ g‘ n Pair g_def show ?thesis using stmt.psimps(8) by
simp
          qed
        qed
      next
        case (e e)
        with 43(1) stmt_def ‘¬ gas st ≤ g‘ g_def show ?thesis using stmt.psimps(8) by simp
      qed
    qed
  qed
next
  case (44 id0 tp ex s e_p e_v cd st)
  define g where "g = costs (BLOCK ((id0, tp), ex) s) e_p e_v cd st"
  show ?case
  proof (rule allI[OF impI])
    fix st6’ assume stmt_def: "stmt (BLOCK ((id0, tp), ex) s) e_p e_v cd st = Normal ((), st6’)"
    show "gas st6’ ≤ gas st"
    proof cases
      assume "gas st ≤ g"
      with 44 stmt_def g_def show ?thesis using stmt.psimps(9) by simp
    next
      assume "¬ gas st ≤ g"
      show ?thesis
      proof (cases ex)
        case None
        then show ?thesis
        proof (cases "decl id0 tp None False cd (memory (st⦇gas := gas st - g⦈)) cd e_v (st⦇gas := gas
st - g⦈)")
          case (n a st’)
```

```
          then show ?thesis
          proof (cases a)
            case (Pair cd' e')
            with 44(1) stmt_def ‘¬ gas st ≤ g‘ None n g_def have "stmt (BLOCK ((id0, tp), ex)
s) e_p e_v cd st = stmt s e_p e' cd' st'" using stmt.psimps(9)[of id0 tp ex s e_p e_v cd st] by (simp
split:unit.split_asm)
            with 44(4)[of "()" "st(|gas := gas st - g|)"] stmt_def ‘¬ gas st ≤ g‘ None n Pair g_def
have "gas st6' ≤ gas st'" by simp
            moreover from n Pair have "gas st' ≤ gas st" using decl_gas_address by simp
            ultimately show ?thesis by simp
          qed
        next
          case (e x)
          with 44 stmt_def ‘¬ gas st ≤ g‘ None g_def show ?thesis using stmt.psimps(9) by simp
        qed
      next
        case (Some ex')
        then show ?thesis
        proof (cases "expr ex' e_p e_v cd (st(|gas := gas st - g|))")
          case (n a st')
          then show ?thesis
          proof (cases a)
            case (Pair v t)
            then show ?thesis
            proof (cases "decl id0 tp (Some (v, t)) False cd (memory st') cd e_v st'")
              case s2: (n a st'')
              then show ?thesis
              proof (cases a)
                case f2: (Pair cd' e')
                with 44(1) stmt_def ‘¬ gas st ≤ g‘ Some n Pair s2 g_def have "stmt (BLOCK ((id0, tp),
ex) s) e_p e_v cd st = stmt s e_p e' cd' st''" using stmt.psimps(9)[of id0 tp ex s e_p e_v cd st] by (simp
split:unit.split_asm)
                with 44(3)[of "()" "st(|gas := gas st - g|)" ex' _ st'] stmt_def ‘¬ gas st ≤ g‘ Some n
Pair s2 f2 g_def have "gas st6' ≤ gas st''" by simp
                moreover from Some n Pair s2 f2 g_def have "gas st'' ≤ gas st'" using
decl_gas_address by simp
                moreover from 44(2)[of "()" "st(|gas := gas st - g|)" ex'] ‘¬ gas st ≤ g‘ Some n Pair
g_def have "gas st' ≤ gas st" by simp
                ultimately show ?thesis by simp
              qed
            next
              case (e x)
              with 44(1) stmt_def ‘¬ gas st ≤ g‘ Some n Pair g_def show ?thesis using stmt.psimps(9)
by simp
            qed
          qed
        next
          case (e e)
          with 44 stmt_def ‘¬ gas st ≤ g‘ Some g_def show ?thesis using stmt.psimps(9) by simp
        qed
      qed
    qed
  qed
qed
```

5.1.5 Termination

```
lemma x1:
  assumes "expr x e_p env cd st = Normal (val, s')"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (x, e_p, env, cd, st))))"
    shows "gas s' < gas st ∨ gas s' = gas st"
  using assms msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(4)[of x e_p env cd st] by auto

lemma x2:
```

```
assumes "(if gas st ≤ c then throw Gas st else (get ⤜ (λs. put (s⦇gas := gas s - c⦈))) st) = Normal
((), s')"
    and "expr x e_p e cd s' = Normal (val, s'a)"
    and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (x, e_p, e, cd, s'))))"
  shows "gas s'a < gas st ∨ gas s'a = gas st"
proof -
  from assms have "gas s' < gas st ∨ gas s' = gas st" by (auto split:if_split_asm)
  with assms show ?thesis using x1[of x e_p e cd s' val s'a] by auto
qed

lemma x2sub:
  assumes "(if gas st ≤ costs (TRANSFER ad ex) e_p e cd st then throw Gas st
        else (get ⤜ (λs. put (s⦇gas := gas s - costs (TRANSFER ad ex) e_p e cd st⦈))) st) =
       Normal ((), s')" and
" expr ex e_p e cd s' = Normal ((KValue vb, Value t), s'a)"
and " msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (ex, e_p, e, cd, s'))))"
and "(⋀ad i xe val e_p e cd st. 0 < costs (EXTERNAL ad i xe val) e_p e cd st)" and "gas s'a ≠ gas st"
shows "gas s'a < gas st"
proof -
  from assms have "gas s' < gas st ∨ gas s' = gas st" by (auto split:if_split_asm)
  with assms show ?thesis using x1[of ex e_p e cd s' "(KValue vb, Value t)" s'a] by auto
qed

lemma x3:
  assumes "(if gas st ≤ c then throw Gas st else (get ⤜ (λs. put (s⦇gas := gas s - c⦈))) st) =
Normal ((), s')"
      and "s'⦇stack := emptyStore⦈ = va"
      and "load False ad xe e_p(ffold (init aa) ⦇address = address e, sender = sender e, svalue = svalue
e, denvalue = fmempty⦈ (fmdom aa)) emptyStore va e cd s' = Normal ((ag, ah, s'd), vc)"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inr (False, ad, xe, e_p, ffold (init aa)
⦇address = address e, sender = sender e, svalue = svalue e, denvalue = fmempty⦈ (fmdom aa), emptyStore,
va, e, cd, s'))))"
      and "c>0"
    shows "gas s'd < gas st"
proof -
  from assms have "gas s'd ≤ gas va" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(5)[of False ad
xe e_p "ffold (init aa) ⦇address = address e, sender = sender e, svalue = svalue e, denvalue = fmempty⦈
(fmdom aa)" emptyStore va e cd s'] by blast
  also from assms(2) have "... = gas s'" by auto
  also from assms(1,5) have "... < gas st" by (auto split:if_split_asm)
  finally show ?thesis .
qed

lemma x4:
  assumes "(if gas st ≤ c then throw Gas st else (get ⤜ (λs. put (s⦇gas := gas s - c⦈))) st) =
Normal ((), s')"
      and "s'⦇stack := emptyStore⦈ = va"
      and "load False ad xe e_p (ffold (init aa) ⦇address = address e, sender = sender e, svalue =
svalue e, denvalue = fmempty⦈ (fmdom aa)) emptyStore va e cd s' =  Normal ((ag, ah, s'd), vc)"
      and "stmt ae e_p ag ah s'd = Normal ((), s'e)"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inr (Inr (ae, e_p, ag, ah, s'd))))"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inr (False, ad, xe, e_p, ffold (init aa)
⦇address = address e, sender = sender e, svalue = svalue e, denvalue = fmempty⦈ (fmdom aa), emptyStore,
va, e, cd, s'))))"
      and "c>0"
    shows "gas s'e < gas st"
proof -
  from assms have "gas s'e ≤ gas s'd" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(7) by blast
  with assms show ?thesis using x3[OF assms(1) assms(2) assms(3) assms(6)] by simp
qed

lemma x5:
  assumes "(if gas st ≤ costs (COMP s1 s2) e_p e cd st then throw Gas st else (get ⤜ (λs. put (s⦇gas
:= gas s - costs (COMP s1 s2) e_p e cd st⦈))) st) = Normal ((), s')"
```

```
      and "stmt s1 e_p e cd s' = Normal ((), s'a)"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inr (Inr (s1, e_p, e, cd, s'))))"
    shows "gas s'a < gas st ∨ gas s'a = gas st"
    using assms msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(7)[of s1 e_p e cd s'] by (auto
split:if_split_asm)

lemma x6:
  assumes "(if gas st ≤ costs (WHILE ex s0) e_p e cd st then throw Gas st else (get ≫= (λs. put
(s⦇gas := gas s - costs (WHILE ex s0) e_p e cd st⦈))) st) = Normal ((), s')"
      and "expr ex e_p e cd s' = Normal (val, s'a)"
      and "stmt s0 e_p e cd s'a = Normal ((), s'b)"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inr (Inr (s0, e_p, e, cd, s'a))))"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (ex, e_p, e, cd, s'))))"
    shows "gas s'b < gas st"
proof -
  from assms have "gas s'b ≤ gas s'a" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(7)[of s0 e_p e
cd s'a] by blast
  also from assms have "... ≤ gas s'" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(4)[of ex e_p e
cd s'] by auto
  also from assms(1) have "... < gas st" using while_not_zero by (auto split:if_split_asm)
  finally show ?thesis .
qed

lemma x7:
  assumes "(if gas st ≤ c then throw Gas st else (get ≫= (λs. put (s⦇gas := gas s - c⦈))) st) =
Normal ((), s')"
      and "expr ad e_p e cd s' = Normal ((KValue vb, Value TAddr), s'a)"
      and "expr val e_p e cd s'a = Normal ((KValue vd, Value ta), s'b)"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (val, e_p, e, cd, s'a))))"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (ad, e_p, e, cd, s'))))"
      and "c>0"
    shows "gas s'b < gas st"
proof -
  from assms(4,3) have "gas s'b ≤ gas s'a" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(4)[of val
e_p e cd s'a] by simp
  also from assms(5,2) have "... ≤ gas s'" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(4)[of ad
e_p e cd s'] by simp
  also from assms(1,6) have "... < gas st" by (auto split:if_split_asm)
  finally show ?thesis .
qed

lemma x8:
  assumes "(if gas st ≤ costs (TRANSFER ad ex) e_p e cd st then throw Gas st else (get ≫= (λs. put
(s⦇gas := gas s - costs (TRANSFER ad ex) e_p e cd st⦈))) st) = Normal ((), s')"
      and "expr ex e_p e cd s' = Normal ((KValue vb, Value t), s'a)"
      and "expr ad e_p e cd s'a = Normal ((KValue vd, Value TAddr), s'b)"
      and "s'b⦇accounts := ab, stack := emptyStore, memory := emptyStore⦈ = s'e"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (ad, e_p, e, cd, s'a))))"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (ex, e_p, e, cd, s'))))"
    shows "gas s'e < gas st"
proof -
  from assms(4) have "gas s'e = gas s'b" by auto
  also from assms(5,3) have "... ≤ gas s'a" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(4)[of ad
e_p e cd s'a] by simp
  also from assms(6,2) have "... ≤ gas s'" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(4)[of ex
e_p e cd s'] by simp
  also from assms(1) have "... < gas st" using transfer_not_zero by (auto split:if_split_asm)
  finally show ?thesis .
qed

lemma x9:
  assumes "(if gas st ≤ costs (BLOCK ((id0, tp), Some a) s) e_p e_v cd st then throw Gas st else (get
≫= (λsa. put (sa⦇gas := gas sa - costs (BLOCK ((id0, tp), Some a) s) e_p e_v cd st⦈))) st) = Normal ((),
s')"
```

```
      and "expr a ep ev cd s' = Normal ((aa, b), s'a)"
      and "decl id0 tp (Some (aa, b)) False cd vb cd ev s'a = Normal ((ab, ba), s'c)"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (a, ep, ev, cd, s')))))"
    shows "gas s'c < gas st ∨ gas s'c = gas st"
proof -
  from assms have "gas s'c = gas s'a" using decl_gas_address[of id0 tp "(Some (aa, b))" False cd vb cd
ev s'a] by simp
  also from assms have "... ≤ gas s'" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(4)[of a ep ev
cd s'] by simp
  also from assms(1) have "... ≤ gas st" by (auto split:if_split_asm)
  finally show ?thesis by auto
qed

lemma x10:
  assumes "(if gas st ≤ costs (BLOCK ((id0, tp), None) s) ep ev cd st then throw Gas st else (get ≫=
(λsa. put (sa(|gas := gas sa - costs (BLOCK ((id0, tp), None) s) ep ev cd st|)))) st) = Normal ((), s')"
      and "decl id0 tp None False cd va cd ev s' = Normal ((a, b), s'b)"
    shows "gas s'b < gas st ∨ gas s'b = gas st"
proof -
  from assms have "gas s'b = gas s'" using decl_gas_address[of id0 tp None False cd va cd ev s'] by
simp
  also from assms(1) have "... ≤ gas st" by (auto split:if_split_asm)
  finally show ?thesis by auto
qed

lemma x11:
  assumes "(if gas st ≤ c then throw Gas st else (get ≫= (λs. put (s(|gas := gas s - c|)))) st) =
Normal ((), s')"
      and "expr ad ep e cd s' = Normal ((KValue vb, Value TAddr), s'a)"
      and "expr val ep e cd s'a = Normal ((KValue vd, Value ta), s'b)"
      and "load True af xe ep (ffold (init ab) (|address = vb, sender = address e, svalue = vd, denvalue
= fmempty|) (fmdom ab)) emptyStore (s'b(|stack := emptyStore, memory := emptyStore|)) e cd s'b = Normal
((ak, al, s'g), vh)"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inr (True, af, xe, ep, ffold (init ab)
(|address = vb, sender = address e, svalue = vd, denvalue = fmempty|) (fmdom ab), emptyStore, s'b(|stack
:= emptyStore, memory := emptyStore|), e, cd, s'b))))"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (val, ep, e, cd, s'a))))"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (ad, ep, e, cd, s')))))"
      and "c>0"
    shows "gas s'g < gas st"
proof -
  from assms have "gas s'g ≤ gas (s'b(|stack := emptyStore, memory := emptyStore|))" using
msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(5)[of True af xe ep "ffold (init ab) (|address = vb, sender =
address e, svalue = vd, denvalue = fmempty|) (fmdom ab)" emptyStore "s'b(|stack := emptyStore, memory :=
emptyStore|)" e cd s'b] by blast
  also have "... = gas s'b" by simp
  also from assms have "... < gas st" using x7[OF assms(1) assms(2) assms(3) assms(6)] by auto
  finally show ?thesis .
qed

lemma x12:
  assumes "(if gas st ≤ c then throw Gas st else (get ≫= (λs. put (s(|gas := gas s - c|)))) st) =
Normal ((), s')"
      and "expr ad ep e cd s' = Normal ((KValue vb, Value TAddr), s'a)"
      and "expr val ep e cd s'a = Normal ((KValue vd, Value ta), s'b)"
      and "load True af xe ep (ffold (init ab) (|address = vb, sender = address e, svalue = vd, denvalue
= fmempty|) (fmdom ab)) emptyStore (s'b(|stack := emptyStore, memory := emptyStore|)) e cd s'b = Normal
((ak, al, s'g), vh)"
      and "stmt ag ep ak al (s'g(|accounts := ala|)) = Normal ((), s'h)"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inr (True, af, xe, ep, ffold (init ab)
(|address = vb, sender = address e, svalue = vd, denvalue = fmempty|) (fmdom ab), emptyStore, s'b(|stack
:= emptyStore, memory := emptyStore|), e, cd, s'b))))"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (val, ep, e, cd, s'a))))"
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inl (Inl (ad, ep, e, cd, s')))))"
```

```
      and "msel_ssel_lexp_expr_load_rexp_stmt_dom (Inr (Inr (Inr (ag, e_p, ak, al, (s'g(|accounts :=
ala|))))))"
      and "c>0"
    shows "gas s'h < gas st"
proof -
  from assms have "gas s'h ≤ gas (s'g(|accounts := ala|))" using msel_ssel_lexp_expr_load_rexp_stmt_dom_gas(7)
by blast
  also from assms have "... < gas st" using x11[OF assms(1) assms(2) assms(3) assms(4)] by auto
  finally show ?thesis .
qed

termination
  apply (relation
      "measures [λx. case x of Inr (Inr (Inr l)) ⇒ gas (snd (snd (snd (snd l))))
                           | Inr (Inr (Inl l)) ⇒ gas (snd (snd (snd (snd l))))
                           | Inr (Inl (Inr l)) ⇒ gas (snd (snd (snd (snd (snd (snd (snd (snd (snd
l)))))))))
                           | Inr (Inl (Inl l)) ⇒ gas (snd (snd (snd (snd l))))
                           | Inl (Inr (Inr l)) ⇒ gas (snd (snd (snd (snd l))))
                           | Inl (Inr (Inl l)) ⇒ gas (snd (snd (snd (snd (snd (snd l))))))
                           | Inl (Inl l) ⇒ gas (snd (snd (snd (snd (snd (snd (snd l))))))),
                λx. case x of Inr (Inr (Inr l)) ⇒ 1
                           | Inr (Inr (Inl l)) ⇒ 0
                           | Inr (Inl (Inr l)) ⇒ 0
                           | Inr (Inl (Inl l)) ⇒ 0
                           | Inl (Inr (Inr l)) ⇒ 0
                           | Inl (Inr (Inl l)) ⇒ 0
                           | Inl (Inl l) ⇒ 0,
                λx. case x of Inr (Inr (Inr l)) ⇒ size (fst l)
                           | Inr (Inr (Inl l)) ⇒ size (fst l)
                           | Inr (Inl (Inr l)) ⇒ size_list size (fst (snd (snd l)))
                           | Inr (Inl (Inl l)) ⇒ size (fst l)
                           | Inl (Inr (Inr l)) ⇒ size (fst l)
                           | Inl (Inr (Inl l)) ⇒ size_list size (fst (snd (snd l)))
                           | Inl (Inl l) ⇒ size_list size (fst (snd (snd (snd l))))]
  ")
  apply simp_all
  apply (simp only: x1)
  apply (simp only: x1)
  apply (simp only: x1)
  apply (auto split: if_split_asm)[1]
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (simp only: x2)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (auto split: if_split_asm)[1]
  apply (auto split: if_split_asm)[1]
  using call_not_zero apply (simp only: x3)
  using call_not_zero apply (simp add: x4)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  using ecall_not_zero apply (simp add: x7)
  using ecall_not_zero apply (auto simp add: x11)[1]
```

```
  using ecall_not_zero apply (auto simp add: x12)[1]
  apply (simp only: x1)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (simp only: x2)
  apply (simp only: x2)
  apply (simp only: x2)
  apply (simp only: x2)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x5)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (simp only: x2)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  apply (simp only: x6)
  apply (auto split: if_split_asm)[1]
  using invoke_not_zero apply (simp only: x3)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x2)
  using external_not_zero apply (simp add: x7)
  using external_not_zero apply (auto simp add: x11)[1]
  using external_not_zero apply (auto simp add: x7)[1]
  apply (auto split: if_split_asm)[1]
  apply (simp add: x2)
  apply (simp add: x8)
  apply (auto split: if_split_asm)[1]
  apply (simp only: x9)
  apply (simp only: x10)
  done
end
```

5.1.6 A minimal cost model

```
fun costs_min :: "S⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ State ⇒ Gas"
  where
  "costs_min SKIP e_p e cd st = 0"
| "costs_min (ASSIGN lv ex) e_p e cd st = 0"
| "costs_min (COMP s1 s2) e_p e cd st = 0"
| "costs_min (ITE ex s1 s2) e_p e cd st = 0"
| "costs_min (WHILE ex s0) e_p e cd st = 1"
| "costs_min (TRANSFER ad ex) e_p e cd st = 1"
| "costs_min (BLOCK ((id0, tp), ex) s) e_p e cd st =0"
| "costs_min (INVOKE _ _) e_p e cd st = 1"
| "costs_min (EXTERNAL _ _ _ _) e_p e cd st = 1"

fun costs_ex :: "E ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ State ⇒ Gas"
  where
  "costs_ex (E.INT _ _) e_p e cd st = 0"
| "costs_ex (UINT _ _) e_p e cd st = 0"
| "costs_ex (ADDRESS _) e_p e cd st = 0"
| "costs_ex (BALANCE _) e_p e cd st = 0"
| "costs_ex THIS e_p e cd st = 0"
| "costs_ex SENDER e_p e cd st = 0"
| "costs_ex VALUE e_p e cd st = 0"
| "costs_ex (TRUE) e_p e cd st = 0"
| "costs_ex (FALSE) e_p e cd st = 0"
| "costs_ex (LVAL _) e_p e cd st = 0"
| "costs_ex (PLUS _ _) e_p e cd st = 0"
| "costs_ex (MINUS _ _) e_p e cd st = 0"
| "costs_ex (EQUAL _ _) e_p e cd st = 0"
| "costs_ex (LESS _ _) e_p e cd st = 0"
| "costs_ex (AND _ _) e_p e cd st = 0"
| "costs_ex (OR _ _) e_p e cd st = 0"
```

```
| "costs_ex (NOT _) ep e cd st = 0"
| "costs_ex (CALL _ _) ep e cd st = 1"
| "costs_ex (ECALL _ _ _ _) ep e cd st = 1"

global_interpretation solidity: statement_with_gas costs_min costs_ex
  defines stmt = "solidity.stmt"
      and lexp   = solidity.lexp
      and expr   = solidity.expr
      and ssel   = solidity.ssel
      and rexp   = solidity.rexp
      and msel   = solidity.msel
      and load   = solidity.load
  by unfold_locales auto

end
```

5.2 The Main Entry Point (Solidity_Main)

```
theory
  Solidity_Main
imports
  Valuetypes
  Storage
  Environment
  Statements
begin
```

This theory is the main entry point into the session Solidity, i.e., it serves the same purpose as *Main* for the session HOL.

It is based on Solidity v0.5.16 https://docs.soliditylang.org/en/v0.5.16/index.html

end

6 A Solidity Evaluation System

This chapter discussed a tactic for symbolically executing Solidity statements and expressions as well as provides a configuration for Isabelle's code generator that allows us to generate an efficient implementation of our executable formal semantics in, e.g., Haskell, SML, or Scala. In our test framework, we use Haskell as a target language.

6.1 Towards a Setup for Symbolic Evaluation of Solidity (Solidity_Symbex)

In this chapter, we lay out the foundations for a tactic for executing Solidity statements and expressions symbolically.

```
theory Solidity_Symbex
imports
  Main
  "HOL-Eisbach.Eisbach"
begin

lemma string_literal_cat: "a+b = String.implode ((String.explode a) @ (String.explode b))"
  by (metis String.implode_explode_eq plus_literal.rep_eq)


lemma string_literal_conv: "(map String.ascii_of y = y) ⟹  (x = String.implode y) = (String.explode
x = y) "
  by auto

lemmas string_literal_opt = Literal.rep_eq zero_literal.rep_eq plus_literal.rep_eq
                            string_literal_cat  string_literal_conv

named_theorems solidity_symbex
method solidity_symbex declares solidity_symbex =
   ((simp add:solidity_symbex cong:unit.case), (simp add:string_literal_opt)?; (code_simp,simp?)+)

declare Let_def [solidity_symbex]
        o_def [solidity_symbex]

end
```

6.2 Solidty Evaluator and Code Generator Setup (Solidity_Evaluator)

```
theory
  Solidity_Evaluator
imports
  Solidity_Main
  "HOL-Library.Code_Target_Numeral"
  "HOL-Library.Sublist"
  "HOL-Library.Finite_Map"
begin
```

6.2.1 Code Generator Setup and Local Tests

Utils

```
definition FAILURE::"String.literal" where "FAILURE = STR ''Failure''"
definition "inta_of_int = int o nat_of_integer"
definition "nat_of_int  = nat_of_integer"
```

```
fun astore :: "Identifier ⇒ Type ⇒ Valuetype ⇒ StorageT * Environment ⇒ StorageT * Environment"
  where "astore i t v (s, e) = (fmupd i v s, (updateEnv i t (Storeloc i) e))"
```

Valuetypes

```
fun dump_Valuetypes::"Types ⇒ Valuetype ⇒ String.literal" where
   "dump_Valuetypes (TSInt _) n = n"
 | "dump_Valuetypes (TUInt _) n = n"
 | "dump_Valuetypes TBool b = (if b = (STR ''True'') then  STR ''true'' else STR ''false'')"
 | "dump_Valuetypes TAddr ad = ad"
```

Generalized Unit Tests

```
lemma  "createSInt 8 500 = STR ''-12''"
  by(eval)

lemma "STR ''-92134039538802366542421159375273829975''
      = createSInt 128 45648483135649456465465452123894894554654654654654646999465"
  by(eval)

lemma "STR ''-128'' = createSInt 8 (-128)"
  by(eval)

lemma "STR ''244'' = (createUInt 8 500)"
  by(eval)

lemma "STR ''22044342891552415597793633092234930760 8''
      = (createUInt 128 456484831356494564654654521238948945546546546546546469994654465465465465416 8)"
  by(eval)

lemma "less (TUInt 144) (TSInt 160) (STR ''5'') (STR ''8'') = Some(STR ''True'', TBool) "
  by(eval)
```

Load: Accounts

```
fun load_Accounts :: "Accounts ⇒ (Address × Balance) list ⇒ Accounts" where
   "load_Accounts acc []              = acc"
 | "load_Accounts acc ((ad, bal)#as) = load_Accounts (updateBalance ad bal acc) as"

definition dump_Accounts :: "Accounts ⇒ Address list ⇒ String.literal"
where
  "dump_Accounts acc = foldl (λ t a . String.implode (    (String.explode t)
                                                      @ (String.explode a)
                                                      @ ''.balance''
                                                      @ ''==''
                                                      @ String.explode (accessBalance acc a)
                                                      @ ''↩''))
                              (STR '''')"

definition init_Account::"(Address × Balance) list => Accounts" where
  "init_Account = load_Accounts emptyAccount"
```

Load: Store

```
type_synonym Data_Store = "(Location × String.literal) list"

fun show_Store::"'a Store ⇒ 'a fset" where
    "show_Store s = (fmran (mapping s))"
```

Load: Memory

```
datatype Data_Memory = MArray "Data_Memory list"
  | MBool bool
  | MInt int
  | MAddress Address

fun
```

```
 loadRec_Memory :: "Location ⇒ Data_Memory ⇒ MemoryT ⇒ MemoryT" where
"loadRec_Memory loc (MArray dat) mem =
(fst (foldl (λ S d . let (s',x) = S in  (loadRec_Memory (hash loc (ShowL_nat x)) d s', Suc x))
(updateStore loc (MPointer loc) mem,0) dat))"
| "loadRec_Memory loc (MBool b) mem = updateStore loc ((MValue ∘ ShowL_bool) b) mem "
| "loadRec_Memory loc (MInt i) mem = updateStore loc ((MValue ∘ ShowL_int) i) mem "
| "loadRec_Memory loc (MAddress ad) mem = updateStore loc (MValue ad) mem"

definition load_Memory :: "Data_Memory list ⇒ MemoryT ⇒ MemoryT" where
"load_Memory dat mem = (let l     = ShowL_nat (toploc mem);
                            (m, _) = foldl (λ (m',x) d . (loadRec_Memory (((hash l) ∘ ShowL_nat) x) d m',
Suc x)) (mem, 0) dat
    in (snd ∘ allocate) m)"


fun  dumprec_Memory:: "Location ⇒ MTypes ⇒ MemoryT ⇒ String.literal ⇒ String.literal ⇒
String.literal" where
"dumprec_Memory loc tp mem ls str = ( case accessStore loc mem of
    Some (MPointer l) ⇒ ( case tp of
        (MTArray x t) ⇒ iter (λ i str' . dumprec_Memory ((hash l o ShowL_int) i) t mem
                             (ls + (STR ''['') + (ShowL_int i) + (STR '']'')) str') str x
       | _ ⇒ FAILURE)
    | Some (MValue v) ⇒ (case tp of
        MTValue t ⇒ str + ls + (STR ''=='') + dump_Valuetypes t v + (STR ''↩'')
       | _ ⇒ FAILURE)
   | None ⇒ FAILURE)"

definition dump_Memory :: "Location ⇒ int ⇒ MTypes ⇒ MemoryT ⇒ String.literal ⇒String.literal
⇒String.literal" where
"dump_Memory loc x t mem ls str = iter (λi. dumprec_Memory ((hash loc (ShowL_int i))) t mem (ls + STR
''['' + (ShowL_int i + STR '']''))) str x"
```

Storage

```
datatype Data_Storage =
    SArray "Data_Storage list" |
    SMap "(String.literal × Data_Storage) list" |
    SBool bool |
    SInt int |
    SAddress Address

definition splitAt::"nat ⇒ String.literal ⇒ String.literal × String.literal" where
"splitAt n xs = (String.implode(take n (String.explode xs)), String.implode(drop n (String.explode
xs)))"


fun splitOn':: "'a ⇒ 'a list ⇒ 'a list ⇒ 'a list list" where
   "splitOn' x [] acc = [rev acc]"
 | "splitOn' x (y#ys) acc = (if x = y then (rev acc)#(splitOn' x ys [])
                                      else splitOn' x ys (y#acc))"

fun splitOn::"'a  ⇒ 'a list ⇒ 'a list list" where
"splitOn x xs = splitOn' x xs []"

definition isSuffixOf::"String.literal ⇒ String.literal ⇒ bool" where
"isSuffixOf s x = suffix (String.explode s) (String.explode x)"

definition tolist :: "Location ⇒ String.literal list" where
"tolist s = map String.implode (splitOn (CHR ''.'') (String.explode s))"

abbreviation convert :: "Location ⇒ Location"
  where "convert loc ≡ (if loc= STR ''True'' then STR ''true'' else
    if loc=STR ''False'' then STR ''false'' else loc)"
```

```
fun go_Storage :: "Location ⇒ (String.literal × STypes) ⇒ (String.literal × STypes)" where
  "go_Storage l (s, STArray _ t) = (s + (STR ''['') + (convert l) + (STR '']''), t)"
| "go_Storage l (s, STMap _ t) = (s + (STR ''['') + (convert l) + (STR '']''), t)"
| "go_Storage l (s, STValue t) = (s + (STR ''['') + (convert l) + (STR '']''), STValue t)"

fun dumpSingle_Storage :: "StorageT ⇒ String.literal ⇒ STypes ⇒ (Location × Location) ⇒
String.literal ⇒ String.literal" where
"dumpSingle_Storage sto id' tp (loc,l) str =
    (case foldr go_Storage (tolist loc) (str + id', tp)  of
        (s, STValue t) ⇒ (case fmlookup sto (loc + l) of
            Some v ⇒ s + (STR ''=='') + dump_Valuetypes t v
           | None ⇒ FAILURE)
        | _ ⇒ FAILURE)"


definition ‹sorted_list_of_set' ≡ map_fun id id (folding_on.F insort [])›

lemma sorted_list_of_fset'_def': ‹sorted_list_of_set' = sorted_list_of_set›
  apply(rule ext)
  by (simp add: sorted_list_of_set'_def sorted_list_of_set_def sorted_key_list_of_set_def)

lemma sorted_list_of_set_sort_remdups' [code]:
  ‹sorted_list_of_set' (set xs) = sort (remdups xs)›
  using sorted_list_of_fset'_def' sorted_list_of_set_sort_remdups
  by metis

definition  locations_map :: "Location ⇒ (Location, 'v) fmap ⇒ Location list" where
"locations_map loc = (filter (isSuffixOf ((STR ''.'')+loc))) ∘  sorted_list_of_set' ∘ fset ∘ fmdom"

definition  locations :: "Location ⇒ 'v Store ⇒ Location list" where
"locations loc = locations_map loc ∘ mapping"

fun dump_Storage ::  "StorageT ⇒ Location ⇒ String.literal ⇒ STypes ⇒ String.literal ⇒
String.literal"
where
"dump_Storage sto loc id' (STArray _ t) str = foldl
      (λ s l . dumpSingle_Storage sto id' t ((splitAt (length (String.explode l) - length (String.explode
loc) - 1) l)) s
          + (STR ''↩'')) str (locations_map loc sto)"
| "dump_Storage sto loc id' (STMap _ t) str =
   foldl (λ s l . dumpSingle_Storage sto id' t (splitAt (length (String.explode l) - length
(String.explode loc) - 1) l) s + (STR ''↩'')) str
(locations_map loc sto)"
  | "dump_Storage sto loc id' (STValue t) str = (case fmlookup sto loc of
      Some v ⇒ str + id' + (STR ''=='') + dump_Valuetypes t v + (STR ''↩'')
    | _ ⇒ str)"


fun loadRec_Storage :: "Location ⇒ Data_Storage ⇒ StorageT ⇒ StorageT" where
"loadRec_Storage loc (SArray dat) sto = fst (foldl (λ S d . let (s', x) = S in (loadRec_Storage (hash loc
(ShowL_nat x)) d s', Suc x)) (sto,0) dat)"
| "loadRec_Storage loc (SMap dat) sto  = ( foldr (λ (k, v) s'. loadRec_Storage (hash loc k) v s') dat
sto)"
| "loadRec_Storage loc (SBool b) sto = fmupd loc (ShowL_bool b) sto"
| "loadRec_Storage loc (SInt i)  sto = fmupd loc (ShowL_int i) sto"
| "loadRec_Storage loc (SAddress ad) sto = fmupd loc ad sto"
```

Environment

```
datatype Data_Environment =
    Memarr "Data_Memory list" |
    CDarr "Data_Memory list" |
    Stoarr "Data_Storage list"|
    Stomap "(String.literal × Data_Storage) list" |
```

```
    Stackbool bool |
    Stobool bool |
    Stackint int |
    Stoint int |
    Stackaddr Address |
    Stoaddr Address

fun loadsimple_Environment :: "(Stack × CalldataT × MemoryT × StorageT × Environment)
                    ⇒ (Identifier × Type × Data_Environment) ⇒ (Stack × CalldataT × MemoryT ×
StorageT × Environment)"
  where
"loadsimple_Environment (k, c, m, s, e) (id', tp, d) = (case d of
    Stackbool b ⇒
        let (k', e') = astack id' tp (KValue (ShowL_bool b)) (k, e)
        in (k', c, m, s, e')
  | Stobool b ⇒
        let (s', e') = astore id' tp (ShowL_bool b) (s, e)
        in (k, c, m, s', e')
  |  Stackint n ⇒
        let (k', e') = astack id' tp (KValue (ShowL_int n)) (k, e)
        in (k', c, m, s, e')
  |  Stoint n ⇒
        let (s', e') = astore id' tp (ShowL_int n) (s, e)
        in (k, c, m, s', e')
  |  Stackaddr ad ⇒
        let (k', e') = astack id' tp (KValue ad) (k, e)
        in (k', c, m, s, e')
  |  Stoaddr ad ⇒
        let (s', e') = astore id' tp ad (s, e)
        in (k, c, m, s', e')
  |  CDarr a ⇒
        let l = ShowL_nat (toploc c);
            c' = load_Memory a c;
            (k', e') = astack id' tp (KCDptr l) (k, e)
        in (k', c', m, s, e')
  |  Memarr a ⇒
        let l = ShowL_nat (toploc m);
            m' = load_Memory a m;
            (k', e') = astack id' tp (KMemptr l) (k, e)
        in (k', c, m', s, e')
  |  Stoarr a ⇒
        let s' = loadRec_Storage id' (SArray a) s;
            e' = updateEnv id' tp (Storeloc id') e
        in (k, c, m, s', e')
  |  Stomap mp ⇒
        let s' = loadRec_Storage id' (SMap mp) s;
            e' = updateEnv id' tp (Storeloc id') e
        in (k, c, m, s', e')
)"

definition load_Environment::"(Stack × CalldataT × MemoryT × StorageT × Environment) ⇒ (Identifier ×
Type × Data_Environment) list
                      ⇒ (Stack × CalldataT × MemoryT × StorageT × Environment)"
  where
"load_Environment = foldl loadsimple_Environment"

definition getValue_Environment :: "Stack ⇒ CalldataT ⇒ MemoryT ⇒ StorageT ⇒ Environment ⇒
Identifier ⇒ String.literal ⇒ String.literal"
  where
"getValue_Environment k c m s e i txt = (case fmlookup (denvalue e) i of
    Some (tp, Stackloc l) ⇒ (case accessStore l k of
        Some (KValue v) ⇒ (case tp of
            Value t ⇒ (txt + i + (STR ''=='') + dump_Valuetypes t v + (STR ''↩''))
          |  _ ⇒ FAILURE)
```

```
      | Some (KCDptr p) ⇒ (case tp of
            Calldata (MTArray x t) ⇒ dump_Memory p x t c i txt
            | _ ⇒ FAILURE)
      | Some (KMemptr p) ⇒ (case tp of
            Memory (MTArray x t) ⇒ dump_Memory p x t m i txt
            | _ ⇒ FAILURE)
      | Some (KStoptr p) ⇒ (case tp of
            Storage t ⇒ dump_Storage s p i t txt
            | _ ⇒ FAILURE))
   | Some (Storage t, Storeloc l) ⇒ dump_Storage s l i t txt
   | _ ⇒ FAILURE
)"
```

```
type_synonym Data_P = "(Address × ((Identifier × Member) list × S))"
```

```
definition dump_Environment :: "Stack  ⇒ CalldataT ⇒ MemoryT ⇒ StorageT ⇒ Environment ⇒ Identifier
list ⇒ String.literal"
  where "dump_Environment k c m s e sl = foldr (getValue_Environment k c m s e) sl  (STR '''')"
```

```
fun loadProc::"Environment_P ⇒ Data_P ⇒ Environment_P"
  where "loadProc e_P (ad, (xs, fb)) = fmupd ad (fmap_of_list xs, fb) e_P"
```

```
fun initStorage::"(Address × Balance) list ⇒ (Address, StorageT) fmap ⇒ (Address, StorageT) fmap"
  where "initStorage [] m = m"
  | "initStorage (x # xs) m = fmupd (fst x) (fmempty) m"
```

6.2.2 Test Setup

```
definition eval::"Gas ⇒ (S ⇒ Environment_P ⇒ Environment ⇒ CalldataT ⇒ (unit, Ex, State)
state_monad)
                ⇒ S ⇒ Address ⇒ Address ⇒ Valuetype ⇒ (Address × Balance) list
                ⇒ Data_P list
                ⇒ (String.literal × Type × Data_Environment) list
          ⇒ String.literal"
  where "eval g stmt_eval stm addr adest aval acc d dat
        = (let (k,c,m,s,e) = load_Environment (emptyStore, emptyStore, emptyStore, fmempty, emptyEnv
addr adest aval) dat;
               e_p        = foldl loadProc fmempty d;
               a          = init_Account acc;
               s'         = fmupd addr s (initStorage acc fmempty);
               z          = (|accounts=a,stack=k,memory=m,storage=s',gas=g|)
           in (
             case (stmt_eval stm e_p e c z) of
              Normal ((), z') ⇒ (dump_Environment (stack z') c (memory z') (the (fmlookup (storage z')
addr)) e (map (λ (a,b,c). a) dat))
                              + (dump_Accounts (accounts z') (map fst acc))
             | Exception Err   ⇒ STR ''Exception''
             | Exception Gas   ⇒ STR ''OutOfGas''))"
```

```
value "eval 1
            stmt
            SKIP
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''),(STR
''115f6e2F70210C14f7DB1AC69737a3CC78435d49'', STR ''100'')]
            []
            [(STR ''v1'', (Value TBool, Stackbool True))]"
```

```
lemma "eval 1000
            stmt
            SKIP
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
```

```
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''), (STR
''115f6e2F70210C14f7DB1AC69737a3CC78435d49'', STR ''100'')]
            []
            [(STR ''v1'', (Value TBool, Stackbool True))]
     = STR ''v1==true↩089Be5381FcEa58aF334101414c04F993947C733.balance==100↩115f6e2F70210C14f7DB1AC69737a3CC78
  by(eval)

value "eval 1000
            stmt
            SKIP
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''), (STR
''115f6e2F70210C14f7DB1AC69737a3CC78435d49'', STR ''100'')]
            []
            [(STR ''v1'', (Storage (STArray 5 (STValue TBool)), Stoarr [SBool True, SBool False, SBool
True, SBool False, SBool True]))]"


lemma "eval 1000
            stmt
            SKIP
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''), (STR
''115f6e2F70210C14f7DB1AC69737a3CC78435d49'', STR ''100'')]
            []
            [(STR ''v1'',(Memory (MTArray 5 (MTValue TBool)), Memarr [MBool True, MBool False, MBool
True, MBool False, MBool True]))]
       = STR ''v1[0]==true↩v1[1]==false↩v1[2]==true↩v1[3]==false↩v1[4]==true↩089Be5381FcEa58aF3341014
  by(eval)

lemma "eval 1000
            stmt
            (ITE FALSE (ASSIGN (Id (STR ''x'')) TRUE) (ASSIGN (Id (STR ''y'')) TRUE))
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''), (STR
''115f6e2F70210C14f7DB1AC69737a3CC78435d49'', STR ''100'')]
            []
            [(STR ''x'', (Value TBool, Stackbool False)), (STR ''y'', (Value TBool, Stackbool False))]
       = STR ''y==true↩x==false↩089Be5381FcEa58aF334101414c04F993947C733.balance==100↩115f6e2F70210C14f7DB1
  by(eval)

lemma "eval 1000
            stmt
            (BLOCK ((STR ''v2'', Value TBool), None) (ASSIGN (Id (STR ''v1'')) (LVAL (Id (STR
''v2'')))))
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''), (STR
''115f6e2F70210C14f7DB1AC69737a3CC78435d49'', STR ''100'')]
            []
            [(STR ''v1'', (Value TBool, Stackbool True))]
        = STR ''v1==false↩089Be5381FcEa58aF334101414c04F993947C733.balance==100↩115f6e2F70210C14f7DB1AC69737a3
  by(eval)

lemma "eval 1000
```

```
            stmt
            (ASSIGN (Id (STR ''a_s120_21_m8'')) (LVAL (Id (STR ''a_s120_21_s8''))))
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100'')]
            []
            [((STR ''a_s120_21_s8''), Storage (STArray 1 (STArray 2 (STValue (TSInt 120)))), Stoarr
[SArray [SInt 347104507864064359095275590289383142, SInt 565831699297331399489670920129618233]]),
             ((STR ''a_s120_21_m8''), Memory (MTArray 1 (MTArray 2 (MTValue (TSInt 120)))), Memarr
[MArray [MInt (290845675805142398428016622247257774), MInt ((-96834026877269277170645294669272226))]]))]
= STR ''a_s120_21_m8[0][0]==347104507864064359095275590289383142↩a_s120_21_m8[0][1]==5658316992973313994896709201
  by(eval)

lemma "eval 1000
            stmt
            (ASSIGN (Ref (STR ''a_s8_32_m0'') [UINT 8 1]) (LVAL (Ref (STR ''a_s8_31_s7'') [UINT 8 0])))
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100'')]
            []
            [(STR ''a_s8_31_s7'', (Storage (STArray 1 (STArray 3 (STValue (TSInt 8)))), Stoarr [SArray
[SInt ((98)), SInt ((-23)), SInt (36)]])),
             (STR ''a_s8_32_m0'', (Memory (MTArray 2 (MTArray 3 (MTValue (TSInt 8)))), Memarr [MArray
[MInt ((-64)), MInt ((39)), MInt ((-125))], MArray [MInt ((-32)), MInt ((-82)), MInt ((-105))]]))]
       = STR ''a_s8_32_m0[0][0]==-64↩a_s8_32_m0[0][1]==39↩a_s8_32_m0[0][2]==-125↩a_s8_32_m0[1][0]==98↩a_
  by(eval)


lemma "eval 1000
            stmt
            SKIP
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''), (STR
''115f6e2F70210C14f7DB1AC69737a3CC78435d49'', STR ''100'')]
            []
            [(STR ''v1'', (Storage (STMap (TUInt 32) (STValue (TUInt 8))), Stomap [(STR ''2129136830'',
SInt (247))]))]
       = STR ''v1[2129136830]==247↩089Be5381FcEa58aF334101414c04F993947C733.balance==100↩115f6e2F70210C14f7DB1
  by(eval)

value "eval 1000
            stmt
            (INVOKE (STR ''m1'') [])
            (STR ''myaddr'')
            (STR '''')
            (STR ''0'')
            [(STR ''myaddr'', STR ''100'')]
            [
              (STR ''myaddr'',
                ([(STR ''m1'', Method ([], SKIP, None))],
                SKIP))
            ]
            [(STR ''x'', (Value TBool, Stackbool True))]"

lemma "eval 1000
            stmt
            (ASSIGN (Id (STR ''v1'')) (CALL (STR ''m1'') []))
            (STR ''myaddr'')
            (STR '''')
            (STR ''0'')
```

```
            [(STR ''myaddr'', STR ''100'')]
            [
              (STR ''myaddr'',
                ([(STR ''m1'', Method ([], SKIP, Some (UINT 8 5)))],
                SKIP))
            ]
            [(STR ''v1'', (Value (TUInt 8), Stackint 0))]
      = STR ''v1==5↩myaddr.balance==100↩''"
  by (eval)

lemma "eval 1000
            stmt
            (ASSIGN (Id (STR ''v1'')) (CALL (STR ''m1'') [E.INT 8 3, E.INT 8 4]))
            (STR ''myaddr'')
            (STR '''')
            (STR ''0'')
            [(STR ''myaddr'', STR ''100'')]
            [
              (STR ''myaddr'',
                ([(STR ''m1'', Method ([(STR ''v2'', Value (TSInt 8)), (STR ''v3'', Value (TSInt 8))],
SKIP, Some (PLUS (LVAL (Id (STR ''v2''))) (LVAL (Id (STR ''v3''))))))],
                SKIP))
            ]
            [(STR ''v1'', (Value (TSInt 8), Stackint 0))]
      = STR ''v1==7↩myaddr.balance==100↩''"
  by (eval)

lemma "eval 1000
            stmt
            (ASSIGN (Id (STR ''v1'')) (ECALL (ADDRESS (STR ''extaddr'')) (STR ''m1'') [E.INT 8 3, E.INT
8 4] (E.UINT 8 0)))
            (STR ''myaddr'')
            (STR '''')
            (STR ''0'')
            [(STR ''myaddr'', STR ''100'')]
            [
              (STR ''extaddr'',
                ([(STR ''m1'', Method ([(STR ''v2'', Value (TSInt 8)), (STR ''v3'', Value (TSInt 8))],
SKIP, Some (PLUS (LVAL (Id (STR ''v2''))) (LVAL (Id (STR ''v3''))))))],
                SKIP))
            ]
            [(STR ''v1'', (Value (TSInt 8), Stackint 0))]
       = STR ''v1==7↩myaddr.balance==100↩''"
  by (eval)

lemma "eval 1000
            stmt
            (TRANSFER (ADDRESS (STR ''myaddr'')) (UINT 256 10))
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''), (STR
''0x2d5F6f401c770eEAdd68deB348948ed4504c4676'', STR ''100'')]
            [
              (STR ''myaddr'',
                ([], SKIP))
            ]
            []
      = STR ''089Be5381FcEa58aF334101414c04F993947C733.balance==90↩0x2d5F6f401c770eEAdd68deB348948ed4504c4676.bal
  by (eval)

value "eval 1000
            stmt
            (TRANSFER (ADDRESS (STR ''myaddr'')) (UINT 256 10))
```

```
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''), (STR
''0x2d5F6f401c770eEAdd68deB348948ed4504c4676'', STR ''100'')]
            [
              (STR ''myaddr'',
                ([], SKIP))
            ]
            []"

lemma "eval 1000
            stmt
            (COMP(COMP(((ASSIGN (Id (STR ''x''))  (E.UINT 8 0))))(TRANSFER (ADDRESS (STR ''myaddr''))
(UINT 256 5)))(SKIP))
            (STR ''089Be5381FcEa58aF334101414c04F993947C733'')
            (STR '''')
            (STR ''0'')
            [(STR ''089Be5381FcEa58aF334101414c04F993947C733'', STR ''100''), (STR
''115f6e2F70210C14f7DB1AC69737a3CC78435d49'', STR ''100'')]
            [
              (STR ''myaddr'',
                ([], SKIP))
            ]
            [(STR ''x'', (Value (TUInt 8), Stackint 9))]
      = STR ''x==0↩089Be5381FcEa58aF334101414c04F993947C733.balance==95↩115f6e2F70210C14f7DB1AC69737a3CC78435d
  by (eval)

value "eval 1000
            stmt
            (EXTERNAL (ADDRESS (STR ''Victim'')) (STR ''withdraw'') [] (E.UINT 8 0))
            (STR ''Victim'')
            (STR '''')
            (STR ''0'')
            [(STR ''Victim'', STR ''100''), (STR ''Attacker'', STR ''100'')]
            [
              (STR ''Attacker'',
                [(STR ''withdraw'', Method ([], EXTERNAL (ADDRESS (STR ''Victim'')) (STR ''withdraw'')
[] (E.UINT 8 0), None))],
                SKIP),
              (STR ''Victim'',
                [(STR ''withdraw'', Method ([], EXTERNAL (ADDRESS (STR ''Attacker'')) (STR ''withdraw'')
[] (E.UINT 8 0), None))],
                SKIP)
            ]
            []"

value "eval 1000
            stmt
            (INVOKE (STR ''withdraw'') [])
            (STR ''Victim'')
            (STR '''')
            (STR ''0'')
            [(STR ''Victim'', STR ''100''), (STR ''Attacker'', STR ''100'')]
            [
              (STR ''Victim'',
                [(STR ''withdraw'', Method ([], INVOKE (STR ''withdraw'') [], None))],
                SKIP)
            ]
            []"
```

6.2.3 The Final Code Export

```
consts ReadL_S    :: "String.literal ⇒ S"
```

```
consts ReadL_acc :: "String.literal ⇒ (String.literal × String.literal) list"
consts ReadL_dat :: "String.literal ⇒ (String.literal × Type × Data_Environment) list"
consts ReadL_P :: "String.literal ⇒ Data_P list"

code_printing
   constant ReadL_S   ⇀ (Haskell) "Prelude.read"
 | constant ReadL_acc ⇀ (Haskell) "Prelude.read"
 | constant ReadL_dat ⇀ (Haskell) "Prelude.read"
 | constant ReadL_P ⇀ (Haskell) "Prelude.read"

fun main_stub :: "String.literal list ⇒ (int × String.literal)"
  where
   "main_stub [credit, stm, saddr, raddr, val, acc, pr, dat]
        = (0, eval (ReadL_nat credit) stmt (ReadL_S stm) saddr raddr val (ReadL_acc acc) (ReadL_P pr)
(ReadL_dat dat))"
 | "main_stub [stm, saddr, raddr, val, acc, pr, dat]
        = (0, eval 1000 stmt (ReadL_S stm) saddr raddr val (ReadL_acc acc) (ReadL_P pr) (ReadL_dat dat))"
 | "main_stub _ = (2,
          STR ''solidity-evaluator [credit] "Statement" "ContractAddress" "OriginAddress" "Value"↩''
        + STR ''  "(Address * Balance) list" "(Address * ((Identifier * Member) list) * Statement)"
"(Variable * Type * Value) list"↩''
        + STR ''↩'')"

generate_file "code/solidity-evaluator/app/Main.hs" = ‹
module Main where
import System.Environment
import Solidity_Evaluator
import Prelude

main :: IO ()
main = do
  args <- getArgs
  Prelude.putStr(snd $ Solidity_Evaluator.main_stub args)
›

export_generated_files _

export_code eval SKIP main_stub
           in Haskell module_name "Solidity_Evaluator" file_prefix "solidity-evaluator/src"
(string_classes)
```

6.2.4 Demonstrating the Symbolic Execution of Solidity

```
abbreviation P1::S
  where "P1 ≡ COMP (ASSIGN (Id (STR ''sa'')) (LVAL (Id (STR ''ma''))))
                  (ASSIGN (Ref (STR ''sa'') [UINT (8::nat) 0]) TRUE)"
abbreviation myenv::Environment
  where "myenv ≡ updateEnv (STR ''ma'') (Memory (MTArray 1 (MTValue TBool))) (Stackloc (STR ''1''))
                (updateEnv (STR ''sa'') (Storage (STArray 1 (STValue TBool))) (Storeloc (STR ''1''))
                (emptyEnv (STR ''ad'') (STR ''ad'') (STR ''0'')))"

abbreviation mystack::Stack
  where "mystack ≡ updateStore (STR ''1'') (KMemptr (STR ''1'')) emptyStore"

abbreviation mystore::StorageT
  where "mystore ≡ fmempty"

abbreviation mymemory::MemoryT
  where "mymemory ≡ updateStore (STR ''0.1'') (MValue (STR ''false'')) emptyStore"

abbreviation mystorage::StorageT
  where "mystorage ≡ fmupd (STR ''0.1'') (STR ''True'') fmempty"
```

```
lemma "( stmt P1 fmempty myenv emptyStore (|accounts=emptyAccount, stack=mystack, memory=mymemory,
storage=fmupd (STR ''ad'') mystorage fmempty, gas=1000|) ) =
  (Normal ((), (|accounts=emptyAccount, stack=mystack, memory=mymemory, storage=fmupd (STR ''ad'')
mystorage fmempty, gas=1000|) ))"
  by(solidity_symbex)

end
```

6.3 Generating an Exectuable of the Evaluator (Compile_Evaluator)

```
theory
  Compile_Evaluator
  imports
    Solidity_Evaluator
begin


compile_generated_files _  (in Solidity_Evaluator) export_files  "solidity-evaluator"  (executable)
  where ‹fn dir =>
    let
       val modules_src =
        Generated_Files.get_files theory‹Solidity_Evaluator›
        |> filter (fn p => String.isSubstring "src" (Path.implode (#path p)))
        |> map (#path #> Path.implode  #> unsuffix ".hs" #> space_explode "/" #> space_implode "."
                      #> unprefix "code.solidity-evaluator.src.");
       val modules_app =
        Generated_Files.get_files theory‹Solidity_Evaluator›
        |> filter (fn p => String.isSubstring "app" (Path.implode (#path p)))
        |> map (#path #> Path.implode  #> unsuffix ".hs" #> space_explode "/" #> space_implode "."
                      #> unprefix "code.solidity-evaluator.app.");
      val _ =
        GHC.new_project dir
          {name = "solidity-evaluator",
           depends =
            [],
           modules = modules_app};
     val _ = writeln (Path.implode dir)
     val res = Generated_Files.execute dir ‹Build› (String.concat [
                 "echo \"\n  default-extensions:  TypeSynonymInstances, FlexibleInstances\" >>
solidity-evaluator.cabal"
                ," && rm -rf src"
                ," && mv code/solidity-evaluator/src src"
                ," && mv code/solidity-evaluator/app/* src/"
                ," && isabelle ghc_stack install --local-bin-path . ‘pwd‘"])
   in
     writeln (res)
   end›


end
```

7 Applications

In this chapter, we discuss various applications of our Solidity semantics.

7.1 Constant Folding (Constant_Folding)

theory *Constant_Folding*
imports
  Solidity_Main
begin

The following function optimizes expressions w.r.t. gas consumption.

```
fun eupdate :: "E ⇒ E"
and lupdate :: "L ⇒ L"
where
  "lupdate (Id i) = Id i"
| "lupdate (Ref i exp) = Ref i (map eupdate exp)"
| "eupdate (E.INT b v) =
    (if (b∈vbits)
      then if v ≥ 0
        then E.INT b (-(2^(b-1)) + (v+2^(b-1)) mod (2^b))
        else E.INT b (2^(b-1) - (-v+2^(b-1)-1) mod (2^b) - 1)
      else E.INT b v)"
| "eupdate (UINT b v) = (if (b∈vbits) then UINT b (v mod (2^b)) else UINT b v)"
| "eupdate (ADDRESS a) = ADDRESS a"
| "eupdate (BALANCE a) = BALANCE a"
| "eupdate THIS = THIS"
| "eupdate SENDER = SENDER"
| "eupdate VALUE = VALUE"
| "eupdate TRUE = TRUE"
| "eupdate FALSE = FALSE"
| "eupdate (LVAL l) = LVAL (lupdate l)"
| "eupdate (PLUS ex1 ex2) =
    (case (eupdate ex1) of
      E.INT b1 v1 ⇒
        if b1 ∈ vbits
          then (case (eupdate ex2) of
            E.INT b2 v2 ⇒
              if b2∈vbits
                then let v=v1+v2 in
                  if v ≥ 0
                    then E.INT (max b1 b2) (-(2^((max b1 b2)-1)) + (v+2^((max b1 b2)-1)) mod (2^(max b1
b2)))
                    else E.INT (max b1 b2) (2^((max b1 b2)-1) - (-v+2^((max b1 b2)-1)-1) mod (2^(max b1
b2)) - 1)
                else (PLUS (E.INT b1 v1) (E.INT b2 v2))
          | UINT b2 v2 ⇒
              if b2∈vbits ∧ b2 < b1
                then let v=v1+v2 in
                  if v ≥ 0
                    then E.INT b1 (-(2^(b1-1)) + (v+2^(b1-1)) mod (2^b1))
                    else E.INT b1 (2^(b1-1) - (-v+2^(b1-1)-1) mod (2^b1) - 1)
                else PLUS (E.INT b1 v1) (UINT b2 v2)
          | _ ⇒ PLUS (E.INT b1 v1) (eupdate ex2))
        else PLUS (E.INT b1 v1) (eupdate ex2)
    | UINT b1 v1 ⇒
        if b1 ∈ vbits
          then (case (eupdate ex2) of
```

```
          UINT b2 v2 ⇒
            if b2 ∈ vbits
              then UINT (max b1 b2) ((v1 + v2) mod (2^(max b1 b2)))
              else (PLUS (UINT b1 v1) (UINT b2 v2))
        | E.INT b2 v2 ⇒
            if b2∈vbits ∧ b1 < b2
              then let v=v1+v2 in
                if v ≥ 0
                  then E.INT b2 (-(2^(b2-1)) + (v+2^(b2-1)) mod (2^b2))
                  else E.INT b2 (2^(b2-1) - (-v+2^(b2-1)-1) mod (2^b2) - 1)
              else PLUS (UINT b1 v1) (E.INT b2 v2)
        | _ ⇒ PLUS (UINT b1 v1) (eupdate ex2))
      else PLUS (UINT b1 v1) (eupdate ex2)
  | _ ⇒ PLUS (eupdate ex1) (eupdate ex2))"
| "eupdate (MINUS ex1 ex2) =
    (case (eupdate ex1) of
     E.INT b1 v1 ⇒
      if b1 ∈ vbits
        then (case (eupdate ex2) of
          E.INT b2 v2 ⇒
            if b2∈vbits
              then let v=v1-v2 in
                if v ≥ 0
                  then E.INT (max b1 b2) (-(2^((max b1 b2)-1)) + (v+2^((max b1 b2)-1)) mod (2^(max b1
b2)))
                  else E.INT (max b1 b2) (2^((max b1 b2)-1) - (-v+2^((max b1 b2)-1)-1) mod (2^(max b1
b2)) - 1)
              else (MINUS (E.INT b1 v1) (E.INT b2 v2))
        | UINT b2 v2 ⇒
            if b2∈vbits ∧ b2 < b1
              then let v=v1-v2 in
                if v ≥ 0
                  then E.INT b1 (-(2^(b1-1)) + (v+2^(b1-1)) mod (2^b1))
                  else E.INT b1 (2^(b1-1) - (-v+2^(b1-1)-1) mod (2^b1) - 1)
              else MINUS (E.INT b1 v1) (UINT b2 v2)
        | _ ⇒ MINUS (E.INT b1 v1) (eupdate ex2))
      else MINUS (E.INT b1 v1) (eupdate ex2)
  | UINT b1 v1 ⇒
      if b1 ∈ vbits
        then (case (eupdate ex2) of
          UINT b2 v2 ⇒
            if b2 ∈ vbits
              then UINT (max b1 b2) ((v1 - v2) mod (2^(max b1 b2)))
              else (MINUS (UINT b1 v1) (UINT b2 v2))
        | E.INT b2 v2 ⇒
            if b2∈vbits ∧ b1 < b2
              then let v=v1-v2 in
                if v ≥ 0
                  then E.INT b2 (-(2^(b2-1)) + (v+2^(b2-1)) mod (2^b2))
                  else E.INT b2 (2^(b2-1) - (-v+2^(b2-1)-1) mod (2^b2) - 1)
              else MINUS (UINT b1 v1) (E.INT b2 v2)
        | _ ⇒ MINUS (UINT b1 v1) (eupdate ex2))
      else MINUS (UINT b1 v1) (eupdate ex2)
  | _ ⇒ MINUS (eupdate ex1) (eupdate ex2))"
| "eupdate (EQUAL ex1 ex2) =
    (case (eupdate ex1) of
     E.INT b1 v1 ⇒
      if b1 ∈ vbits
        then (case (eupdate ex2) of
          E.INT b2 v2 ⇒
            if b2∈vbits
              then if v1 = v2
                then TRUE
                else FALSE
```

```
              else EQUAL (E.INT b1 v1) (E.INT b2 v2)
        | UINT b2 v2 ⇒
            if b2∈vbits ∧ b2 < b1
              then if v1 = v2
                then TRUE
                else FALSE
              else EQUAL (E.INT b1 v1) (UINT b2 v2)
        | _ ⇒ EQUAL (E.INT b1 v1) (eupdate ex2))
      else EQUAL (E.INT b1 v1) (eupdate ex2)
  | UINT b1 v1 ⇒
      if b1 ∈ vbits
        then (case (eupdate ex2) of
          UINT b2 v2 ⇒
            if b2 ∈ vbits
              then if v1 = v2
                then TRUE
                else FALSE
              else EQUAL (E.INT b1 v1) (UINT b2 v2)
        | E.INT b2 v2 ⇒
            if b2∈vbits ∧ b1 < b2
              then if v1 = v2
                  then TRUE
                  else FALSE
              else EQUAL (UINT b1 v1) (E.INT b2 v2)
        | _ ⇒ EQUAL (UINT b1 v1) (eupdate ex2))
      else EQUAL (UINT b1 v1) (eupdate ex2)
  | _ ⇒ EQUAL (eupdate ex1) (eupdate ex2))"
| "eupdate (LESS ex1 ex2) =
  (case (eupdate ex1) of
    E.INT b1 v1 ⇒
      if b1 ∈ vbits
        then (case (eupdate ex2) of
          E.INT b2 v2 ⇒
            if b2∈vbits
              then if v1 < v2
                then TRUE
                else FALSE
              else LESS (E.INT b1 v1) (E.INT b2 v2)
        | UINT b2 v2 ⇒
            if b2∈vbits ∧ b2 < b1
              then if v1 < v2
                then TRUE
                else FALSE
              else LESS (E.INT b1 v1) (UINT b2 v2)
        | _ ⇒ LESS (E.INT b1 v1) (eupdate ex2))
      else LESS (E.INT b1 v1) (eupdate ex2)
  | UINT b1 v1 ⇒
      if b1 ∈ vbits
        then (case (eupdate ex2) of
          UINT b2 v2 ⇒
            if b2 ∈ vbits
              then if v1 < v2
                then TRUE
                else FALSE
              else LESS (E.INT b1 v1) (UINT b2 v2)
        | E.INT b2 v2 ⇒
            if b2∈vbits ∧ b1 < b2
              then if v1 < v2
                  then TRUE
                  else FALSE
              else LESS (UINT b1 v1) (E.INT b2 v2)
        | _ ⇒ LESS (UINT b1 v1) (eupdate ex2))
      else LESS (UINT b1 v1) (eupdate ex2)
  | _ ⇒ LESS (eupdate ex1) (eupdate ex2))"
```

```
| "eupdate (AND ex1 ex2) =
    (case (eupdate ex1) of
      TRUE ⇒ (case (eupdate ex2) of
                 TRUE ⇒ TRUE
               | FALSE ⇒ FALSE
               | _ ⇒ AND TRUE (eupdate ex2))
    | FALSE ⇒ (case (eupdate ex2) of
                 TRUE ⇒ FALSE
               | FALSE ⇒ FALSE
               | _ ⇒ AND FALSE (eupdate ex2))
    | _ ⇒ AND (eupdate ex1) (eupdate ex2))"
| "eupdate (OR ex1 ex2) =
    (case (eupdate ex1) of
      TRUE ⇒ (case (eupdate ex2) of
                 TRUE ⇒ TRUE
               | FALSE ⇒ TRUE
               | _ ⇒ OR TRUE (eupdate ex2))
    | FALSE ⇒ (case (eupdate ex2) of
                 TRUE ⇒ TRUE
               | FALSE ⇒ FALSE
               | _ ⇒ OR FALSE (eupdate ex2))
    | _ ⇒ OR (eupdate ex1) (eupdate ex2))"
| "eupdate (NOT ex1) =
    (case (eupdate ex1) of
      TRUE ⇒ FALSE
    | FALSE ⇒ TRUE
    | _ ⇒ NOT (eupdate ex1))"
| "eupdate (CALL i xs) = CALL i xs"
| "eupdate (ECALL e i xs r) = ECALL e i xs r"

value "eupdate (UINT 8 250)"
lemma "eupdate (UINT 8 250)
      =UINT 8 250"
  by(simp)
lemma "eupdate (UINT 8 500)
      = UINT 8 244"
  by(simp)
lemma "eupdate (E.INT 8 (-100))
      = E.INT 8 (- 100)"
  by(simp)
lemma "eupdate (E.INT 8 (-150))
      = E.INT 8 106"
  by(simp)
lemma "eupdate (PLUS (UINT 8 100) (UINT 8 100))
      = UINT 8 200"
  by(simp)
lemma "eupdate (PLUS (UINT 8 257) (UINT 16 100))
      = UINT 16 101"
  by(simp)
lemma "eupdate (PLUS (E.INT 8 100) (UINT 8 250))
      = PLUS (E.INT 8 100) (UINT 8 250)"
  by(simp)
lemma  "eupdate (PLUS (E.INT 8 250) (UINT 8 500))
      = PLUS (E.INT 8 (- 6)) (UINT 8 244)"
  by(simp)
lemma "eupdate (PLUS (E.INT 16 250) (UINT 8 500))
      = E.INT 16 494"
  by(simp)
lemma "eupdate (EQUAL (UINT 16 250) (UINT 8 250))
      = TRUE"
  by(simp)
lemma  "eupdate (EQUAL (E.INT 16 100) (UINT 8 100))
      = TRUE"
  by(simp)
```

```
lemma "eupdate (EQUAL (E.INT 8 100) (UINT 8 100))
      = EQUAL (E.INT 8 100) (UINT 8 100)"
  by(simp)

lemma update_bounds_int:
  assumes "eupdate ex = (E.INT b v)" and "b∈vbits"
  shows "(v < 2^(b-1)) ∧ v ≥ -(2^(b-1))"
proof (cases ex)
  case (INT b' v')
  then show ?thesis
  proof cases
    assume "b'∈vbits"
    show ?thesis
    proof cases
      let ?x="-(2^(b'-1)) + (v'+2^(b'-1)) mod 2^b'"
      assume "v'≥0"
      with ‘b'∈vbits‘ have "eupdate (E.INT b' v') = E.INT b' ?x" by simp
      with assms have "b=b'" and "v=?x" using INT by (simp,simp)
      moreover from ‘b'∈vbits‘ have "b'>0" by auto
      hence "?x < 2 ^(b'-1)" using upper_bound2[of b' "(v' + 2 ^ (b' - 1)) mod 2^b'"] by simp
      moreover have "?x ≥ -(2^(b'-1))" by simp
      ultimately show ?thesis by simp
    next
      let ?x="2^(b'-1) - (-v'+2^(b'-1)-1) mod (2^b') - 1"
      assume "¬v'≥0"
      with ‘b'∈vbits‘ have "eupdate (E.INT b' v') = E.INT b' ?x" by simp
      with assms have "b=b'" and "v=?x" using INT by (simp,simp)
      moreover have "(-v'+2^(b'-1)-1) mod (2^b')≥0" by simp
      hence "?x < 2 ^(b'-1)" by arith
      moreover from ‘b'∈vbits‘ have "b'>0" by auto
      hence "?x ≥ -(2^(b'-1))" using lower_bound2[of b' v'] by simp
      ultimately show ?thesis by simp
    qed
  next
    assume "¬ b'∈vbits"
    with assms show ?thesis using INT by simp
  qed
next
  case (UINT b' v')
  with assms show ?thesis
  proof cases
    assume "b'∈vbits"
    with assms show ?thesis using UINT by simp
  next
    assume "¬ b'∈vbits"
    with assms show ?thesis using UINT by simp
  qed
next
  case (ADDRESS x3)
  with assms show ?thesis by simp
next
  case (BALANCE x4)
  with assms show ?thesis by simp
next
  case THIS
  with assms show ?thesis by simp
next
  case SENDER
  with assms show ?thesis by simp
next
  case VALUE
  with assms show ?thesis by simp
next
  case TRUE
```

```
  with assms show ?thesis by simp
next
  case FALSE
  with assms show ?thesis by simp
next
  case (LVAL x7)
  with assms show ?thesis by simp
next
  case p: (PLUS e1 e2)
  show ?thesis
  proof (cases "eupdate e1")
    case i: (INT b1 v1)
    show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i2: (INT b2 v2)
        then show ?thesis
        proof cases
          let ?v="v1+v2"
          assume "b2∈vbits"
          show ?thesis
          proof cases
            let ?x="-(2^((max b1 b2)-1)) + (?v+2^((max b1 b2)-1)) mod 2^(max b1 b2)"
            assume "?v≥0"
            with `b1∈vbits` `b2∈vbits` i i2 have "eupdate (PLUS e1 e2) = E.INT (max b1 b2) ?x" by
simp
            with assms have "b=max b1 b2" and "v=?x" using p by (simp,simp)
            moreover from `b1∈vbits` have "max b1 b2>0" by auto
            hence "?x < 2 ^(max b1 b2 - 1)"
              using upper_bound2[of "max b1 b2" "(?v + 2 ^ (max b1 b2 - 1)) mod 2^max b1 b2"] by simp
            moreover have "?x ≥ -(2^(max b1 b2-1))" by simp
            ultimately show ?thesis by simp
          next
            let ?x="2^((max b1 b2)-1) - (-?v+2^((max b1 b2)-1)-1) mod (2^(max b1 b2)) - 1"
            assume "¬?v≥0"
            with `b1∈vbits` `b2∈vbits` i i2 have "eupdate (PLUS e1 e2) = E.INT (max b1 b2) ?x" by
simp
            with assms have "b=max b1 b2" and "v=?x" using p by (simp,simp)
            moreover have "(-?v+2^(max b1 b2-1)-1) mod (2^max b1 b2)≥0" by simp
            hence "?x < 2 ^(max b1 b2-1)" by arith
            moreover from `b1∈vbits` have "max b1 b2>0" by auto
            hence "?x ≥ -(2^(max b1 b2-1))" using lower_bound2[of "max b1 b2" ?v] by simp
            ultimately show ?thesis by simp
          qed
        next
          assume "b2∉vbits"
          with p i i2 `b1∈vbits` show ?thesis using assms by simp
        qed
      next
        case u: (UINT b2 v2)
        then show ?thesis
        proof cases
          let ?v="v1+v2"
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "b2<b1"
            then show ?thesis
            proof cases
              let ?x="(-(2^(b1-1)) + (?v+2^(b1-1)) mod (2^b1))"
              assume "?v≥0"
              with `b1∈vbits` `b2∈vbits` `b2<b1` i u have "eupdate (PLUS e1 e2) = E.INT b1 ?x" by
```

```
simp
          with assms have "b=b1" and "v=?x" using p by (simp,simp)
          moreover from ‘b1∈vbits‘ have "b1>0" by auto
          hence "?x < 2 ^(b1 - 1)" using upper_bound2[of b1] by simp
          moreover have "?x ≥ -(2^(b1-1))" by simp
          ultimately show ?thesis by simp
        next
          let ?x="2^(b1-1) - (-?v+2^(b1-1)-1) mod (2^b1) - 1"
          assume "¬?v≥0"
          with ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ i u have "eupdate (PLUS e1 e2) = E.INT b1 ?x" by
simp
          with assms have "b=b1" and "v=?x" using p i u by (simp,simp)
          moreover have "(-?v+2^(b1-1)-1) mod 2^b1≥0" by simp
          hence "?x < 2 ^(b1-1)" by arith
          moreover from ‘b1∈vbits‘ have "b1>0" by auto
          hence "?x ≥ -(2^(b1-1))" using lower_bound2[of b1 ?v] by simp
          ultimately show ?thesis by simp
        qed
      next
        assume "¬ b2<b1"
        with p i u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "b2∉vbits"
      with p i u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case THIS
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case SENDER
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case VALUE
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case TRUE
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case FALSE
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LVAL x7)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (AND x121 x122)
```

```
        with p i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (OR x131 x132)
        with p i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (NOT x131)
        with p i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (CALL x181 x182)
        with p i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with p i ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with p i show ?thesis using assms by simp
    qed
  next
    case u: (UINT b1 v1)
    show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i: (INT b2 v2)
        then show ?thesis
        proof cases
          let ?v="v1+v2"
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "b1<b2"
            then show ?thesis
            proof cases
              let ?x="(-(2^(b2-1)) + (?v+2^(b2-1)) mod (2^b2))"
              assume "?v≥0"
              with ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ i u have "eupdate (PLUS e1 e2) = E.INT b2 ?x" by
simp
              with assms have "b=b2" and "v=?x" using p by (simp,simp)
              moreover from ‘b2∈vbits‘ have "b2>0" by auto
              hence "?x < 2 ^(b2 - 1)" using upper_bound2[of b2] by simp
              moreover have "?x ≥ -(2^(b2-1))" by simp
              ultimately show ?thesis by simp
            next
              let ?x="2^(b2-1) - (-?v+2^(b2-1)-1) mod (2^b2) - 1"
              assume "¬?v≥0"
              with ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ i u have "eupdate (PLUS e1 e2) = E.INT b2 ?x" by
simp
              with assms have "b=b2" and "v=?x" using p i u by (simp,simp)
              moreover have "(-?v+2^(b2-1)-1) mod 2^b2≥0" by simp
              hence "?x < 2 ^(b2-1)" by arith
              moreover from ‘b2∈vbits‘ have "b2>0" by auto
              hence "?x ≥ -(2^(b2-1))" using lower_bound2[of b2 ?v] by simp
              ultimately show ?thesis by simp
            qed
          next
            assume "¬ b1<b2"
            with p i u ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
          assume "b2∉vbits"
          with p i u ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
```

```
    next
      case u2: (UINT b2 v2)
      then show ?thesis
      proof cases
        assume "b2∈vbits"
        with ‘b1∈vbits‘ u u2 p show ?thesis using assms by simp
      next
        assume "¬b2∈vbits"
        with p u u2 ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      case (ADDRESS x3)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case THIS
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case SENDER
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case VALUE
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case TRUE
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case FALSE
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LVAL x7)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (NOT x131)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with p u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    assume "¬ b1∈vbits"
```

```
      with p u show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with p show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with p show ?thesis using assms by simp
  next
    case THIS
    with p show ?thesis using assms by simp
  next
    case SENDER
    with p show ?thesis using assms by simp
  next
    case VALUE
    with p show ?thesis using assms by simp
  next
    case TRUE
    with p show ?thesis using assms by simp
  next
    case FALSE
    with p show ?thesis using assms by simp
  next
    case (LVAL x7)
    with p show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with p show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with p show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with p show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with p show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with p show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with p show ?thesis using assms by simp
  next
    case (NOT x131)
    with p show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with p show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with p show ?thesis using assms by simp
  qed
next
  case m: (MINUS e1 e2)
  show ?thesis
  proof (cases "eupdate e1")
    case i: (INT b1 v1)
    with m show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
```

```
    case i2: (INT b2 v2)
    then show ?thesis
    proof cases
      let ?v="v1-v2"
      assume "b2∈vbits"
      with ‘b1 ∈ vbits‘ have
        u_def: "eupdate (MINUS e1 e2) =
        (let v = v1 - v2
          in if 0 ≤ v
            then E.INT (max b1 b2)
              (- (2 ^ (max b1 b2 - 1)) + (v + 2 ^ (max b1 b2 - 1)) mod 2 ^ max b1 b2)
            else E.INT (max b1 b2)
              (2 ^ (max b1 b2 - 1) - (- v + 2 ^ (max b1 b2 - 1) - 1) mod 2 ^ max b1 b2 - 1))"
        using i i2 eupdate.simps(11)[of e1 e2] by simp
      show ?thesis
      proof cases
        let ?x="-(2^((max b1 b2)-1)) + (?v+2^((max b1 b2)-1)) mod 2^(max b1 b2)"
        assume "?v≥0"
        with u_def have "eupdate (MINUS e1 e2) = E.INT (max b1 b2) ?x" by simp
        with assms have "b=max b1 b2" and "v=?x" using m by (simp,simp)
        moreover from ‘b1∈vbits‘ have "max b1 b2>0" by auto
        hence "?x < 2 ^(max b1 b2 - 1)"
          using upper_bound2[of "max b1 b2" "(?v + 2 ^ (max b1 b2 - 1)) mod 2^max b1 b2"] by simp
        moreover have "?x ≥ -(2^(max b1 b2-1))" by simp
        ultimately show ?thesis by simp
      next
        let ?x="2^((max b1 b2)-1) - (-?v+2^((max b1 b2)-1)-1) mod (2^(max b1 b2)) - 1"
        assume "¬?v≥0"
        with u_def have "eupdate (MINUS e1 e2) = E.INT (max b1 b2) ?x" using u_def by simp
        with assms have "b=max b1 b2" and "v=?x" using m by (simp,simp)
        moreover have "(-?v+2^(max b1 b2-1)-1) mod (2^max b1 b2)≥0" by simp
        hence "?x < 2 ^(max b1 b2-1)" by arith
        moreover from ‘b1∈vbits‘ have "max b1 b2>0" by auto
        hence "?x ≥ -(2^(max b1 b2-1))" using lower_bound2[of "max b1 b2" ?v] by simp
        ultimately show ?thesis by simp
      qed
    next
      assume "b2∉vbits"
      with m i i2 ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case u: (UINT b2 v2)
    then show ?thesis
    proof cases
      let ?v="v1-v2"
      assume "b2∈vbits"
      show ?thesis
      proof cases
        assume "b2<b1"
        with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have
          u_def: "eupdate (MINUS e1 e2) =
          (let v = v1 - v2
            in if 0 ≤ v
              then E.INT b1 (- (2 ^ (b1 - 1)) + (v + 2 ^ (b1 - 1)) mod 2 ^ b1)
              else E.INT b1 (2 ^ (b1 - 1) - (- v + 2 ^ (b1 - 1) - 1) mod 2 ^ b1 - 1))"
          using i u eupdate.simps(11)[of e1 e2] by simp
        then show ?thesis
        proof cases
          let ?x="(-(2^(b1-1)) + (?v+2^(b1-1)) mod (2^b1))"
          assume "?v≥0"
          with u_def have "eupdate (MINUS e1 e2) = E.INT b1 ?x" by simp
          with assms have "b=b1" and "v=?x" using m by (simp,simp)
          moreover from ‘b1∈vbits‘ have "b1>0" by auto
          hence "?x < 2 ^(b1 - 1)" using upper_bound2[of b1] by simp
```

```
          moreover have "?x ≥ -(2^(b1-1))" by simp
          ultimately show ?thesis by simp
        next
          let ?x="2^(b1-1) - (-?v+2^(b1-1)-1) mod (2^b1) - 1"
          assume "¬?v≥0"
          with u_def have "eupdate (MINUS e1 e2) = E.INT b1 ?x" by simp
          with assms have "b=b1" and "v=?x" using m i u by (simp,simp)
          moreover have "(-?v+2^(b1-1)-1) mod 2^b1≥0" by simp
          hence "?x < 2 ^(b1-1)" by arith
          moreover from ‘b1∈vbits‘ have "b1>0" by auto
          hence "?x ≥ -(2^(b1-1))" using lower_bound2[of b1 ?v] by simp
          ultimately show ?thesis by simp
        qed
      next
        assume "¬ b2<b1"
        with m i u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "b2∉vbits"
      with m i u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case THIS
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case SENDER
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case VALUE
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case TRUE
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case FALSE
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LVAL x7)
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with m i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
```

```
        case (NOT x131)
        with m i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (CALL x181 x182)
        with m i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with m i ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with m i show ?thesis using assms by simp
    qed
  next
    case u: (UINT b1 v1)
    show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i: (INT b2 v2)
        then show ?thesis
        proof cases
          let ?v="v1-v2"
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "b1<b2"
            with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have
              u_def: "eupdate (MINUS e1 e2) =
              (let v = v1 - v2
                in if 0 ≤ v
                  then E.INT b2 (- (2 ^ (b2 - 1)) + (v + 2 ^ (b2 - 1)) mod 2 ^ b2)
                  else E.INT b2 (2 ^ (b2 - 1) - (- v + 2 ^ (b2 - 1) - 1) mod 2 ^ b2 - 1))"
              using i u eupdate.simps(11)[of e1 e2] by simp
            then show ?thesis
            proof cases
              let ?x="(-(2^(b2-1)) + (?v+2^(b2-1)) mod (2^b2))"
              assume "?v≥0"
              with u_def have "eupdate (MINUS e1 e2) = E.INT b2 ?x" by simp
              with assms have "b=b2" and "v=?x" using m by (simp,simp)
              moreover from ‘b2∈vbits‘ have "b2>0" by auto
              hence "?x < 2 ^(b2 - 1)" using upper_bound2[of b2] by simp
              moreover have "?x ≥ -(2^(b2-1))" by simp
              ultimately show ?thesis by simp
            next
              let ?x="2^(b2-1) - (-?v+2^(b2-1)-1) mod (2^b2) - 1"
              assume "¬?v≥0"
              with u_def have "eupdate (MINUS e1 e2) = E.INT b2 ?x" by simp
              with assms have "b=b2" and "v=?x" using m i u by (simp,simp)
              moreover have "(-?v+2^(b2-1)-1) mod 2^b2≥0" by simp
              hence "?x < 2 ^(b2-1)" by arith
              moreover from ‘b2∈vbits‘ have "b2>0" by auto
              hence "?x ≥ -(2^(b2-1))" using lower_bound2[of b2 ?v] by simp
              ultimately show ?thesis by simp
            qed
          next
            assume "¬ b1<b2"
            with m i u ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
          assume "b2∉vbits"
          with m i u ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
```

```
      next
        case u2: (UINT b2 v2)
        then show ?thesis
        proof cases
          assume "b2∈vbits"
          with ‘b1∈vbits‘ u u2 m show ?thesis using assms by simp
        next
          assume "¬b2∈vbits"
          with m u u2 ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
        case (ADDRESS x3)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (BALANCE x4)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case THIS
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case SENDER
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case VALUE
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case TRUE
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case FALSE
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (LVAL x7)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (PLUS x81 x82)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (MINUS x91 x92)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (EQUAL x101 x102)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (LESS x111 x112)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (AND x121 x122)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (OR x131 x132)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (NOT x131)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (CALL x181 x182)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with m u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
```

```
      with m u show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with m show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with m show ?thesis using assms by simp
  next
    case THIS
    with m show ?thesis using assms by simp
  next
    case SENDER
    with m show ?thesis using assms by simp
  next
    case VALUE
    with m show ?thesis using assms by simp
  next
    case TRUE
    with m show ?thesis using assms by simp
  next
    case FALSE
    with m show ?thesis using assms by simp
  next
    case (LVAL x7)
    with m show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with m show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with m show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with m show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with m show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with m show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with m show ?thesis using assms by simp
  next
    case (NOT x131)
    with m show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with m show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with m show ?thesis using assms by simp
  qed
next
  case e: (EQUAL e1 e2)
  show ?thesis
  proof (cases "eupdate e1")
    case i: (INT b1 v1)
    show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
```

```
    case i2: (INT b2 v2)
    then show ?thesis
    proof cases
      assume "b2∈vbits"
      show ?thesis
      proof cases
        assume "v1=v2"
        with assms show ?thesis using e i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
      next
        assume "¬ v1=v2"
        with assms show ?thesis using e i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
      qed
    next
      assume "b2∉vbits"
      with e i i2 ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case u: (UINT b2 v2)
    then show ?thesis
    proof cases
      assume "b2∈vbits"
      show ?thesis
      proof cases
        assume "b2<b1"
        then show ?thesis
        proof cases
          assume "v1=v2"
          with assms show ?thesis using e i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
        next
          assume "¬ v1=v2"
          with assms show ?thesis using e i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
        qed
      next
        assume "¬ b2<b1"
        with e i u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "b2∉vbits"
      with e i u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case THIS
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case SENDER
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case VALUE
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case TRUE
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case FALSE
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LVAL x7)
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
```

```
      next
        case (PLUS x81 x82)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (MINUS x91 x92)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (EQUAL x101 x102)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (LESS x111 x112)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (AND x121 x122)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (OR x131 x132)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (NOT x131)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (CALL x181 x182)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with e i show ?thesis using assms by simp
    qed
  next
    case u: (UINT b1 v1)
    show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i: (INT b2 v2)
        then show ?thesis
        proof cases
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "b1<b2"
            then show ?thesis
            proof cases
              assume "v1=v2"
              with assms show ?thesis using e i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            next
              assume "¬ v1=v2"
              with assms show ?thesis using e i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            qed
          next
            assume "¬ b1<b2"
            with e i u ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
          assume "b2∉vbits"
          with e i u ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
        case u2: (UINT b2 v2)
```

```
    then show ?thesis
    proof cases
      assume "b2∈vbits"
      show ?thesis
      proof cases
        assume "v1=v2"
        with assms show ?thesis using e u u2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
      next
        assume "¬ v1=v2"
        with assms show ?thesis using e u u2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
      qed
    next
      assume "¬b2∈vbits"
      with e u u2 ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case THIS
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case SENDER
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case VALUE
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case TRUE
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case FALSE
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LVAL x7)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (NOT x131)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with e u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
```

```
        case (ECALL x191 x192 x193 x194)
        with e u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with e u show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with e show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with e show ?thesis using assms by simp
  next
    case THIS
    with e show ?thesis using assms by simp
  next
    case SENDER
    with e show ?thesis using assms by simp
  next
    case VALUE
    with e show ?thesis using assms by simp
  next
    case TRUE
    with e show ?thesis using assms by simp
  next
    case FALSE
    with e show ?thesis using assms by simp
  next
    case (LVAL x7)
    with e show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with e show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with e show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with e show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with e show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with e show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with e show ?thesis using assms by simp
  next
    case (NOT x131)
    with e show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with e show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with e show ?thesis using assms by simp
  qed
next
  case l: (LESS e1 e2)
  show ?thesis
  proof (cases "eupdate e1")
    case i: (INT b1 v1)
```

```
show ?thesis
proof cases
  assume "b1∈vbits"
  show ?thesis
  proof (cases "eupdate e2")
    case i2: (INT b2 v2)
    then show ?thesis
    proof cases
      assume "b2∈vbits"
      show ?thesis
      proof cases
        assume "v1<v2"
        with assms show ?thesis using l i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
      next
        assume "¬ v1<v2"
        with assms show ?thesis using l i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
      qed
    next
      assume "b2∉vbits"
      with l i i2 ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case u: (UINT b2 v2)
    then show ?thesis
    proof cases
      assume "b2∈vbits"
      show ?thesis
      proof cases
        assume "b2<b1"
        then show ?thesis
        proof cases
          assume "v1<v2"
          with assms show ?thesis using l i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
        next
          assume "¬ v1<v2"
          with assms show ?thesis using l i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
        qed
      next
        assume "¬ b2<b1"
        with l i u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "b2∉vbits"
      with l i u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with l i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with l i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case THIS
    with l i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case SENDER
    with l i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case VALUE
    with l i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case TRUE
    with l i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
```

```
        case FALSE
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (LVAL x7)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (PLUS x81 x82)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (MINUS x91 x92)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (EQUAL x101 x102)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (LESS x111 x112)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (AND x121 x122)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (OR x131 x132)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (NOT x131)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (CALL x181 x182)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with l i show ?thesis using assms by simp
    qed
  next
    case u: (UINT b1 v1)
    show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i: (INT b2 v2)
        then show ?thesis
        proof cases
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "b1<b2"
            then show ?thesis
            proof cases
              assume "v1<v2"
              with assms show ?thesis using l i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            next
              assume "¬ v1<v2"
              with assms show ?thesis using l i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            qed
          next
            assume "¬ b1<b2"
            with l i u ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
```

```
      assume "b2∉vbits"
      with l i u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case u2: (UINT b2 v2)
    then show ?thesis
    proof cases
      assume "b2∈vbits"
      show ?thesis
      proof cases
        assume "v1<v2"
        with assms show ?thesis using l u u2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
      next
        assume "¬ v1<v2"
        with assms show ?thesis using l u u2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
      qed
    next
      assume "¬b2∈vbits"
      with l u u2 ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case THIS
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case SENDER
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case VALUE
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case TRUE
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case FALSE
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LVAL x7)
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with l u ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (NOT x131)
```

```
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    assume "¬ b1∈vbits"
    with l u show ?thesis using assms by simp
  qed
next
  case (ADDRESS x3)
  with l show ?thesis using assms by simp
next
  case (BALANCE x4)
  with l show ?thesis using assms by simp
next
  case THIS
  with l show ?thesis using assms by simp
next
  case SENDER
  with l show ?thesis using assms by simp
next
  case VALUE
  with l show ?thesis using assms by simp
next
  case TRUE
  with l show ?thesis using assms by simp
next
  case FALSE
  with l show ?thesis using assms by simp
next
  case (LVAL x7)
  with l show ?thesis using assms by simp
next
  case (PLUS x81 x82)
  with l show ?thesis using assms by simp
next
  case (MINUS x91 x92)
  with l show ?thesis using assms by simp
next
  case (EQUAL x101 x102)
  with l show ?thesis using assms by simp
next
  case (LESS x111 x112)
  with l show ?thesis using assms by simp
next
  case (AND x121 x122)
  with l show ?thesis using assms by simp
next
  case (OR x131 x132)
  with l show ?thesis using assms by simp
next
  case (NOT x131)
  with l show ?thesis using assms by simp
next
  case (CALL x181 x182)
  with l show ?thesis using assms by simp
next
  case (ECALL x191 x192 x193 x194)
  with l show ?thesis using assms by simp
qed
```

```
next
  case a: (AND e1 e2)
  show ?thesis
  proof (cases "eupdate e1")
    case (INT x11 x12)
    with a show ?thesis using assms by simp
  next
    case (UINT x21 x22)
    with a show ?thesis using assms by simp
  next
    case (ADDRESS x3)
    with a show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with a show ?thesis using assms by simp
  next
    case THIS
    with a show ?thesis using assms by simp
  next
    case SENDER
    with a show ?thesis using assms by simp
  next
    case VALUE
    with a show ?thesis using assms by simp
  next
    case t: TRUE
    show ?thesis
    proof (cases "eupdate e2")
      case (INT x11 x12)
      with a t show ?thesis using assms by simp
    next
      case (UINT x21 x22)
      with a t show ?thesis using assms by simp
    next
      case (ADDRESS x3)
      with a t show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with a t show ?thesis using assms by simp
    next
      case THIS
      with a t show ?thesis using assms by simp
    next
      case SENDER
      with a t show ?thesis using assms by simp
    next
      case VALUE
      with a t show ?thesis using assms by simp
    next
      case TRUE
      with a t show ?thesis using assms by simp
    next
      case FALSE
      with a t show ?thesis using assms by simp
    next
      case (LVAL x7)
      with a t show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with a t show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with a t show ?thesis using assms by simp
    next
```

```
      case (EQUAL x101 x102)
      with a t show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with a t show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with a t show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with a t show ?thesis using assms by simp
    next
      case (NOT x131)
      with a t show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with a t show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with a t show ?thesis using assms by simp
    qed
  next
    case f: FALSE
    show ?thesis
    proof (cases "eupdate e2")
      case (INT x11 x12)
      with a f show ?thesis using assms by simp
    next
      case (UINT x21 x22)
      with a f show ?thesis using assms by simp
    next
      case (ADDRESS x3)
      with a f show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with a f show ?thesis using assms by simp
    next
      case THIS
      with a f show ?thesis using assms by simp
    next
      case SENDER
      with a f show ?thesis using assms by simp
    next
      case VALUE
      with a f show ?thesis using assms by simp
    next
      case TRUE
      with a f show ?thesis using assms by simp
    next
      case FALSE
      with a f show ?thesis using assms by simp
    next
      case (LVAL x7)
      with a f show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with a f show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with a f show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with a f show ?thesis using assms by simp
    next
```

```
      case (LESS x111 x112)
      with a f show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with a f show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with a f show ?thesis using assms by simp
    next
      case (NOT x131)
      with a f show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with a f show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with a f show ?thesis using assms by simp
    qed
  next
    case (LVAL x7)
    with a show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with a show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with a show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with a show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with a show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with a show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with a show ?thesis using assms by simp
  next
    case (NOT x131)
    with a show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with a show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with a show ?thesis using assms by simp
  qed
next
  case o: (OR e1 e2)
  show ?thesis
  proof (cases "eupdate e1")
    case (INT x11 x12)
    with o show ?thesis using assms by simp
  next
    case (UINT x21 x22)
    with o show ?thesis using assms by simp
  next
    case (ADDRESS x3)
    with o show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with o show ?thesis using assms by simp
```

```
next
  case THIS
  with o show ?thesis using assms by simp
next
  case SENDER
  with o show ?thesis using assms by simp
next
  case VALUE
  with o show ?thesis using assms by simp
next
  case t: TRUE
  show ?thesis
  proof (cases "eupdate e2")
    case (INT x11 x12)
    with o t show ?thesis using assms by simp
  next
    case (UINT x21 x22)
    with o t show ?thesis using assms by simp
  next
    case (ADDRESS x3)
    with o t show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with o t show ?thesis using assms by simp
  next
    case THIS
    with o t show ?thesis using assms by simp
  next
    case SENDER
    with o t show ?thesis using assms by simp
  next
    case VALUE
    with o t show ?thesis using assms by simp
  next
    case TRUE
    with o t show ?thesis using assms by simp
  next
    case FALSE
    with o t show ?thesis using assms by simp
  next
    case (LVAL x7)
    with o t show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with o t show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with o t show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with o t show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with o t show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with o t show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with o t show ?thesis using assms by simp
  next
    case (NOT x131)
    with o t show ?thesis using assms by simp
  next
```

```
      case (CALL x181 x182)
      with o t show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with o t show ?thesis using assms by simp
    qed
  next
    case f: FALSE
    show ?thesis
    proof (cases "eupdate e2")
      case (INT x11 x12)
      with o f show ?thesis using assms by simp
    next
      case (UINT x21 x22)
      with o f show ?thesis using assms by simp
    next
      case (ADDRESS x3)
      with o f show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with o f show ?thesis using assms by simp
    next
      case THIS
      with o f show ?thesis using assms by simp
    next
      case SENDER
      with o f show ?thesis using assms by simp
    next
      case VALUE
      with o f show ?thesis using assms by simp
    next
      case TRUE
      with o f show ?thesis using assms by simp
    next
      case FALSE
      with o f show ?thesis using assms by simp
    next
      case (LVAL x7)
      with o f show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with o f show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with o f show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with o f show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with o f show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with o f show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with o f show ?thesis using assms by simp
    next
      case (NOT x131)
      with o f show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with o f show ?thesis using assms by simp
    next
```

```
      case (ECALL x191 x192 x193 x194)
      with o f show ?thesis using assms by simp
    qed
  next
    case (LVAL x7)
    with o show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with o show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with o show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with o show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with o show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with o show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with o show ?thesis using assms by simp
  next
    case (NOT x131)
    with o show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with o show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with o show ?thesis using assms by simp
  qed
next
  case o: (NOT e1)
  show ?thesis
  proof (cases "eupdate e1")
    case (INT x11 x12)
    with o show ?thesis using assms by simp
  next
    case (UINT x21 x22)
    with o show ?thesis using assms by simp
  next
    case (ADDRESS x3)
    with o show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with o show ?thesis using assms by simp
  next
    case THIS
    with o show ?thesis using assms by simp
  next
    case SENDER
    with o show ?thesis using assms by simp
  next
    case VALUE
    with o show ?thesis using assms by simp
  next
    case t: TRUE
    with o show ?thesis using assms by simp
  next
    case f: FALSE
    with o show ?thesis using assms by simp
```

```
  next
    case (LVAL x7)
    with o show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with o show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with o show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with o show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with o show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with o show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with o show ?thesis using assms by simp
  next
    case (NOT x131)
    with o show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with o show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with o show ?thesis using assms by simp
  qed
next
  case (CALL x181 x182)
  with assms show ?thesis by simp
next
  case (ECALL x191 x192 x193 x194)
  with assms show ?thesis by simp
qed

lemma update_bounds_uint:
  assumes "eupdate ex = UINT b v" and "b∈vbits"
  shows "v < 2^b ∧ v ≥ 0"
proof (cases ex)
  case (INT b' v')
  with assms show ?thesis
  proof cases
    assume "b'∈vbits"
    show ?thesis
    proof cases
      assume "v'≥0"
      with INT show ?thesis using assms ‘b'∈vbits‘ by simp
    next
      assume "¬ v'≥0"
      with INT show ?thesis using assms ‘b'∈vbits‘ by simp
    qed
  next
    assume "¬ b'∈vbits"
    with INT show ?thesis using assms by simp
  qed
next
  case (UINT b' v')
  then show ?thesis
  proof cases
    assume "b'∈vbits"
```

```
      with UINT show ?thesis using assms by auto
    next
      assume "¬ b'∈vbits"
      with UINT show ?thesis using assms by auto
    qed
  next
    case (ADDRESS x3)
    with assms show ?thesis by simp
  next
    case (BALANCE x4)
    with assms show ?thesis by simp
  next
    case THIS
    with assms show ?thesis by simp
  next
    case SENDER
    with assms show ?thesis by simp
  next
    case VALUE
    with assms show ?thesis by simp
  next
    case TRUE
    with assms show ?thesis by simp
  next
    case FALSE
    with assms show ?thesis by simp
  next
    case (LVAL x7)
    with assms show ?thesis by simp
  next
    case p: (PLUS e1 e2)
    show ?thesis
    proof (cases "eupdate e1")
      case i: (INT b1 v1)
      with p show ?thesis
      proof cases
        assume "b1∈vbits"
        show ?thesis
        proof (cases "eupdate e2")
          case i2: (INT b2 v2)
          then show ?thesis
          proof cases
            let ?v="v1+v2"
            assume "b2∈vbits"
            show ?thesis
            proof cases
              assume "?v≥0"
              with assms show ?thesis using p i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
            next
              assume "¬?v≥0"
              with assms show ?thesis using p i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
            qed
          next
            assume "b2∉vbits"
            with p i i2 ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
          case u: (UINT b2 v2)
          then show ?thesis
          proof cases
            let ?v="v1+v2"
            assume "b2∈vbits"
            show ?thesis
            proof cases
```

```
        assume "b2<b1"
        then show ?thesis
        proof cases
          assume "?v≥0"
          with assms show ?thesis using p i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
        next
          assume "¬?v≥0"
          with assms show ?thesis using p i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
        qed
      next
        assume "¬ b2<b1"
        with p i u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "b2∉vbits"
      with p i u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case THIS
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case SENDER
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case VALUE
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case TRUE
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case FALSE
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LVAL x7)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (NOT x131)
    with p i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (CALL x181 x182)
```

```
        with p i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with p i ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with p i show ?thesis using assms by simp
    qed
  next
    case u: (UINT b1 v1)
    with p show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i: (INT b2 v2)
        then show ?thesis
        proof cases
          let ?v="v1+v2"
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "b1<b2"
            then show ?thesis
            proof cases
              assume "?v≥0"
              with assms show ?thesis using p i u ‘b1∈vbits‘  ‘b2∈vbits‘ ‘b1<b2‘ by simp
            next
              assume "¬?v≥0"
              with assms show ?thesis using p i u ‘b1∈vbits‘  ‘b2∈vbits‘ ‘b1<b2‘ by simp
            qed
          next
            assume "¬ b1<b2"
            with p i u ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
          assume "b2∉vbits"
          with p i u ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
        case u2: (UINT b2 v2)
        then show ?thesis
        proof cases
          let ?x="((v1 + v2) mod (2^(max b1 b2)))"
          assume "b2∈vbits"
          with ‘b1∈vbits‘ u u2 have "eupdate (PLUS e1 e2) = UINT (max b1 b2) ?x" by simp
          with assms have "b=max b1 b2" and "v=?x" using p by (simp,simp)
          moreover from ‘b1∈vbits‘ have "max b1 b2>0" by auto
          hence "?x < 2 ^(max b1 b2)" by simp
          moreover have "?x ≥ 0" by simp
          ultimately show ?thesis by simp
        next
          assume "¬b2∈vbits"
          with p u u2 ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
        case (ADDRESS x3)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (BALANCE x4)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case THIS
```

```
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case SENDER
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case VALUE
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case TRUE
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case FALSE
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (LVAL x7)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (PLUS x81 x82)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (MINUS x91 x92)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (EQUAL x101 x102)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (LESS x111 x112)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (AND x121 x122)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (OR x131 x132)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (NOT x131)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (CALL x181 x182)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with p u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with p u show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with p show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with p show ?thesis using assms by simp
  next
    case THIS
    with p show ?thesis using assms by simp
  next
    case SENDER
    with p show ?thesis using assms by simp
  next
    case VALUE
    with p show ?thesis using assms by simp
  next
```

```
      case TRUE
      with p show ?thesis using assms by simp
    next
      case FALSE
      with p show ?thesis using assms by simp
    next
      case (LVAL x7)
      with p show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with p show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with p show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with p show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with p show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with p show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with p show ?thesis using assms by simp
    next
      case (NOT x131)
      with p show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with p show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with p show ?thesis using assms by simp
    qed
  next
    case m: (MINUS e1 e2)
    show ?thesis
    proof (cases "eupdate e1")
      case i: (INT b1 v1)
      with m show ?thesis
      proof cases
        assume "b1∈vbits"
        show ?thesis
        proof (cases "eupdate e2")
          case i2: (INT b2 v2)
          then show ?thesis
          proof cases
            let ?v="v1-v2"
            assume "b2∈vbits"
            show ?thesis
            proof cases
              assume "?v≥0"
              with assms show ?thesis using m i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
            next
              assume "¬?v≥0"
              with assms show ?thesis using m i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
            qed
          next
            assume "b2∉vbits"
            with m i i2 ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
```

```
      case u: (UINT b2 v2)
      then show ?thesis
      proof cases
        let ?v="v1-v2"
        assume "b2∈vbits"
        show ?thesis
        proof cases
          assume "b2<b1"
          show ?thesis
          proof cases
            assume "?v≥0"
            with assms show ?thesis using m i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
          next
            assume "¬?v≥0"
            with assms show ?thesis using m i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
          qed
        next
          assume "¬ b2<b1"
          with m i u ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
        assume "b2∉vbits"
        with m i u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      case (ADDRESS x3)
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case THIS
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case SENDER
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case VALUE
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case TRUE
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case FALSE
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LVAL x7)
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with m i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
```

```
        case (OR x131 x132)
        with m i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (NOT x131)
        with m i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (CALL x181 x182)
        with m i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with m i ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with m i show ?thesis using assms by simp
    qed
  next
    case u: (UINT b1 v1)
    with m show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i: (INT b2 v2)
        then show ?thesis
        proof cases
          let ?v="v1-v2"
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "b1<b2"
            show ?thesis
            proof cases
              assume "?v≥0"
              with assms show ?thesis using m i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            next
              assume "¬?v≥0"
              with assms show ?thesis using m i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            qed
          next
            assume "¬ b1<b2"
            with m i u ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
          assume "b2∉vbits"
          with m i u ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
        case u2: (UINT b2 v2)
        then show ?thesis
        proof cases
          let ?x="((v1 - v2) mod (2^(max b1 b2)))"
          assume "b2∈vbits"
          with ‘b1∈vbits‘ u u2 have "eupdate (MINUS e1 e2) = UINT (max b1 b2) ?x" by simp
          with assms have "b=max b1 b2" and "v=?x" using m by (simp,simp)
          moreover from ‘b1∈vbits‘ have "max b1 b2>0" by auto
          hence "?x < 2 ^(max b1 b2)" by simp
          moreover have "?x ≥ 0" by simp
          ultimately show ?thesis by simp
        next
          assume "¬b2∈vbits"
          with m u u2 ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
```

```
      case (ADDRESS x3)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case THIS
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case SENDER
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case VALUE
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case TRUE
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case FALSE
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LVAL x7)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (NOT x131)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with m u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    assume "¬ b1∈vbits"
    with m u show ?thesis using assms by simp
  qed
next
  case (ADDRESS x3)
  with m show ?thesis using assms by simp
next
  case (BALANCE x4)
  with m show ?thesis using assms by simp
next
  case THIS
  with m show ?thesis using assms by simp
```

```
    next
      case SENDER
      with m show ?thesis using assms by simp
    next
      case VALUE
      with m show ?thesis using assms by simp
    next
      case TRUE
      with m show ?thesis using assms by simp
    next
      case FALSE
      with m show ?thesis using assms by simp
    next
      case (LVAL x7)
      with m show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with m show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with m show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with m show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with m show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with m show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with m show ?thesis using assms by simp
    next
      case (NOT x131)
      with m show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with m show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with m show ?thesis using assms by simp
    qed
  next
    case e: (EQUAL e1 e2)
    show ?thesis
    proof (cases "eupdate e1")
      case i: (INT b1 v1)
      show ?thesis
      proof cases
        assume "b1∈vbits"
        show ?thesis
        proof (cases "eupdate e2")
          case i2: (INT b2 v2)
          then show ?thesis
          proof cases
            assume "b2∈vbits"
            show ?thesis
            proof cases
              assume "v1=v2"
              with assms show ?thesis using e i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
            next
              assume "¬ v1=v2"
              with assms show ?thesis using e i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
```

```
      qed
    next
      assume "b2∉vbits"
      with e i i2 ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case u: (UINT b2 v2)
    then show ?thesis
    proof cases
      assume "b2∈vbits"
      show ?thesis
      proof cases
        assume "b2<b1"
        then show ?thesis
        proof cases
          assume "v1=v2"
          with assms show ?thesis using e i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
        next
          assume "¬ v1=v2"
          with assms show ?thesis using e i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
        qed
      next
        assume "¬ b2<b1"
        with e i u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "b2∉vbits"
      with e i u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case THIS
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case SENDER
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case VALUE
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case TRUE
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case FALSE
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LVAL x7)
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with e i ‘b1∈vbits‘ show ?thesis using assms by simp
  next
    case (LESS x111 x112)
```

```
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (AND x121 x122)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (OR x131 x132)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (NOT x131)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (CALL x181 x182)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with e i ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with e i show ?thesis using assms by simp
    qed
  next
    case u: (UINT b1 v1)
    show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i: (INT b2 v2)
        then show ?thesis
        proof cases
          let ?v="v1+v2"
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "b1<b2"
            then show ?thesis
            proof cases
              assume "v1=v2"
              with assms show ?thesis using e i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            next
              assume "¬ v1=v2"
              with assms show ?thesis using e i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            qed
          next
            assume "¬ b1<b2"
            with e i u ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
          assume "b2∉vbits"
          with e i u ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
        case u2: (UINT b2 v2)
        then show ?thesis
        proof cases
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "v1=v2"
            with assms show ?thesis using e u u2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
          next
            assume "¬ v1=v2"
            with assms show ?thesis using e u u2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
```

```
        qed
      next
        assume "¬b2∈vbits"
        with e u u2 ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      case (ADDRESS x3)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case THIS
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case SENDER
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case VALUE
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case TRUE
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case FALSE
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LVAL x7)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (NOT x131)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with e u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
    assume "¬ b1∈vbits"
    with e u show ?thesis using assms by simp
  qed
next
  case (ADDRESS x3)
  with e show ?thesis using assms by simp
```

```
  next
    case (BALANCE x4)
    with e show ?thesis using assms by simp
  next
    case THIS
    with e show ?thesis using assms by simp
  next
    case SENDER
    with e show ?thesis using assms by simp
  next
    case VALUE
    with e show ?thesis using assms by simp
  next
    case TRUE
    with e show ?thesis using assms by simp
  next
    case FALSE
    with e show ?thesis using assms by simp
  next
    case (LVAL x7)
    with e show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with e show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with e show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with e show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with e show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with e show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with e show ?thesis using assms by simp
  next
    case (NOT x131)
    with e show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with e show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with e show ?thesis using assms by simp
  qed
next
  case l: (LESS e1 e2)
  show ?thesis
  proof (cases "eupdate e1")
    case i: (INT b1 v1)
    show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i2: (INT b2 v2)
        then show ?thesis
        proof cases
          assume "b2∈vbits"
          show ?thesis
```

```
        proof cases
          assume "v1<v2"
          with assms show ?thesis using l i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
        next
          assume "¬ v1<v2"
          with assms show ?thesis using l i i2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
        qed
      next
        assume "b2∉vbits"
        with l i i2 ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      case u: (UINT b2 v2)
      then show ?thesis
      proof cases
        assume "b2∈vbits"
        show ?thesis
        proof cases
          assume "b2<b1"
          then show ?thesis
          proof cases
            assume "v1<v2"
            with assms show ?thesis using l i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
          next
            assume "¬ v1<v2"
            with assms show ?thesis using l i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b2<b1‘ by simp
          qed
        next
          assume "¬ b2<b1"
          with l i u ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
        assume "b2∉vbits"
        with l i u ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      case (ADDRESS x3)
      with l i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with l i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case THIS
      with l i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case SENDER
      with l i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case VALUE
      with l i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case TRUE
      with l i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case FALSE
      with l i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LVAL x7)
      with l i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with l i ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
```

```
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (EQUAL x101 x102)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (LESS x111 x112)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (AND x121 x122)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (OR x131 x132)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (NOT x131)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (CALL x181 x182)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      next
        case (ECALL x191 x192 x193 x194)
        with l i ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      assume "¬ b1∈vbits"
      with l i show ?thesis using assms by simp
    qed
  next
    case u: (UINT b1 v1)
    show ?thesis
    proof cases
      assume "b1∈vbits"
      show ?thesis
      proof (cases "eupdate e2")
        case i: (INT b2 v2)
        then show ?thesis
        proof cases
          let ?v="v1+v2"
          assume "b2∈vbits"
          show ?thesis
          proof cases
            assume "b1<b2"
            then show ?thesis
            proof cases
              assume "v1<v2"
              with assms show ?thesis using l i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            next
              assume "¬ v1<v2"
              with assms show ?thesis using l i u ‘b1∈vbits‘ ‘b2∈vbits‘ ‘b1<b2‘ by simp
            qed
          next
            assume "¬ b1<b2"
            with l i u ‘b1∈vbits‘ show ?thesis using assms by simp
          qed
        next
          assume "b2∉vbits"
          with l i u ‘b1∈vbits‘ show ?thesis using assms by simp
        qed
      next
        case u2: (UINT b2 v2)
        then show ?thesis
        proof cases
          assume "b2∈vbits"
          show ?thesis
```

```
        proof cases
          assume "v1<v2"
          with assms show ?thesis using l u u2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
        next
          assume "¬ v1<v2"
          with assms show ?thesis using l u u2 ‘b1∈vbits‘ ‘b2∈vbits‘ by simp
        qed
      next
        assume "¬b2∈vbits"
        with l u u2 ‘b1∈vbits‘ show ?thesis using assms by simp
      qed
    next
      case (ADDRESS x3)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case THIS
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case SENDER
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case VALUE
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case TRUE
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case FALSE
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LVAL x7)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (NOT x131)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with l u ‘b1∈vbits‘ show ?thesis using assms by simp
    qed
  next
```

```
      assume "¬ b1∈vbits"
      with l u show ?thesis using assms by simp
    qed
  next
    case (ADDRESS x3)
    with l show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with l show ?thesis using assms by simp
  next
    case THIS
    with l show ?thesis using assms by simp
  next
    case SENDER
    with l show ?thesis using assms by simp
  next
    case VALUE
    with l show ?thesis using assms by simp
  next
    case TRUE
    with l show ?thesis using assms by simp
  next
    case FALSE
    with l show ?thesis using assms by simp
  next
    case (LVAL x7)
    with l show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with l show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with l show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with l show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with l show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with l show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with l show ?thesis using assms by simp
  next
    case (NOT x131)
    with l show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with l show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with l show ?thesis using assms by simp
  qed
next
  case a: (AND e1 e2)
  show ?thesis
  proof (cases "eupdate e1")
    case (INT x11 x12)
    with a show ?thesis using assms by simp
  next
    case (UINT x21 x22)
    with a show ?thesis using assms by simp
```

```
next
  case (ADDRESS x3)
  with a show ?thesis using assms by simp
next
  case (BALANCE x4)
  with a show ?thesis using assms by simp
next
  case THIS
  with a show ?thesis using assms by simp
next
  case SENDER
  with a show ?thesis using assms by simp
next
  case VALUE
  with a show ?thesis using assms by simp
next
  case t: TRUE
  show ?thesis
  proof (cases "eupdate e2")
    case (INT x11 x12)
    with a t show ?thesis using assms by simp
  next
    case (UINT x21 x22)
    with a t show ?thesis using assms by simp
  next
    case (ADDRESS x3)
    with a t show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with a t show ?thesis using assms by simp
  next
    case THIS
    with a t show ?thesis using assms by simp
  next
    case SENDER
    with a t show ?thesis using assms by simp
  next
    case VALUE
    with a t show ?thesis using assms by simp
  next
    case TRUE
    with a t show ?thesis using assms by simp
  next
    case FALSE
    with a t show ?thesis using assms by simp
  next
    case (LVAL x7)
    with a t show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with a t show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with a t show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with a t show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with a t show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with a t show ?thesis using assms by simp
  next
```

```
      case (OR x131 x132)
      with a t show ?thesis using assms by simp
    next
      case (NOT x131)
      with a t show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with a t show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with a t show ?thesis using assms by simp
    qed
  next
    case f: FALSE
    show ?thesis
    proof (cases "eupdate e2")
      case (INT x11 x12)
      with a f show ?thesis using assms by simp
    next
      case (UINT x21 x22)
      with a f show ?thesis using assms by simp
    next
      case (ADDRESS x3)
      with a f show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with a f show ?thesis using assms by simp
    next
      case THIS
      with a f show ?thesis using assms by simp
    next
      case SENDER
      with a f show ?thesis using assms by simp
    next
      case VALUE
      with a f show ?thesis using assms by simp
    next
      case TRUE
      with a f show ?thesis using assms by simp
    next
      case FALSE
      with a f show ?thesis using assms by simp
    next
      case (LVAL x7)
      with a f show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with a f show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with a f show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with a f show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with a f show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with a f show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with a f show ?thesis using assms by simp
    next
```

```
      case (NOT x131)
      with a f show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with a f show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with a f show ?thesis using assms by simp
    qed
  next
    case (LVAL x7)
    with a show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with a show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with a show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with a show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with a show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with a show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with a show ?thesis using assms by simp
  next
    case (NOT x131)
    with a show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with a show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with a show ?thesis using assms by simp
  qed
next
  case o: (OR e1 e2)
  show ?thesis
  proof (cases "eupdate e1")
    case (INT x11 x12)
    with o show ?thesis using assms by simp
  next
    case (UINT x21 x22)
    with o show ?thesis using assms by simp
  next
    case (ADDRESS x3)
    with o show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with o show ?thesis using assms by simp
  next
    case THIS
    with o show ?thesis using assms by simp
  next
    case SENDER
    with o show ?thesis using assms by simp
  next
    case VALUE
    with o show ?thesis using assms by simp
```

```
next
  case t: TRUE
  show ?thesis
  proof (cases "eupdate e2")
    case (INT x11 x12)
    with o t show ?thesis using assms by simp
  next
    case (UINT x21 x22)
    with o t show ?thesis using assms by simp
  next
    case (ADDRESS x3)
    with o t show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with o t show ?thesis using assms by simp
  next
    case THIS
    with o t show ?thesis using assms by simp
  next
    case SENDER
    with o t show ?thesis using assms by simp
  next
    case VALUE
    with o t show ?thesis using assms by simp
  next
    case TRUE
    with o t show ?thesis using assms by simp
  next
    case FALSE
    with o t show ?thesis using assms by simp
  next
    case (LVAL x7)
    with o t show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with o t show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with o t show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with o t show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with o t show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with o t show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with o t show ?thesis using assms by simp
  next
    case (NOT x131)
    with o t show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with o t show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with o t show ?thesis using assms by simp
  qed
next
  case f: FALSE
  show ?thesis
```

```
  proof (cases "eupdate e2")
    case (INT x11 x12)
    with o f show ?thesis using assms by simp
  next
    case (UINT x21 x22)
    with o f show ?thesis using assms by simp
  next
    case (ADDRESS x3)
    with o f show ?thesis using assms by simp
  next
    case (BALANCE x4)
    with o f show ?thesis using assms by simp
  next
    case THIS
    with o f show ?thesis using assms by simp
  next
    case SENDER
    with o f show ?thesis using assms by simp
  next
    case VALUE
    with o f show ?thesis using assms by simp
  next
    case TRUE
    with o f show ?thesis using assms by simp
  next
    case FALSE
    with o f show ?thesis using assms by simp
  next
    case (LVAL x7)
    with o f show ?thesis using assms by simp
  next
    case (PLUS x81 x82)
    with o f show ?thesis using assms by simp
  next
    case (MINUS x91 x92)
    with o f show ?thesis using assms by simp
  next
    case (EQUAL x101 x102)
    with o f show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with o f show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with o f show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with o f show ?thesis using assms by simp
  next
    case (NOT x131)
    with o f show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with o f show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with o f show ?thesis using assms by simp
  qed
next
  case (LVAL x7)
  with o show ?thesis using assms by simp
next
  case (PLUS x81 x82)
  with o show ?thesis using assms by simp
```

```
    next
      case (MINUS x91 x92)
      with o show ?thesis using assms by simp
    next
      case (EQUAL x101 x102)
      with o show ?thesis using assms by simp
    next
      case (LESS x111 x112)
      with o show ?thesis using assms by simp
    next
      case (AND x121 x122)
      with o show ?thesis using assms by simp
    next
      case (OR x131 x132)
      with o show ?thesis using assms by simp
    next
      case (NOT x131)
      with o show ?thesis using assms by simp
    next
      case (CALL x181 x182)
      with o show ?thesis using assms by simp
    next
      case (ECALL x191 x192 x193 x194)
      with o show ?thesis using assms by simp
    qed
  next
    case o: (NOT x)
    show ?thesis
    proof (cases "eupdate x")
      case (INT x11 x12)
      with o show ?thesis using assms by simp
    next
      case (UINT x21 x22)
      with o show ?thesis using assms by simp
    next
      case (ADDRESS x3)
      with o show ?thesis using assms by simp
    next
      case (BALANCE x4)
      with o show ?thesis using assms by simp
    next
      case THIS
      with o show ?thesis using assms by simp
    next
      case SENDER
      with o show ?thesis using assms by simp
    next
      case VALUE
      with o show ?thesis using assms by simp
    next
      case t: TRUE
      with o show ?thesis using assms by simp
    next
      case f: FALSE
      with o show ?thesis using assms by simp
    next
      case (LVAL x7)
      with o show ?thesis using assms by simp
    next
      case (PLUS x81 x82)
      with o show ?thesis using assms by simp
    next
      case (MINUS x91 x92)
      with o show ?thesis using assms by simp
```

```
  next
    case (EQUAL x101 x102)
    with o show ?thesis using assms by simp
  next
    case (LESS x111 x112)
    with o show ?thesis using assms by simp
  next
    case (AND x121 x122)
    with o show ?thesis using assms by simp
  next
    case (OR x131 x132)
    with o show ?thesis using assms by simp
  next
    case (NOT x131)
    with o show ?thesis using assms by simp
  next
    case (CALL x181 x182)
    with o show ?thesis using assms by simp
  next
    case (ECALL x191 x192 x193 x194)
    with o show ?thesis using assms by simp
  qed
next
  case (CALL x181 x182)
  with assms show ?thesis by simp
next
  case (ECALL x191 x192 x193 x194)
  with assms show ?thesis by simp
qed

lemma no_gas:
  assumes "¬ gas st > 0"
  shows "expr ex ep env cd st = Exception Gas"
proof (cases ex)
  case (INT x11 x12)
  with assms show ?thesis by simp
next
  case (UINT x21 x22)
  with assms show ?thesis by simp
next
  case (ADDRESS x3)
  with assms show ?thesis by simp
next
  case (BALANCE x4)
  with assms show ?thesis by simp
next
  case THIS
  with assms show ?thesis by simp
next
  case SENDER
  with assms show ?thesis by simp
next
  case VALUE
  with assms show ?thesis by simp
next
  case TRUE
  with assms show ?thesis by simp
next
  case FALSE
  with assms show ?thesis by simp
next
  case (LVAL x10)
  with assms show ?thesis by simp
next
```

```
  case (PLUS x111 x112)
  with assms show ?thesis by simp
next
  case (MINUS x121 x122)
  with assms show ?thesis by simp
next
  case (EQUAL x131 x132)
  with assms show ?thesis by simp
next
  case (LESS x141 x142)
  with assms show ?thesis by simp
next
  case (AND x151 x152)
  with assms show ?thesis by simp
next
  case (OR x161 x162)
  with assms show ?thesis by simp
next
  case (NOT x17)
  with assms show ?thesis by simp
next
  case (CALL x181 x182)
  with assms show ?thesis by simp
next
  case (ECALL x191 x192 x193 x194)
  with assms show ?thesis by simp
qed

lemma lift_eq:
  assumes "expr e1 ep env cd st = expr e1' ep env cd st"
      and "⋀st' rv. expr e1 ep env cd st = Normal (rv, st') ⟹ expr e2 ep env cd st'= expr e2' ep env
cd st'"
    shows "lift expr f e1 e2 ep env cd st=lift expr f e1' e2' ep env cd st"
proof (cases "expr e1 ep env cd st")
  case s1: (n a st')
  then show ?thesis
  proof (cases a)
    case f1:(Pair a b)
    then show ?thesis
    proof (cases a)
      case k1:(KValue x1)
      then show ?thesis
      proof (cases b)
        case v1: (Value x1)
        then show ?thesis
        proof (cases "expr e2 ep env cd st'")
          case s2: (n a' st'')
          then show ?thesis
          proof (cases a')
            case f2:(Pair a' b')
            then show ?thesis
            proof (cases a')
              case (KValue x1')
              with s1 f1 k1 v1 assms(1) assms(2) show ?thesis by auto
            next
              case (KCDptr x2)
              with s1 f1 k1 v1 assms(1) assms(2) show ?thesis by auto
            next
              case (KMemptr x2')
              with s1 f1 k1 v1 assms(1) assms(2) show ?thesis by auto
            next
              case (KStoptr x3')
              with s1 f1 k1 v1 assms(1) assms(2) show ?thesis by auto
            qed
```

```
          qed
        next
          case (e e)
          then show ?thesis using k1 s1 v1 assms(1) assms(2) f1 by auto
        qed
      next
        case (Calldata x2)
        then show ?thesis using k1 s1 assms(1) f1 by auto
      next
        case (Memory x2)
        then show ?thesis using k1 s1 assms(1) f1 by auto
      next
        case (Storage x3)
        then show ?thesis using k1 s1 assms(1) f1 by auto
      qed
    next
      case (KCDptr x2)
      then show ?thesis using s1 assms(1) f1 by fastforce
    next
      case (KMemptr x2)
      then show ?thesis using s1 assms(1) f1 by fastforce
    next
      case (KStoptr x3)
      then show ?thesis using s1 assms(1) f1 by fastforce
    qed
  qed
next
  case (e e)
  then show ?thesis using assms(1) by simp
qed

lemma ssel_eq_ssel:
  "(⋀i st. i ∈ set ix ⟹ expr i ep env cd st = expr (f i) ep env cd st)
  ⟹ ssel tp loc ix ep env cd st = ssel tp loc (map f ix) ep env cd st"
proof (induction ix arbitrary: tp loc ep env cd st)
  case Nil
  then show ?case by simp
next
  case c1: (Cons i ix)
  then show ?case
  proof (cases tp)
    case tp1: (STArray al tp)
    then show ?thesis
    proof (cases "expr i ep env cd st")
      case s1: (n a st')
      then show ?thesis
      proof (cases a)
        case f1: (Pair a b)
        then show ?thesis
        proof (cases a)
          case k1: (KValue v)
          then show ?thesis
          proof (cases b)
            case v1: (Value t)
            then show ?thesis
            proof (cases "less t (TUInt 256) v (ShowL_int al)")
              case None
              with v1 k1 tp1 s1 c1.prems f1 show ?thesis by simp
            next
              case s2: (Some a)
              then show ?thesis
              proof (cases a)
                case p1: (Pair a b)
                then show ?thesis
```

```
              proof (cases b)
                case (TSInt x1)
                with s2 p1 v1 k1 tp1 s1 c1.prems f1 show ?thesis by simp
              next
                case (TUInt x2)
                with s2 p1 v1 k1 tp1 s1 c1.prems f1 show ?thesis by simp
              next
                case b1: TBool
                show ?thesis
                proof cases
                  assume "a = ShowL_bool True"
                  from c1.IH[OF c1.prems] have
                    "ssel tp (hash loc v) ix ep env cd st' = ssel tp (hash loc v) (map f ix) ep env cd
st'"
                    by simp
                  with mp s2 b1 p1 v1 k1 tp1 s1 c1.prems f1 show ?thesis by simp
                next
                  assume "¬ a = ShowL_bool True"
                  with s2 p1 v1 k1 tp1 s1 c1.prems f1 show ?thesis by simp
                qed
              next
                case TAddr
                with s2 p1 v1 k1 tp1 s1 c1.prems f1 show ?thesis by simp
              qed
            qed
          qed
        next
          case (Calldata x2)
          with k1 tp1 s1 c1.prems f1 show ?thesis by simp
        next
          case (Memory x2)
          with k1 tp1 s1 c1.prems f1 show ?thesis by simp
        next
          case (Storage x3)
          with k1 tp1 s1 c1.prems f1 show ?thesis by simp
        qed
      next
        case (KCDptr x2)
        with tp1 s1 c1.prems f1 show ?thesis by simp
      next
        case (KMemptr x2)
        with tp1 s1 c1.prems f1 show ?thesis by simp
      next
        case (KStoptr x3)
        with tp1 s1 c1.prems f1 show ?thesis by simp
      qed
    qed
  next
    case (e e)
    with tp1 c1.prems show ?thesis by simp
  qed
next
  case tp1: (STMap _ t)
  then show ?thesis
  proof (cases "expr i ep env cd st")
    case s1: (n a s)
    then show ?thesis
    proof (cases a)
      case f1: (Pair a b)
      then show ?thesis
      proof (cases a)
        case k1: (KValue v)
        from c1.IH[OF c1.prems] have
          "ssel tp (hash loc v) ix ep env cd st = ssel tp (hash loc v) (map f ix) ep env cd st" by
```

```
simp
          with k1 tp1 s1 c1 f1 show ?thesis by simp
        next
          case (KCDptr x2)
          with tp1 s1 c1.prems f1 show ?thesis by simp
        next
          case (KMemptr x2)
          with tp1 s1 c1.prems f1 show ?thesis by simp
        next
          case (KStoptr x3)
          with tp1 s1 c1.prems f1 show ?thesis by simp
        qed
      qed
    next
      case (e e)
      with tp1 c1.prems show ?thesis by simp
    qed
  next
    case (STValue x2)
    then show ?thesis by simp
  qed
qed

lemma msel_eq_msel:
"(⋀i st. i ∈ set ix ⟹ expr i ep env cd st = expr (f i) ep env cd st) ⟹
          msel c tp loc ix ep env cd st = msel c tp loc (map f ix) ep env cd st"
proof (induction ix arbitrary: c tp loc ep env cd st)
  case Nil
  then show ?case by simp
next
  case c1: (Cons i ix)
  then show ?case
  proof (cases tp)
    case tp1: (MTArray al tp)
    then show ?thesis
    proof (cases ix)
      case Nil
      thus ?thesis using tp1 c1.prems by auto
    next
      case c2: (Cons a list)
      then show ?thesis
      proof (cases "expr i ep env cd st")
        case s1: (n a st')
        then show ?thesis
        proof (cases a)
          case f1: (Pair a b)
          then show ?thesis
          proof (cases a)
            case k1: (KValue v)
            then show ?thesis
            proof (cases b)
              case v1: (Value t)
              then show ?thesis
              proof (cases "less t (TUInt 256) v (ShowL_int al)")
                case None
                with v1 k1 tp1 s1 c1.prems f1 show ?thesis using c2 by simp
              next
                case s2: (Some a)
                then show ?thesis
                proof (cases a)
                  case p1: (Pair a b)
                  then show ?thesis
                  proof (cases b)
                    case (TSInt x1)
```

```
    with s2 p1 v1 k1 tp1 s1 c1.prems f1 show ?thesis using c2 by simp
  next
    case (TUInt x2)
    with s2 p1 v1 k1 tp1 s1 c1.prems f1 show ?thesis using c2 by simp
  next
    case b1: TBool
    show ?thesis
    proof cases
      assume "a = ShowL_bool True"
      then show ?thesis
      proof (cases c)
        case True
        then show ?thesis
        proof (cases "accessStore (hash loc v) (memory st')")
          case None
          with s2 b1 p1 v1 k1 tp1 s1 c1.prems f1 True show ?thesis using c2 by simp
        next
          case s3: (Some a)
          then show ?thesis
          proof (cases a)
            case (MValue x1)
            with s2 s3 b1 p1 v1 k1 tp1 s1 c1.prems f1 True show ?thesis using c2 by
simp

          next
            case mp: (MPointer l)
            from c1.IH[OF c1.prems]
              have "msel c tp l ix ep env cd st' = msel c tp l (map f ix) ep env cd
st'" by simp

            with mp s2 s3 b1 p1 v1 k1 tp1 s1 c1.prems f1 True show ?thesis using c2
by simp

          qed
        qed
      next
        case False
        then show ?thesis
        proof (cases "accessStore (hash loc v) cd")
          case None
          with s2 b1 p1 v1 k1 tp1 s1 c1.prems f1 False show ?thesis using c2 by simp
        next
          case s3: (Some a)
          then show ?thesis
          proof (cases a)
            case (MValue x1)
            with s2 s3 b1 p1 v1 k1 tp1 s1 c1.prems f1 False show ?thesis using c2 by
simp

          next
            case mp: (MPointer l)
            from c1.IH[OF c1.prems]
              have "msel c tp l ix ep env cd st' = msel c tp l (map f ix) ep env cd
st'" by simp

            with mp s2 s3 b1 p1 v1 k1 tp1 s1 c1.prems f1 False show ?thesis using c2
by simp

          qed
        qed
      qed
    next
      assume "¬ a = ShowL_bool True"
      with s2 p1 v1 k1 tp1 s1 c1.prems f1 show ?thesis using c2 by simp
    qed
  next
    case TAddr
    with s2 p1 v1 k1 tp1 s1 c1.prems f1 show ?thesis using c2 by simp
  qed
qed
```

```
            qed
          next
            case (Calldata x2)
            with k1 tp1 s1 c1.prems f1 show ?thesis using c2 by simp
          next
            case (Memory x2)
            with k1 tp1 s1 c1.prems f1 show ?thesis using c2 by simp
          next
            case (Storage x3)
            with k1 tp1 s1 c1.prems f1 show ?thesis using c2 by simp
          qed
        next
          case (KCDptr x2)
          with tp1 s1 c1.prems f1 show ?thesis using c2 by simp
        next
          case (KMemptr x2)
          with tp1 s1 c1.prems f1 show ?thesis using c2 by simp
        next
          case (KStoptr x3)
          with tp1 s1 c1.prems f1 show ?thesis using c2 by simp
        qed
      qed
    next
      case (e e)
      with tp1 c1.prems show ?thesis using c2 by simp
    qed
  qed
next
  case (MTValue x2)
  then show ?thesis by simp
qed
qed

lemma ref_eq:
  assumes "⋀e st. e ∈ set ex ⟹ expr e ep env cd st = expr (f e) ep env cd st"
  shows "rexp (Ref i ex) ep env cd st=rexp (Ref i (map f ex)) ep env cd st"
proof (cases "fmlookup (denvalue env) i")
  case None
  then show ?thesis by simp
next
  case s1: (Some a)
  then show ?thesis
  proof (cases a)
    case p1: (Pair tp b)
    then show ?thesis
    proof (cases b)
      case k1: (Stackloc l)
      then show ?thesis
      proof (cases "accessStore l (stack st)")
        case None
        with s1 p1 k1 show ?thesis by simp
      next
        case s2: (Some a')
        then show ?thesis
        proof (cases a')
          case (KValue _)
          with s1 s2 p1 k1 show ?thesis by simp
        next
          case cp: (KCDptr cp)
          then show ?thesis
          proof (cases tp)
            case (Value x1)
            with mp s1 s2 p1 k1 show ?thesis by simp
          next
```

```
          case mt: (Calldata ct)
          from msel_eq_msel have
            "msel False ct cp ex ep env cd st=msel False ct cp (map f ex) ep env cd st" using assms
by blast
          thus ?thesis using s1 s2 p1 k1 mt cp by simp
        next
          case mt: (Memory mt)
          from msel_eq_msel have
            "msel True mt cp ex ep env cd st=msel True mt cp (map f ex) ep env cd st" using assms by
blast
          thus ?thesis using s1 s2 p1 k1 mt cp by simp
        next
          case (Storage x3)
          with cp s1 s2 p1 k1 show ?thesis by simp
        qed
      next
        case mp: (KMemptr mp)
        then show ?thesis
        proof (cases tp)
          case (Value x1)
          with mp s1 s2 p1 k1 show ?thesis by simp
        next
          case mt: (Calldata ct)
          from msel_eq_msel have
            "msel True ct mp ex ep env cd st=msel True ct mp (map f ex) ep env cd st" using assms by
blast
          thus ?thesis using s1 s2 p1 k1 mt mp by simp
        next
          case mt: (Memory mt)
          from msel_eq_msel have
            "msel True mt mp ex ep env cd st=msel True mt mp (map f ex) ep env cd st" using assms by
blast
          thus ?thesis using s1 s2 p1 k1 mt mp by simp
        next
          case (Storage x3)
          with mp s1 s2 p1 k1 show ?thesis by simp
        qed
      next
        case sp: (KStoptr sp)
        then show ?thesis
        proof (cases tp)
          case (Value x1)
          then show ?thesis using s1 s2 p1 k1 sp by simp
        next
          case (Calldata x2)
          then show ?thesis using s1 s2 p1 k1 sp by simp
        next
          case (Memory x2)
          then show ?thesis using s1 s2 p1 k1 sp by simp
        next
          case st: (Storage stp)
          from ssel_eq_ssel have
            "ssel stp sp ex ep env cd st=ssel stp sp (map f ex) ep env cd st" using assms by blast
          thus ?thesis using s1 s2 p1 k1 st sp by simp
        qed
      qed
    qed
  next
    case sl:(Storeloc sl)
    then show ?thesis
    proof (cases tp)
      case (Value x1)
      then show ?thesis using s1 p1 sl by simp
    next
```

```
        case (Calldata x2)
        then show ?thesis using s1 p1 sl by simp
      next
        case (Memory x2)
        then show ?thesis using s1 p1 sl by simp
      next
        case st: (Storage stp)
        from ssel_eq_ssel have
          "ssel stp sl ex ep env cd st=ssel stp sl (map f ex) ep env cd st" using assms by blast
        thus ?thesis using s1 sl p1 st by simp
      qed
    qed
  qed
qed
```

The following theorem proves that the update function preserves the semantics of expressions.

```
theorem update_correctness:
    "⋀st lb lv. expr ex ep env cd st = expr (eupdate ex) ep env cd st"
    "⋀st. rexp lv ep env cd st = rexp (lupdate lv) ep env cd st"
proof (induction ex and lv)
  case (Id x)
  then show ?case by simp
next
  case (Ref d ix)
  then show ?case using ref_eq[where f="eupdate"] by simp
next
  case (INT b v)
  then show ?case
  proof (cases "gas st > 0")
    case True
    then show ?thesis
    proof cases
      assume "b∈vbits"
      show ?thesis
      proof cases
        let ?m_def = "(-(2^(b-1)) + (v+2^(b-1)) mod (2^b))"
        assume "v ≥ 0"

        from ‘b∈vbits‘ True have
          "expr (E.INT b v) ep env cd st = Normal ((KValue (createSInt b v), Value (TSInt b)), st)" by
simp
        also have "createSInt b v = createSInt b ?m_def" using ‘b∈vbits‘ ‘v ≥ 0‘ by auto
        also from ‘v ≥ 0‘ ‘b∈vbits‘ True have
          "Normal ((KValue (createSInt b ?m_def), Value (TSInt b)),st) = expr (eupdate (E.INT b v)) ep
env cd st"
          by simp
        finally show "expr (E.INT b v) ep env cd st = expr (eupdate (E.INT b v)) ep env cd st" by simp
      next
        let ?m_def = "(2^(b-1) - (-v+2^(b-1)-1) mod (2^b) - 1)"
        assume "¬ v ≥ 0"

        from ‘b∈vbits‘ True have
          "expr (E.INT b v) ep env cd st = Normal ((KValue (createSInt b v), Value (TSInt b)), st)" by
simp
        also have "createSInt b v = createSInt b ?m_def" using ‘b∈vbits‘ ‘¬ v ≥ 0‘ by auto
        also from ‘¬ v ≥ 0‘ ‘b∈vbits‘ True have
          "Normal ((KValue (createSInt b ?m_def), Value (TSInt b)),st) =expr (eupdate (E.INT b v)) ep
env cd st"
          by simp
        finally show "expr (E.INT b v) ep env cd st = expr (eupdate (E.INT b v)) ep env cd st" by simp
      qed
    next
      assume "¬ b∈vbits"
      thus ?thesis by auto
```

```
    qed
  next
    case False
    then show ?thesis using no_gas by simp
  qed
next
  case (UINT x1 x2)
  then show ?case by simp
next
  case (ADDRESS x)
  then show ?case by simp
next
  case (BALANCE x)
  then show ?case by simp
next
  case THIS
  then show ?case by simp
next
  case SENDER
  then show ?case by simp
next
  case VALUE
  then show ?case by simp
next
  case TRUE
  then show ?case by simp
next
  case FALSE
  then show ?case by simp
next
  case (LVAL x)
  then show ?case by simp
next
  case p: (PLUS e1 e2)
  show ?case
  proof (cases "eupdate e1")
    case i: (INT b1 v1)
    with p.IH have expr1: "expr e1 ep env cd st = expr (E.INT b1 v1) ep env cd st" by simp
    then show ?thesis
    proof (cases "gas st > 0")
      case True
      then show ?thesis
      proof (cases)
        assume "b1 ∈ vbits"
        with expr1 True
          have "expr e1 ep env cd st=Normal ((KValue (createSInt b1 v1), Value (TSInt b1)),st)" by simp
        moreover from i ‘b1 ∈ vbits‘
          have "v1 < 2^(b1-1)" and "v1 ≥ -(2^(b1-1))" using update_bounds_int by auto
        moreover from ‘b1 ∈ vbits‘ have "0 < b1" by auto
        ultimately have r1: "expr e1 ep env cd st = Normal ((KValue (ShowL_int v1), Value (TSInt
b1)),st)"
          using createSInt_id[of v1 b1] by simp
        thus ?thesis
        proof (cases "eupdate e2")
          case i2: (INT b2 v2)
          with p.IH have expr2: "expr e2 ep env cd st = expr (E.INT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            let ?v="v1 + v2"
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createSInt b2 v2), Value (TSInt b2)),st)" by
simp
            moreover from i2 ‘b2 ∈ vbits‘
```

```
    have "v2 < 2^(b2-1)" and "v2 ≥ -(2^(b2-1))" using update_bounds_int by auto
  moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
  ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TSInt
b2)),st)"
    using createSInt_id[of v2 b2] by simp
  thus ?thesis
  proof (cases)
    let ?x="- (2 ^ (max b1 b2 - 1)) + (?v + 2 ^ (max b1 b2 - 1)) mod 2 ^ max b1 b2"
    assume "?v≥0"
    hence "createSInt (max b1 b2) ?v = (ShowL_int ?x)" by simp
    moreover have "add (TSInt b1) (TSInt b2) (ShowL_int v1) (ShowL_int v2)
      = Some (createSInt (max b1 b2) ?v, TSInt (max b1 b2))"
      using Read_ShowL_id add_def olift.simps(1)[of "(+)" b1 b2] by simp
    ultimately have "expr (PLUS e1 e2) ep env cd st
      = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)" using r1 r2 True by
simp
    moreover have "expr (eupdate (PLUS e1 e2)) ep env cd st
      = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)"
    proof -
      from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘?v≥0‘
        have "eupdate (PLUS e1 e2) = E.INT (max b1 b2) ?x" using i i2 by simp
      moreover have "expr (E.INT (max b1 b2) ?x) ep env cd st
        = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)"
      proof -
        from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have "max b1 b2 ∈ vbits" using vbits_max by simp
        with True have "expr (E.INT (max b1 b2) ?x) ep env cd st
          = Normal ((KValue (createSInt (max b1 b2) ?x), Value (TSInt (max b1 b2))),st)" by
simp
        moreover from ‘0 < b1‘
          have "?x < 2 ^ (max b1 b2 - 1)" using upper_bound3 by simp
        moreover from ‘0 < b1‘ have "0 < max b1 b2" using max_def by simp
        ultimately show ?thesis using createSInt_id[of ?x "max b1 b2"] by simp
      qed
      ultimately show ?thesis by simp
    qed
    ultimately show ?thesis by simp
  next
    let ?x="2^(max b1 b2 -1) - (-?v+2^(max b1 b2-1)-1) mod (2^max b1 b2) - 1"
    assume "¬ ?v≥0"
    hence "createSInt (max b1 b2) ?v = (ShowL_int ?x)" by simp
    moreover have "add (TSInt b1) (TSInt b2) (ShowL_int v1) (ShowL_int v2)
      = Some (createSInt (max b1 b2) ?v, TSInt (max b1 b2))"
      using Read_ShowL_id add_def olift.simps(1)[of "(+)" b1 b2] by simp
    ultimately have "expr (PLUS e1 e2) ep env cd st
      = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)" using True r1 r2 by
simp
    moreover have "expr (eupdate (PLUS e1 e2)) ep env cd st
      = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)"
    proof -
      from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘¬?v≥0‘
        have "eupdate (PLUS e1 e2) = E.INT (max b1 b2) ?x" using i i2 by simp
      moreover have "expr (E.INT (max b1 b2) ?x) ep env cd st
        = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)"
      proof -
        from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have "max b1 b2 ∈ vbits" using vbits_max by simp
        with True have "expr (E.INT (max b1 b2) ?x) ep env cd st
          = Normal ((KValue (createSInt (max b1 b2) ?x), Value (TSInt (max b1 b2))),st)" by
simp
        moreover from ‘0 < b1‘
          have "?x ≥ - (2 ^ (max b1 b2 - 1))" using lower_bound2[of "max b1 b2" ?v] by simp
        moreover from ‘b1 > 0‘ have "2^(max b1 b2 -1) > (0::nat)" by simp
          hence "2^(max b1 b2 -1) - (-?v+2^(max b1 b2-1)-1) mod (2^max b1 b2) - 1 < 2 ^ (max
b1 b2 - 1)"
          by (smt (verit, best) Euclidean_Division.pos_mod_sign not_exp_less_eq_0_int)
```

```
            moreover from ‘0 < b1‘ have "0 < max b1 b2" using max_def by simp
            ultimately show ?thesis using createSInt_id[of ?x "max b1 b2"] by simp
          qed
          ultimately show ?thesis by simp
        qed
        ultimately show ?thesis by simp
      qed
    next
      assume "¬ b2 ∈ vbits"
      with p i i2 show ?thesis by simp
    qed
  next
    case u2: (UINT b2 v2)
    with p.IH have expr2: "expr e2 ep env cd st = expr (UINT b2 v2) ep env cd st" by simp
    then show ?thesis
    proof (cases)
      let ?v="v1 + v2"
      assume "b2 ∈ vbits"
      with expr2 True
        have "expr e2 ep env cd st=Normal ((KValue (createUInt b2 v2), Value (TUInt b2)),st)" by
simp
      moreover from u2 ‘b2 ∈ vbits‘
        have "v2 < 2^b2" and "v2 ≥ 0" using update_bounds_uint by auto
      moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
      ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TUInt
b2)),st)"
        using createUInt_id[of v2 b2] by simp
      thus ?thesis
      proof (cases)
        assume "b2<b1"
        thus ?thesis
        proof (cases)
          let ?x="- (2 ^ (b1 - 1)) + (?v + 2 ^ (b1 - 1)) mod 2 ^ b1"
          assume "?v≥0"
          hence "createSInt b1 ?v = (ShowL_int ?x)" using ‘b2<b1‘ by auto
          moreover have "add (TSInt b1) (TUInt b2) (ShowL_int v1) (ShowL_int v2)
            = Some (createSInt b1 ?v, TSInt b1)"
            using Read_ShowL_id add_def olift.simps(3)[of "(+)" b1 b2] ‘b2<b1‘ by simp
          ultimately have "expr (PLUS e1 e2) ep env cd st
            = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)" using r1 r2 True by simp
          moreover have "expr (eupdate (PLUS e1 e2)) ep env cd st
            = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)"
          proof -
            from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘?v≥0‘ ‘b2<b1‘
              have "eupdate (PLUS e1 e2) = E.INT b1 ?x" using i u2 by simp
            moreover have "expr (E.INT b1 ?x) ep env cd st
              = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)"
            proof -
              from ‘b1 ∈ vbits‘ True have "expr (E.INT b1 ?x) ep env cd st
                = Normal ((KValue (createSInt b1 ?x), Value (TSInt b1)),st)" by simp
              moreover from ‘0 < b1‘ have "?x < 2 ^ (b1 - 1)" using upper_bound2 by simp
              ultimately show ?thesis using createSInt_id[of ?x "b1"] ‘0 < b1‘ by simp
            qed
            ultimately show ?thesis by simp
          qed
          ultimately show ?thesis by simp
        next
          let ?x="2^(b1 -1) - (-?v+2^(b1-1)-1) mod (2^b1) - 1"
          assume "¬ ?v≥0"
          hence "createSInt b1 ?v = (ShowL_int ?x)" by simp
          moreover have "add (TSInt b1) (TUInt b2) (ShowL_int v1) (ShowL_int v2)
            = Some (createSInt b1 ?v, TSInt b1)"
            using Read_ShowL_id add_def olift.simps(3)[of "(+)" b1 b2] ‘b2<b1‘ by simp
          ultimately have "expr (PLUS e1 e2) ep env cd st
```

```
          = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)" using r1 r2 True by simp
        moreover have "expr (eupdate (PLUS e1 e2)) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)"
        proof -
          from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘¬?v≥0‘ ‘b2<b1‘
            have "eupdate (PLUS e1 e2) = E.INT b1 ?x" using i u2 by simp
          moreover have "expr (E.INT b1 ?x) ep env cd st
            = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)"
          proof -
            from ‘b1 ∈ vbits‘ True have "expr (E.INT b1 ?x) ep env cd st
              = Normal ((KValue (createSInt b1 ?x), Value (TSInt b1)),st)" by simp
            moreover from ‘0 < b1‘ have "?x ≥ - (2 ^ (b1 - 1))" using upper_bound2 by simp
            moreover have "2^(b1-1) - (-?v+2^(b1-1)-1) mod (2^b1) - 1 < 2 ^ (b1 - 1)"
              by (smt (verit, best) Euclidean_Division.pos_mod_sign zero_less_power)
            ultimately show ?thesis using createSInt_id[of ?x b1] ‘0 < b1‘ by simp
          qed
          ultimately show ?thesis by simp
        qed
        ultimately show ?thesis by simp
      qed
    next
      assume "¬ b2 < b1"
      with p i u2 show ?thesis by simp
    qed
  next
    assume "¬ b2 ∈ vbits"
    with p i u2 show ?thesis by simp
  qed
next
  case (ADDRESS _)
  with p i show ?thesis by simp
next
  case (BALANCE _)
  with p i show ?thesis by simp
next
  case THIS
  with p i show ?thesis by simp
next
  case SENDER
  with p i show ?thesis by simp
next
  case VALUE
  with p i show ?thesis by simp
next
  case TRUE
  with p i show ?thesis by simp
next
  case FALSE
  with p i show ?thesis by simp
next
  case (LVAL _)
  with p i show ?thesis by simp
next
  case (PLUS _ _)
  with p i show ?thesis by simp
next
  case (MINUS _ _)
  with p i show ?thesis by simp
next
  case (EQUAL _ _)
  with p i show ?thesis by simp
next
  case (LESS _ _)
  with p i show ?thesis by simp
```

```
        next
          case (AND _ _)
          with p i show ?thesis by simp
        next
          case (OR _ _)
          with p i show ?thesis by simp
        next
          case (NOT _)
          with p i show ?thesis by simp
        next
          case (CALL x181 x182)
          with p i show ?thesis by simp
        next
          case (ECALL x191 x192 x193 x194)
          with p i show ?thesis by simp
        qed
      next
        assume "¬ b1 ∈ vbits"
        with p i show ?thesis by simp
      qed
    next
      case False
      then show ?thesis using no_gas by simp
    qed
  next
    case u: (UINT b1 v1)
    with p.IH have expr1: "expr e1 ep env cd st = expr (UINT b1 v1) ep env cd st" by simp
    then show ?thesis
    proof (cases "gas st > 0")
      case True
      then show ?thesis
      proof (cases)
        assume "b1 ∈ vbits"
        with expr1 True
          have "expr e1 ep env cd st=Normal ((KValue (createUInt b1 v1), Value (TUInt b1)),st)" by simp
        moreover from u ‘b1 ∈ vbits‘
          have "v1 < 2^b1" and "v1 ≥ 0" using update_bounds_uint by auto
        moreover from ‘b1 ∈ vbits‘ have "0 < b1" by auto
        ultimately have r1: "expr e1 ep env cd st = Normal ((KValue (ShowL_int v1), Value (TUInt
b1)),st)"
          by simp
        thus ?thesis
        proof (cases "eupdate e2")
          case u2: (UINT b2 v2)
          with p.IH have expr2: "expr e2 ep env cd st = expr (UINT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            let ?v="v1 + v2"
            let ?x="?v mod 2 ^ max b1 b2"
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createUInt b2 v2), Value (TUInt b2)),st)" by
simp
            moreover from u2 ‘b2 ∈ vbits‘
              have "v2 < 2^b2" and "v2 ≥ 0" using update_bounds_uint by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TUInt
b2)),st)"
              by simp
            moreover have "add (TUInt b1) (TUInt b2) (ShowL_int v1) (ShowL_int v2)
              = Some (createUInt (max b1 b2) ?v, TUInt (max b1 b2))"
              using Read_ShowL_id add_def olift.simps(2)[of "(+)" b1 b2] by simp
            ultimately have "expr (PLUS e1 e2) ep env cd st
              = Normal ((KValue (ShowL_int ?x), Value (TUInt (max b1 b2))),st)" using r1 True by simp
```

```
        moreover have "expr (eupdate (PLUS e1 e2)) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TUInt (max b1 b2))),st)"
        proof -
          from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘
            have "eupdate (PLUS e1 e2) = UINT (max b1 b2) ?x" using u u2 by simp
          moreover have "expr (UINT (max b1 b2) ?x) ep env cd st
            = Normal ((KValue (ShowL_int ?x), Value (TUInt (max b1 b2))),st)"
          proof -
            from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have "max b1 b2 ∈ vbits" using vbits_max by simp
            with True have "expr (UINT (max b1 b2) ?x) ep env cd st
              = Normal ((KValue (createUInt (max b1 b2) ?x), Value (TUInt (max b1 b2))),st)" by
simp
            moreover from ‘0 < b1‘
              have "?x < 2 ^ (max b1 b2)" by simp
            moreover from ‘0 < b1‘ have "0 < max b1 b2" using max_def by simp
            ultimately show ?thesis by simp
          qed
          ultimately show ?thesis by simp
        qed
        ultimately show ?thesis by simp
      next
        assume "¬ b2 ∈ vbits"
        with p u u2 show ?thesis by simp
      qed
    next
      case i2: (INT b2 v2)
      with p.IH have expr2: "expr e2 ep env cd st = expr (E.INT b2 v2) ep env cd st" by simp
      then show ?thesis
      proof (cases)
        let ?v="v1 + v2"
        assume "b2 ∈ vbits"
        with expr2 True
          have "expr e2 ep env cd st=Normal ((KValue (createSInt b2 v2), Value (TSInt b2)),st)" by
simp
        moreover from i2 ‘b2 ∈ vbits‘
          have "v2 < 2^(b2-1)" and "v2 ≥ -(2^(b2-1))" using update_bounds_int by auto
        moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
        ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TSInt
b2)),st)"
          using createSInt_id[of v2 b2] by simp
        thus ?thesis
        proof (cases)
          assume "b1<b2"
          thus ?thesis
          proof (cases)
            let ?x="- (2 ^ (b2 - 1)) + (?v + 2 ^ (b2 - 1)) mod 2 ^ b2"
            assume "?v≥0"
            hence "createSInt b2 ?v = (ShowL_int ?x)" using ‘b1<b2‘ by auto
            moreover have "add (TUInt b1) (TSInt b2) (ShowL_int v1) (ShowL_int v2)
              = Some (createSInt b2 ?v, TSInt b2)"
              using Read_ShowL_id add_def olift.simps(4)[of "(+)" b1 b2] ‘b1<b2‘ by simp
            ultimately have "expr (PLUS e1 e2) ep env cd st
              = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)" using r1 r2 True by simp
            moreover have "expr (eupdate (PLUS e1 e2)) ep env cd st
              = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)"
            proof -
              from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘?v≥0‘ ‘b1<b2‘
                have "eupdate (PLUS e1 e2) = E.INT b2 ?x" using u i2 by simp
              moreover have "expr (E.INT b2 ?x) ep env cd st
                = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)"
              proof -
                from ‘b2 ∈ vbits‘ True have "expr (E.INT b2 ?x) ep env cd st
                  = Normal ((KValue (createSInt b2 ?x), Value (TSInt b2)),st)" by simp
                moreover from ‘0 < b2‘ have "?x < 2 ^ (b2 - 1)" using upper_bound2 by simp
```

```
          ultimately show ?thesis using createSInt_id[of ?x "b2"] ‘0 < b2‘ by simp
        qed
        ultimately show ?thesis by simp
      qed
      ultimately show ?thesis by simp
    next
      let ?x="2^(b2 -1) - (-?v+2^(b2-1)-1) mod (2^b2) - 1"
      assume "¬ ?v≥0"
      hence "createSInt b2 ?v = (ShowL_int ?x)" by simp
      moreover have "add (TUInt b1) (TSInt b2) (ShowL_int v1) (ShowL_int v2)
        = Some (createSInt b2 ?v, TSInt b2)"
        using Read_ShowL_id add_def olift.simps(4)[of "(+)" b1 b2] ‘b1<b2‘ by simp
      ultimately have "expr (PLUS e1 e2) ep env cd st
        = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)" using r1 r2 True by simp
      moreover have "expr (eupdate (PLUS e1 e2)) ep env cd st
        = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)"
      proof -
        from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘¬?v≥0‘ ‘b1<b2‘
          have "eupdate (PLUS e1 e2) = E.INT b2 ?x" using u i2 by simp
        moreover have "expr (E.INT b2 ?x) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)"
        proof -
          from ‘b2 ∈ vbits‘ True have "expr (E.INT b2 ?x) ep env cd st
            = Normal ((KValue (createSInt b2 ?x), Value (TSInt b2)),st)" by simp
          moreover from ‘0 < b2‘ have "?x ≥ - (2 ^ (b2 - 1))" using upper_bound2 by simp
          moreover have "2^(b2-1) - (-?v+2^(b2-1)-1) mod (2^b2) - 1 < 2 ^ (b2 - 1)"
            by (smt (verit, best) Euclidean_Division.pos_mod_sign zero_less_power)
          ultimately show ?thesis using createSInt_id[of ?x b2] ‘0 < b2‘ by simp
        qed
        ultimately show ?thesis by simp
      qed
      ultimately show ?thesis by simp
    qed
  next
    assume "¬ b1 < b2"
    with p u i2 show ?thesis by simp
  qed
next
  assume "¬ b2 ∈ vbits"
  with p u i2 show ?thesis by simp
qed
next
  case (ADDRESS _)
  with p u show ?thesis by simp
next
  case (BALANCE _)
  with p u show ?thesis by simp
next
  case THIS
  with p u show ?thesis by simp
next
  case SENDER
  with p u show ?thesis by simp
next
  case VALUE
  with p u show ?thesis by simp
next
  case TRUE
  with p u show ?thesis by simp
next
  case FALSE
  with p u show ?thesis by simp
next
  case (LVAL _)
```

```
        with p u show ?thesis by simp
      next
        case (PLUS _ _)
        with p u show ?thesis by simp
      next
        case (MINUS _ _)
        with p u show ?thesis by simp
      next
        case (EQUAL _ _)
        with p u show ?thesis by simp
      next
        case (LESS _ _)
        with p u show ?thesis by simp
      next
        case (AND _ _)
        with p u show ?thesis by simp
      next
        case (OR _ _)
        with p u show ?thesis by simp
      next
        case (NOT _)
        with p u show ?thesis by simp
      next
        case (CALL x181 x182)
        with p u show ?thesis by simp
      next
        case (ECALL x191 x192 x193 x194)
        with p u show ?thesis by simp
      qed
    next
      assume "¬ b1 ∈ vbits"
      with p u show ?thesis by simp
    qed
  next
    case False
    then show ?thesis using no_gas by simp
  qed
next
  case (ADDRESS x3)
  with p show ?thesis by simp
next
  case (BALANCE x4)
  with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case THIS
  with p show ?thesis by simp
next
  case SENDER
  with p show ?thesis by simp
next
  case VALUE
  with p show ?thesis by simp
next
  case TRUE
  with p show ?thesis by simp
next
  case FALSE
  with p show ?thesis by simp
next
  case (LVAL x7)
  with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case (PLUS x81 x82)
  with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
```

```
  next
    case (MINUS x91 x92)
    with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (EQUAL x101 x102)
    with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (LESS x111 x112)
    with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (AND x121 x122)
    with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (OR x131 x132)
    with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (NOT x131)
    with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (CALL x181 x182)
    with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (ECALL x191 x192 x193 x194)
    with p show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  qed
next
  case m: (MINUS e1 e2)
  show ?case
  proof (cases "eupdate e1")
    case i: (INT b1 v1)
    with m.IH have expr1: "expr e1 ep env cd st = expr (E.INT b1 v1) ep env cd st" by simp
    then show ?thesis
    proof (cases "gas st > 0")
      case True
      show ?thesis
      proof (cases)
        assume "b1 ∈ vbits"
        with expr1 True
          have "expr e1 ep env cd st=Normal ((KValue (createSInt b1 v1), Value (TSInt b1)),st)" by simp
        moreover from i ‘b1 ∈ vbits‘
          have "v1 < 2^(b1-1)" and "v1 ≥ -(2^(b1-1))" using update_bounds_int by auto
        moreover from ‘b1 ∈ vbits‘ have "0 < b1" by auto
        ultimately have r1: "expr e1 ep env cd st = Normal ((KValue (ShowL_int v1), Value (TSInt
b1)),st)"
          using createSInt_id[of v1 b1] by simp
        thus ?thesis
        proof (cases "eupdate e2")
          case i2: (INT b2 v2)
          with m.IH have expr2: "expr e2 ep env cd st = expr (E.INT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            let ?v="v1 - v2"
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createSInt b2 v2), Value (TSInt b2)),st)" by
simp
            moreover from i2 ‘b2 ∈ vbits‘
              have "v2 < 2^(b2-1)" and "v2 ≥ -(2^(b2-1))" using update_bounds_int by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TSInt
b2)),st)"
              using createSInt_id[of v2 b2] by simp

            from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have
```

```
    u_def: "eupdate (MINUS e1 e2) =
    (let v = v1 - v2
      in if 0 ≤ v
        then E.INT (max b1 b2)
          (- (2 ^ (max b1 b2 - 1)) + (v + 2 ^ (max b1 b2 - 1)) mod 2 ^ max b1 b2)
        else E.INT (max b1 b2)
          (2 ^ (max b1 b2 - 1) - (- v + 2 ^ (max b1 b2 - 1) - 1) mod 2 ^ max b1 b2 - 1))"
    using i i2 eupdate.simps(11)[of e1 e2] by simp

  show ?thesis
  proof (cases)
    let ?x="- (2 ^ (max b1 b2 - 1)) + (?v + 2 ^ (max b1 b2 - 1)) mod 2 ^ max b1 b2"
    assume "?v≥0"
    hence "createSInt (max b1 b2) ?v = (ShowL_int ?x)" by simp
    moreover have "sub (TSInt b1) (TSInt b2) (ShowL_int v1) (ShowL_int v2)
      = Some (createSInt (max b1 b2) ?v, TSInt (max b1 b2))"
      using Read_ShowL_id sub_def olift.simps(1)[of "(-)" b1 b2] by simp
    ultimately have "expr (MINUS e1 e2) ep env cd st
      = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)" using r1 r2 True by
simp
    moreover have "expr (eupdate (MINUS e1 e2)) ep env cd st
      = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)"
    proof -
      from u_def have "eupdate (MINUS e1 e2) = E.INT (max b1 b2) ?x" using ‘?v≥0‘ by simp
      moreover have "expr (E.INT (max b1 b2) ?x) ep env cd st
        = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)"
      proof -
        from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have "max b1 b2 ∈ vbits" using vbits_max by simp
        with True have "expr (E.INT (max b1 b2) ?x) ep env cd st
          = Normal ((KValue (createSInt (max b1 b2) ?x), Value (TSInt (max b1 b2))),st)" by
simp
        moreover from ‘0 < b1‘
          have "?x < 2 ^ (max b1 b2 - 1)" using upper_bound2 by simp
        moreover from ‘0 < b1‘ have "0 < max b1 b2" using max_def by simp
        ultimately show ?thesis using createSInt_id[of ?x "max b1 b2"] by simp
      qed
      ultimately show ?thesis by simp
    qed
    ultimately show ?thesis by simp
  next
    let ?x="2^(max b1 b2 -1) - (-?v+2^(max b1 b2-1)-1) mod (2^max b1 b2) - 1"
    assume "¬ ?v≥0"
    hence "createSInt (max b1 b2) ?v = (ShowL_int ?x)" by simp
    moreover have "sub (TSInt b1) (TSInt b2) (ShowL_int v1) (ShowL_int v2)
      = Some (createSInt (max b1 b2) ?v, TSInt (max b1 b2))"
      using Read_ShowL_id sub_def olift.simps(1)[of "(-)" b1 b2] by simp
    ultimately have "expr (MINUS e1 e2) ep env cd st
      = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)" using r1 r2 True by
simp
    moreover have "expr (eupdate (MINUS e1 e2)) ep env cd st
      = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)"
    proof -
      from u_def have "eupdate (MINUS e1 e2) = E.INT (max b1 b2) ?x" using ‘¬ ?v≥0‘ by
simp
      moreover have "expr (E.INT (max b1 b2) ?x) ep env cd st
        = Normal ((KValue (ShowL_int ?x), Value (TSInt (max b1 b2))),st)"
      proof -
        from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have "max b1 b2 ∈ vbits" using vbits_max by simp
        with True have "expr (E.INT (max b1 b2) ?x) ep env cd st
          = Normal ((KValue (createSInt (max b1 b2) ?x), Value (TSInt (max b1 b2))),st)" by
simp
        moreover from ‘0 < b1‘
          have "?x ≥ - (2 ^ (max b1 b2 - 1))" using lower_bound2[of "max b1 b2" ?v] by simp
        moreover from ‘b1 > 0‘ have "2^(max b1 b2 -1) > (0::nat)" by simp
```

```
                hence "2^(max b1 b2 -1) - (-?v+2^(max b1 b2-1)-1) mod (2^max b1 b2) - 1 < 2 ^ (max
b1 b2 - 1)"
                by (smt (verit, best) Euclidean_Division.pos_mod_sign not_exp_less_eq_0_int)
              moreover from ‘0 < b1‘ have "0 < max b1 b2" using max_def by simp
              ultimately show ?thesis using createSInt_id[of ?x "max b1 b2"] by simp
            qed
            ultimately show ?thesis by simp
          qed
          ultimately show ?thesis by simp
        qed
      next
        assume "¬ b2 ∈ vbits"
        with m i i2 show ?thesis by simp
      qed
    next
      case u: (UINT b2 v2)
      with m.IH have expr2: "expr e2 ep env cd st = expr (UINT b2 v2) ep env cd st" by simp
      then show ?thesis
      proof (cases)
        let ?v="v1 - v2"
        assume "b2 ∈ vbits"
        with expr2 True
          have "expr e2 ep env cd st=Normal ((KValue (createUInt b2 v2), Value (TUInt b2)),st)" by
simp
        moreover from u ‘b2 ∈ vbits‘
          have "v2 < 2^b2" and "v2 ≥ 0" using update_bounds_uint by auto
        moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
        ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TUInt
b2)),st)"
          using createUInt_id[of v2 b2] by simp
        thus ?thesis
        proof (cases)
          assume "b2<b1"
          with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have
            u_def: "eupdate (MINUS e1 e2) =
            (let v = v1 - v2
              in if 0 ≤ v
                then E.INT b1 (- (2 ^ (b1 - 1)) + (v + 2 ^ (b1 - 1)) mod 2 ^ b1)
                else E.INT b1 (2 ^ (b1 - 1) - (- v + 2 ^ (b1 - 1) - 1) mod 2 ^ b1 - 1))"
            using i u eupdate.simps(11)[of e1 e2] by simp
          show ?thesis
          proof (cases)
            let ?x="- (2 ^ (b1 - 1)) + (?v + 2 ^ (b1 - 1)) mod 2 ^ b1"
            assume "?v≥0"
            hence "createSInt b1 ?v = (ShowL_int ?x)" using ‘b2<b1‘ by auto
            moreover have "sub (TSInt b1) (TUInt b2) (ShowL_int v1) (ShowL_int v2)
              = Some (createSInt b1 ?v, TSInt b1)"
              using Read_ShowL_id sub_def olift.simps(3)[of "(-)" b1 b2] ‘b2<b1‘ by simp
            ultimately have "expr (MINUS e1 e2) ep env cd st
              = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)" using r1 r2 True by simp
            moreover have "expr (eupdate (MINUS e1 e2)) ep env cd st
              = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)"
            proof -
              from u_def have "eupdate (MINUS e1 e2) = E.INT b1 ?x" using ‘?v≥0‘ by simp
              moreover have "expr (E.INT b1 ?x) ep env cd st
                = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)"
              proof -
                from ‘b1 ∈ vbits‘ True have "expr (E.INT b1 ?x) ep env cd st
                  = Normal ((KValue (createSInt b1 ?x), Value (TSInt b1)),st)" by simp
                moreover from ‘0 < b1‘ have "?x < 2 ^ (b1 - 1)" using upper_bound2 by simp
                ultimately show ?thesis using createSInt_id[of ?x "b1"] ‘0 < b1‘ by simp
              qed
              ultimately show ?thesis by simp
            qed
```

```
        ultimately show ?thesis by simp
      next
        let ?x="2^(b1 -1) - (-?v+2^(b1-1)-1) mod (2^b1) - 1"
        assume "¬ ?v≥0"
        hence "createSInt b1 ?v = (ShowL_int ?x)" by simp
        moreover have "sub (TSInt b1) (TUInt b2) (ShowL_int v1) (ShowL_int v2)
          = Some (createSInt b1 ?v, TSInt b1)"
          using Read_ShowL_id sub_def olift.simps(3)[of "(-)" b1 b2] ‘b2<b1‘ by simp
        ultimately have "expr (MINUS e1 e2) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)" using r1 r2 True by simp
        moreover have "expr (eupdate (MINUS e1 e2)) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)"
        proof -
          from u_def have "eupdate (MINUS e1 e2) = E.INT b1 ?x" using ‘¬ ?v≥0‘ by simp
          moreover have "expr (E.INT b1 ?x) ep env cd st
            = Normal ((KValue (ShowL_int ?x), Value (TSInt b1)),st)"
          proof -
            from ‘b1 ∈ vbits‘ True have "expr (E.INT b1 ?x) ep env cd st
              = Normal ((KValue (createSInt b1 ?x), Value (TSInt b1)),st)" by simp
            moreover from ‘0 < b1‘ have "?x ≥ - (2 ^ (b1 - 1))" using upper_bound2 by simp
            moreover have "2^(b1-1) - (-?v+2^(b1-1)-1) mod (2^b1) - 1 < 2 ^ (b1 - 1)"
              by (smt (verit, best) Euclidean_Division.pos_mod_sign zero_less_power)
            ultimately show ?thesis using createSInt_id[of ?x b1] ‘0 < b1‘ by simp
          qed
          ultimately show ?thesis by simp
        qed
        ultimately show ?thesis by simp
      qed
    next
      assume "¬ b2 < b1"
      with m i u show ?thesis by simp
    qed
  next
    assume "¬ b2 ∈ vbits"
    with m i u show ?thesis by simp
  qed
next
  case (ADDRESS _)
  with m i show ?thesis by simp
next
  case (BALANCE _)
  with m i show ?thesis by simp
next
  case THIS
  with m i show ?thesis by simp
next
  case SENDER
  with m i show ?thesis by simp
next
  case VALUE
  with m i show ?thesis by simp
next
  case TRUE
  with m i show ?thesis by simp
next
  case FALSE
  with m i show ?thesis by simp
next
  case (LVAL _)
  with m i show ?thesis by simp
next
  case (PLUS _ _)
  with m i show ?thesis by simp
next
```

```
        case (MINUS _ _)
        with m i show ?thesis by simp
      next
        case (EQUAL _ _)
        with m i show ?thesis by simp
      next
        case (LESS _ _)
        with m i show ?thesis by simp
      next
        case (AND _ _)
        with m i show ?thesis by simp
      next
        case (OR _ _)
        with m i show ?thesis by simp
      next
        case (NOT _)
        with m i show ?thesis by simp
      next
        case (CALL x181 x182)
        with m i show ?thesis by simp
      next
        case (ECALL x191 x192 x193 x194)
        with m i show ?thesis by simp
      qed
    next
      assume "¬ b1 ∈ vbits"
      with m i show ?thesis by simp
    qed
  next
    case False
    then show ?thesis using no_gas by simp
  qed
next
  case u: (UINT b1 v1)
  with m.IH have expr1: "expr e1 ep env cd st = expr (UINT b1 v1) ep env cd st" by simp
  then show ?thesis
  proof (cases "gas st > 0")
    case True
    show ?thesis
    proof (cases)
      assume "b1 ∈ vbits"
      with expr1 True
        have "expr e1 ep env cd st=Normal ((KValue (createUInt b1 v1), Value (TUInt b1)),st)" by simp
      moreover from u ‘b1 ∈ vbits‘
        have "v1 < 2^b1" and "v1 ≥ 0" using update_bounds_uint by auto
      moreover from ‘b1 ∈ vbits‘ have "0 < b1" by auto
      ultimately have r1: "expr e1 ep env cd st = Normal ((KValue (ShowL_int v1), Value (TUInt
b1)),st)"
        by simp
      thus ?thesis
      proof (cases "eupdate e2")
        case u2: (UINT b2 v2)
        with m.IH have expr2: "expr e2 ep env cd st = expr (UINT b2 v2) ep env cd st" by simp
        then show ?thesis
        proof (cases)
          let ?v="v1 - v2"
          let ?x="?v mod 2 ^ max b1 b2"
          assume "b2 ∈ vbits"
          with expr2 True
            have "expr e2 ep env cd st=Normal ((KValue (createUInt b2 v2), Value (TUInt b2)),st)" by
simp
          moreover from u2 ‘b2 ∈ vbits‘
            have "v2 < 2^b2" and "v2 ≥ 0" using update_bounds_uint by auto
          moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
```

```
        ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TUInt
b2)),st)"
          by simp
        moreover have "sub (TUInt b1) (TUInt b2) (ShowL_int v1) (ShowL_int v2)
          = Some (createUInt (max b1 b2) ?v, TUInt (max b1 b2))"
          using Read_ShowL_id sub_def olift.simps(2)[of "(-)" b1 b2] by simp
        ultimately have "expr (MINUS e1 e2) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TUInt (max b1 b2))),st)" using r1 True by simp
        moreover have "expr (eupdate (MINUS e1 e2)) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TUInt (max b1 b2))),st)"
        proof -
          from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘
            have "eupdate (MINUS e1 e2) = UINT (max b1 b2) ?x" using u u2 by simp
          moreover have "expr (UINT (max b1 b2) ?x) ep env cd st
            = Normal ((KValue (ShowL_int ?x), Value (TUInt (max b1 b2))),st)"
          proof -
            from ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have "max b1 b2 ∈ vbits" using vbits_max by simp
            with True have "expr (UINT (max b1 b2) ?x) ep env cd st
              = Normal ((KValue (createUInt (max b1 b2) ?x), Value (TUInt (max b1 b2))),st)" by
simp
            moreover from ‘0 < b1‘
              have "?x < 2 ^ (max b1 b2)" by simp
            moreover from ‘0 < b1‘ have "0 < max b1 b2" using max_def by simp
            ultimately show ?thesis by simp
          qed
          ultimately show ?thesis by simp
        qed
        ultimately show ?thesis by simp
      next
        assume "¬ b2 ∈ vbits"
        with m u u2 show ?thesis by simp
      qed
    next
      case i: (INT b2 v2)
      with m.IH have expr2: "expr e2 ep env cd st = expr (E.INT b2 v2) ep env cd st" by simp
      then show ?thesis
      proof (cases)
        let ?v="v1 - v2"
        assume "b2 ∈ vbits"
        with expr2 True
          have "expr e2 ep env cd st=Normal ((KValue (createSInt b2 v2), Value (TSInt b2)),st)" by
simp
        moreover from i ‘b2 ∈ vbits‘
          have "v2 < 2^(b2-1)" and "v2 ≥ -(2^(b2-1))" using update_bounds_int by auto
        moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
        ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TSInt
b2)),st)"
          using createSInt_id[of v2 b2] by simp
        thus ?thesis
        proof (cases)
          assume "b1<b2"
          with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ have
            u_def: "eupdate (MINUS e1 e2) =
            (let v = v1 - v2
              in if 0 ≤ v
                then E.INT b2 (- (2 ^ (b2 - 1)) + (v + 2 ^ (b2 - 1)) mod 2 ^ b2)
                else E.INT b2 (2 ^ (b2 - 1) - (- v + 2 ^ (b2 - 1) - 1) mod 2 ^ b2 - 1))"
            using u i by simp
          show ?thesis
          proof (cases)
            let ?x="- (2 ^ (b2 - 1)) + (?v + 2 ^ (b2 - 1)) mod 2 ^ b2"
            assume "?v≥0"
            hence "createSInt b2 ?v = (ShowL_int ?x)" using ‘b1<b2‘ by auto
            moreover have "sub (TUInt b1) (TSInt b2) (ShowL_int v1) (ShowL_int v2)
```

```
          = Some (createSInt b2 ?v, TSInt b2)"
          using Read_ShowL_id sub_def olift.simps(4)[of "(-)" b1 b2] ‘b1<b2‘ by simp
        ultimately have "expr (MINUS e1 e2) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)" using r1 r2 True by simp
        moreover have "expr (eupdate (MINUS e1 e2)) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)"
        proof -
          from u_def have "eupdate (MINUS e1 e2) = E.INT b2 ?x" using ‘?v≥0‘ by simp
          moreover have "expr (E.INT b2 ?x) ep env cd st
            = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)"
          proof -
            from ‘b2 ∈ vbits‘ True have "expr (E.INT b2 ?x) ep env cd st
              = Normal ((KValue (createSInt b2 ?x), Value (TSInt b2)),st)" by simp
            moreover from ‘0 < b2‘ have "?x < 2 ^ (b2 - 1)" using upper_bound2 by simp
            ultimately show ?thesis using createSInt_id[of ?x "b2"] ‘0 < b2‘ by simp
          qed
          ultimately show ?thesis by simp
        qed
        ultimately show ?thesis by simp
      next
        let ?x="2^(b2 -1) - (-?v+2^(b2-1)-1) mod (2^b2) - 1"
        assume "¬ ?v≥0"
        hence "createSInt b2 ?v = (ShowL_int ?x)" by simp
        moreover have "sub (TUInt b1) (TSInt b2) (ShowL_int v1) (ShowL_int v2)
          = Some (createSInt b2 ?v, TSInt b2)"
          using Read_ShowL_id sub_def olift.simps(4)[of "(-)" b1 b2] ‘b1<b2‘ by simp
        ultimately have "expr (MINUS e1 e2) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)" using r1 r2 True by simp
        moreover have "expr (eupdate (MINUS e1 e2)) ep env cd st
          = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)"
        proof -
          from u_def have "eupdate (MINUS e1 e2) = E.INT b2 ?x" using ‘¬ ?v≥0‘ by simp
          moreover have "expr (E.INT b2 ?x) ep env cd st
            = Normal ((KValue (ShowL_int ?x), Value (TSInt b2)),st)"
          proof -
            from ‘b2 ∈ vbits‘ True have "expr (E.INT b2 ?x) ep env cd st
              = Normal ((KValue (createSInt b2 ?x), Value (TSInt b2)),st)" by simp
            moreover from ‘0 < b2‘ have "?x ≥ - (2 ^ (b2 - 1))" using upper_bound2 by simp
            moreover have "2^(b2-1) - (-?v+2^(b2-1)-1) mod (2^b2) - 1 < 2 ^ (b2 - 1)"
              by (smt (verit, best) Euclidean_Division.pos_mod_sign zero_less_power)
            ultimately show ?thesis using createSInt_id[of ?x b2] ‘0 < b2‘ by simp
          qed
          ultimately show ?thesis by simp
        qed
        ultimately show ?thesis by simp
      qed
    next
      assume "¬ b1 < b2"
      with m u i show ?thesis by simp
    qed
  next
    assume "¬ b2 ∈ vbits"
    with m u i show ?thesis by simp
  qed
next
  case (ADDRESS _)
  with m u show ?thesis by simp
next
  case (BALANCE _)
  with m u show ?thesis by simp
next
  case THIS
  with m u show ?thesis by simp
next
```

```
          case SENDER
          with m u show ?thesis by simp
        next
          case VALUE
          with m u show ?thesis by simp
        next
          case TRUE
          with m u show ?thesis by simp
        next
          case FALSE
          with m u show ?thesis by simp
        next
          case (LVAL _)
          with m u show ?thesis by simp
        next
          case (PLUS _ _)
          with m u show ?thesis by simp
        next
          case (MINUS _ _)
          with m u show ?thesis by simp
        next
          case (EQUAL _ _)
          with m u show ?thesis by simp
        next
          case (LESS _ _)
          with m u show ?thesis by simp
        next
          case (AND _ _)
          with m u show ?thesis by simp
        next
          case (OR _ _)
          with m u show ?thesis by simp
        next
          case (NOT _)
          with m u show ?thesis by simp
        next
          case (CALL x181 x182)
          with m u show ?thesis by simp
        next
          case (ECALL x191 x192 x193 x194)
          with m u show ?thesis by simp
        qed
      next
        assume "¬ b1 ∈ vbits"
        with m u show ?thesis by simp
      qed
    next
      case False
      then show ?thesis using no_gas by simp
    qed
  next
    case (ADDRESS x3)
    with m show ?thesis by simp
  next
    case (BALANCE x4)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case THIS
    with m show ?thesis by simp
  next
    case SENDER
    with m show ?thesis by simp
  next
    case VALUE
```

```
    with m show ?thesis by simp
  next
    case TRUE
    with m show ?thesis by simp
  next
    case FALSE
    with m show ?thesis by simp
  next
    case (LVAL x7)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (PLUS x81 x82)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (MINUS x91 x92)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (EQUAL x101 x102)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (LESS x111 x112)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (AND x121 x122)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (OR x131 x132)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (NOT x131)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (CALL x181 x182)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
  next
    case (ECALL x191 x192 x193 x194)
    with m show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
  qed
next
  case e: (EQUAL e1 e2)
  show ?case
  proof (cases "eupdate e1")
    case i: (INT b1 v1)
    with e.IH have expr1: "expr e1 ep env cd st = expr (E.INT b1 v1) ep env cd st" by simp
    then show ?thesis
    proof (cases "gas st > 0")
      case True
      then show ?thesis
      proof (cases)
        assume "b1 ∈ vbits"
        with expr1 True
          have "expr e1 ep env cd st=Normal ((KValue (createSInt b1 v1), Value (TSInt b1)),st)" by simp
        moreover from i ‘b1 ∈ vbits‘
          have "v1 < 2^(b1-1)" and "v1 ≥ -(2^(b1-1))" using update_bounds_int by auto
        moreover from ‘b1 ∈ vbits‘ have "0 < b1" by auto
        ultimately have r1: "expr e1 ep env cd st = Normal ((KValue (ShowL_int v1), Value (TSInt
b1)),st)"
          using createSInt_id[of v1 b1] by simp
        thus ?thesis
        proof (cases "eupdate e2")
          case i2: (INT b2 v2)
          with e.IH have expr2: "expr e2 ep env cd st = expr (E.INT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
```

```
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createSInt b2 v2), Value (TSInt b2)),st)" by
simp
            moreover from i2 ‘b2 ∈ vbits‘
              have "v2 < 2^(b2-1)" and "v2 ≥ -(2^(b2-1))" using update_bounds_int by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TSInt
b2)),st)"
              using createSInt_id[of v2 b2] by simp
            with r1 True have "expr (EQUAL e1 e2) ep env cd st=
              Normal ((KValue (createBool ((ReadL_int (ShowL_int v1))=((ReadL_int (ShowL_int v2))))), Value
TBool),st)"
              using equal_def plift.simps(1)[of "(=)"] by simp
            hence "expr (EQUAL e1 e2) ep env cd st=Normal ((KValue (createBool (v1=v2)), Value
TBool),st)"
              using Read_ShowL_id by simp
            with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ True show ?thesis using i i2 by simp
          next
            assume "¬ b2 ∈ vbits"
            with e i i2 show ?thesis by simp
          qed
        next
          case u: (UINT b2 v2)
          with e.IH have expr2: "expr e2 ep env cd st = expr (UINT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createUInt b2 v2), Value (TUInt b2)),st)" by
simp
            moreover from u ‘b2 ∈ vbits‘
              have "v2 < 2^b2" and "v2 ≥ 0" using update_bounds_uint by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TUInt
b2)),st)"
              using createUInt_id[of v2 b2] by simp
            thus ?thesis
            proof (cases)
              assume "b2<b1"
              with r1 r2 True have "expr (EQUAL e1 e2) ep env cd st=
                Normal ((KValue (createBool ((ReadL_int (ShowL_int v1))=((ReadL_int (ShowL_int v2))))), Value
TBool),st)"
                using equal_def plift.simps(3)[of "(=)"] by simp
              hence "expr (EQUAL e1 e2) ep env cd st=Normal ((KValue (createBool (v1=v2)), Value
TBool),st)"
                using Read_ShowL_id by simp
              with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘b2<b1‘ True show ?thesis using i u by simp
            next
              assume "¬ b2 < b1"
              with e i u show ?thesis by simp
            qed
          next
            assume "¬ b2 ∈ vbits"
            with e i u show ?thesis by simp
          qed
        next
          case (ADDRESS _)
          with e i show ?thesis by simp
        next
          case (BALANCE _)
          with e i show ?thesis by simp
        next
          case THIS
```

```
        with e i show ?thesis by simp
      next
        case SENDER
        with e i show ?thesis by simp
      next
        case VALUE
        with e i show ?thesis by simp
      next
        case TRUE
        with e i show ?thesis by simp
      next
        case FALSE
        with e i show ?thesis by simp
      next
        case (LVAL _)
        with e i show ?thesis by simp
      next
        case (PLUS _ _)
        with e i show ?thesis by simp
      next
        case (MINUS _ _)
        with e i show ?thesis by simp
      next
        case (EQUAL _ _)
        with e i show ?thesis by simp
      next
        case (LESS _ _)
        with e i show ?thesis by simp
      next
        case (AND _ _)
        with e i show ?thesis by simp
      next
        case (OR _ _)
        with e i show ?thesis by simp
      next
        case (NOT _)
        with e i show ?thesis by simp
      next
        case (CALL x181 x182)
        with e i show ?thesis by simp
      next
        case (ECALL x191 x192 x193 x194)
        with e i show ?thesis by simp
      qed
    next
      assume "¬ b1 ∈ vbits"
      with e i show ?thesis by simp
    qed
  next
    case False
    then show ?thesis using no_gas by simp
  qed
next
  case u: (UINT b1 v1)
  with e.IH have expr1: "expr e1 ep env cd st = expr (UINT b1 v1) ep env cd st" by simp
  then show ?thesis
  proof (cases "gas st > 0")
    case True
    then show ?thesis
    proof (cases)
      assume "b1 ∈ vbits"
      with expr1 True
        have "expr e1 ep env cd st=Normal ((KValue (createUInt b1 v1), Value (TUInt b1)),st)" by simp
      moreover from u ‘b1 ∈ vbits‘
```

```
          have "v1 < 2^b1" and "v1 ≥ 0" using update_bounds_uint by auto
        moreover from ‘b1 ∈ vbits‘ have "0 < b1" by auto
        ultimately have r1: "expr e1 ep env cd st = Normal ((KValue (ShowL_int v1), Value (TUInt b1)),st)"
          by simp
        thus ?thesis
        proof (cases "eupdate e2")
          case u2: (UINT b2 v2)
          with e.IH have expr2: "expr e2 ep env cd st = expr (UINT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createUInt b2 v2), Value (TUInt b2)),st)" by simp
            moreover from u2 ‘b2 ∈ vbits‘
              have "v2 < 2^b2" and "v2 ≥ 0" using update_bounds_uint by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TUInt b2)),st)"
              by simp
            with r1 True have "expr (EQUAL e1 e2) ep env cd st=
              Normal ((KValue (createBool ((ReadL_int (ShowL_int v1))=((ReadL_int (ShowL_int v2))))), Value TBool),st)"
              using equal_def plift.simps(2)[of "(=)"] by simp
            hence "expr (EQUAL e1 e2) ep env cd st=Normal ((KValue (createBool (v1=v2)), Value TBool),st)"
              using Read_ShowL_id by simp
            with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ show ?thesis using u u2 True by simp
          next
            assume "¬ b2 ∈ vbits"
            with e u u2 show ?thesis by simp
          qed
        next
          case i: (INT b2 v2)
          with e.IH have expr2: "expr e2 ep env cd st = expr (E.INT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            let ?v="v1 + v2"
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createSInt b2 v2), Value (TSInt b2)),st)" by simp
            moreover from i ‘b2 ∈ vbits‘
              have "v2 < 2^(b2-1)" and "v2 ≥ -(2^(b2-1))" using update_bounds_int by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TSInt b2)),st)"
              using createSInt_id[of v2 b2] by simp
            thus ?thesis
            proof (cases)
              assume "b1<b2"
              with r1 r2 True have "expr (EQUAL e1 e2) ep env cd st=
                Normal ((KValue (createBool ((ReadL_int (ShowL_int v1))=((ReadL_int (ShowL_int v2))))), Value TBool),st)"
                using equal_def plift.simps(4)[of "(=)"] by simp
              hence "expr (EQUAL e1 e2) ep env cd st=Normal ((KValue (createBool (v1=v2)), Value TBool),st)"
                using Read_ShowL_id by simp
              with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘b1<b2‘ True show ?thesis using u i by simp
            next
              assume "¬ b1 < b2"
              with e u i show ?thesis by simp
            qed
```

```
        next
          assume "¬ b2 ∈ vbits"
          with e u i show ?thesis by simp
        qed
      next
        case (ADDRESS _)
        with e u show ?thesis by simp
      next
        case (BALANCE _)
        with e u show ?thesis by simp
      next
        case THIS
        with e u show ?thesis by simp
      next
        case SENDER
        with e u show ?thesis by simp
      next
        case VALUE
        with e u show ?thesis by simp
      next
        case TRUE
        with e u show ?thesis by simp
      next
        case FALSE
        with e u show ?thesis by simp
      next
        case (LVAL _)
        with e u show ?thesis by simp
      next
        case (PLUS _ _)
        with e u show ?thesis by simp
      next
        case (MINUS _ _)
        with e u show ?thesis by simp
      next
        case (EQUAL _ _)
        with e u show ?thesis by simp
      next
        case (LESS _ _)
        with e u show ?thesis by simp
      next
        case (AND _ _)
        with e u show ?thesis by simp
      next
        case (OR _ _)
        with e u show ?thesis by simp
      next
        case (NOT _)
        with e u show ?thesis by simp
      next
        case (CALL x181 x182)
        with e u show ?thesis by simp
      next
        case (ECALL x191 x192 x193 x194)
        with e u show ?thesis by simp
      qed
    next
      assume "¬ b1 ∈ vbits"
      with e u show ?thesis by simp
    qed
  next
    case False
    then show ?thesis using no_gas by simp
  qed
```

```
  next
    case (ADDRESS x3)
    with e show ?thesis by simp
  next
    case (BALANCE x4)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case THIS
    with e show ?thesis by simp
  next
    case SENDER
    with e show ?thesis by simp
  next
    case VALUE
    with e show ?thesis by simp
  next
    case TRUE
    with e show ?thesis by simp
  next
    case FALSE
    with e show ?thesis by simp
  next
    case (LVAL x7)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (PLUS x81 x82)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (MINUS x91 x92)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (EQUAL x101 x102)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (LESS x111 x112)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (AND x121 x122)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (OR x131 x132)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (NOT x131)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (CALL x181 x182)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
  next
    case (ECALL x191 x192 x193 x194)
    with e show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
  qed
next
  case l: (LESS e1 e2)
  show ?case
  proof (cases "eupdate e1")
    case i: (INT b1 v1)
    with l.IH have expr1: "expr e1 ep env cd st = expr (E.INT b1 v1) ep env cd st" by simp
    then show ?thesis
    proof (cases "gas st > 0")
      case True
      then show ?thesis
      proof (cases)
        assume "b1 ∈ vbits"
```

```
        with expr1 True
          have "expr e1 ep env cd st=Normal ((KValue (createSInt b1 v1), Value (TSInt b1)),st)" by simp
        moreover from i ‘b1 ∈ vbits‘
          have "v1 < 2^(b1-1)" and "v1 ≥ -(2^(b1-1))" using update_bounds_int by auto
        moreover from ‘b1 ∈ vbits‘ have "0 < b1" by auto
        ultimately have r1: "expr e1 ep env cd st = Normal ((KValue (ShowL_int v1), Value (TSInt
b1)),st)"
          using createSInt_id[of v1 b1] by simp
        thus ?thesis
        proof (cases "eupdate e2")
          case i2: (INT b2 v2)
          with l.IH have expr2: "expr e2 ep env cd st = expr (E.INT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createSInt b2 v2), Value (TSInt b2)),st)" by
simp
            moreover from i2 ‘b2 ∈ vbits‘
              have "v2 < 2^(b2-1)" and "v2 ≥ -(2^(b2-1))" using update_bounds_int by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TSInt
b2)),st)"
              using createSInt_id[of v2 b2] by simp
            with r1 True have "expr (LESS e1 e2) ep env cd st=
              Normal ((KValue (createBool ((ReadL_int (ShowL_int v1))<((ReadL_int (ShowL_int v2))))), Value
TBool),st)"
              using less_def plift.simps(1)[of "(<)"] by simp
            hence "expr (LESS e1 e2) ep env cd st=Normal ((KValue (createBool (v1<v2)), Value
TBool),st)"
              using Read_ShowL_id by simp
            with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ show ?thesis using i i2 True by simp
          next
            assume "¬ b2 ∈ vbits"
            with l i i2 show ?thesis by simp
          qed
        next
          case u: (UINT b2 v2)
          with l.IH have expr2: "expr e2 ep env cd st = expr (UINT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createUInt b2 v2), Value (TUInt b2)),st)" by
simp
            moreover from u ‘b2 ∈ vbits‘
              have "v2 < 2^b2" and "v2 ≥ 0" using update_bounds_uint by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TUInt
b2)),st)"
              using createUInt_id[of v2 b2] by simp
            thus ?thesis
            proof (cases)
              assume "b2<b1"
              with r1 r2 True have "expr (LESS e1 e2) ep env cd st=
                Normal ((KValue (createBool ((ReadL_int (ShowL_int v1))<((ReadL_int (ShowL_int v2))))), Value
TBool),st)"
                using less_def plift.simps(3)[of "(<)"] by simp
              hence "expr (LESS e1 e2) ep env cd st=Normal ((KValue (createBool (v1<v2)), Value
TBool),st)"
                using Read_ShowL_id by simp
              with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘b2<b1‘ show ?thesis using i u True by simp
            next
              assume "¬ b2 < b1"
```

```
          with l i u show ?thesis by simp
        qed
      next
        assume "¬ b2 ∈ vbits"
        with l i u show ?thesis by simp
      qed
    next
      case (ADDRESS _)
      with l i show ?thesis by simp
    next
      case (BALANCE _)
      with l i show ?thesis by simp
    next
      case THIS
      with l i show ?thesis by simp
    next
      case SENDER
      with l i show ?thesis by simp
    next
      case VALUE
      with l i show ?thesis by simp
    next
      case TRUE
      with l i show ?thesis by simp
    next
      case FALSE
      with l i show ?thesis by simp
    next
      case (LVAL _)
      with l i show ?thesis by simp
    next
      case (PLUS _ _)
      with l i show ?thesis by simp
    next
      case (MINUS _ _)
      with l i show ?thesis by simp
    next
      case (EQUAL _ _)
      with l i show ?thesis by simp
    next
      case (LESS _ _)
      with l i show ?thesis by simp
    next
      case (AND _ _)
      with l i show ?thesis by simp
    next
      case (OR _ _)
      with l i show ?thesis by simp
    next
      case (NOT _)
      with l i show ?thesis by simp
    next
      case (CALL x181 x182)
      with l i show ?thesis by simp
    next
      case (ECALL x191 x192 x193 x194)
      with l i show ?thesis by simp
    qed
  next
    assume "¬ b1 ∈ vbits"
    with l i show ?thesis by simp
  qed
next
  case False
```

```
      then show ?thesis using no_gas by simp
    qed
  next
    case u: (UINT b1 v1)
    with l.IH have expr1: "expr e1 ep env cd st = expr (UINT b1 v1) ep env cd st" by simp
    then show ?thesis
    proof (cases "gas st > 0")
      case True
      then show ?thesis
      proof (cases)
        assume "b1 ∈ vbits"
        with expr1 True
          have "expr e1 ep env cd st=Normal ((KValue (createUInt b1 v1), Value (TUInt b1)),st)" by simp
        moreover from u ‘b1 ∈ vbits‘
          have "v1 < 2^b1" and "v1 ≥ 0" using update_bounds_uint by auto
        moreover from ‘b1 ∈ vbits‘ have "0 < b1" by auto
        ultimately have r1: "expr e1 ep env cd st = Normal ((KValue (ShowL_int v1), Value (TUInt
b1)),st)"
          by simp
        thus ?thesis
        proof (cases "eupdate e2")
          case u2: (UINT b2 v2)
          with l.IH have expr2: "expr e2 ep env cd st = expr (UINT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createUInt b2 v2), Value (TUInt b2)),st)" by
simp
            moreover from u2 ‘b2 ∈ vbits‘
              have "v2 < 2^b2" and "v2 ≥ 0" using update_bounds_uint by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TUInt
b2)),st)"
              by simp
            with r1 True have "expr (LESS e1 e2) ep env cd st=Normal ((KValue (createBool ((ReadL_int
(ShowL_int v1))<((ReadL_int (ShowL_int v2))))), Value TBool),st)" using less_def plift.simps(2)[of "(<)"]
by simp
            hence "expr (LESS e1 e2) ep env cd st=Normal ((KValue (createBool (v1<v2)), Value
TBool),st)" using Read_ShowL_id by simp
            with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ show ?thesis using u u2 True by simp
          next
            assume "¬ b2 ∈ vbits"
            with l u u2 show ?thesis by simp
          qed
        next
          case i: (INT b2 v2)
          with l.IH have expr2: "expr e2 ep env cd st = expr (E.INT b2 v2) ep env cd st" by simp
          then show ?thesis
          proof (cases)
            let ?v="v1 + v2"
            assume "b2 ∈ vbits"
            with expr2 True
              have "expr e2 ep env cd st=Normal ((KValue (createSInt b2 v2), Value (TSInt b2)),st)" by
simp
            moreover from i ‘b2 ∈ vbits‘
              have "v2 < 2^(b2-1)" and "v2 ≥ -(2^(b2-1))" using update_bounds_int by auto
            moreover from ‘b2 ∈ vbits‘ have "0 < b2" by auto
            ultimately have r2: "expr e2 ep env cd st = Normal ((KValue (ShowL_int v2), Value (TSInt
b2)),st)"
              using createSInt_id[of v2 b2] by simp
            thus ?thesis
            proof (cases)
              assume "b1<b2"
```

```
        with r1 r2 True have "expr (LESS e1 e2) ep env cd st=
          Normal ((KValue (createBool ((ReadL_int (ShowL_int v1))<((ReadL_int (ShowL_int v2))))), Value
TBool),st)"
          using less_def plift.simps(4)[of "(<)"] by simp
        hence "expr (LESS e1 e2) ep env cd st=Normal ((KValue (createBool (v1<v2)), Value
TBool),st)"
          using Read_ShowL_id by simp
        with ‘b1 ∈ vbits‘ ‘b2 ∈ vbits‘ ‘b1<b2‘ show ?thesis using u i True by simp
      next
        assume "¬ b1 < b2"
        with l u i show ?thesis by simp
      qed
    next
      assume "¬ b2 ∈ vbits"
      with l u i show ?thesis by simp
    qed
  next
    case (ADDRESS _)
    with l u show ?thesis by simp
  next
    case (BALANCE _)
    with l u show ?thesis by simp
  next
    case THIS
    with l u show ?thesis by simp
  next
    case SENDER
    with l u show ?thesis by simp
  next
    case VALUE
    with l u show ?thesis by simp
  next
    case TRUE
    with l u show ?thesis by simp
  next
    case FALSE
    with l u show ?thesis by simp
  next
    case (LVAL _)
    with l u show ?thesis by simp
  next
    case (PLUS _ _)
    with l u show ?thesis by simp
  next
    case (MINUS _ _)
    with l u show ?thesis by simp
  next
    case (EQUAL _ _)
    with l u show ?thesis by simp
  next
    case (LESS _ _)
    with l u show ?thesis by simp
  next
    case (AND _ _)
    with l u show ?thesis by simp
  next
    case (OR _ _)
    with l u show ?thesis by simp
  next
    case (NOT _)
    with l u show ?thesis by simp
  next
    case (CALL x181 x182)
    with l u show ?thesis by simp
```

```
      next
        case (ECALL x191 x192 x193 x194)
        with l u show ?thesis by simp
      qed
    next
      assume "¬ b1 ∈ vbits"
      with l u show ?thesis by simp
    qed
  next
    case False
    then show ?thesis using no_gas by simp
  qed
next
  case (ADDRESS x3)
  with l show ?thesis by simp
next
  case (BALANCE x4)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case THIS
  with l show ?thesis by simp
next
  case SENDER
  with l show ?thesis by simp
next
  case VALUE
  with l show ?thesis by simp
next
  case TRUE
  with l show ?thesis by simp
next
  case FALSE
  with l show ?thesis by simp
next
  case (LVAL x7)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case (PLUS x81 x82)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case (MINUS x91 x92)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case (EQUAL x101 x102)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case (LESS x111 x112)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case (AND x121 x122)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case (OR x131 x132)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case (NOT x131)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
next
  case (CALL x181 x182)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
next
  case (ECALL x191 x192 x193 x194)
  with l show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
qed
```

```
next
  case a: (AND e1 e2)
  show ?case
  proof (cases "eupdate e1")
    case (INT x11 x12)
    with a show ?thesis by simp
  next
    case (UINT x21 x22)
    with a show ?thesis by simp
  next
    case (ADDRESS x3)
    with a show ?thesis by simp
  next
    case (BALANCE x4)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case THIS
    with a show ?thesis by simp
  next
    case SENDER
    with a show ?thesis by simp
  next
    case VALUE
    with a show ?thesis by simp
  next
    case t: TRUE
    show ?thesis
    proof (cases "eupdate e2")
      case (INT x11 x12)
      with a t show ?thesis by simp
    next
      case (UINT x21 x22)
      with a t show ?thesis by simp
    next
      case (ADDRESS x3)
      with a t show ?thesis by simp
    next
      case (BALANCE x4)
      with a t show ?thesis by simp
    next
      case THIS
      with a t show ?thesis by simp
    next
      case SENDER
      with a t show ?thesis by simp
    next
      case VALUE
      with a t show ?thesis by simp
    next
      case TRUE
      with a t show ?thesis by simp
    next
      case FALSE
      with a t show ?thesis by simp
    next
      case (LVAL x7)
      with a t show ?thesis by simp
    next
      case (PLUS x81 x82)
      with a t show ?thesis by simp
    next
      case (MINUS x91 x92)
      with a t show ?thesis by simp
    next
```

```
      case (EQUAL x101 x102)
      with a t show ?thesis by simp
    next
      case (LESS x111 x112)
      with a t show ?thesis by simp
    next
      case (AND x121 x122)
      with a t show ?thesis by simp
    next
      case (OR x131 x132)
      with a t show ?thesis by simp
    next
      case (NOT x131)
      with a t show ?thesis by simp
    next
      case (CALL x181 x182)
      with a t show ?thesis by simp
    next
      case (ECALL x191 x192 x193 x194)
      with a t show ?thesis by simp
    qed
  next
    case f: FALSE
    show ?thesis
    proof (cases "eupdate e2")
      case (INT b v)
      with a f show ?thesis by simp
    next
      case (UINT b v)
      with a f show ?thesis by simp
    next
      case (ADDRESS x3)
      with a f show ?thesis by simp
    next
      case (BALANCE x4)
      with a f show ?thesis by simp
    next
      case THIS
      with a f show ?thesis by simp
    next
      case SENDER
      with a f show ?thesis by simp
    next
      case VALUE
      with a f show ?thesis by simp
    next
      case TRUE
      with a f show ?thesis by simp
    next
      case FALSE
      with a f show ?thesis by simp
    next
      case (LVAL x7)
      with a f show ?thesis by simp
    next
      case (PLUS x81 x82)
      with a f show ?thesis by simp
    next
      case (MINUS x91 x92)
      with a f show ?thesis by simp
    next
      case (EQUAL x101 x102)
      with a f show ?thesis by simp
    next
```

```
      case (LESS x111 x112)
      with a f show ?thesis by simp
    next
      case (AND x121 x122)
      with a f show ?thesis by simp
    next
      case (OR x131 x132)
      with a f show ?thesis by simp
    next
      case (NOT x131)
      with a f show ?thesis by simp
    next
      case (CALL x181 x182)
      with a f show ?thesis by simp
    next
      case (ECALL x191 x192 x193 x194)
      with a f show ?thesis by simp
    qed
  next
    case (LVAL x7)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case p: (PLUS x81 x82)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (MINUS x91 x92)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (EQUAL x101 x102)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (LESS x111 x112)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (AND x121 x122)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (OR x131 x132)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (NOT x131)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (CALL x181 x182)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
  next
    case (ECALL x191 x192 x193 x194)
    with a show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
  qed
next
  case o: (OR e1 e2)
  show ?case
  proof (cases "eupdate e1")
    case (INT x11 x12)
    with o show ?thesis by simp
  next
    case (UINT x21 x22)
    with o show ?thesis by simp
  next
    case (ADDRESS x3)
    with o show ?thesis by simp
  next
    case (BALANCE x4)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
```

```
next
  case THIS
  with o show ?thesis by simp
next
  case SENDER
  with o show ?thesis by simp
next
  case VALUE
  with o show ?thesis by simp
next
  case t: TRUE
  show ?thesis
  proof (cases "eupdate e2")
    case (INT x11 x12)
    with o t show ?thesis by simp
  next
    case (UINT x21 x22)
    with o t show ?thesis by simp
  next
    case (ADDRESS x3)
    with o t show ?thesis by simp
  next
    case (BALANCE x4)
    with o t show ?thesis by simp
  next
    case THIS
    with o t show ?thesis by simp
  next
    case SENDER
    with o t show ?thesis by simp
  next
    case VALUE
    with o t show ?thesis by simp
  next
    case TRUE
    with o t show ?thesis by simp
  next
    case FALSE
    with o t show ?thesis by simp
  next
    case (LVAL x7)
    with o t show ?thesis by simp
  next
    case (PLUS x81 x82)
    with o t show ?thesis by simp
  next
    case (MINUS x91 x92)
    with o t show ?thesis by simp
  next
    case (EQUAL x101 x102)
    with o t show ?thesis by simp
  next
    case (LESS x111 x112)
    with o t show ?thesis by simp
  next
    case (AND x121 x122)
    with o t show ?thesis by simp
  next
    case (OR x131 x132)
    with o t show ?thesis by simp
  next
    case (NOT x131)
    with o t show ?thesis by simp
  next
```

```
      case (CALL x181 x182)
      with o t show ?thesis by simp
    next
      case (ECALL x191 x192 x193 x194)
      with o t show ?thesis by simp
    qed
  next
    case f: FALSE
    show ?thesis
    proof (cases "eupdate e2")
      case (INT b v)
      with o f show ?thesis by simp
    next
      case (UINT b v)
      with o f show ?thesis by simp
    next
      case (ADDRESS x3)
      with o f show ?thesis by simp
    next
      case (BALANCE x4)
      with o f show ?thesis by simp
    next
      case THIS
      with o f show ?thesis by simp
    next
      case SENDER
      with o f show ?thesis by simp
    next
      case VALUE
      with o f show ?thesis by simp
    next
      case TRUE
      with o f show ?thesis by simp
    next
      case FALSE
      with o f show ?thesis by simp
    next
      case (LVAL x7)
      with o f show ?thesis by simp
    next
      case (PLUS x81 x82)
      with o f show ?thesis by simp
    next
      case (MINUS x91 x92)
      with o f show ?thesis by simp
    next
      case (EQUAL x101 x102)
      with o f show ?thesis by simp
    next
      case (LESS x111 x112)
      with o f show ?thesis by simp
    next
      case (AND x121 x122)
      with o f show ?thesis by simp
    next
      case (OR x131 x132)
      with o f show ?thesis by simp
    next
      case (NOT x131)
      with o f show ?thesis by simp
    next
      case (CALL x181 x182)
      with o f show ?thesis by simp
    next
```

```
      case (ECALL x191 x192 x193 x194)
      with o f show ?thesis by simp
    qed
  next
    case (LVAL x7)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case p: (PLUS x81 x82)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (MINUS x91 x92)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (EQUAL x101 x102)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (LESS x111 x112)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (AND x121 x122)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (OR x131 x132)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (NOT x131)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by auto
  next
    case (CALL x181 x182)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
  next
    case (ECALL x191 x192 x193 x194)
    with o show ?thesis using lift_eq[of e1 ep env cd st "eupdate e1" e2 "eupdate e2"] by simp
  qed
next
  case o: (NOT e)
  show ?case
  proof (cases "eupdate e")
    case (INT x11 x12)
    with o show ?thesis by simp
  next
    case (UINT x21 x22)
    with o show ?thesis by simp
  next
    case (ADDRESS x3)
    with o show ?thesis by simp
  next
    case (BALANCE x4)
    with o show ?thesis by simp
  next
    case THIS
    with o show ?thesis by simp
  next
    case SENDER
    with o show ?thesis by simp
  next
    case VALUE
    with o show ?thesis by simp
  next
    case t: TRUE
    with o show ?thesis by simp
  next
    case f: FALSE
    with o show ?thesis by simp
```

```
  next
    case (LVAL x7)
    with o show ?thesis by simp
  next
    case p: (PLUS x81 x82)
    with o show ?thesis by simp
  next
    case (MINUS x91 x92)
    with o show ?thesis by simp
  next
    case (EQUAL x101 x102)
    with o show ?thesis by simp
  next
    case (LESS x111 x112)
    with o show ?thesis by simp
  next
    case (AND x121 x122)
    with o show ?thesis by simp
  next
    case (OR x131 x132)
    with o show ?thesis by simp
  next
    case (NOT x131)
    with o show ?thesis by simp
  next
    case (CALL x181 x182)
    with o show ?thesis by simp
  next
    case (ECALL x191 x192 x193 x194)
    with o show ?thesis by simp
  qed
next
  case (CALL x181 x182)
  show ?case by simp
next
  case (ECALL x191 x192 x193 x194)
  show ?case by simp
qed

end
```

7.2 Reentrancy (Reentrancy)

In the following we use our semantics to verify a contract implementing a simple token. The contract is defined by definition *victim* and consist of one state variable and two methods:

- The state variable "balance" is a mapping which assigns a balance to each address.
- Method "deposit" allows to send money to the contract which is then added to the sender's balance.
- Method "withdraw" allows to withdraw the callers balance.

We then verify that the following invariant (defined by INV) is preserved by both methods: The difference between

- the contracts own account-balance and
- the sum of all the balances kept in the contracts state variable is larger than a certain threshold.

There are two things to note here: First, Solidity implicitly triggers the call of a so-called fallback method whenever we transfer money to a contract. In particular if another contract calls "withdraw", this triggers an implict call to the callee's fallback method. This functionality was exploited in the infamous DAO attack which we demonstrate it in terms of an example later on. Since we do not know all potential contracts which call "withdraw", we need to verify our invariant for all possible Solidity programs. Thus, the core result here is a lemma which shows that the invariant is preserved by every Solidity program which is not executed in the context of our own contract. For our own methods we show that the invariant holds after executing it. Since our own program as well as the unknown program may depend on each other both properties are encoded in a single lemma (*secure*) which is then proved by induction over all statements. The final result is then given in terms of two corollaries for the corresponding methods of our contract.

The second thing to note is that we were not able to verify that the difference is indeed constant. During verification it turned out that this is not the case since in the fallback method a contract could just send us additional money withouth calling "deposit". In such a case the difference would change. In particular it would grow. However, we were able to verify that the difference does never shrink which is what we actually want to ensure.

```
theory Reentrancy
imports Solidity_Evaluator
begin
```

7.2.1 Example of Re-entrancy

```
value "eval 1000
         stmt
         (COMP
           (EXTERNAL (ADDRESS (STR ''Victim'')) (STR ''deposit'') [] (UINT 256 10))
           (EXTERNAL (ADDRESS (STR ''Victim'')) (STR ''withdraw'') [] (UINT 256 0)))
         (STR ''Attacker'')
         (STR '''')
         (STR ''0'')
         [(STR ''Victim'', STR ''100''), (STR ''Attacker'', STR ''100'')]
         [
           (STR ''Attacker'',
             [],
             ITE
               (LESS (BALANCE THIS) (UINT 256 125))
               (EXTERNAL (ADDRESS (STR ''Victim'')) (STR ''withdraw'') [] (UINT 256 0))
               SKIP),
           (STR ''Victim'',
             [
               (STR ''balance'', Var (STMap TAddr (STValue (TUInt 256)))),
               (STR ''deposit'', Method ([], ASSIGN (Ref (STR ''balance'') [SENDER]) VALUE, None)),
               (STR ''withdraw'', Method ([],
               ITE
                 (LESS (UINT 256 0) (LVAL (Ref (STR ''balance'') [SENDER])))
                 (COMP
                   (TRANSFER SENDER (LVAL (Ref (STR ''balance'') [SENDER])))
                   (ASSIGN (Ref (STR ''balance'') [SENDER]) (UINT 256 0)))
                 SKIP
               , None))],
             SKIP)
         ]
         []"
```

7.2.2 Definition of Contract

```
abbreviation myrexp::L
  where "myrexp ≡ Ref (STR ''balance'') [SENDER]"

abbreviation mylval::E
  where "mylval ≡ LVAL myrexp"

abbreviation assign::S
  where "assign ≡ ASSIGN (Ref (STR ''balance'') [SENDER]) (UINT 256 0)"

abbreviation transfer::S
  where "transfer ≡ TRANSFER SENDER (LVAL (Id (STR ''bal'')))"
```

abbreviation *comp::S*
  where *"comp ≡ COMP assign transfer"*

abbreviation *keep::S*
  where *"keep ≡ BLOCK ((STR ''bal'', Value (TUInt 256)), Some mylval) comp"*

abbreviation *deposit::S*
  where *"deposit ≡ ASSIGN (Ref (STR ''balance'') [SENDER]) (PLUS (LVAL (Ref (STR ''balance'') [SENDER])) VALUE)"*

definition *victim::"(Identifier, Member) fmap"*
  where *"victim ≡ fmap_of_list [*
        (STR ''balance'', Var (STMap TAddr (STValue (TUInt 256)))),
        (STR ''deposit'', Method ([], deposit, None)),
        (STR ''withdraw'', Method ([], keep, None))]"

7.2.3 Definition of Invariant

abbreviation *"SUMM s ≡ ∑(ad,x)|fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x. ReadL$_{int}$ x"*

abbreviation *"POS s ≡ ∀ad x. fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ⟶ ReadL$_{int}$ x ≥ 0"*

abbreviation *"INV st s val bal ≡*
  fmlookup (storage st) (STR ''Victim'') = Some s ∧ ReadL$_{int}$ (accessBalance (accounts st) (STR ''Victim'')) - val ≥ bal ∧ bal ≥ 0"

definition *frame_def: "frame bal st ≡ (∃s. INV st s (SUMM s) bal ∧ POS s)"*

7.2.4 Verification

lemma *conj3: "P ⟹ Q ⟹ R ⟹ P ∧ (Q ∧ R)"* **by** *simp*

lemma *fmfinite: "finite ({(ad, x). fmlookup y ad = Some x})"*
proof -
  have *"{(ad, x). fmlookup y ad = Some x} ⊆ dom (fmlookup y) × ran (fmlookup y)"*
  proof
    fix *x* **assume** *"x ∈ {(ad, x). fmlookup y ad = Some x}"*
    then have *"fmlookup y (fst x) = Some (snd x)"* **by** *auto*
    then have *"fst x ∈ dom (fmlookup y)"* **and** *"snd x ∈ ran (fmlookup y)"* **using** *Map.ranI* **by** *(blast,metis)*
    then show *"x ∈ dom (fmlookup y) × ran (fmlookup y)"* **by** *(simp add: mem_Times_iff)*
  qed
  thus *?thesis* **by** *(simp add: finite_ran finite_subset)*
qed

lemma *fmlookup_finite:*
  fixes *f :: "'a ⇒ 'a"*
    and *y :: "('a, 'b) fmap"*
  assumes *"inj_on (λ(ad, x). (f ad, x)) {(ad, x). (fmlookup y ∘ f) ad = Some x}"*
  shows *"finite {(ad, x). (fmlookup y ∘ f) ad = Some x}"*
proof *(cases rule: inj_on_finite[OF assms(1), of "{(ad, x)|ad x. (fmlookup y) ad = Some x}"])*
  case *1*
  then show *?case* **by** *auto*
next
  case *2*
  then have **: "finite {(ad, x) |ad x. fmlookup y ad = Some x}"* **using** *fmfinite[of y]* **by** *simp*
  show *?case* **using** *finite_image_set[OF *, of "λ(ad, x). (ad, x)"]* **by** *simp*
qed

lemma *balance_inj: "inj_on (λ(ad, x). (ad + (STR ''.'' + STR ''balance''),x)) {(ad, x). (fmlookup y ∘ f) ad = Some x}"*
proof

```
  fix v v' assume asm1: "v ∈ {(ad, x). (fmlookup y ∘ f) ad = Some x}" and asm2: "v' ∈ {(ad, x).
(fmlookup y ∘ f) ad = Some x}"
  and *:"(case v of (ad, x) ⇒ (ad + (STR ''.'' + STR ''balance''),x)) = (case v' of (ad, x) ⇒ (ad +
(STR ''.'' + STR ''balance''),x))"
  then obtain ad ad' x x' where **: "v = (ad,x)" and ***: "v'=(ad',x')" by auto
  moreover from * ** *** have "ad + (STR ''.'' + STR ''balance'') = ad' + (STR ''.'' + STR
''balance'')" by simp
  with * ** have "ad = ad'" using String_Cancel[of ad "STR ''.'' + STR ''balance''" ad' ] by simp
  moreover from asm1 asm2 ** *** have "(fmlookup y ∘ f) ad = Some x" and "(fmlookup y ∘ f) ad' =
Some x'" by auto
  with calculation(3) have "x=x'" by simp
  ultimately show "v=v'" by simp
qed

lemma transfer_frame:
assumes "Accounts.transfer ad adv v (accounts st) = Some acc"
    and "frame bal st"
    and "ad ≠ STR ''Victim''"
  shows "frame bal (st⦇accounts := acc⦈)"
proof (cases "adv = STR ''Victim''")
  case True
  define st' where "st' = st⦇accounts := acc, stack := emptyStore, memory := emptyStore⦈"
  from True assms(2) transfer_mono[OF assms(1)] have "(∃s. fmlookup (storage st) (STR ''Victim'') =
Some s ∧ ReadL_int (accessBalance acc (STR ''Victim'')) - (SUMM s) ≥ bal ∧ bal ≥ 0)" by (auto simp
add:frame_def)
  then have "(∃s. fmlookup (storage st') (STR ''Victim'') = Some s ∧ ReadL_int (accessBalance (accounts
st') (STR ''Victim'')) - (SUMM s) ≥ bal ∧ bal ≥ 0)" by (auto simp add: st'_def)
  then show ?thesis using assms(2) by (auto simp add:st'_def frame_def)
next
  case False
  define st' where "st' = st⦇accounts := acc, stack := emptyStore, memory := emptyStore⦈"
  from False assms(2) assms(3) transfer_eq[OF assms(1)] have "(∃s. fmlookup (storage st) (STR
''Victim'') = Some s ∧ ReadL_int (accessBalance acc (STR ''Victim'')) - (SUMM s) ≥ bal ∧ bal ≥ 0)"
by (auto simp add:frame_def)
  then have "(∃s. fmlookup (storage st') (STR ''Victim'') = Some s ∧ ReadL_int (accessBalance (accounts
st') (STR ''Victim'')) - (SUMM s) ≥ bal ∧ bal ≥ 0)" by (auto simp add: st'_def)
  then show ?thesis using assms(2) by (auto simp add:st'_def frame_def)
qed

lemma decl_frame:
  assumes "frame bal st"
      and "decl a1 a2 a3 cp cd mem c env st = Normal (rv, st')"
    shows "frame bal st'"
proof (cases a2)
  case (Value t)
  then show ?thesis
  proof (cases a3)
    case None
    with Value show ?thesis using assms by (auto simp add:frame_def)
  next
    case (Some a)
    show ?thesis
    proof (cases a)
      case (Pair a b)
      then show ?thesis
      proof (cases a)
        case (KValue v)
        then show ?thesis
        proof (cases b)
          case v2: (Value t')
          show ?thesis
          proof (cases "Valuetypes.convert t' t v")
            case None
            with Some Pair KValue Value v2 show ?thesis using assms by simp
```

```
          next
            case s2: (Some a)
            show ?thesis
            proof (cases a)
              case p2: (Pair a b)
              with Some Pair KValue Value v2 s2 show ?thesis using assms by (auto simp add:frame_def)
            qed
          qed
        next
          case (Calldata x2)
          with Some Pair KValue Value show ?thesis using assms by simp
        next
          case (Memory x3)
          with Some Pair KValue Value show ?thesis using assms by simp
        next
          case (Storage x4)
          with Some Pair KValue Value show ?thesis using assms by simp
        qed
      next
        case (KCDptr x2)
        with Some Pair Value show ?thesis using assms by simp
      next
        case (KMemptr x3)
        with Some Pair Value show ?thesis using assms by simp
      next
        case (KStoptr x4)
        with Some Pair Value show ?thesis using assms by simp
      qed
    qed
  qed
next
  case (Calldata x2)
  then show ?thesis
  proof (cases cp)
    case True
    then show ?thesis
    proof (cases x2)
      case (MTArray x t)
      then show ?thesis
      proof (cases a3)
        case None
        with Calldata show ?thesis using assms by simp
      next
        case (Some a)
        show ?thesis
        proof (cases a)
          case (Pair a b)
          then show ?thesis
          proof (cases a)
            case (KValue x1)
            with Calldata Some Pair show ?thesis using assms by simp
          next
            case (KCDptr p)
            define l where "l = ShowL_nat (toploc c)"
            obtain c' where c_def: "∃x. (x, c') = allocate c" by simp
            show ?thesis
            proof (cases "cpm2m p l x t cd c'")
              case None
              with Calldata MTArray Some Pair KCDptr l_def c_def True show ?thesis using assms by
simp
            next
              case s2: (Some a)
              with Calldata MTArray Some Pair KCDptr l_def c_def True show ?thesis using assms by
(auto simp add:frame_def)
```

```
          qed
        next
          case (KMemptr p)
          define l where "l = ShowL_nat (toploc c)"
          obtain c' where c_def: "∃x. (x, c') = allocate c" by simp
          show ?thesis
          proof (cases "cpm2m p l x t mem c'")
            case None
            with Calldata MTArray Some Pair KMemptr l_def c_def True show ?thesis using assms by
simp
          next
            case s2: (Some a)
            with Calldata MTArray Some Pair KMemptr l_def c_def True show ?thesis using assms by
(auto simp add:frame_def)
          qed
        next
          case (KStoptr x4)
          with Calldata Some Pair show ?thesis using assms by simp
        qed
      qed
    qed
  next
    case (MTValue x2)
    with Calldata show ?thesis using assms by simp
  qed
 next
  case False
  with Calldata show ?thesis using assms by simp
 qed
next
 case (Memory x3)
 then show ?thesis
 proof (cases x3)
  case (MTArray x t)
  then show ?thesis
  proof (cases a3)
    case None
    with Memory MTArray None show ?thesis using assms by (auto simp add:frame_def simp add:Let_def)
  next
    case (Some a)
    then show ?thesis
    proof (cases a)
      case (Pair a b)
      then show ?thesis
      proof (cases a)
        case (KValue x1)
        with Memory MTArray Some Pair show ?thesis using assms by simp
      next
        case (KCDptr p)
        define m l where "m = memory st" and "l = ShowL_nat (toploc m)"
        obtain m' where m'_def: "∃x. (x, m') = allocate m" by simp
        then show ?thesis
        proof (cases "cpm2m p l x t cd m'")
          case None
          with Memory MTArray Some Pair KCDptr m_def l_def m'_def show ?thesis using assms by simp
        next
          case s2: (Some a)
          with Memory MTArray Some Pair KCDptr m_def l_def m'_def show ?thesis using assms by (auto
simp add:frame_def)
        qed
      next
        case (KMemptr p)
        then show ?thesis
        proof (cases cp)
```

```
            case True
            define m l where "m = memory st" and "l = ShowL_nat (toploc m)"
            obtain m' where m'_def: "∃x. (x, m') = allocate m" by simp
            then show ?thesis
            proof (cases "cpm2m p l x t mem m'")
              case None
              with Memory MTArray Some Pair KMemptr True m_def l_def m'_def show ?thesis using assms
by simp
            next
              case s2: (Some a)
              with Memory MTArray Some Pair KMemptr True m_def l_def m'_def show ?thesis using assms
by (auto simp add:frame_def)
            qed
          next
            case False
            with Memory MTArray Some Pair KMemptr show ?thesis using assms by (auto simp
add:frame_def)
          qed
        next
          case (KStoptr p)
          then show ?thesis
          proof (cases b)
            case (Value x1)
            with Memory MTArray Some Pair KStoptr show ?thesis using assms by simp
          next
            case (Calldata x2)
            with Memory MTArray Some Pair KStoptr show ?thesis using assms by simp
          next
            case m2: (Memory x3)
            with Memory MTArray Some Pair KStoptr show ?thesis using assms by simp
          next
            case (Storage x4)
            then show ?thesis
            proof (cases x4)
              case (STArray x' t')
              define m l where "m = memory st" and "l = ShowL_nat (toploc m)"
              obtain m' where m'_def: "∃x. (x, m') = allocate m" by simp
              from assms(2) Memory MTArray Some Pair KStoptr Storage STArray m_def l_def m'_def
              obtain s where *: "fmlookup (storage st) (address env) = Some s" using Let_def by (auto
simp add: Let_def split:option.split_asm)
              then show ?thesis
              proof (cases "cps2m p l x' t' s m'")
                case None
                with Memory MTArray Some Pair KStoptr Storage STArray m_def l_def m'_def * show
?thesis using assms by simp
              next
                case s2: (Some a)
                with Memory MTArray Some Pair KStoptr Storage STArray m_def l_def m'_def * show
?thesis using assms by (auto simp add:frame_def)
              qed
            next
              case (STMap x21 x22)
              with Memory MTArray Some Pair KStoptr Storage show ?thesis using assms by simp
            next
              case (STValue x3)
              with Memory MTArray Some Pair KStoptr Storage show ?thesis using assms by simp
            qed
          qed
        qed
      qed
    qed
  next
    case (MTValue x2)
    with Memory show ?thesis using assms by simp
```

```
  qed
next
  case (Storage x4)
  then show ?thesis
  proof (cases x4)
    case (STArray x t)
    then show ?thesis
    proof (cases a3)
      case None
      with Storage STArray show ?thesis using assms by simp
    next
      case (Some a)
      then show ?thesis
      proof (cases a)
        case (Pair a b)
        then show ?thesis
        proof (cases a)
          case (KValue x1)
          with Storage STArray Some Pair show ?thesis using assms by simp
        next
          case (KCDptr x2)
          with Storage STArray Some Pair show ?thesis using assms by simp
        next
          case (KMemptr x3)
          with Storage STArray Some Pair show ?thesis using assms by simp
        next
          case (KStoptr x4)
          with Storage STArray Some Pair show ?thesis using assms by (auto simp add:frame_def)
        qed
      qed
    qed
  next
    case (STMap t t')
    then show ?thesis
    proof (cases a3)
      case None
      with Storage STMap show ?thesis using assms by simp
    next
      case (Some a)
      then show ?thesis
      proof (cases a)
        case (Pair a b)
        then show ?thesis
        proof (cases a)
          case (KValue x1)
          with Storage STMap Some Pair show ?thesis using assms by simp
        next
          case (KCDptr x2)
          with Storage STMap Some Pair show ?thesis using assms by simp
        next
          case (KMemptr x3)
          with Storage STMap Some Pair show ?thesis using assms by simp
        next
          case (KStoptr x4)
          with Storage STMap Some Pair show ?thesis using assms by (auto simp add:frame_def)
        qed
      qed
    qed
  next
    case (STValue x3)
    with Storage show ?thesis using assms by simp
  qed
qed
```

```
context statement_with_gas
begin

lemma secureassign:
  assumes "stmt assign ep env cd st = Normal((), st')"
      and "fmlookup (storage st) (STR ''Victim'') = Some s"
      and "address env = (STR ''Victim'')"
      and "fmlookup (denvalue env) (STR ''balance'') = Some (Storage (STMap TAddr (STValue (TUInt
256))), Storeloc STR ''balance'')"
      and "accessStore x (stack st) = Some (KValue (accessStorage (TUInt 256) (sender env + (STR ''.''
+ STR ''balance'')) s))"
      and "ReadL_int (accessBalance (accounts st) (STR ''Victim'')) - (SUMM s) ≥ bal"
      and "POS s"
    obtains s'
    where "fmlookup (storage st') (STR ''Victim'') = Some s'"
      and "ReadL_int (accessBalance (accounts st') (STR ''Victim'')) - (SUMM s' + ReadL_int (accessStorage
(TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s)) ≥ bal"
      and "accessStore x (stack st') = Some (KValue (accessStorage (TUInt 256) (sender env + (STR ''.''
+ STR ''balance'')) s))"
      and "POS s'"
proof -
  define st'' where "st'' = st⦇gas := gas st - costs assign ep env cd st⦈"
  define st''' where "st''' = st''⦇gas := gas st'' - costs_e (UINT 256 0) ep env cd st''⦈"
  define st'''' where "st'''' = st'''⦇gas := gas st''' - costs_e SENDER ep env cd st'''⦈"

  from assms(1) have c1: "gas st > costs assign ep env cd st" by (auto split:if_split_asm)
  have c2: "gas st'' > costs_e (UINT 256 0) ep env cd st''"
  proof (rule ccontr)
    assume "¬ costs_e (UINT 256 0) ep env cd st'' < gas st''"
    with c1 show False using assms(1) st''_def st'''_def by auto
  qed
  hence *:"expr (UINT 256 0) ep env cd st'' = Normal ((KValue (createUInt 256 0),Value (TUInt 256)),
st''')" using expr.simps(2)[of 256 0 ep env cd "st⦇gas := gas st - costs assign ep env cd st⦈"]
st''_def st'''_def by simp
  moreover have "gas st''' > costs_e SENDER ep env cd st'''"
  proof (rule ccontr)
    assume "¬ costs_e SENDER ep env cd st''' < gas st'''"
    with c1 c2 show False using assms(1,4) st''_def st'''_def by auto
  qed
  with assms(4) have **:"lexp (Ref (STR ''balance'') [SENDER]) ep env cd st''' = Normal ((LStoreloc
((sender env) + (STR ''.'' +  STR ''balance'')), Storage (STValue (TUInt 256))), st'''')" using
st''''_def by simp
  moreover have "Valuetypes.convert (TUInt 256) (TUInt 256) (ShowL_int 0) = Some (ShowL_int 0, TUInt
256)" by simp
  moreover from * ** st''_def assms(1) obtain s'' where ***: "fmlookup (storage st'''') (address env)
= Some s''" by (auto split:if_split_asm option.split_asm)
  ultimately have ****:"st' = st''''⦇storage := fmupd (STR ''Victim'') (fmupd ((sender env) + (STR
''.'' + STR ''balance'')) (ShowL_int 0) s'') (storage st)⦈" using c1 st''_def st'''_def st''''_def
assms(1,3) by auto
  moreover define s' where s'_def: "s' = (fmupd ((sender env) + (STR ''.'' + STR ''balance''))
(ShowL_int 0) s'')"
  ultimately have "fmlookup (storage st') (STR ''Victim'') = Some s'"
              and *****:"fmlookup s' ((sender env) + (STR ''.'' +  STR ''balance'')) = Some (ShowL_int
0)" by simp_all
  moreover have "SUMM s' + ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR
''balance'')) s) = SUMM s"
  proof -
    have s1: "SUMM s = (∑(ad,x)|fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad
≠ sender env. ReadL_int x) + ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR
''balance'')) s)"
    proof (cases "fmlookup s (sender env + (STR ''.'' + STR ''balance'')) = None")
      case True
      then have "accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s = ShowL_int 0"
```

```
by simp
      moreover have "{(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x} = {(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"
      proof
        show "{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x} ⊆ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"
        proof
          fix x
          assume "x ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x}"
          then show "x ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}" using True by auto
        qed
      next
        show "{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env} ⊆ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x }"
        proof
          fix x
          assume "x ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"
          then show "x ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x}" using True by auto
        qed
      qed
      then have "SUMM s = (∑(ad,x)|fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env. ReadL_int x)" by simp
      ultimately show ?thesis using Read_ShowL_id by simp
    next
      case False
      then obtain val where val_def: "fmlookup s (sender env + (STR ''.'' + STR ''balance'')) = Some val" by auto

      have "inj_on (λ(ad, x). (ad + (STR ''.'' + STR ''balance''), x)) {(ad, x). (fmlookup s ∘ (λad. ad + (STR ''.'' + STR ''balance''))) ad = Some x}" using balance_inj by simp
      then have "finite {(ad, x). (fmlookup s ∘ (λad. ad + (STR ''.'' + STR ''balance''))) ad = Some x}" using fmlookup_finite[of "λad. (ad + (STR ''.'' + STR ''balance''))" s] by simp
      then have sum1: "finite ({(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env})" using finite_subset[of "{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}" "{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x}"] by auto
      moreover have sum2: "(sender env,val) ∉ {(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}" by simp
      moreover from sum1 x1 val_def have "insert (sender env,val) {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env} = {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x}" by auto
      ultimately show ?thesis using sum.insert[OF sum1 sum2, of "λ(ad,x). ReadL_int x"] val_def by simp
    qed
    moreover have s2: "SUMM s' = (∑(ad,x)|fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env. ReadL_int x)"
    proof -
      have "inj_on (λ(ad, x). (ad + (STR ''.'' + STR ''balance''), x)) {(ad, x). (fmlookup s' ∘ (λad. ad + (STR ''.'' + STR ''balance''))) ad = Some x}" using balance_inj by simp
      then have "finite {(ad, x). (fmlookup s' ∘ (λad. ad + (STR ''.'' + STR ''balance''))) ad = Some x}" using fmlookup_finite[of "λad. (ad + (STR ''.'' + STR ''balance''))" s'] by simp
      then have sum1: "finite ({(ad,x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env})" using finite_subset[of "{(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}" "{(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x}"] by auto
      moreover have sum2: "(sender env,ShowL_int 0) ∉ {(ad,x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}" by simp
      moreover from ***** have "insert (sender env,ShowL_int 0) {(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env} = {(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x}" by auto
      ultimately show ?thesis using sum.insert[OF sum1 sum2, of "λ(ad,x). ReadL_int x"] Read_ShowL_id by simp
```

```
    qed
    moreover have s3: "(∑(ad,x)|fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠
sender env. ReadL_int x)
                =(∑(ad,x)|fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender
env. ReadL_int x)"
    proof -
      have "{(ad,x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env} =
{(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"
      proof
        show "{(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}
          ⊆ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"
        proof
          fix xx
          assume "xx ∈ {(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠
sender env}"
          then obtain ad x where "xx = (ad,x)" and "fmlookup s' (ad + (STR ''.'' + STR ''balance''))
= Some x" and "ad ≠ sender env" by auto
          moreover have "s''=s" using assms(2,3) s'_def *** st''''_def st'''_def st''_def by simp
          moreover from ‘ad ≠ sender env‘ have "ad + (STR ''.'' + STR ''balance'') ≠ (sender env) +
(STR ''.'' + STR ''balance'')" using String_Cancel[where c="(STR ''.'' + STR ''balance'')"] by auto
          ultimately show "xx ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧
ad ≠ sender env}" using s'_def by simp
        qed
      next
        show "{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}
          ⊆ {(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"
        proof
          fix xx
          assume "xx ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠
sender env}"
          then obtain ad x where "xx = (ad,x)" and "fmlookup s (ad + (STR ''.'' + STR ''balance'')) =
Some x" and "ad ≠ sender env" by auto
          moreover have "s''=s" using assms(2,3) s'_def *** st''''_def st'''_def st''_def by simp
          moreover from ‘ad ≠ sender env‘ have "ad + (STR ''.'' + STR ''balance'') ≠ (sender env) +
(STR ''.'' + STR ''balance'')" using String_Cancel[where c="(STR ''.'' + STR ''balance'')"] by auto
          ultimately show "xx ∈ {(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧
ad ≠ sender env}" using s'_def by simp
        qed
      qed
      thus ?thesis by simp
    qed
    ultimately have "SUMM s' = SUMM s - ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' +
STR ''balance'')) s) "
    proof -
      from s2 have "SUMM s' = (∑(ad,x)|fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad
≠ sender env. ReadL_int x)" by simp
      also from s3 have "... = (∑(ad,x)|fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad
≠ sender env. ReadL_int x)" by simp
      also from s1 have "... = SUMM s - ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' +
STR ''balance'')) s) " by simp
      finally show ?thesis .
    qed
    then show ?thesis by simp
  qed
  moreover have "POS s'"
  proof (rule allI[OF allI])
    fix x xa
    show "fmlookup s' (x + (STR ''.'' + STR ''balance'')) = Some xa ⟶ 0 ≤ (⌈xa⌉::int)"
    proof
      assume a1: "fmlookup s' (x + (STR ''.'' + STR ''balance'')) = Some xa"
      show "0 ≤ (⌈xa⌉::int)"
      proof (cases "x = sender env")
        case True
        then show ?thesis using s'_def a1 using Read_ShowL_id by auto
```

```
      next
        case False
        moreover have "s''=s" using assms(2,3) s'_def *** st''''_def st'''_def st''_def by simp
        ultimately have "fmlookup s (x + (STR ''.'' + STR ''balance'')) = Some xa" using s'_def a1
String_Cancel by (auto split:if_split_asm)
        then show ?thesis using assms(7) by simp
      qed
    qed
  qed
  moreover have "ReadL_int (accessBalance (accounts st') (STR ''Victim'')) = ReadL_int (accessBalance
(accounts st) (STR ''Victim''))" using **** st''_def st'''_def st''''_def by simp
  moreover from assms(5) have "accessStore x (stack st') = Some (KValue (accessStorage (TUInt 256)
(sender env + (STR ''.'' + STR ''balance'')) s) )"using **** st''_def st'''_def st''''_def by simp
  ultimately show ?thesis using assms(6) that by simp
qed


lemma securesender:
  assumes "expr SENDER ep env cd st = Normal((KValue v,t), st')"
      and "fmlookup (storage st) (STR ''Victim'') = Some s"
      and "ReadL_int (accessBalance (accounts st) (STR ''Victim'')) - SUMM s ≥ bal ∧ POS s"
    obtains s' where
      "v = sender env"
      and "t = Value TAddr"
      and "fmlookup (storage st') (STR ''Victim'') = Some s'"
      and "ReadL_int (accessBalance (accounts st') (STR ''Victim'')) - SUMM s' ≥ bal ∧ POS s'"
  using assms by (auto split:if_split_asm)

lemma securessel:
  assumes "ssel type loc [] ep env cd st = Normal (x, st')"
      and "fmlookup (storage st) (STR ''Victim'') = Some s"
      and "ReadL_int (accessBalance (accounts st) (STR ''Victim'')) - SUMM s ≥ bal ∧ POS s"
    obtains s' where
      "x = (loc, type)"
      and "fmlookup (storage st') (STR ''Victim'') = Some s'"
      and "ReadL_int (accessBalance (accounts st') (STR ''Victim'')) - SUMM s' ≥ bal ∧ POS s'"
  using assms by auto

lemma securessel2:
  assumes "ssel (STMap TAddr (STValue (TUInt 256))) (STR ''balance'') [SENDER] ep env cd st = Normal
((loc, type), st')"
      and "fmlookup (storage st) (STR ''Victim'') = Some s"
      and "ReadL_int (accessBalance (accounts st) (STR ''Victim'')) - SUMM s ≥ bal ∧ POS s"
    obtains s' where
      "loc = sender env + (STR ''.'' + STR ''balance'')"
      and "type = STValue (TUInt 256)"
      and "fmlookup (storage st') (STR ''Victim'') = Some s'"
      and "ReadL_int (accessBalance (accounts st') (STR ''Victim'')) - SUMM s' ≥ bal ∧ POS s'"
proof -
  from assms(1) obtain v t st'' st''' x
    where *: "expr SENDER ep env cd st = Normal ((KValue v, t), st'')"
      and **: "ssel (STValue (TUInt 256)) (hash (STR ''balance'') v) [] ep env cd st'' = Normal
(x,st''')"
      and "st' = st'''"
    by (auto split:if_split_asm)
  moreover obtain s'' where "v =sender env"
      and "t = Value TAddr"
      and ***:"fmlookup (storage st'') (STR ''Victim'') = Some s''"
      and ****: "ReadL_int (accessBalance (accounts st'') (STR ''Victim'')) - SUMM s'' ≥ bal ∧ POS s''"
    using securesender[OF * assms(2,3)] by auto
  moreover obtain s''' where "x = (hash (STR ''balance'') v, STValue (TUInt 256))"
      and "fmlookup (storage st''') (STR ''Victim'') = Some s'''"
      and "ReadL_int (accessBalance (accounts st''') (STR ''Victim'')) - SUMM s''' ≥ bal ∧ POS s'''"
    using securessel[OF ** *** ****] by auto
```

```
  ultimately show ?thesis using assms(1) that by simp
qed

lemma securerexp:
  assumes "rexp myrexp e_p env cd st = Normal ((v, t), st')"
      and "fmlookup (denvalue env) (STR ''balance'') = Some (Storage (STMap TAddr (STValue (TUInt
256))), Storeloc STR ''balance'')"
      and "fmlookup (storage st) (STR ''Victim'') = Some s"
      and "ReadL_int (accessBalance (accounts st) (STR ''Victim'')) - SUMM s ≥ bal ∧ POS s"
      and "address env = STR ''Victim''"
    obtains s' where
      "fmlookup (storage st') (address env) = Some s'"
      and "v = KValue (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s')"
      and "t = Value (TUInt 256)"
      and "ReadL_int (accessBalance (accounts st') (STR ''Victim'')) - SUMM s' ≥ bal ∧ POS s'"
proof -
  from assms(1,2) obtain l' t'' st'' s
    where *: "ssel (STMap TAddr (STValue (TUInt 256))) (STR ''balance'') [SENDER] e_p env cd st = Normal
((l', STValue t''), st'')"
      and "fmlookup (storage st'') (address env) = Some s"
      and "v = KValue (accessStorage t'' l' s)"
      and "t = Value t''" and "st'=st''"
    by (simp split:if_split_asm option.split_asm)
  moreover obtain s'' where
      "fmlookup (storage st'') (STR ''Victim'') = Some s''"
      and "ReadL_int (accessBalance (accounts st'') (STR ''Victim'')) - SUMM s'' ≥ bal ∧ POS s''"
      and "l'=sender env + (STR ''.'' + STR ''balance'')" and "t'' = TUInt 256" using securessel2[OF *
assms(3,4)] by blast
  ultimately show ?thesis using assms(1,5) that by auto
qed

lemma securelval:
  assumes "expr mylval ep env cd st = Normal((v,t), st')"
      and "fmlookup (denvalue env) (STR ''balance'') = Some (Storage (STMap TAddr (STValue (TUInt
256))), Storeloc STR ''balance'')"
      and "fmlookup (storage st) (STR ''Victim'') = Some s"
      and "ReadL_int (accessBalance (accounts st) (STR ''Victim'')) - SUMM s ≥ bal ∧ bal ≥ 0 ∧ POS s"
      and "address env = STR ''Victim''"
    obtains s' where "fmlookup (storage st') (STR ''Victim'') = Some s'"
      and "v = KValue (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s')"
      and "t = Value (TUInt 256)"
      and "ReadL_int (accessBalance (accounts st') (STR ''Victim'')) - SUMM s' ≥ bal ∧ bal ≥ 0 ∧ POS
s'"
proof -
  define st'' where "st'' = st⦇gas := gas st - costs_e mylval ep env cd st⦈"
  with assms(3,4) have *: "fmlookup (storage st'') (STR ''Victim'') = Some s"
    and **: "ReadL_int (accessBalance (accounts st'') (STR ''Victim'')) - SUMM s ≥ bal ∧ POS s" by
simp+

  from assms(1) st''_def obtain v' t' st''' where ***: "rexp myrexp ep env cd st'' =  Normal ((v, t),
st''')"
  and "v' = v"
  and "t' = t"
  and "st''' = st'"
    by (simp split:if_split_asm)

  with securerexp[OF *** assms(2) * **] show ?thesis using assms(1,4,5) that by auto
qed

lemma plus_frame:
  assumes "expr (PLUS (LVAL (Ref (STR ''balance'') [SENDER])) VALUE) ep env cd st = Normal (kv, st')"
      and "ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s) +
ReadL_int (svalue env) < 2^256"
      and "ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s) +
```

```
ReadL_int (svalue env) ≥ 0"
      and "fmlookup (storage st) (STR ''Victim'') = Some s"
      and "ReadL_int (accessBalance (accounts st) (STR ''Victim'')) - SUMM s ≥ bal"
      and "fmlookup (denvalue env) (STR ''balance'') = Some (Storage (STMap TAddr (STValue (TUInt
256))), Storeloc STR ''balance'')"
      and "address env = (STR ''Victim'')"
    shows "kv = (KValue (ShowL_int (ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR
''balance'')) s) + ReadL_int (svalue env))), Value (TUInt 256))"
      and "fmlookup (storage st') (STR ''Victim'') = Some s"
      and "ReadL_int (accessBalance (accounts st') (STR ''Victim'')) = ReadL_int (accessBalance (accounts
st) (STR ''Victim''))"
proof -
  define st0 where "st0 = st⦇gas := gas st - costs_e (PLUS (LVAL (Ref (STR ''balance'') [SENDER]))
VALUE) ep env cd st⦈"
  define st1 where "st1 = st0⦇gas := gas st0 - costs_e (LVAL (Ref (STR ''balance'') [SENDER])) ep env cd
st0⦈"
  define st2 where "st2 = st1⦇gas := gas st1 - costs_e SENDER ep env cd st1⦈"

  define st3 where "st3 = st2⦇gas := gas st2 - costs_e VALUE ep env cd st2⦈"
  from assms(1) obtain v1 t1 v2 t2 st'' st''' st'''' v t where
    *: "expr (LVAL (Ref (STR ''balance'') [SENDER])) ep env cd st0 = Normal ((KValue v1, Value t1),
st'')"
    and **: "expr VALUE ep env cd st'' = Normal ((KValue v2, Value t2), st''')"
    and ***: "add t1 t2 v1 v2 = Some (v,t)"
    and ****: "expr (PLUS (LVAL (Ref (STR ''balance'') [SENDER])) VALUE) ep env cd st = Normal ((KValue
v, Value t), st'''')"
    using st0_def by (auto simp del: expr.simps simp add:expr.simps(11) split:if_split_asm
result.split_asm Stackvalue.split_asm Type.split_asm option.split_asm)

  moreover have "expr (LVAL (Ref (STR ''balance'') [SENDER])) ep env cd st0 = Normal ((KValue
(accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s), Value (TUInt 256)), st'')"
    and "st'' = st2"
  proof -
    from * obtain l' t' s'' where *****: "ssel (STMap TAddr (STValue (TUInt 256))) (STR ''balance'')
[SENDER] ep env cd st1 = Normal ((l', STValue t'), st'')"
      and ******: "fmlookup (storage st'') (address env) = Some s''" and "v1 = (accessStorage t' l'
s'')" and "t' = t1"
      using st0_def st1_def assms(4,6) by (auto simp del: accessStorage.simps ssel.simps
split:if_split_asm option.split_asm STypes.split_asm result.split_asm)
    moreover from ***** have "expr SENDER ep env cd st1 = Normal ((KValue (sender env), Value TAddr),
st2)" using st2_def by (simp split:if_split_asm)
    with ***** have "st'' = st2" and "l' = sender env + (STR ''.'' + STR ''balance'')" and "t' =
TUInt 256" by auto
    moreover from ****** ‘st'' = st2‘ have "s''=s" using st0_def st1_def st2_def assms(4,7) by auto
    ultimately show "expr (LVAL (Ref (STR ''balance'') [SENDER])) ep env cd st0 = Normal ((KValue
(accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s), Value (TUInt 256)), st'')"
    and "st'' = st2" using * by (auto split:if_split_asm)
  qed

  moreover from ** ‘st'' = st2‘ have "expr VALUE ep env cd st2 = Normal ((KValue (svalue env), Value
(TUInt 256)), st3)" and "st''' = st3" using st1_def st3_def by (auto split:if_split_asm)
  moreover have "add (TUInt 256) (TUInt 256) (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR
''balance'')) s) (svalue env) = Some (ShowL_int (ReadL_int (accessStorage (TUInt 256) (sender env + (STR
''.'' + STR ''balance'')) s) + ReadL_int (svalue env)), TUInt 256)" (is "?LHS = ?RHS")
  proof -
    have "?LHS = Some (createUInt 256 (⌈(accessStorage (TUInt 256) (sender env + (STR ''.'' + STR
''balance'')) s)⌉+ ⌈(svalue env)⌉), TUInt 256)" using add_def olift.simps(2)[of "(+)" 256 256] by simp
    with assms(2,3) show "?LHS = ?RHS" by simp
  qed
  ultimately have "v= (ShowL_int (ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR
''balance'')) s) + ReadL_int (svalue env)))" and "t = TUInt 256" and "st' = st3" using st0_def assms(1)
by (auto split:if_split_asm)
  with assms(1) **** have "kv = (KValue (ShowL_int (ReadL_int (accessStorage (TUInt 256) (sender env +
(STR ''.'' + STR ''balance'')) s) + ReadL_int (svalue env))), Value (TUInt 256))" using st0_def by simp
```

```
  moreover from assms(4) st0_def st1_def st2_def st3_def ‘st’ = st3‘ have "fmlookup (storage st’)
(STR ’’Victim’’) = Some s" by simp
  moreover from assms(5) st0_def st1_def st2_def st3_def ‘st’ = st3‘ have "ReadL_int (accessBalance
(accounts st’) (STR ’’Victim’’)) - SUMM s ≥ bal" by simp
  moreover have "ReadL_int (accessBalance (accounts st’) (STR ’’Victim’’)) = ReadL_int (accessBalance
(accounts st) (STR ’’Victim’’))" using st0_def st1_def st2_def st3_def ‘st’ = st3‘ by simp
  ultimately show "kv = (KValue (ShowL_int (ReadL_int (accessStorage (TUInt 256) (sender env + (STR ’’.’’
+ STR ’’balance’’)) s) + ReadL_int (svalue env))), Value (TUInt 256))"
   and "fmlookup (storage st’) (STR ’’Victim’’) = Some s"
   and "ReadL_int (accessBalance (accounts st’) (STR ’’Victim’’)) = ReadL_int (accessBalance (accounts st)
(STR ’’Victim’’))" by auto
qed



lemma deposit_frame:
  assumes "stmt deposit ep env cd st = Normal ((), st’)"
      and "fmlookup (storage st) (STR ’’Victim’’) = Some s"
      and "address env = (STR ’’Victim’’)"
      and "fmlookup (denvalue env) (STR ’’balance’’) = Some (Storage (STMap TAddr (STValue (TUInt
256))), Storeloc STR ’’balance’’)"
      and "ReadL_int (accessBalance (accounts st) (STR ’’Victim’’)) - SUMM s ≥ bal + ReadL_int (svalue
env)"
      and "ReadL_int (accessStorage (TUInt 256) (sender env + (STR ’’.’’ + STR ’’balance’’)) s) +
ReadL_int (svalue env) < 2^256"
      and "ReadL_int (accessStorage (TUInt 256) (sender env + (STR ’’.’’ + STR ’’balance’’)) s) +
ReadL_int (svalue env) ≥ 0"
      and "POS s"
    obtains s’
    where "fmlookup (storage st’) (STR ’’Victim’’) = Some s’"
      and "ReadL_int (accessBalance (accounts st’) (STR ’’Victim’’)) - SUMM s’ ≥ bal"
      and "POS s’"
proof -
  define st0 where "st0 = st⦇gas := gas st - costs deposit ep env cd st⦈"

  from assms(1) st0_def obtain kv st’’ where *: "expr (PLUS (LVAL (Ref (STR ’’balance’’) [SENDER]))
VALUE) ep env cd st0 = Normal (kv, st’’)" by (auto simp del: expr.simps split:if_split_asm
result.split_asm)
  moreover have "fmlookup (storage st0) (STR ’’Victim’’) = Some s" using st0_def assms(2) by simp
  moreover from assms(5) have "ReadL_int (accessBalance (accounts st0) (STR ’’Victim’’)) - SUMM s ≥
bal + ReadL_int (svalue env)" using st0_def by simp
  ultimately have **: "kv = (KValue ⌊(⌈accessStorage (TUInt 256) (sender env + (STR ’’.’’ + STR
’’balance’’)) s⌉::int) + ⌈svalue env⌉⌋, Value (TUInt 256))"
    and st’’_s:"fmlookup (storage st’’) STR ’’Victim’’ = Some s"
    and ac: "ReadL_int (accessBalance (accounts st’’) (STR ’’Victim’’)) = ReadL_int (accessBalance
(accounts st0) (STR ’’Victim’’))"
    using plus_frame[OF _ assms(6,7) _ _ assms(4,3), of ep cd st0 kv st’’] by auto

  define v where "v= (⌈accessStorage (TUInt 256) (sender env + (STR ’’.’’ + STR ’’balance’’)) s⌉::int)
+ ⌈svalue env⌉"
  moreover from * ** assms(1) st0_def obtain rl st’’’ where ***: "lexp (Ref (STR ’’balance’’)
[SENDER]) ep env cd st’’ = Normal (rl, st’’’)" by (auto simp del:expr.simps lexp.simps
accessStorage.simps split:if_split_asm result.split_asm)
  moreover from *** assms have "rl = (LStoreloc ((sender env) + (STR ’’.’’ +  STR ’’balance’’)),
Storage (STValue (TUInt 256)))" and st’’’_def: "st’’’ = st’’⦇gas := gas st’’ - costs_e SENDER ep env cd
st’’⦈"
  proof -
    from *** assms(4) obtain l’ t’ where
      "fmlookup (denvalue env) (STR ’’balance’’) = Some (Storage (STMap TAddr (STValue (TUInt 256))),
Storeloc (STR ’’balance’’))"
      and *:"ssel (STMap TAddr (STValue (TUInt 256))) (STR ’’balance’’) [SENDER] ep env cd st’’ =
Normal ((l’,t’), st’’’)"
      and "rl = (LStoreloc l’, Storage t’)"
      by (auto simp del: ssel.simps split:if_split_asm option.split_asm result.split_asm)
```

moreover from * **have** *"ssel (STMap TAddr (STValue (TUInt 256))) (STR ''balance'') [SENDER] ep env cd st'' = Normal ((((sender env) + (STR ''.'' + STR ''balance'')), STValue (TUInt 256)), st''⦇gas := gas st'' - costs$_e$ SENDER ep env cd st''⦈)"* **by** *(simp split:if_split_asm)*
ultimately show *"rl = (LStoreloc ((sender env) + (STR ''.'' + STR ''balance'')), Storage (STValue (TUInt 256)))"* **and** *st'''_def: "st''' = st''⦇gas := gas st'' - costs$_e$ SENDER ep env cd st''⦈"* **by** *auto*
qed
moreover have *"Valuetypes.convert (TUInt 256) (TUInt 256) (ShowL$_{int}$ v) = Some (ShowL$_{int}$ v, TUInt 256)"* **by** *simp*

moreover from *st''_s st'''_def* **have** *s'''_s: "fmlookup (storage st''') (STR ''Victim'') = Some s"* **by** *simp*
ultimately have *****:"st' = st'''⦇storage := fmupd (STR ''Victim'') (fmupd ((sender env) + (STR ''.'' + STR ''balance'')) (ShowL$_{int}$ v) s) (storage st''')⦈"*
using *assms(1) * ** st0_def assms(3)* **by** *(auto simp del:expr.simps lexp.simps accessStorage.simps split:if_split_asm)*

moreover define *s'* **where** *"s' = (fmupd ((sender env) + (STR ''.'' + STR ''balance'')) (ShowL$_{int}$ v) s)"*
ultimately have *"fmlookup (storage st') (STR ''Victim'') = Some s'"*
and *******:"fmlookup s' ((sender env) + (STR ''.'' + STR ''balance'')) = Some (ShowL$_{int}$ v)"* **by** *simp_all*

moreover have *"SUMM s' = SUMM s + ⌈svalue env⌉"*
proof -
have *s1: "SUMM s = (∑(ad,x)|fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env. ReadL$_{int}$ x) + ReadL$_{int}$ (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s)"*
proof *(cases "fmlookup s (sender env + (STR ''.'' + STR ''balance'')) = None")*
case *True*
then have *"accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance'')) s = ShowL$_{int}$ 0"* **by** *simp*
moreover have *"{(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x} = {(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"*
proof
show *"{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x} ⊆ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"*
proof
fix *x*
assume *"x ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x}"*
then show *"x ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"* **using** *True* **by** *auto*
qed
next
show *"{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env} ⊆ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x }"*
proof
fix *x*
assume *"x ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"*
then show *"x ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x}"* **using** *True* **by** *auto*
qed
qed
then have *"SUMM s = (∑(ad,x)|fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env. ReadL$_{int}$ x)"* **by** *simp*
ultimately show *?thesis* **using** *Read_ShowL_id* **by** *simp*
next
case *False*
then obtain *val* **where** *val_def: "fmlookup s (sender env + (STR ''.'' + STR ''balance'')) = Some val"* **by** *auto*

have *"inj_on (λ(ad, x). (ad + (STR ''.'' + STR ''balance''), x)) {(ad, x). (fmlookup s ∘ (λad. ad + (STR ''.'' + STR ''balance''))) ad = Some x}"* **using** *balance_inj* **by** *simp*
then have *"finite {(ad, x). (fmlookup s ∘ (λad. ad + (STR ''.'' + STR ''balance''))) ad = Some*

```
x}" using fmlookup_finite[of "λad. (ad + (STR ''.'' + STR ''balance''))" s] by simp
      then have sum1: "finite ({(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad
≠ sender env})" using finite_subset[of "{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) =
Some x ∧ ad ≠ sender env}" "{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x}"] by
auto
      moreover have sum2: "(sender env,val) ∉ {(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance''))
= Some x ∧ ad ≠ sender env}" by simp
      moreover from sum1 x1 val_def have "insert (sender env,val) {(ad, x). fmlookup s (ad + (STR
''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env} = {(ad, x). fmlookup s (ad + (STR ''.'' + STR
''balance'')) = Some x}" by auto
      ultimately show ?thesis using sum.insert[OF sum1 sum2, of "λ(ad,x). ReadL_int x"] val_def by simp
    qed
    moreover have s2: "SUMM s' = (∑(ad,x)|fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x
∧ ad ≠ sender env. ReadL_int x) + v"
    proof -
      have "inj_on (λ(ad, x). (ad + (STR ''.'' + STR ''balance''), x)) {(ad, x). (fmlookup s' ∘ (λad.
ad + (STR ''.'' + STR ''balance''))) ad = Some x}" using balance_inj by simp
      then have "finite {(ad, x). (fmlookup s' ∘ (λad. ad + (STR ''.'' + STR ''balance''))) ad = Some
x}" using fmlookup_finite[of "λad. (ad + (STR ''.'' + STR ''balance''))" s'] by simp
      then have sum1: "finite ({(ad,x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad
≠ sender env})" using finite_subset[of "{(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) =
Some x ∧ ad ≠ sender env}" "{(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x}"] by
auto
      moreover have sum2: "(sender env,ShowL_int v) ∉ {(ad,x). fmlookup s' (ad + (STR ''.'' + STR
''balance'')) = Some x ∧ ad ≠ sender env}" by simp
      moreover from ***** have "insert (sender env,ShowL_int v) {(ad, x). fmlookup s' (ad + (STR
''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env} = {(ad, x). fmlookup s' (ad + (STR ''.'' + STR
''balance'')) = Some x}" by auto
      ultimately show ?thesis using sum.insert[OF sum1 sum2, of "λ(ad,x). ReadL_int x"] Read_ShowL_id
by simp
    qed
    moreover have s3: "(∑(ad,x)|fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠
sender env. ReadL_int x)
                =(∑(ad,x)|fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender
env. ReadL_int x)"
    proof -
      have "{(ad,x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env} =
{(ad,x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"
      proof
        show "{(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}
          ⊆ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"
        proof
          fix xx
          assume "xx ∈ {(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠
sender env}"
          then obtain ad x where "xx = (ad,x)" and "fmlookup s' (ad + (STR ''.'' + STR ''balance''))
= Some x" and "ad ≠ sender env" by auto
          then have "fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x" using s'_def
String_Cancel[of ad "(STR ''.'' + STR ''balance'')" "sender env"] by (simp split:if_split_asm)
          with `ad ≠ sender env` `xx = (ad,x)` show "xx ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR
''balance'')) = Some x ∧ ad ≠ sender env}" by simp
        qed
      next
        show "{(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}
          ⊆ {(ad, x). fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠ sender env}"
        proof
          fix xx
          assume "xx ∈ {(ad, x). fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad ≠
sender env}"
          then obtain ad x where "xx = (ad,x)" and "fmlookup s (ad + (STR ''.'' + STR ''balance'')) =
Some x" and "ad ≠ sender env" by auto
          then have "fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x" using s'_def
String_Cancel[of ad "(STR ''.'' + STR ''balance'')" "sender env"] by (auto split:if_split_asm)
          with `ad ≠ sender env` `xx = (ad,x)` show "xx ∈ {(ad, x). fmlookup s' (ad + (STR ''.'' +
```

```
STR ''balance'')) = Some x ∧ ad ≠ sender env}" by simp
        qed
      qed
      thus ?thesis by simp
    qed
    moreover from s'_def v_def have "ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' +
STR ''balance'')) s') = ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' + STR ''balance''))
s) + ⌈svalue env⌉" using Read_ShowL_id by (simp split:option.split_asm)
    ultimately have "SUMM s' = SUMM s + ⌈svalue env⌉"
    proof -
      from s2 have "SUMM s' = (∑(ad,x)|fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad
≠ sender env. ReadL_int x) + v" by simp
      also from s3 have "... = (∑(ad,x)|fmlookup s (ad + (STR ''.'' + STR ''balance'')) = Some x ∧ ad
≠ sender env. ReadL_int x) + v" by simp
      also from s1 have "... = SUMM s - ReadL_int (accessStorage (TUInt 256) (sender env + (STR ''.'' +
STR ''balance'')) s) + v" by simp
      finally show ?thesis using v_def by simp
    qed
    then show ?thesis by simp
  qed
  moreover have "POS s'"
  proof (rule allI[OF allI])
    fix ad xa
    show "fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some xa ⟶ 0 ≤ (⌈xa⌉::int)"
    proof
      assume a1: "fmlookup s' (ad + (STR ''.'' + STR ''balance'')) = Some xa"
      show "0 ≤ (⌈xa⌉::int)"
      proof (cases "ad = sender env")
        case True
        then show ?thesis using s'_def assms(7) Read_ShowL_id a1 v_def by auto
      next
        case False
        then show ?thesis using s'_def assms(7,8) using Read_ShowL_id a1 v_def by (auto
split:if_split_asm)
      qed
    qed
  qed
  moreover have "ReadL_int (accessBalance (accounts st') (STR ''Victim'')) = ReadL_int (accessBalance
(accounts st) (STR ''Victim''))" using **** ac st0_def st'''_def by simp
  ultimately show ?thesis using that assms(5) by simp
qed


lemma secure:
        "address ev1 ≠ (STR ''Victim'') ∧ fmlookup ep1 (STR ''Victim'') = Some (victim, SKIP) ⟶
(∀rv1 st1' bal. frame bal st1 ∧ msel c1 t1 l1 xe1 ep1 ev1 cd1 st1 = Normal (rv1, st1') ⟶ frame bal
st1')"
        "address ev2 ≠ (STR ''Victim'') ∧ fmlookup ep2 (STR ''Victim'') = Some (victim, SKIP) ⟶
(∀rv2 st2' bal. frame bal st2 ∧ ssel t2 l2 xe2 ep2 ev2 cd2 st2 = Normal (rv2, st2') ⟶ frame bal
st2')"
        "address ev5 ≠ (STR ''Victim'') ∧ fmlookup ep5 (STR ''Victim'') = Some (victim, SKIP) ⟶
(∀rv3 st5' bal. frame bal st5 ∧ lexp l5 ep5 ev5 cd5 st5 = Normal (rv3, st5') ⟶ frame bal st5')"
        "address ev4 ≠ (STR ''Victim'') ∧ fmlookup ep4 (STR ''Victim'') = Some (victim, SKIP) ⟶
(∀rv4 st4' bal. frame bal st4 ∧ expr e4 ep4 ev4 cd4 st4 = Normal (rv4, st4') ⟶ frame bal st4')"
        "address lev ≠ (STR ''Victim'') ∧ fmlookup lep (STR ''Victim'') = Some (victim, SKIP) ⟶ (∀ev
cd st st' bal. load lcp lis lxs lep lev0 lcd0 lst0 lev lcd lst = Normal ((ev, cd, st), st') ⟶ (frame
bal lst0 ⟶ frame bal st) ∧ (frame bal lst ⟶ frame bal st') ∧ address lev0 = address ev ∧ sender
lev0 = sender ev ∧ svalue lev0 = svalue ev)"
        "address ev3 ≠ (STR ''Victim'') ∧ fmlookup ep3 (STR ''Victim'') = Some (victim, SKIP) ⟶
(∀rv3 st3' bal. frame bal st3 ∧ rexp l3 ep3 ev3 cd3 st3 = Normal (rv3, st3') ⟶ frame bal st3')"
        "(fmlookup ep6 (STR ''Victim'') = Some (victim, SKIP) ⟶
          (∀st6'. stmt s6 ep6 ev6 cd6 st6 = Normal((), st6') ⟶
            ((address ev6 ≠ (STR ''Victim'') ⟶ (∀bal. frame bal st6 ⟶ frame bal st6'))
           ∧ (address ev6 = (STR ''Victim'') ⟶
```

```
              (∀s val bal x. s6 = transfer
              ∧ INV st6 s (SUMM s + ReadL_int val) bal ∧ POS s
              ∧ fmlookup (denvalue ev6) (STR ''bal'') = Some (Value (TUInt 256), Stackloc x)
              ∧ accessStore x (stack st6) = Some (KValue val)
              ∧ sender ev6 ≠ address ev6
              ⟶ (∃s'. fmlookup (storage st6') (STR ''Victim'') = Some s'
                  ∧ ReadL_int (accessBalance (accounts st6') (STR ''Victim'')) - (SUMM s') ≥ bal ∧ bal
≥ 0 ∧ POS s')) ∧
              (∀s bal x. s6 = comp
              ∧ INV st6 s (SUMM s) bal ∧ POS s
              ∧ fmlookup (denvalue ev6) (STR ''bal'') = Some (Value (TUInt 256), Stackloc x)
              ∧ fmlookup (denvalue ev6) (STR ''balance'') = Some (Storage (STMap TAddr (STValue (TUInt
256))), Storeloc STR ''balance'')
              ∧ accessStore x (stack st6) = Some (KValue (accessStorage (TUInt 256) (sender ev6 + (STR
''.'' + STR ''balance'')) s))
              ∧ sender ev6 ≠ address ev6
              ⟶ (∃s'. fmlookup (storage st6') (STR ''Victim'') = Some s'
                  ∧ ReadL_int (accessBalance (accounts st6') (STR ''Victim'')) - (SUMM s') ≥ bal ∧ bal
≥ 0 ∧ POS s')) ∧
              (∀s bal. s6 = keep
              ∧ INV st6 s (SUMM s) bal ∧ POS s
              ∧ fmlookup (denvalue ev6) (STR ''balance'') = Some (Storage (STMap TAddr (STValue (TUInt
256))), Storeloc STR ''balance'')
              ∧ sender ev6 ≠ address ev6
              ⟶ (∃s'. fmlookup (storage st6') (STR ''Victim'') = Some s'
                  ∧ ReadL_int (accessBalance (accounts st6') (STR ''Victim'')) - (SUMM s') ≥ bal ∧ bal
≥ 0 ∧ POS s'))
))))"
proof (induct rule: msel_ssel_lexp_expr_load_rexp_stmt.induct
[where ?P1.0="λc1 t1 l1 xe1 ep1 ev1 cd1 st1. address ev1 ≠ (STR ''Victim'') ∧ fmlookup ep1 (STR
''Victim'') = Some (victim, SKIP) ⟶ (∀rv1 st1' bal. frame bal st1 ∧ msel c1 t1 l1 xe1 ep1 ev1 cd1
st1 = Normal (rv1, st1') ⟶ frame bal st1')"
   and ?P2.0="λt2 l2 xe2 ep2 ev2 cd2 st2. address ev2 ≠ (STR ''Victim'') ∧ fmlookup ep2 (STR
''Victim'') = Some (victim, SKIP) ⟶ (∀rv2 st2' bal. frame bal st2 ∧ ssel t2 l2 xe2 ep2 ev2 cd2 st2 =
Normal (rv2, st2') ⟶ frame bal st2')"
   and ?P3.0="λl5 ep5 ev5 cd5 st5. address ev5 ≠ (STR ''Victim'') ∧ fmlookup ep5 (STR ''Victim'') =
Some (victim, SKIP) ⟶ (∀rv5 st5' bal. frame bal st5 ∧ lexp l5 ep5 ev5 cd5 st5 = Normal (rv5, st5')
⟶ frame bal st5')"
   and ?P4.0="λe4 ep4 ev4 cd4 st4. address ev4 ≠ (STR ''Victim'') ∧ fmlookup ep4 (STR ''Victim'') =
Some (victim, SKIP) ⟶ (∀rv4 st4' bal. frame bal st4 ∧ expr e4 ep4 ev4 cd4 st4 = Normal (rv4, st4')
⟶ frame bal st4')"
   and ?P5.0="λlcp lis lxs lep lev0 lcd0 lst0 lev lcd lst. address lev ≠ (STR ''Victim'') ∧ fmlookup
lep (STR ''Victim'') = Some (victim, SKIP) ⟶ (∀ev cd st st' bal. load lcp lis lxs lep lev0 lcd0 lst0
lev lcd lst = Normal ((ev, cd, st), st') ⟶ (frame bal lst0 ⟶ frame bal st) ∧ (frame bal lst ⟶
frame bal st') ∧ address lev0 = address ev ∧ sender lev0 = sender ev ∧ svalue lev0 = svalue ev)"
   and ?P6.0="λl3 ep3 ev3 cd3 st3. address ev3 ≠ (STR ''Victim'') ∧ fmlookup ep3 (STR ''Victim'') =
Some (victim, SKIP) ⟶ (∀rv3 st3' bal. frame bal st3 ∧ rexp l3 ep3 ev3 cd3 st3 = Normal (rv3, st3')
⟶ frame bal st3')"
   and ?P7.0="λs6 ep6 ev6 cd6 st6.
(fmlookup ep6 (STR ''Victim'') = Some (victim, SKIP) ⟶
          (∀st6'. stmt s6 ep6 ev6 cd6 st6 = Normal((), st6') ⟶
            ((address ev6 ≠ (STR ''Victim'') ⟶ (∀bal. frame bal st6 ⟶ frame bal st6'))
           ∧ (address ev6 = (STR ''Victim'') ⟶
              (∀s val bal x. s6 = transfer
              ∧ INV st6 s (SUMM s + ReadL_int val) bal ∧ POS s
              ∧ fmlookup (denvalue ev6) (STR ''bal'') = Some (Value (TUInt 256), Stackloc x)
              ∧ accessStore x (stack st6) = Some (KValue val)
              ∧ sender ev6 ≠ address ev6
              ⟶ (∃s'. fmlookup (storage st6') (STR ''Victim'') = Some s'
                  ∧ ReadL_int (accessBalance (accounts st6') (STR ''Victim'')) - (SUMM s') ≥ bal ∧ bal
≥ 0 ∧ POS s')) ∧
              (∀s bal x. s6 = comp
              ∧ INV st6 s (SUMM s) bal ∧ POS s
              ∧ fmlookup (denvalue ev6) (STR ''bal'') = Some (Value (TUInt 256), Stackloc x)
```

```
            ∧ fmlookup (denvalue ev6) (STR ''balance'') = Some (Storage (STMap TAddr (STValue (TUInt
256))), Storeloc STR ''balance'')
            ∧ accessStore x (stack st6) = Some (KValue (accessStorage (TUInt 256) (sender ev6 + (STR
''.'' + STR ''balance'')) s))
            ∧ sender ev6 ≠ address ev6
            ⟶ (∃s'. fmlookup (storage st6') (STR ''Victim'') = Some s'
               ∧ ReadL_int (accessBalance (accounts st6') (STR ''Victim'')) - (SUMM s') ≥ bal ∧ bal
≥ 0 ∧ POS s')) ∧
            (∀s bal. s6 = keep
            ∧ INV st6 s (SUMM s) bal ∧ POS s
            ∧ fmlookup (denvalue ev6) (STR ''balance'') = Some (Storage (STMap TAddr (STValue (TUInt
256))), Storeloc STR ''balance'')
            ∧ sender ev6 ≠ address ev6
            ⟶ (∃s'. fmlookup (storage st6') (STR ''Victim'') = Some s'
               ∧ ReadL_int (accessBalance (accounts st6') (STR ''Victim'')) - (SUMM s') ≥ bal ∧ bal
≥ 0 ∧ POS s'))
))))"
])
  case (1 uu uv uw ux uy uz st)
  then show ?case by simp
next
  case (2 va vb vc vd ve vf vg st)
  then show ?case by simp
next
  case (3 vh al t loc x e_p env cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address env ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ msel vh (MTArray al t) loc [x] e_p env cd st = Normal (rv1, st')"
      moreover from * obtain v4 t4 st4' where **: "expr x e_p env cd st = Normal ((v4, t4), st4')" by
(auto split: result.split_asm)
      moreover from * ** have "frame bal st4'" using 3(1) asm by (auto split:if_split_asm)
      ultimately show "frame bal st'" by (simp split:Stackvalue.split_asm Type.split_asm if_split_asm)
    qed
  qed
next
  case (4 mm al t loc x y ys e_p env cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address env ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ msel mm (MTArray al t) loc (x # y # ys) e_p env cd st = Normal (rv1,
st')"
      moreover from * obtain v4 t4 st'' where **: "expr x e_p env cd st = Normal ((KValue v4, Value
t4), st'')" by (auto split: result.split_asm Stackvalue.split_asm Type.split_asm)
      moreover from * ** have f1: "frame bal st''" using 4(1) asm by (auto split:if_split_asm)
      moreover from * ** have ***: "Valuetypes.less t4 (TUInt 256) v4 ⌊al⌋ = Some (⌈True⌉, TBool)" by
(auto split: result.split_asm Stackvalue.split_asm Type.split_asm if_split_asm)
      moreover from * ** *** obtain vb st''' where ****: "(applyf (λst. if mm then memory st else cd)
st'') = Normal (vb, st''')"
        and f2: "frame bal st'''" using f1 by (simp split:Stackvalue.split_asm Type.split_asm
if_split_asm)
      moreover from * ** *** **** obtain l1 where *****: "accessStore (hash loc v4) vb = Some
(MPointer l1)"
        by (simp split: Type.split_asm if_split_asm option.split_asm Memoryvalue.split_asm)
      moreover from * ** *** **** ***** obtain l1' st'''' where ******: "msel mm t l1 (y # ys) e_p env
cd st''' = Normal (l1', st'''')"
        by (simp split: Type.split_asm if_split_asm option.split_asm Memoryvalue.split_asm)
      ultimately have "st' = st''''" by simp
```

```
      moreover have x1: "∀rv1' st1' bal. (frame bal st''') ∧ local.msel mm t l1 (y # ys) e_p env cd
st''' = Normal (rv1', st1') ⟶ frame bal st1'" using 4(2)[OF ** _ _ _ *** _ *****] **** asm apply
safe by auto
      ultimately show "frame bal st'" using f2 ****** by blast
    qed
  qed
next
  case (5 tp loc vi vj vk st)
  then show ?case by simp
next
  case (6 vl vm vn vo vp vq vr st)
  then show ?case by simp
next
  case (7 al t loc x xs e_p env cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address env ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ ssel (STArray al t) loc (x # xs) e_p env cd st = Normal (rv1, st')"
      moreover from * obtain v4 t4 st4' where **: "expr x e_p env cd st = Normal ((KValue v4, Value
t4), st4')" by (auto split: result.split_asm Stackvalue.split_asm Type.split_asm)
      moreover from * ** have f1: "frame bal st4'" using 7(1) asm by (auto split:if_split_asm)
      moreover from * ** have ***: "Valuetypes.less t4 (TUInt 256) v4 ⌊al⌋ = Some (⌈True⌉, TBool)" by
(auto split: result.split_asm Stackvalue.split_asm Type.split_asm if_split_asm)
      moreover from * ** *** obtain l1' st''' where ****: "ssel t (hash loc v4) xs e_p env
cd st4' = Normal (l1', st''')" by (simp split: Type.split_asm if_split_asm option.split_asm
Memoryvalue.split_asm)
      ultimately have "st' = st'''" by simp
      moreover have "∀rv' st2' bal. (frame bal  st4') ∧ ssel t (hash loc v4) xs e_p env cd st4' =
Normal (rv', st2') ⟶ frame bal st2'" using 7(2)[OF ** _ _ _ ***] asm apply safe by auto
      ultimately show "frame bal st'" using f1 **** by blast
    qed
  qed
next
  case (8 vs t loc x xs e_p env cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address env ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ ssel (STMap vs t) loc (x # xs) e_p env cd st = Normal (rv1, st')"
      moreover from * obtain v4 t4 st4' where **: "expr x e_p env cd st = Normal ((KValue v4, t4),
st4')" by (auto split: result.split_asm Stackvalue.split_asm)
      moreover from * ** have ***: "frame bal st4'" using 8(1) asm by (auto split:if_split_asm)
      moreover from * ** *** obtain l1' st''' where ****:"ssel t (hash loc v4) xs e_p env cd st4' =
Normal (l1', st''')" by simp
      moreover from *** **** have "frame bal st'''" using 8(2)[OF **,of "KValue v4" t4 v4] asm by
blast
      ultimately show "frame bal st'" by (simp split:Stackvalue.split_asm)
    qed
  qed
next
  case (9 i vt e vu st)
  then show ?case by (auto split:option.split_asm result.split_asm Denvalue.split_asm)
next
  case (10 i r e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
```

```
fix rv1 and st' and bal
assume *: "frame bal st ∧ lexp (Ref i r) e_p e cd st = Normal (rv1, st')"
show "frame bal st'"
proof (cases "fmlookup (denvalue e) i")
  case None
  with * show ?thesis by simp
next
  case (Some a)
  then show ?thesis
  proof (cases a)
    case (Pair tp b)
    then show ?thesis
    proof (cases b)
      case (Stackloc l')
      then show ?thesis
      proof (cases "accessStore l' (stack st)")
        case None
        with * show ?thesis using Some Pair Stackloc by simp
      next
        case s2: (Some k)
        then show ?thesis
        proof (cases k)
          case (KValue x1)
          with * show ?thesis using Some Pair Stackloc s2 by simp
        next
          case (KCDptr x2)
          with * show ?thesis using Some Pair Stackloc s2 by simp
        next
          case (KMemptr l'')
          then show ?thesis
          proof (cases tp)
            case (Value x1)
            with * show ?thesis using Some Pair Stackloc s2 KMemptr by simp
          next
            case (Calldata x2)
            with * show ?thesis using Some Pair Stackloc s2 KMemptr by simp
          next
            case (Memory x3)
            with * Some Pair Stackloc KMemptr s2 obtain l1' t1' where "msel True x3 l'' r e_p e
cd st = Normal ((l1', t1'), st')" by (auto split: result.split_asm)
            with * 10(1)[OF Some Pair Stackloc _ _ KMemptr, of "Some k" st x3] show ?thesis us-
ing s2 Memory asm by auto
          next
            case (Storage x4)
            with * show ?thesis using Some Pair Stackloc s2 KMemptr by simp
          qed
        next
          case (KStoptr l'')
          then show ?thesis
          proof (cases tp)
            case (Value x1)
            with * show ?thesis using Some Pair Stackloc s2 KStoptr by simp
          next
            case (Calldata x2)
            with * show ?thesis using Some Pair Stackloc s2 KStoptr by simp
          next
            case (Memory x3)
            with * show ?thesis using Some Pair Stackloc s2 KStoptr by simp
          next
            case (Storage x4)
            with * Some Pair Stackloc KStoptr s2 obtain l1' t1' where "ssel x4 l'' r e_p e cd st
= Normal ((l1', t1'), st')" by (auto split: result.split_asm)
            with * 10(2)[OF Some Pair Stackloc _ _ KStoptr, of "Some k" st x4] show ?thesis us-
ing s2 Storage asm by auto
```

```
              qed
            qed
          qed
        next
          case (Storeloc l'')
          then show ?thesis
          proof (cases tp)
            case (Value x1)
            with * show ?thesis using Some Pair Storeloc by simp
          next
            case (Calldata x2)
            with * show ?thesis using Some Pair Storeloc by simp
          next
            case (Memory x3)
            with * show ?thesis using Some Pair Storeloc by simp
          next
            case (Storage x4)
            with * Some Pair Storeloc obtain l1' t1' where "ssel x4 l'' r e_p e cd st = Normal ((l1',
t1'), st')" by (auto split: result.split_asm)
            with * 10(3)[OF Some Pair Storeloc Storage] asm show ?thesis by auto
          qed
        qed
      qed
    qed
  qed
qed
next
  case (11 b x e_p e cd st)
  then show ?case by (simp add:frame_def)
next
  case (12 b x e_p e cd st)
  then show ?case by (simp add:frame_def)
next
  case (13 ad e_p e cd st)
  then show ?case by (simp add:frame_def)
next
  case (14 ad e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (BALANCE ad) e_p e cd st = Normal (rv1, st')"
      moreover from * obtain adv st'' where **:"expr ad e_p e cd (st⦇gas:=gas st - (costs_e
(BALANCE ad) e_p e cd st)⦈) = Normal ((KValue adv, Value TAddr),st'')" by (auto split:if_split_asm
result.split_asm Stackvalue.split_asm Types.split_asm Type.split_asm)
      with * ** have "frame bal st''" using 14(1) asm by (auto simp add:frame_def split:if_split_asm)
      moreover from * ** have "st' = st''" by (simp split:if_split_asm)
      ultimately show "frame bal st'" by simp
    qed
  qed
next
  case (15 e_p e cd st)
  then show ?case by (simp add:frame_def)
next
  case (16 e_p e cd st)
  then show ?case by (simp add:frame_def)
next
  case (17 e_p e cd st)
  then show ?case by (simp add:frame_def)
next
  case (18 e_p e cd st)
  then show ?case by (simp add:frame_def)
```

```
next
  case (19 e_p e cd st)
  then show ?case by (simp add:frame_def)
next
  case (20 x e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (NOT x) e_p e cd st = Normal (rv1, st')"
      then have f1: "frame bal (st⦇gas:=gas st - (costs_e (NOT x) e_p e cd st)⦈)" by (simp
add:frame_def)
      moreover from * obtain v t st'' where **: "expr x e_p e cd (st⦇gas:=gas st - (costs_e (NOT x) e_p
e cd st)⦈) = Normal ((KValue v, Value t), st'')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** have ***: "frame bal st''" using 20(1) asm by (auto simp add:frame_def
split:if_split_asm)
      show "frame bal st'"
      proof (cases "v = ShowL_bool True")
        case True
        with * ** *** obtain x st''' where "expr FALSE e_p e cd st'' = Normal (x, st''')"
          and "frame bal st'''" by (auto simp add:frame_def split:if_split_asm)
        with * ** *** True show ?thesis by (auto split: if_split_asm)
      next
        case False
        with * ** *** obtain x st''' where "expr TRUE e_p e cd st'' = Normal (x, st''')"
          and "frame bal st'''" by (auto simp add:frame_def split:if_split_asm)
        with * ** *** False show ?thesis by (auto split: if_split_asm)
      qed
    qed
  qed
next
  case (21 e1 e2 e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (PLUS e1 e2) e_p e cd st = Normal (rv1, st')"
      moreover from * obtain v1 t1 st'' where **: "expr e1 e_p e cd (st⦇gas:=gas st - (costs_e (PLUS
e1 e2) e_p e cd st)⦈) = Normal ((KValue v1, Value t1), st'')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** have ***: "frame bal st''" using 21(1) asm by (auto simp add:frame_def
split:if_split_asm)
      moreover from * ** *** obtain v2 t2 st''' where ****: "expr e2 e_p e cd st'' = Normal ((KValue
v2, Value t2), st''')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** *** **** have "frame bal st'''" using 21(2)[OF _ **] asm by (auto
split:if_split_asm)
      moreover from * ** **** obtain v t where "add t1 t2 v1 v2 = Some (v, t)" by (auto
split:if_split_asm option.split_asm)
      ultimately show "frame bal st'" by (auto split:if_split_asm)
    qed
  qed
next
  case (22 e1 e2 e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
```

```
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (MINUS e1 e2) e_p e cd st = Normal (rv1, st')"
      moreover from * obtain v1 t1 st'' where **: "expr e1 e_p e cd (st(|gas:=gas st - (costs_e (MINUS
e1 e2) e_p e cd st)|)) = Normal ((KValue v1, Value t1), st'')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** have ***: "frame bal st''" using 22(1) asm by (auto simp add:frame_def
split:if_split_asm)
      moreover from * ** *** obtain v2 t2 st''' where ****: "expr e2 e_p e cd st'' = Normal ((KValue
v2, Value t2), st''')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** *** **** have "frame bal st'''" using 22(2)[OF _ **] asm by (auto
split:if_split_asm)
      moreover from * ** **** obtain v t where "sub t1 t2 v1 v2 = Some (v, t)" by (auto
split:if_split_asm option.split_asm)
      ultimately show "frame bal st'" by (auto split:if_split_asm)
    qed
  qed
next
  case (23 e1 e2 e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (LESS e1 e2) e_p e cd st = Normal (rv1, st')"
      moreover from * obtain v1 t1 st'' where **: "expr e1 e_p e cd (st(|gas:=gas st - (costs_e (LESS
e1 e2) e_p e cd st)|)) = Normal ((KValue v1, Value t1), st'')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** have ***: "frame bal st''" using 23(1) asm by (auto simp add:frame_def
split:if_split_asm)
      moreover from * ** *** obtain v2 t2 st''' where ****: "expr e2 e_p e cd st'' = Normal ((KValue
v2, Value t2), st''')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** *** **** have "frame bal st'''" using 23(2)[OF _ **] asm by (auto
split:if_split_asm)
      moreover from * ** **** obtain v t where "Valuetypes.less t1 t2 v1 v2 = Some (v, t)" by (auto
split:if_split_asm option.split_asm)
      ultimately show "frame bal st'" by (auto split:if_split_asm)
    qed
  qed
next
  case (24 e1 e2 e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (EQUAL e1 e2) e_p e cd st = Normal (rv1, st')"
      moreover from * obtain v1 t1 st'' where **: "expr e1 e_p e cd (st(|gas:=gas st - (costs_e (EQUAL
e1 e2) e_p e cd st)|)) = Normal ((KValue v1, Value t1), st'')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** have ***: "frame bal st''" using 24(1) asm by (auto simp add:frame_def
split:if_split_asm)
      moreover from * ** *** obtain v2 t2 st''' where ****: "expr e2 e_p e cd st'' = Normal ((KValue
```

```
v2, Value t2), st''')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** *** **** have "frame bal st'''" using 24(2)[OF _ **] asm by (auto
split:if_split_asm)
      moreover from * ** **** obtain v t where "Valuetypes.equal t1 t2 v1 v2 = Some (v, t)" by (auto
split:if_split_asm option.split_asm)
      ultimately show "frame bal st'" by (auto split:if_split_asm)
    qed
  qed
next
  case (25 e1 e2 e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (AND e1 e2) e_p e cd st = Normal (rv1, st')"
      moreover from * obtain v1 t1 st'' where **: "expr e1 e_p e cd (st⦇gas:=gas st - (costs_e (AND e1
e2) e_p e cd st)⦈) = Normal ((KValue v1, Value t1), st'')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** have ***: "frame bal st''" using 25(1) asm by (auto simp add:frame_def
split:if_split_asm)
      moreover from * ** *** obtain v2 t2 st''' where ****: "expr e2 e_p e cd st'' = Normal ((KValue
v2, Value t2), st''')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** *** **** have "frame bal st'''" using 25(2)[OF _ **] asm by (auto
split:if_split_asm)
      moreover from * ** **** obtain v t where "Valuetypes.vtand t1 t2 v1 v2 = Some (v, t)" by (auto
split:if_split_asm option.split_asm)
      ultimately show "frame bal st'" by (auto split:if_split_asm)
    qed
  qed
next
  case (26 e1 e2 e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (OR e1 e2) e_p e cd st = Normal (rv1, st')"
      moreover from * obtain v1 t1 st'' where **: "expr e1 e_p e cd (st⦇gas:=gas st - (costs_e (OR e1
e2) e_p e cd st)⦈) = Normal ((KValue v1, Value t1), st'')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** have ***: "frame bal st''" using 26(1) asm by (auto simp add:frame_def
split:if_split_asm)
      moreover from * ** *** obtain v2 t2 st''' where ****: "expr e2 e_p e cd st'' = Normal ((KValue
v2, Value t2), st''')"
        by (auto split:if_split_asm result.split_asm prod.split_asm Stackvalue.split_asm
Type.split_asm)
      moreover from * ** *** **** have "frame bal st'''" using 26(2)[OF _ **] asm by (auto
split:if_split_asm)
      moreover from * ** **** obtain v t where "Valuetypes.vtor t1 t2 v1 v2 = Some (v, t)" by (auto
split:if_split_asm option.split_asm)
      ultimately show "frame bal st'" by (auto split:if_split_asm)
    qed
  qed
next
  case (27 i e_p e cd st)
```

```
  show ?case using 27(1)[of "()" "st⦇gas:=gas st - (costs_e (LVAL i) e_p e cd st)⦈"] apply safe by
(auto simp add:frame_def split:if_split_asm)
next
  case (28 i xe e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (CALL i xe) e_p e cd st = Normal (rv1, st')"
      moreover from * have a1: "(applyf (costs_e (CALL i xe) e_p e cd) ≫= (λg. assert Gas (λst. gas st
≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st = Normal ((), st⦇gas := gas st - costs_e (CALL i xe)
e_p e cd st⦈)" by auto
      moreover from * obtain ct bla where **: "fmlookup e_p (address e) = Some (ct, bla)"
        by (auto split:if_split_asm option.split_asm)
      moreover from * ** obtain fp f x where ***: "fmlookup ct i = Some (Method (fp, f, Some x))"
        by (auto split:if_split_asm option.split_asm Member.split_asm)
      moreover define e' where "e' = ffold_init ct (emptyEnv (address e) (sender e) (svalue e)) (fmdom
ct)"
      moreover from * ** *** obtain e'' cd' st'' st''' where ****: "load False fp xe e_p e' emptyStore
(st⦇gas:=gas st - (costs_e (CALL i xe) e_p e cd st), stack:=emptyStore⦈) e cd (st⦇gas:=gas st - (costs_e
(CALL i xe) e_p e cd st)⦈) = Normal ((e'', cd', st''), st''')"
        using e'_def by (auto split:if_split_asm result.split_asm)
      moreover from * **** have f1: "frame bal st''" and ad: "address e' = address e''"
        using asm 28(1)[OF _ ** _ *** _ _ _ _ e'_def, of _ "st⦇gas := gas st - costs_e (CALL i xe) e_p
e cd st⦈" bla "(fp, f, Some x)" fp "(f, Some x)" f "Some x" x "st⦇gas := gas st - costs_e (CALL i xe)
e_p e cd st, stack := emptyStore⦈" "st⦇gas := gas st - costs_e (CALL i xe) e_p e cd st⦈"] by (auto simp
add:frame_def split:if_split_asm result.split_asm)
      moreover from e'_def have ad2: "address e = address e'" using ffold_init_ad_same[of ct
"(emptyEnv (address e) (sender e) (svalue e))" "(fmdom ct)" e'] by simp
      moreover from * ** *** **** e'_def obtain st'''' where *****: "stmt f e_p e'' cd' st'' = Normal
((), st'''')" by (auto split:if_split_asm result.split_asm)
      moreover from f1 ad ad2 asm ***** have f2:"frame bal st''''"
        using 28(2)[OF a1 ** _ *** _ _ _ _ e'_def _ ****, of bla "(fp, f, Some x)" "(f, Some x)" f
"Some x" x e'' "(cd', st'')" "cd'" "st''" st''' st''' "()" st''] by (simp add:frame_def)
      moreover from * ** *** **** ***** f1 f2 e'_def obtain rv st''''' where ******: "expr x e_p e''
cd' st'''' = Normal (rv, st''''')" by (auto split:if_split_asm result.split_asm)
      ultimately have "st' = st'''''⦇stack:=stack st''', memory := memory st'''⦈" apply safe by auto
      moreover from f1 f2 ad ad2 asm a1 ***** ****** have "∀rv4 st4' bal.
        frame bal st'''' ∧
        local.expr x e_p e'' cd' st'''' = Normal (rv4, st4') ⟶
        frame bal st4'" using e'_def asm 28(3)[OF a1 ** _ *** _ _ _ _ e'_def _ **** _ _ _ _ *****, of
bla "(fp, f, Some x)" " (f, Some x)" "Some x" x "(cd', st'')" st'' st''' st''' "()"] apply safe by auto
      with ****** f2 have "frame bal st'''''" by blast
      ultimately show "frame bal st'" by (simp add:frame_def)
    qed
  qed
next
  case (29 ad i xe val e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv1 and st' and bal
      assume *: "frame bal st ∧ expr (ECALL ad i xe val) e_p e cd st = Normal (rv1, st')"
      moreover from * have a1: "(applyf (costs_e (ECALL ad i xe val) e_p e cd) ≫= (λg. assert Gas
(λst. gas st ≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st = Normal ((), st⦇gas := gas st - costs_e
(ECALL ad i xe val) e_p e cd st⦈)" by auto
      moreover from * obtain adv st'' where **: "expr ad e_p e cd (st⦇gas:=gas st - (costs_e (ECALL ad
i xe val) e_p e cd st)⦈) = Normal ((KValue adv, Value TAddr), st'')"
        by (auto split:if_split_asm result.split_asm Stackvalue.split_asm Type.split_asm
Types.split_asm)
```

```
    moreover from * ** have f1: "frame bal st''"using asm 29(1) by (auto simp add:frame_def split:if_split_asm)
    moreover from * ** obtain ct bla where ***: "fmlookup ep adv = Some (ct, bla)"
      by (auto split:if_split_asm option.split_asm)
    moreover from * ** *** obtain fp f x where ****: "fmlookup ct i = Some (Method (fp, f, Some x))"
      by (auto split:if_split_asm option.split_asm Member.split_asm)
    moreover from * ** *** **** obtain v t st''' where *****: "expr val ep e cd st'' = Normal ((KValue v, Value t), st''')" by (auto split:if_split_asm result.split_asm Stackvalue.split_asm Type.split_asm)
    moreover from f1 ***** asm have f2: "frame bal st'''" and f3: "frame bal (st'''⦇stack := emptyStore, memory := emptyStore⦈)" using asm 29(2)[OF a1 ** _ _ _ _ *** _ ****] by (auto simp add:frame_def)
    moreover define e' where "e' = ffold_init ct (emptyEnv adv (address e) v) (fmdom ct)"
    moreover from * ** *** **** ***** obtain e'' cd' st'''' st''''' where *******: "load True fp xe ep e' emptyStore (st'''⦇stack:=emptyStore, memory:=emptyStore⦈) e cd st''' = Normal ((e'', cd', st''''), st''''')"
      using e'_def by (auto split:if_split_asm result.split_asm option.split_asm)
    moreover have "(∀ev cda st st' bal.
      local.load True fp xe ep e' emptyStore (st'''⦇stack := emptyStore, memory := emptyStore⦈) e cd st''' = Normal ((ev, cda, st), st') ⟶
      (frame bal (st'''⦇stack := emptyStore, memory := emptyStore⦈) ⟶ frame bal st) ∧
      (frame bal st''' ⟶ frame bal st') ∧ address e' = address ev ∧ sender e' = sender ev ∧ svalue e' = svalue ev)"
      using 29(3)[OF a1 ** _ _ _ _ *** _ **** _ _ _ _ ***** _ _ _ e'_def, of "KValue adv" "Value TAddr" TAddr bla "(fp, f, Some x)" fp "(f, Some x)" f "Some x" x "KValue v" "Value t" t "st'''⦇stack := emptyStore, memory := emptyStore⦈" st'''] asm ******* by simp
    then have "frame bal st'''' ∧ frame bal st''''' ∧ address e' = address e''" using ******* f2 f3 by blast
    then have f4: "frame bal st''''" and ad1: "address e' = address e''" by auto
    moreover from * ** *** **** ***** ******* e'_def obtain acc where ******: "Accounts.transfer (address e) adv v (accounts st'''') = Some acc" by (auto split:if_split_asm option.split_asm)
    then have ******: "Accounts.transfer (address e) adv v (accounts st'''') = Some acc" by (auto split:if_split_asm option.split_asm)
    moreover from f4 have f5: "frame bal (st''''⦇accounts := acc⦈)" using transfer_frame[OF ******] asm by simp
    moreover from e'_def have ad2: "adv = address e'" using ffold_init_ad_same[of ct "(emptyEnv adv (address e) v)" "(fmdom ct)" e'] by simp
    moreover from * ** *** **** ***** ****** ******* ****** obtain st'''''' where ********: "stmt f ep e'' cd' (st''''⦇accounts := acc⦈) = Normal ((), st'''''')"
      using e'_def by (auto simp del: transfer.simps split:if_split_asm result.split_asm)
    moreover have "adv ≠ STR ''Victim''"
    proof (rule ccontr)
      assume "¬ adv ≠ STR ''Victim''"
      with asm ** *** **** show False using victim_def fmap_of_list_SomeD[of "[(STR ''balance'', Var (STMap TAddr (STValue (TUInt 256)))), (STR ''deposit'', Method ([], deposit, None)), (STR ''withdraw'', Method ([], keep, None))]"] by auto
    qed
    with ad1 ad2 have ad: "address e'' ≠ STR ''Victim'' ∧ fmlookup ep (STR ''Victim'') = Some (victim, SKIP)" using asm by simp
    then have "(∀bal. frame bal (st''''⦇accounts := acc⦈) ⟶ frame bal st'''''')" using 29(4)[OF a1 ** _ _ _ _ *** _ **** _ _ _ _ ***** _ _ _ _ _ ******* _ _ _ ******, of "KValue adv" "Value TAddr" TAddr bla "(fp, f, Some x)" "(f, Some x)" f "Some x" x "KValue v" "Value t" t e'' "(cd', st'''')" cd'  st''''' st''''' "()" "st''''⦇accounts := acc⦈"] ******** e'_def by auto
    then have f4: "frame bal st''''''" using f5 ******** by auto
    moreover from * ** *** **** ***** ****** ******* ******** obtain rv st''''''' where *********: "expr x ep e'' cd' st'''''' = Normal (rv, st''''''')"
      using e'_def by (auto split:if_split_asm result.split_asm)
    ultimately have "st' = st'''''''⦇stack:=stack st''''', memory := memory st'''''⦈" apply safe by auto
    moreover from ad have "∀rv4 st4' bal.
      frame bal st'''''' ∧
      local.expr x ep e'' cd' st'''''' = Normal (rv4, st4') ⟶
      frame bal st4'"
```

```
        using e'_def 29(5)[OF a1 ** _ _ _ _ *** _ **** _ _ _ _ ***** _ _ _ _ _ ******* _ _ _ ****** _
********, of "KValue adv" "Value TAddr" TAddr bla "(fp, f, Some x)"] by auto
      then have"frame bal st''''''''" using f4 ********* by blast
      ultimately show "frame bal st'" by (simp add:frame_def)
    qed
  qed
next
  case (30 cp i_p t_p pl e el e_p e_v' cd' st' e_v cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e_v ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF allI[OF allI[OF impI]]]]])
      fix ev and cda and sta and st'a and bal
      assume *: "local.load cp ((i_p, t_p) # pl) (e # el) e_p e_v' cd' st' e_v cd st = Normal ((ev, cda,
sta), st'a)"
      moreover from * obtain v t st'' where **: "expr e e_p e_v cd st = Normal ((v,t),st'')" by (auto
split: result.split_asm)
      moreover from * ** obtain cd'' e_v'' st''' where ***: "decl i_p t_p (Some (v,t)) cp cd (memory
st'') cd' e_v' st' = Normal ((cd'', e_v''),st''')" by (auto split: result.split_asm)
      moreover from *** have ad: "address e_v' = address e_v'' ∧ sender e_v' = sender e_v'' ∧ svalue e_v'
= svalue e_v''" using decl_gas_address by simp
      moreover from * ** *** obtain ev' cda' sta' st'a' where ****: "local.load cp pl el e_p e_v''
cd'' st''' e_v cd st''= Normal ((ev', cda', sta'), st'a')" by (auto split: result.split_asm)
      ultimately have "ev = ev'" and "sta = sta'" and "st'a = st'a'" by simp+

      from **** asm have IH: "(frame bal st''' ⟶ frame bal sta') ∧
        (frame bal st'' ⟶ frame bal st'a') ∧
        address e_v'' = address ev' ∧ sender e_v'' = sender ev' ∧ svalue e_v'' = svalue ev'" us-
ing 30(2)[OF ** _ _ _ ***, of st'' "()" cd'' e_v'' st''' st''' "()" st''] apply safe by (auto simp
add:frame_def)
      show "(frame bal st' ⟶ frame bal sta) ∧ (frame bal st ⟶ frame bal st'a) ∧ address e_v' =
address ev ∧ sender e_v' = sender ev ∧ svalue e_v' = svalue ev"
      proof (rule conj3)
        show "frame bal st' ⟶ frame bal sta"
        proof
          assume "frame bal st'"
          with * ** *** have "frame bal st'''" using decl_frame by simp
          with IH have "frame bal sta'" by simp
          with `sta = sta'` show "frame bal sta" by simp
        qed
      next
        show "frame bal st ⟶ frame bal st'a"
        proof
          assume "frame bal st"
          with ** have "frame bal st''" using 30(1) asm by simp
          with IH have "frame bal st'a'" by simp
          with `st'a = st'a'` show "frame bal st'a" by simp
        qed
      next
        from ad IH show "address e_v' = address ev ∧ sender e_v' = sender ev ∧ svalue e_v' = svalue ev"
using `ev = ev'` by simp
      qed
    qed
  qed
next
  case (31 vv vw vx vy vz wa wb wc wd st)
  then show ?case by simp
next
  case (32 we wf wg wh wi wj wk wl wm st)
  then show ?case by simp
next
  case (33 wn wo e_v' cd' st' e_v cd st)
  then show ?case by simp
```

```
next
  case (34 i e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv3 and st3' and bal
      assume *: "frame bal st ∧ local.rexp (L.Id i) e_p e cd st = Normal (rv3, st3')"
      show "frame bal st3'"
      proof (cases "fmlookup (denvalue e) i")
        case None
        with * show ?thesis by simp
      next
        case (Some a)
        then show ?thesis
        proof (cases a)
          case (Pair tp b)
          then show ?thesis
          proof (cases b)
            case (Stackloc l)
            then show ?thesis
            proof (cases "accessStore l (stack st)")
              case None
              with * Some Pair Stackloc show ?thesis by (auto split: Type.split_asm STypes.split_asm)
            next
              case s2: (Some a)
              with * Some Pair Stackloc s2 show ?thesis by (auto split: Type.split_asm
STypes.split_asm Stackvalue.split_asm)
            qed
          next
            case (Storeloc x2)
            with * Some Pair Storeloc show ?thesis by (auto split: Type.split_asm STypes.split_asm
option.split_asm)
          qed
        qed
      qed
    qed
  qed
next
  case (35 i r e_p e cd st)
  show ?case (is "_ ⟶ ?RHS")
  proof
    assume asm: "address e ≠ (STR ''Victim'') ∧ fmlookup e_p (STR ''Victim'') = Some (victim, SKIP)"
    show ?RHS
    proof (rule allI[OF allI[OF allI[OF impI]]])
      fix rv3 and st3' and bal
      assume *: "frame bal st ∧ local.rexp (L.Ref i r) e_p e cd st = Normal (rv3, st3')"
      show "frame bal st3'"
      proof (cases "fmlookup (denvalue e) i")
        case None
        with * show ?thesis by simp
      next
        case (Some a)
        then show ?thesis
        proof (cases a)
          case (Pair tp b)
          then show ?thesis
          proof (cases b)
            case (Stackloc l')
            then show ?thesis
            proof (cases "accessStore l' (stack st)")
              case None
              with * Some Pair Stackloc show ?thesis by simp
```

```
      next
        case s2: (Some a)
        then show ?thesis
        proof (cases a)
          case (KValue x1)
          with * Some Pair Stackloc s2 show ?thesis by simp
        next
          case (KCDptr l'')
          then show ?thesis
          proof (cases tp)
            case (Value x1)
            with * Some Pair Stackloc s2 KCDptr show ?thesis by simp
          next
            case (Calldata t)
            with * Some Pair Stackloc s2 KCDptr obtain l''' t' st' where **: "msel False t l'' r
e_p e cd st = Normal ((l''',t'), st')" by (auto split: Type.split_asm STypes.split_asm result.split_asm)
            then have "∀rv1 st1' bal.
            frame bal st ∧
            local.msel False t l'' r e_p e cd st = Normal (rv1, st1') ⟶
            frame bal st1'" using asm 35(1)[OF Some Pair Stackloc _ s2 KCDptr Calldata] by auto
            with * ** have f2: "frame bal st'" by blast
            then show ?thesis
            proof (cases t')
              case (MTArray x t'')
              then show ?thesis
              proof (cases "accessStore l''' cd")
                case None
                with * ** Some Pair Stackloc s2 KCDptr Calldata MTArray show ?thesis by simp
              next
                case s3: (Some a)
                then show ?thesis
                proof (cases a)
                  case (MValue x1)
                  with * ** Some Pair Stackloc s2 KCDptr Calldata MTArray s3 show ?thesis by
simp
                next
                  case (MPointer x2)
                  with * ** f2 Some Pair Stackloc s2 KCDptr Calldata MTArray s3 show ?thesis by
simp
                qed
              qed
            next
              case (MTValue t''')
              then show ?thesis
              proof (cases "accessStore l''' cd")
                case None
                with * ** Some Pair Stackloc s2 KCDptr Calldata MTValue show ?thesis by simp
              next
                case s3: (Some a)
                then show ?thesis
                proof (cases a)
                  case (MValue x1)
                  with * ** f2 Some Pair Stackloc s2 KCDptr Calldata MTValue s3 show ?thesis by
simp
                next
                  case (MPointer x2)
                  with * ** Some Pair Stackloc s2 KCDptr Calldata MTValue s3 show ?thesis by
simp
                qed
              qed
            qed
          next
            case (Memory x3)
            with * Some Pair Stackloc s2 KCDptr show ?thesis by simp
```

```
          next
            case (Storage x4)
            with * Some Pair Stackloc s2 KCDptr show ?thesis by simp
          qed
        next
          case (KMemptr l'')
          then show ?thesis
          proof (cases tp)
            case (Value x1)
            with * Some Pair Stackloc s2 KMemptr show ?thesis by simp
          next
            case (Calldata x2)
            with * Some Pair Stackloc s2 KMemptr show ?thesis by simp
          next
            case (Memory t)
            with * Some Pair Stackloc s2 KMemptr obtain l''' t' st' where **: "msel True t l'' r
e_p e cd st = Normal ((l''',t'), st')" by (auto split: Type.split_asm STypes.split_asm result.split_asm)
            then have "∀rv1 st1' bal. frame bal st ∧
            local.msel True t l'' r e_p e cd st = Normal (rv1, st1') ⟶
            frame bal st1'" using asm 35(2)[OF Some Pair Stackloc _ s2 KMemptr Memory, of st] by
auto
            with * ** have f2: "frame bal st'" by blast
            then show ?thesis
            proof (cases t')
              case (MTArray x11 x12)
              then show ?thesis
              proof (cases "accessStore l''' (memory st')")
                case None
                with * ** Some Pair Stackloc s2 KMemptr Memory MTArray show ?thesis by simp
              next
                case s3: (Some a)
                then show ?thesis
                proof (cases a)
                  case (MValue x1)
                  with * ** Some Pair Stackloc s2 KMemptr Memory MTArray s3 show ?thesis by simp
                next
                  case (MPointer x2)
                  with * ** f2 Some Pair Stackloc s2 KMemptr Memory MTArray s3 show ?thesis by
simp
                qed
              qed
            next
              case (MTValue x2)
              then show ?thesis
              proof (cases "accessStore l''' (memory st')")
                case None
                with * ** Some Pair Stackloc s2 KMemptr Memory MTValue show ?thesis by simp
              next
                case s3: (Some a)
                then show ?thesis
                proof (cases a)
                  case (MValue x1)
                  with * ** f2 Some Pair Stackloc s2 KMemptr Memory MTValue s3 show ?thesis by
simp
                next
                  case (MPointer x2)
                  with * ** Some Pair Stackloc s2 KMemptr Memory MTValue s3 show ?thesis by simp
                qed
              qed
            qed
          next
            case (Storage x4)
            with * Some Pair Stackloc s2 KMemptr show ?thesis by simp
          qed
```

```
            next
              case (KStoptr l'')
              then show ?thesis
              proof (cases tp)
                case (Value x1)
                with * Some Pair Stackloc s2 KStoptr show ?thesis by simp
              next
                case (Calldata x2)
                with * Some Pair Stackloc s2 KStoptr show ?thesis by simp
              next
                case (Memory x3)
                with * Some Pair Stackloc s2 KStoptr show ?thesis by simp
              next
                case (Storage t)
                with * Some Pair Stackloc s2 KStoptr obtain l''' t' st' where **: "ssel t l'' r e_p e
cd st = Normal ((l''',t'), st')" by (auto split: Type.split_asm STypes.split_asm result.split_asm)
                then have "∀rv2 st2' bal.
                frame bal st ∧
                local.ssel t l'' r e_p e cd st = Normal (rv2, st2') ⟶
                frame bal st2'" using asm 35(3)[OF Some Pair Stackloc _ s2 KStoptr Storage, of st] by
auto
                with * ** have "frame bal st'" by blast
                with * ** Some Pair Stackloc s2 KStoptr Storage show ?thesis by (simp split:
STypes.split_asm option.split_asm)
              qed
            qed
          qed
        next
          case (Storeloc l')
          then show ?thesis
          proof (cases tp)
            case (Value x1)
            with * Some Pair Storeloc show ?thesis by simp
          next
            case (Calldata x2)
            with * Some Pair Storeloc show ?thesis by simp
          next
            case (Memory x3)
            with * Some Pair Storeloc show ?thesis by simp
          next
            case (Storage t)
            with * Some Pair Storeloc obtain l'' t' st' where **: "ssel t l' r e_p e cd st = Normal
((l'',t'), st')" by (auto split: result.split_asm)
            then have "∀rv2 st2' bal.
            frame bal st ∧
            local.ssel t l' r e_p e cd st = Normal (rv2, st2') ⟶
            frame bal st2'" using asm 35(4)[OF Some Pair Storeloc Storage] by auto
            with * ** have "frame bal st'" by blast
            with * ** Some Pair Storeloc Storage show ?thesis by (simp split: STypes.split_asm
option.split_asm)
          qed
        qed
      qed
    qed
  qed
qed
next
  case (36 e_p e cd st)
  show ?case (is "?LHS ⟶ ?RHS")
  proof
    assume *: "fmlookup e_p STR ''Victim'' = Some (victim, SKIP)"
    show ?RHS (is "∀st6'. ?RHS st6'")
    proof
      fix st6'
```

```
        show "?RHS st6'" (is "?LHS ⟶ ?RHS")
        proof
          assume t0: "stmt SKIP e_p e cd st = Normal ((), st6')"
          show ?RHS (is "?LHS ∧ ?RHS")
          proof
            show "?LHS"
            proof
              assume ad: "address e ≠ STR ''Victim''"
              show "∀bal. frame bal st ⟶ frame bal st6'"
              proof
                fix bal
                show "frame bal st ⟶ frame bal st6'"
                proof
                  assume "frame bal st"
                  with t0 * show "frame bal st6'" by (auto simp add: frame_def split:if_split_asm)
                qed
              qed
            qed
          next
            show "?RHS" (is "?LHS ⟶ ?RHS")
            proof
              assume "address e = STR ''Victim''"
              show ?RHS (is "?A ∧ (?B ∧ ?C)")
              proof (rule conj3)
                show ?A (is "∀s val bal x. ?LHS s val bal x")
                proof (rule allI[OF allI[OF allI[OF allI]]])
                  fix s val bal x
                  show "?LHS s val bal x" (is "?LHS ⟶ ?RHS")
                  proof
                    assume ?LHS
                    then show ?RHS by simp
                  qed
                qed
              next
                show ?B (is "∀s bal x. ?LHS s bal x")
                proof (rule allI[OF allI[OF allI]])
                  fix s bal x
                  show "?LHS s bal x" (is "?LHS ⟶ ?RHS")
                  proof
                    assume ?LHS
                    then show ?RHS by simp
                  qed
                qed
              next
                show ?C (is "∀s bal. ?LHS s bal")
                proof (rule allI[OF allI])
                  fix s bal
                  show "?LHS s bal" (is "?LHS ⟶ ?RHS")
                  proof
                    assume ?LHS
                    then show ?RHS by simp
                  qed
                qed
              qed
            qed
          qed
        qed
      qed
    qed
  next
    case (37 lv ex e_p env cd st)
    show ?case (is "?LHS ⟶ ?RHS")
    proof
      assume 0: "fmlookup e_p STR ''Victim'' = Some (victim, SKIP)"
```

```
  show ?RHS (is "∀st6'. ?RHS st6'")
  proof
    fix st6'
    show "?RHS st6'" (is "?LHS ⟶ ?RHS")
    proof
      assume *: "stmt (ASSIGN lv ex) e_p env cd st = Normal ((), st6')"
      show ?RHS (is "?LHS ∧ ?RHS")
      proof
        show "?LHS"
        proof
          assume asm: "address env ≠ STR ''Victim''"
          show "∀bal. frame bal st ⟶ frame bal st6'"
          proof
            fix bal
            show "frame bal st ⟶ frame bal st6'"
            proof
              assume "frame bal st"
              with * have a1: "(applyf (costs (ASSIGN lv ex) e_p env cd) ≫= (λg. assert Gas (λst. gas
st ≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st =
                Normal ((), st⦇gas:=gas st - costs (ASSIGN lv ex) e_p env cd st⦈)"
                and f1: "frame bal (st⦇gas:=gas st - costs (ASSIGN lv ex) e_p env cd st⦈)" by (auto
simp add:frame_def)
              moreover from * obtain kv kt st' where **: "expr ex e_p env cd (st⦇gas:=gas
st - costs (ASSIGN lv ex) e_p env cd st⦈) = Normal ((kv, kt), st')" by (auto split:if_split_asm
result.split_asm)
              ultimately have "∀rv4 st4' (ev4'::Environment) bal.
                frame bal (st⦇gas := gas st - costs (ASSIGN lv ex) e_p env cd st⦈) ∧
                local.expr ex e_p env cd (st⦇gas := gas st - costs (ASSIGN lv ex) e_p env cd st⦈) =
Normal (rv4, st4') ⟶
                frame bal st4'" using asm 0 37(1) by simp
              with f1 ** have f2: "frame bal st'" by blast
              show "frame bal st6'"
              proof (cases kv)
                case (KValue v)
                then show ?thesis
                proof (cases kt)
                  case (Value t)
                  with * ** KValue obtain rv rt st'' where ***: "lexp lv e_p env cd st' = Normal
((rv,rt), st'')" by (auto split:if_split_asm result.split_asm)
                  with KValue Value have "∀rv5 st5' (ev5'::Environment) bal.
                  frame bal st' ∧
                  local.lexp lv e_p env cd st' = Normal (rv5, st5') ⟶
                  frame bal st5'" using asm 0 37(2)[OF a1 **] by simp
                  with f2 *** have f3: "frame bal st''" by blast
                  then show ?thesis
                  proof (cases rv)
                    case (LStackloc l')
                    then show ?thesis
                    proof (cases rt)
                      case v2: (Value t')
                      then show ?thesis
                      proof (cases "Valuetypes.convert t t' v")
                        case None
                        with * ** *** KValue Value LStackloc v2 show ?thesis by (auto
split:if_split_asm)
                      next
                        case (Some a)
                        then show ?thesis
                        proof (cases a)
                          case (Pair v' b)
                          with * ** *** KValue Value LStackloc v2 Some have "st6' = st'' ⦇stack :=
updateStore l' (KValue v') (stack st'')⦈" by (auto split:if_split_asm)
                          with f3 show ?thesis by (simp add:frame_def)
                        qed
```

```
qed
next
  case (Calldata x2)
  with * ** *** KValue Value LStackloc show ?thesis by (auto split:if_split_asm)
next
  case (Memory x3)
  with * ** *** KValue Value LStackloc show ?thesis by (auto split:if_split_asm)
next
  case (Storage x4)
  with * ** *** KValue Value LStackloc show ?thesis by (auto split:if_split_asm)
qed
next
  case (LMemloc l')
  then show ?thesis
  proof (cases rt)
    case v2: (Value t')
    with * ** *** KValue Value LMemloc show ?thesis by (auto split:if_split_asm)
  next
    case (Calldata x2)
    with * ** *** KValue Value LMemloc show ?thesis by (auto split:if_split_asm)
  next
    case (Memory x3)
    then show ?thesis
    proof (cases x3)
      case (MTArray x11 x12)
      with * ** *** KValue Value LMemloc Memory show ?thesis by (auto
split:if_split_asm)
    next
      case (MTValue t')
      then show ?thesis
      proof (cases "Valuetypes.convert t t' v")
        case None
        with * ** *** KValue Value LMemloc Memory MTValue show ?thesis by (auto
split:if_split_asm)
      next
        case (Some a)
        then show ?thesis
        proof (cases a)
          case (Pair v' b)
          with * ** *** KValue Value LMemloc Memory MTValue Some have "st6' = st''
(|memory := updateStore l' (MValue v') (memory st'')|)" by (auto split:if_split_asm)
          with f3 show ?thesis by (simp add:frame_def)
        qed
      qed
    qed
  next
    case (Storage x4)
    with * ** *** KValue Value LMemloc Storage show ?thesis by (auto
split:if_split_asm)
  qed
next
  case (LStoreloc l')
  then show ?thesis
  proof (cases rt)
    case v2: (Value x1)
    with * ** *** KValue Value LStoreloc show ?thesis by (auto split:if_split_asm)
  next
    case (Calldata x2)
    with * ** *** KValue Value LStoreloc show ?thesis by (auto split:if_split_asm)
  next
    case (Memory x3)
    with * ** *** KValue Value LStoreloc show ?thesis by (auto split:if_split_asm)
  next
    case (Storage x4)
```

```
                then show ?thesis
                proof (cases x4)
                  case (STArray x11 x12)
                  with * ** *** KValue Value LStoreloc Storage show ?thesis by (auto
split:if_split_asm)
                next
                  case (STMap x21 x22)
                  with * ** *** KValue Value LStoreloc Storage show ?thesis by (auto
split:if_split_asm)
                next
                  case (STValue t')
                  then show ?thesis
                  proof (cases "Valuetypes.convert t t' v")
                    case None
                    with * ** *** KValue Value LStoreloc Storage STValue show ?thesis by (auto
split:if_split_asm)
                  next
                    case (Some a)
                    then show ?thesis
                    proof (cases a)
                      case (Pair v' b)
                      then show ?thesis
                      proof (cases "fmlookup (storage st'') (address env)")
                        case None
                        with * ** *** KValue Value LStoreloc Storage STValue Some Pair show
?thesis by (auto split:if_split_asm)
                      next
                        case s2: (Some s)
                        with * ** *** KValue Value LStoreloc Storage STValue Some Pair
have "st6' = st''⦇storage := fmupd (address env) (fmupd l' v' s) (storage st'')⦈" by (auto
split:if_split_asm)
                        with f3 show ?thesis using asm by (simp add:frame_def)
                      qed
                    qed
                  qed
                qed
              qed
            qed
          next
            case (Calldata x2)
            with * ** KValue show ?thesis by (auto split:if_split_asm)
          next
            case (Memory x3)
            with * ** KValue show ?thesis by (auto split:if_split_asm)
          next
            case (Storage x4)
            with * ** KValue show ?thesis by (auto split:if_split_asm)
          qed
        next
          case (KCDptr p)
          then show ?thesis
          proof (cases kt)
            case (Value t)
            with * ** KCDptr show ?thesis by (auto split:if_split_asm)
          next
            case (Calldata x2)
            then show ?thesis
            proof (cases x2)
              case (MTArray x t)
              with * ** KCDptr Calldata obtain rv rt st'' where ***: "lexp lv e_p env cd st' =
Normal ((rv,rt), st'')" by (auto split:if_split_asm result.split_asm)
              with KCDptr Calldata MTArray have "∀rv5 st5' (ev5'::Environment) bal.
              frame bal st' ∧
              local.lexp lv e_p env cd st' = Normal (rv5, st5') ⟶
```

```
                    frame bal st5'" using asm 0 37(3)[OF a1 **] by auto
                    with f2 *** have f3: "frame bal st''" by blast
                    then show ?thesis
                    proof (cases rv)
                      case (LStackloc l')
                      then show ?thesis
                      proof (cases rt)
                        case (Value x1)
                        with * ** *** KCDptr Calldata MTArray LStackloc show ?thesis by (auto
split:if_split_asm)
                      next
                        case c2: (Calldata x2)
                        with * ** *** KCDptr Calldata MTArray LStackloc show ?thesis by (auto
split:if_split_asm)
                      next
                        case (Memory x3)
                        with f3 * ** *** KCDptr Calldata MTArray LStackloc show ?thesis by (auto
simp add:frame_def split:if_split_asm)
                      next
                        case (Storage x4)
                        then show ?thesis
                        proof (cases "accessStore l' (stack st'')")
                          case None
                          with * ** *** KCDptr Calldata MTArray LStackloc Storage show ?thesis by
(simp split:if_split_asm)
                        next
                          case (Some sv)
                          then show ?thesis
                          proof (cases sv)
                            case (KValue x1)
                            with * ** *** KCDptr Calldata MTArray LStackloc Storage Some show
?thesis by (simp split:if_split_asm)
                          next
                            case c2: (KCDptr x2)
                            with * ** *** KCDptr Calldata MTArray LStackloc Storage Some show
?thesis by (simp split:if_split_asm)
                          next
                            case (KMemptr x3)
                            with * ** *** KCDptr Calldata MTArray LStackloc Storage Some show
?thesis by (simp split:if_split_asm)
                          next
                            case (KStoptr p')
                            then show ?thesis
                            proof (cases "fmlookup (storage st'') (address env)")
                              case None
                              with * ** *** KCDptr Calldata MTArray LStackloc Storage Some KStoptr
show ?thesis by (simp split:if_split_asm)
                            next
                              case s2: (Some s)
                              then show ?thesis
                              proof (cases "cpm2s p p' x t cd s")
                                case None
                                with * ** *** KCDptr Calldata MTArray LStackloc Storage Some KStoptr
s2 show ?thesis by (simp split:if_split_asm)
                              next
                                case s3: (Some s')
                                with * ** *** KCDptr Calldata MTArray LStackloc Storage Some KStoptr
s2 have "st6' = st'' (|storage := fmupd (address env) s' (storage st'')|)" by (auto split:if_split_asm)
                                with f3 show ?thesis using asm by (simp add:frame_def)
                              qed
                            qed
                          qed
                        qed
                      qed
```

```
                      next
                        case (LMemloc l')
                        then show ?thesis
                        proof (cases "cpm2m p l' x t cd (memory st'')")
                          case None
                          with * ** *** KCDptr Calldata MTArray LMemloc show ?thesis by (auto
split:if_split_asm)
                        next
                          case (Some m)
                          with * ** *** KCDptr Calldata MTArray LMemloc have "st6' = st'' ⦇memory :=
m⦈" by (auto split:if_split_asm)
                          with f3 show ?thesis using asm by (simp add:frame_def)
                        qed
                      next
                        case (LStoreloc l')
                        then show ?thesis
                        proof (cases "fmlookup (storage st'') (address env)")
                          case None
                          with * ** *** KCDptr Calldata MTArray LStoreloc show ?thesis by (auto
split:if_split_asm)
                        next
                          case (Some s)
                          then show ?thesis
                          proof (cases "cpm2s p l' x t cd s")
                            case None
                            with * ** *** KCDptr Calldata MTArray LStoreloc Some show ?thesis by (auto
split:if_split_asm)
                          next
                            case s2: (Some s')
                            with * ** *** KCDptr Calldata MTArray LStoreloc Some s2 have "st6' = st''
⦇storage := fmupd (address env) s' (storage st'')⦈" by (auto split:if_split_asm)
                            with f3 show ?thesis using asm by (simp add:frame_def)
                          qed
                        qed
                      qed
                    next
                      case (MTValue x2)
                      with * ** KCDptr Calldata show ?thesis by (simp split:if_split_asm)
                    qed
                  next
                    case (Memory x3)
                    with * ** KCDptr show ?thesis by (simp split:if_split_asm)
                  next
                    case (Storage x4)
                    with * ** KCDptr show ?thesis by (simp split:if_split_asm)
                  qed
                next
                  case (KMemptr p)
                  then show ?thesis
                  proof (cases kt)
                    case (Value t)
                    with * ** KMemptr show ?thesis by (auto split:if_split_asm)
                  next
                    case (Calldata x2)
                    with * ** KMemptr show ?thesis by (simp split:if_split_asm)
                  next
                    case (Memory x3)
                    then show ?thesis
                    proof (cases x3)
                      case (MTArray x t)
                      with * ** KMemptr Memory obtain rv rt st'' where ***: "lexp lv e_p env cd st' =
Normal ((rv,rt), st'')" by (auto split:if_split_asm result.split_asm)
                      with KMemptr Memory MTArray have "∀rv5 st5' (ev5'::Environment) bal.
                      frame bal st' ∧
```

```
                    local.lexp lv e_p env cd st' = Normal (rv5, st5') ⟶
                    frame bal st5'" using asm 0 37(4)[OF a1 **] by auto
                    with f2 *** have f3: "frame bal st''" by blast
                    then show ?thesis
                    proof (cases rv)
                      case (LStackloc l')
                      then show ?thesis
                      proof (cases rt)
                        case (Value x1)
                        with * ** *** KMemptr Memory MTArray LStackloc show ?thesis by (auto
split:if_split_asm)
                      next
                        case (Calldata x2)
                        with * ** *** KMemptr Memory MTArray LStackloc show ?thesis by (auto
split:if_split_asm)
                      next
                        case m3: (Memory x3)
                        with f3 * ** *** KMemptr Memory MTArray LStackloc show ?thesis by (auto simp
add:frame_def split:if_split_asm)
                      next
                        case (Storage x4)
                        then show ?thesis
                        proof (cases "accessStore l' (stack st'')")
                          case None
                          with * ** *** KMemptr Memory MTArray LStackloc Storage show ?thesis by
(simp split:if_split_asm)
                        next
                          case (Some sv)
                          then show ?thesis
                          proof (cases sv)
                            case (KValue x1)
                            with * ** *** KMemptr Memory MTArray LStackloc Storage Some show ?thesis
by (simp split:if_split_asm)
                          next
                            case (KCDptr x2)
                            with * ** *** KMemptr Memory MTArray LStackloc Storage Some show ?thesis
by (simp split:if_split_asm)
                          next
                            case m2: (KMemptr x3)
                            with * ** *** KMemptr Memory MTArray LStackloc Storage Some show ?thesis
by (simp split:if_split_asm)
                          next
                            case (KStoptr p')
                            then show ?thesis
                            proof (cases "fmlookup (storage st'') (address env)")
                              case None
                              with * ** *** KMemptr Memory MTArray LStackloc Storage Some KStoptr
show ?thesis by (simp split:if_split_asm)
                            next
                              case s2: (Some s)
                              then show ?thesis
                              proof (cases "cpm2s p p' x t (memory st'') s")
                                case None
                                with * ** *** KMemptr Memory MTArray LStackloc Storage Some KStoptr
s2 show ?thesis by (simp split:if_split_asm)
                              next
                                case s3: (Some s')
                                with * ** *** KMemptr Memory MTArray LStackloc Storage Some KStoptr
s2 have "st6' = st'' (|storage := fmupd (address env) s' (storage st'')|)" by (auto split:if_split_asm)
                                with f3 show ?thesis using asm by (simp add:frame_def)
                              qed
                            qed
                          qed
                        qed
```

```
                    qed
                  next
                    case (LMemloc l')
                    with * ** *** KMemptr Memory MTArray LMemloc have "st6' = st'' ⦇memory :=
updateStore l' (MPointer p) (memory st'')⦈" by (auto split:if_split_asm)
                    with f3 show ?thesis using asm by (simp add:frame_def)
                  next
                    case (LStoreloc l')
                    then show ?thesis
                    proof (cases "fmlookup (storage st'') (address env)")
                      case None
                      with * ** *** KMemptr Memory MTArray LStoreloc show ?thesis by (auto
split:if_split_asm)
                    next
                      case (Some s)
                      then show ?thesis
                      proof (cases "cpm2s p l' x t (memory st'') s")
                        case None
                        with * ** *** KMemptr Memory MTArray LStoreloc Some show ?thesis by (auto
split:if_split_asm)
                      next
                        case s2: (Some s')
                        with * ** *** KMemptr Memory MTArray LStoreloc Some s2 have "st6' = st''
⦇storage := fmupd (address env) s' (storage st'')⦈" by (auto split:if_split_asm)
                        with f3 show ?thesis using asm by (simp add:frame_def)
                      qed
                    qed
                  qed
                next
                  case (MTValue x2)
                  with * ** KMemptr Memory show ?thesis by (simp split:if_split_asm)
                qed
              next
                case (Storage x4)
                with * ** KMemptr show ?thesis by (simp split:if_split_asm)
              qed
            next
              case (KStoptr p)
              then show ?thesis
              proof (cases kt)
                case (Value t)
                with * ** KStoptr show ?thesis by (auto split:if_split_asm)
              next
                case (Calldata x2)
                with * ** KStoptr show ?thesis by (simp split:if_split_asm)
              next
                case (Storage x3)
                then show ?thesis
                proof (cases x3)
                  case (STArray x t)
                  with * ** KStoptr Storage obtain rv rt st'' where ***: "lexp lv e_p env cd st' =
Normal ((rv,rt), st'')" by (auto split:if_split_asm result.split_asm)
                  with KStoptr Storage STArray have "∀rv5 st5' bal.
                  frame bal st' ∧
                  local.lexp lv e_p env cd st' = Normal (rv5, st5') ⟶
                  frame bal st5'" using asm 0 37(5)[OF a1 **] by auto
                  with f2 *** have f3: "frame bal st''" by blast
                  then show ?thesis
                  proof (cases rv)
                    case (LStackloc l')
                    then show ?thesis
                    proof (cases rt)
                      case (Value x1)
                      with * ** *** KStoptr Storage STArray LStackloc show ?thesis by (auto
```

```
split:if_split_asm)
                        next
                          case (Calldata x2)
                          with * ** *** KStoptr Storage STArray LStackloc show ?thesis by (auto
split:if_split_asm)
                        next
                          case (Memory x3)
                          then show ?thesis
                          proof (cases "accessStore l' (stack st'')")
                            case None
                            with * ** *** KStoptr Storage STArray LStackloc Memory show ?thesis by
(simp split:if_split_asm)
                          next
                            case (Some sv)
                            then show ?thesis
                            proof (cases sv)
                              case (KValue x1)
                              with * ** *** KStoptr Storage STArray LStackloc Memory Some show ?thesis
by (simp split:if_split_asm)
                            next
                              case (KCDptr x2)
                              with * ** *** KStoptr Storage STArray LStackloc Memory Some show ?thesis
by (simp split:if_split_asm)
                            next
                              case (KMemptr p')
                              then show ?thesis
                              proof (cases "fmlookup (storage st'') (address env)")
                                case None
                                with * ** *** KStoptr Storage STArray LStackloc Memory Some KMemptr
show ?thesis by (simp split:if_split_asm)
                              next
                                case s2: (Some s)
                                then show ?thesis
                                proof (cases "cps2m p p' x t s (memory st'')")
                                  case None
                                  with * ** *** KStoptr Storage STArray LStackloc Memory Some KMemptr
s2 show ?thesis by (simp split:if_split_asm)
                                next
                                  case s3: (Some m)
                                  with * ** *** KStoptr Storage STArray LStackloc Memory Some KMemptr
s2 have "st6' = st'' (|memory := m|)" by (auto split:if_split_asm)
                                  with f3 show ?thesis using asm by (simp add:frame_def)
                                qed
                              qed
                            next
                              case m2: (KStoptr x3)
                              with * ** *** KStoptr Storage STArray LStackloc Memory Some show ?thesis
by (simp split:if_split_asm)
                            qed
                          qed
                        next
                          case st2: (Storage x4)
                          with f3 * ** *** KStoptr Storage STArray LStackloc show ?thesis by (auto
simp add:frame_def split:if_split_asm)
                        qed
                      next
                        case (LStoreloc l')
                        then show ?thesis
                        proof (cases "fmlookup (storage st'') (address env)")
                          case None
                          with * ** *** KStoptr Storage STArray LStoreloc show ?thesis by (auto
split:if_split_asm)
                        next
                          case (Some s)
```

```
                    then show ?thesis
                    proof (cases "copy p l' x t s")
                      case None
                      with * ** *** KStoptr Storage STArray LStoreloc Some show ?thesis by (auto
split:if_split_asm)
                    next
                      case s2: (Some s')
                      with * ** ***  KStoptr Storage STArray LStoreloc Some s2 have "st6' = st''
(|storage := fmupd (address env) s' (storage st'')|)" by (auto split:if_split_asm)
                      with f3 show ?thesis using asm by (simp add:frame_def)
                    qed
                  qed
                next
                  case (LMemloc l')
                  then show ?thesis
                  proof (cases "fmlookup (storage st'') (address env)")
                    case None
                    with * ** *** KStoptr Storage STArray LMemloc show ?thesis by (auto
split:if_split_asm)
                  next
                    case (Some s)
                    then show ?thesis
                    proof (cases "cps2m p l' x t s (memory st'')")
                      case None
                      with * ** *** KStoptr Storage STArray LMemloc Some show ?thesis by (auto
split:if_split_asm)
                    next
                      case s2: (Some m)
                      with * ** ***  KStoptr Storage STArray LMemloc Some s2 have "st6' = st''
(|memory := m|)" by (auto split:if_split_asm)
                      with f3 show ?thesis using asm by (simp add:frame_def)
                    qed
                  qed
                qed
              next
                case (STMap t t')
                with * ** KStoptr Storage obtain l' rt st'' where ***: "lexp lv e_p env cd st' =
Normal ((LStackloc l',rt), st'')" by (auto split:if_split_asm result.split_asm LType.split_asm)
                with KStoptr Storage STMap have "∀rv5 st5' (ev5'::Environment) bal.
                frame bal st' ∧
                local.lexp lv e_p env cd st' = Normal (rv5, st5') ⟶
                frame bal st5'" using asm 0 37(6)[OF a1 **] by auto
                with f2 *** have f3: "frame bal st''" by blast
                moreover from * ** *** KStoptr Storage STMap have "st6' = st'' (|stack :=
updateStore l' (KStoptr p) (stack st'')|)" by (auto split:if_split_asm)
                ultimately show ?thesis using asm f3 by (simp add:frame_def)
              next
                case (STValue x2)
                with * ** KStoptr Storage show ?thesis by (simp split:if_split_asm)
              qed
            next
              case (Memory x4)
              with * ** KStoptr show ?thesis by (simp split:if_split_asm)
            qed
          qed
        qed
      qed
    qed
  next
    show "?RHS" (is "?LHS ⟶ ?RHS")
    proof
      assume "address env = STR ''Victim''"
      show ?RHS (is "?A ∧ (?B ∧ ?C)")
      proof (rule conj3)
```

```
            show ?A (is "∀s val bal x. ?LHS s val bal x")
            proof (rule allI[OF allI[OF allI[OF allI]]])
              fix s val bal x
              show "?LHS s val bal x" (is "?LHS ⟶ ?RHS")
              proof
                assume ?LHS
                then show ?RHS by simp
              qed
            qed
          next
            show ?B (is "∀s bal x. ?LHS s bal x")
            proof (rule allI[OF allI[OF allI]])
              fix s bal x
              show "?LHS s bal x" (is "?LHS ⟶ ?RHS")
              proof
                assume ?LHS
                then show ?RHS by simp
              qed
            qed
          next
            show ?C (is "∀s bal. ?LHS s bal")
            proof (rule allI[OF allI])
              fix s bal
              show "?LHS s bal" (is "?LHS ⟶ ?RHS")
              proof
                assume ?LHS
                then show ?RHS by simp
              qed
            qed
          qed
        qed
      qed
    qed
  qed
qed
next
  case (38 s1 s2 e_p e cd st)
  show ?case (is "?LHS ⟶ ?RHS")
  proof
    assume 0: "fmlookup e_p STR ''Victim'' = Some (victim, SKIP)"
    show ?RHS (is "∀st6'. ?RHS st6'")
    proof
      fix st6'
      show "?RHS st6'" (is "?LHS ⟶ ?RHS")
      proof
        assume *: "stmt (COMP s1 s2) e_p e cd st = Normal ((), st6')"
        show ?RHS (is "?LHS ∧ ?RHS")
        proof
          show "?LHS"
          proof
            assume asm: "address e ≠ STR ''Victim''"
            show "∀bal. frame bal st ⟶ frame bal st6'"
            proof
              fix bal
              show "frame bal st ⟶ frame bal st6'"
              proof
                assume "frame bal st"
                with * have a1: "(applyf (costs (COMP s1 s2) e_p e cd) ≫= (λg. assert Gas (λst. gas st
≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st =
                  Normal ((), st⦇gas:=gas st - costs (COMP s1 s2) e_p e cd st⦈)"
                  and f1: "frame bal (st⦇gas:=gas st - costs (COMP s1 s2) e_p e cd st⦈)" by (auto simp
add:frame_def)
                then have "∀rv4 st4' bal.
                  frame bal (st⦇gas := gas st - costs (COMP s1 s2) e_p e cd st⦈) ∧
```

```
              stmt s1 e_p e cd (st(|gas := gas st - costs (COMP s1 s2) e_p e cd st|)) = Normal (rv4,
st4') ⟶
              frame bal st4'" using asm 0 38(1) by (simp add:frame_def)
            moreover from * obtain st' where **: "stmt s1 e_p e cd (st(|gas:=gas st - costs (COMP
s1 s2) e_p e cd st|)) = Normal ((), st')" by (auto split:if_split_asm result.split_asm)
            ultimately have f2: "frame bal st'" using f1 by blast

            have "∀rv4 st4' bal.
              frame bal st' ∧
              stmt s2 e_p e cd st' = Normal (rv4, st4') ⟶
              frame bal st4'" using asm 0 38(2)[OF a1 **]  by (simp add:frame_def)
            moreover from * ** obtain st'' where ***: "stmt s2 e_p e cd st' = Normal ((), st'')"
by (auto split:if_split_asm result.split_asm)
            ultimately have f3: "frame bal st''" using f2 by blast

            from a1 * ** *** have "st6' = st''" by (simp split:if_split_asm)
            with f3 asm show "frame bal st6'" by simp
          qed
        qed
      qed
    next
      show "?RHS" (is "?LHS ⟶ ?RHS")
      proof
        assume ad: "address e = STR ''Victim''"
        show ?RHS (is "?A ∧ ?B ∧ ?C")
        proof (rule conj3)
          show ?A (is "∀s val bal x. ?LHS s val bal x")
          proof (rule allI[OF allI[OF allI[OF allI]]])
            fix s val bal x
            show "?LHS s val bal x" (is "?LHS ⟶ ?RHS")
            proof
              assume ?LHS
              then show ?RHS by simp
            qed
          qed
        next
          show ?B (is "∀s bal x. ?LHS s bal x")
          proof (rule allI[OF allI[OF allI]])
            fix s bal x
            show "?LHS s bal x" (is "?LHS ⟶ ?RHS")
            proof
              assume ?LHS(is "?A1 ∧ ?A2 ∧ ?A3 ∧ ?A4 ∧ ?A5 ∧ ?A6")
              then have ?A1 and ?A2 and ?A3 and ?A4 and ?A5 and ?A6 by auto
              with * have c1: "gas st > costs comp e_p e cd st" by (auto split:if_split_asm)
              with `?A1` * obtain st'' where 00: "stmt assign e_p e cd (st(|gas := gas st - costs
comp e_p e cd st|)) = Normal((), st'')" by (auto split:result.split_asm)
              moreover from `?A2` have "fmlookup (storage (st(|gas := gas st - costs comp e_p e cd
st|))) (STR ''Victim'') = Some s" by simp
              moreover from `?A2` have "ReadL_int (accessBalance (accounts (st(|gas := gas st -
costs comp e_p e cd st|))) (STR ''Victim'')) - (SUMM s) ≥ bal ∧ bal ≥ 0" by simp
              moreover from `?A3` have "POS s" by simp
              moreover from `?A6` have "accessStore x (stack (st(|gas := gas st - costs comp e_p e
cd st|))) = Some (KValue (accessStorage (TUInt 256) (sender e + (STR ''.'' + STR ''balance'')) s))" by
simp
              ultimately obtain s'' where "fmlookup (storage st'') (STR ''Victim'') = Some s''"
                  and "ReadL_int (accessBalance (accounts st'') (STR ''Victim'')) - (SUMM s'' +
ReadL_int (accessStorage (TUInt 256) (sender e + (STR ''.'' + STR ''balance'')) s)) ≥ bal ∧ bal ≥ 0"
                  and **: "accessStore x (stack st'') = Some (KValue (accessStorage (TUInt 256)
(sender e + (STR ''.'' + STR ''balance'')) s))"
                  and "POS s''"
                using secureassign[OF 00 _ ad `?A5`] that by blast
              moreover from c1 `?A1` * 00 obtain st''' where ***: "stmt transfer e_p e cd st'' =
Normal((), st''')" and "st6' = st'''" by auto
```

```
                moreover from ‘?A1‘ 00 have x1: "stmt s1 ep e cd (st(|gas := gas st - costs (COMP s1
s2) ep e cd st|)) = Normal((), st’’)" by simp
                moreover from * have x2: "(applyf (costs (COMP s1 s2) ep e cd) ⤜ (λg. assert Gas
                                                    (λst. gas st ≤ g)
                                                    (modify (λst. st(|gas := gas st - g|)))))
                 st = Normal ((), st(|gas:=gas st - costs (COMP s1 s2) ep e cd st|))" by (simp split:
if_split_asm)
                ultimately show "∃s’. fmlookup (storage st6’) (STR ’’Victim’’) = Some s’
                  ∧ ReadL_int (accessBalance (accounts st6’) (STR ’’Victim’’)) - (SUMM s’) ≥ bal ∧
bal ≥ 0 ∧ POS s’"
                  using 38(2)[OF x2 x1] ‘?A1‘ ‘?A4‘ ad 0 ** ‘?A6‘ by simp
              qed
            qed
          next
            show ?C (is "∀s bal. ?LHS s bal")
            proof (rule allI[OF allI])
              fix s bal
              show "?LHS s bal" (is "?LHS ⟶ ?RHS")
              proof
                assume ?LHS
                then show ?RHS by simp
              qed
            qed
          qed
        qed
      qed
    qed
  qed
  qed
next
  case (39 ex s1 s2 ep e cd st)
  show ?case (is "?LHS ⟶ ?RHS")
  proof
    assume 0: "fmlookup ep STR ’’Victim’’ = Some (victim, SKIP)"
    show ?RHS (is "∀st6’. ?RHS st6’")
    proof
      fix st6’
      show "?RHS st6’" (is "?LHS ⟶ ?RHS")
      proof
        assume *: "stmt (ITE ex s1 s2) ep e cd st = Normal ((), st6’)"
        show ?RHS (is "?LHS ∧ ?RHS")
        proof
          show "?LHS"
          proof
            assume asm: "address e ≠ STR ’’Victim’’"
            show "∀bal. frame bal st ⟶ frame bal st6’"
            proof
              fix bal
              show "frame bal st ⟶ frame bal st6’"
              proof
                assume "frame bal st"
                with * have a1: "(applyf (costs (ITE ex s1 s2) ep e cd) ⤜ (λg. assert Gas (λst. gas
st ≤ g) (modify (λst. st(|gas := gas st - g|))))) st =
                Normal ((), st(|gas:=gas st - costs (ITE ex s1 s2) ep e cd st|))"
                and f1: "frame bal (st(|gas:=gas st - costs (ITE ex s1 s2) ep e cd st|))" by (auto simp
add:frame_def)

                from * obtain b st’ where **: "expr ex ep e cd (st(|gas:=gas st - costs (ITE ex s1
s2) ep e cd st|)) = Normal ((KValue b, Value TBool), st’)" by (auto split:if_split_asm result.split_asm
prod.split_asm Stackvalue.split_asm Type.split_asm Types.split_asm)
                moreover from asm have "∀rv4 st4’ bal.
                  frame bal (st(|gas := gas st - costs (ITE ex s1 s2) ep e cd st|)) ∧
                  expr ex ep e cd (st(|gas := gas st - costs (ITE ex s1 s2) ep e cd st|)) = Normal (rv4,
st4’) ⟶
```

```
                frame bal st4'" using 0 39(1)[OF a1] by (simp add:frame_def)
              ultimately have f2: "frame bal st'" using f1 by blast

              show "frame bal st6'"
              proof (cases "b = ShowL_bool True")
                case True
                then have "∀st6' bal.
                frame bal st' ∧
                local.stmt s1 e_p e cd st' = Normal ((), st6') ⟶
                frame bal st6'" using asm 0 39(2)[OF a1 **, of "KValue b" "Value TBool" b TBool] by
(simp add:frame_def)
                moreover from * ** True obtain st'' where ***: "stmt s1 e_p e cd st' = Normal ((),
st'')" by (auto split:if_split_asm result.split_asm)
                ultimately have "frame bal st''" using f2 by blast
                moreover from a1 * ** True *** have "st6' = st''" by (simp split:if_split_asm)
                ultimately show "frame bal st6'" using asm by simp
              next
                case False
                then have "∀st6' bal.
                frame bal st' ∧
                local.stmt s2 e_p e cd st' = Normal ((), st6') ⟶
                frame bal st6'" using 0 asm 39(3)[OF a1 **, of "KValue b" "Value TBool" b TBool] by
(simp add:frame_def)
                moreover from * ** False obtain st'' where ***: "stmt s2 e_p e cd st' = Normal ((),
st'')" by (auto split:if_split_asm result.split_asm)
                ultimately have "frame bal st''" using f2 by blast
                moreover from a1 * ** False *** have "st6' = st''" by (simp split:if_split_asm)
                ultimately show "frame bal st6'" using asm by simp
              qed
            qed
          qed
        qed
      next
        show "?RHS" (is "?LHS ⟶ ?RHS")
        proof
          assume "address e = STR ''Victim''"
          show ?RHS (is "?A ∧ (?B ∧ ?C)")
          proof (rule conj3)
            show ?A (is "∀s val bal x. ?LHS s val bal x")
            proof (rule allI[OF allI[OF allI[OF allI]]])
              fix s val bal x
              show "?LHS s val bal x" (is "?LHS ⟶ ?RHS")
              proof
                assume ?LHS
                then show ?RHS by simp
              qed
            qed
          next
            show ?B (is "∀s bal x. ?LHS s bal x")
            proof (rule allI[OF allI[OF allI]])
              fix s bal x
              show "?LHS s bal x" (is "?LHS ⟶ ?RHS")
              proof
                assume ?LHS
                then show ?RHS by simp
              qed
            qed
          next
            show ?C (is "∀s bal. ?LHS s bal")
            proof (rule allI[OF allI])
              fix s bal
              show "?LHS s bal" (is "?LHS ⟶ ?RHS")
              proof
                assume ?LHS
```

```
                then show ?RHS by simp
              qed
            qed
          qed
        qed
      qed
    qed
  qed
 qed
next
  case (40 ex s0 e_p e cd st)
  show ?case (is "?LHS ⟶ ?RHS")
  proof
    assume 0: "fmlookup e_p STR ''Victim'' = Some (victim, SKIP)"
    show ?RHS (is "∀st6'. ?RHS st6'")
    proof
      fix st6'
      show "?RHS st6'" (is "?LHS ⟶ ?RHS")
      proof
        assume *: "stmt (WHILE ex s0) e_p e cd st = Normal ((), st6')"
        show ?RHS (is "?LHS ∧ ?RHS")
        proof
          show "?LHS"
          proof
            assume asm: "address e ≠ STR ''Victim''"
            show "∀bal. frame bal st ⟶ frame bal st6'"
            proof
              fix bal
              show "frame bal st ⟶ frame bal st6'"
              proof
                assume "frame bal st"
                with * have a1: "(applyf (costs (WHILE ex s0) e_p e cd) ≫= (λg. assert Gas (λst. gas st
≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st =
                Normal ((), st⦇gas:=gas st - costs (WHILE ex s0) e_p e cd st⦈)"
                and f1: "frame bal (st⦇gas:=gas st - costs (WHILE ex s0) e_p e cd st⦈)" by (auto simp
add:frame_def)

                from * obtain b st' where **: "expr ex e_p e cd (st⦇gas:=gas st - costs (WHILE ex s0)
e_p e cd st⦈) = Normal ((KValue b, Value TBool), st')" by (auto split:if_split_asm result.split_asm
prod.split_asm Stackvalue.split_asm Type.split_asm Types.split_asm)
                moreover from asm have "∀rv4 st4' bal.
                  frame bal (st⦇gas := gas st - costs (WHILE ex s0) e_p e cd st⦈) ∧
                  expr ex e_p e cd (st⦇gas := gas st - costs (WHILE ex s0) e_p e cd st⦈) = Normal (rv4,
st4') ⟶
                  frame bal st4'" using 0 40(1)[OF a1] by (simp add:frame_def)
                ultimately have f2: "frame bal st'" using f1 by blast

                show "frame bal st6'"
                proof (cases "b = ShowL_bool True")
                  case True
                  then have "∀st6' bal.
                  frame bal st' ∧
                  local.stmt s0 e_p e cd st' = Normal ((), st6') ⟶
                  frame bal st6'" using 0 asm 40(2)[OF a1 **, of "KValue b" "Value TBool" b TBool] by
(simp add:frame_def)
                  moreover from * ** True obtain st'' where ***: "stmt s0 e_p e cd st' = Normal ((),
st'')" by (auto split:if_split_asm result.split_asm)
                  ultimately have f3: "frame bal st''" using f2 by blast

                  have "∀st6' bal.
                  frame bal st'' ∧
                  local.stmt (WHILE ex s0) e_p e cd st'' = Normal ((), st6') ⟶
                  frame bal st6'" using 0 asm 40(3)[OF a1 ** _ _ _ _ True ***] by (simp add:frame_def)
                  moreover from * ** True *** obtain st''' where ****: "stmt (WHILE ex s0) e_p e cd
```

```
st'' = Normal ((), st''')" by (auto split:if_split_asm result.split_asm)
                  ultimately have "frame bal st'''" using f3 by blast

                  moreover from a1 * ** True *** **** have "st6' = st'''" by (simp
split:if_split_asm)
                  ultimately show "frame bal st6'" using asm by simp
                next
                  case False
                  then show "frame bal st6'" using * ** f1 f2 asm by (simp split:if_split_asm)
                qed
              qed
            qed
          qed
        next
          show "?RHS" (is "?LHS ⟶ ?RHS")
          proof
            assume "address e = STR ''Victim''"
            show ?RHS (is "?A ∧ (?B ∧ ?C)")
            proof (rule conj3)
              show ?A (is "∀s val bal x. ?LHS s val bal x")
              proof (rule allI[OF allI[OF allI[OF allI]]])
                fix s val bal x
                show "?LHS s val bal x" (is "?LHS ⟶ ?RHS")
                proof
                  assume ?LHS
                  then show ?RHS by simp
                qed
              qed
            next
              show ?B (is "∀s bal x. ?LHS s bal x")
              proof (rule allI[OF allI[OF allI]])
                fix s bal x
                show "?LHS s bal x" (is "?LHS ⟶ ?RHS")
                proof
                  assume ?LHS
                  then show ?RHS by simp
                qed
              qed
            next
              show ?C (is "∀s bal. ?LHS s bal")
              proof (rule allI[OF allI])
                fix s bal
                show "?LHS s bal" (is "?LHS ⟶ ?RHS")
                proof
                  assume ?LHS
                  then show ?RHS by simp
                qed
              qed
            qed
          qed
        qed
      qed
    qed
  qed
next
  case (41 i xe e_p e cd st)
  show ?case (is "?LHS ⟶ ?RHS")
  proof
    assume 0: "fmlookup e_p STR ''Victim'' = Some (victim, SKIP)"
    show ?RHS (is "∀st6'. ?RHS st6'")
    proof
      fix st'
      show "?RHS st'" (is "?LHS ⟶ ?RHS")
      proof
```

```
        assume *: "stmt (INVOKE i xe) e_p e cd st = Normal ((), st')"
        show ?RHS (is "?LHS ∧ ?RHS")
        proof
          show "?LHS"
          proof
            assume asm: "address e ≠ STR ''Victim''"
            show "∀ bal. frame bal st ⟶ frame bal st'"
            proof
              fix bal
              show "frame bal st ⟶ frame bal st'"
              proof
                assume ff: "frame bal st"
                moreover from * have a1: "(applyf (costs (INVOKE i xe) e_p e cd) ≫= (λg. assert Gas
(λst. gas st ≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st = Normal ((), st⦇gas := gas st - costs
(INVOKE i xe) e_p e cd st⦈)" by auto
                moreover from * obtain ct bla where **: "fmlookup e_p (address e) = Some (ct, bla)"
                  by (auto split:if_split_asm option.split_asm)
                moreover from * ** obtain fp f where ***: "fmlookup ct i = Some (Method (fp, f,
None))"
                  by (auto split:if_split_asm option.split_asm Member.split_asm)
                moreover define e' where "e' = ffold_init ct (emptyEnv (address e) (sender e) (svalue
e)) (fmdom ct)"
                moreover from * ** *** obtain e'' cd' st'' st''' where ****: "load False fp xe e_p e'
emptyStore (st⦇gas:=gas st - (costs (INVOKE i xe) e_p e cd st), stack:=emptyStore⦈) e cd (st⦇gas:=gas st
- (costs (INVOKE i xe) e_p e cd st)⦈) = Normal ((e'', cd', st''), st''')"
                  using e'_def by (auto split:if_split_asm result.split_asm)
                moreover from * **** have f1: "frame bal st''" and ad: "address e' = address e''"
                  using asm ff 0 41(1)[OF a1 ** _ *** _ _ _ _ e'_def, of bla "(fp, f, None)" fp "(f,
None)" f None] by (auto simp add:frame_def)
                moreover from e'_def have ad2: "address e = address e'" using ffold_init_ad_same[of
ct "(emptyEnv (address e) (sender e) (svalue e))" "(fmdom ct)" e'] by simp
                moreover from * ** *** **** e'_def obtain st'''' where *****: "stmt f e_p e'' cd'
st'' = Normal ((), st'''')" by (auto split:if_split_asm result.split_asm)
                ultimately have "st' = st''''⦇stack:=stack st''', memory := memory st'''⦈" using *
apply safe by simp
                moreover from f1 ad ad2 asm ***** have f2:"frame bal st''''"
                  using 41(2)[OF a1 ** _ *** _ _ _ _ e'_def _ ****] using 0 * by (auto simp
add:frame_def)
                ultimately show "frame bal st'" by (simp add:frame_def)
              qed
            qed
          qed
        next
          show "?RHS" (is "?LHS ⟶ ?RHS")
          proof
            assume "address e = STR ''Victim''"
            show ?RHS (is "?A ∧ (?B ∧ ?C)")
            proof (rule conj3)
              show ?A (is "∀ s val bal x. ?LHS s val bal x")
              proof (rule allI[OF allI[OF allI[OF allI]]])
                fix s val bal x
                show "?LHS s val bal x" (is "?LHS ⟶ ?RHS")
                proof
                  assume ?LHS
                  then show ?RHS by simp
                qed
              qed
            next
              show ?B (is "∀ s bal x. ?LHS s bal x")
              proof (rule allI[OF allI[OF allI]])
                fix s bal x
                show "?LHS s bal x" (is "?LHS ⟶ ?RHS")
                proof
                  assume ?LHS
```

```
                then show ?RHS by simp
              qed
            qed
          next
            show ?C (is "∀s bal. ?LHS s bal")
            proof (rule allI[OF allI])
              fix s bal
              show "?LHS s bal" (is "?LHS ⟶ ?RHS")
              proof
                assume ?LHS
                then show ?RHS by simp
              qed
            qed
          qed
        qed
      qed
    qed
  qed
qed
next
  case (42 ad i xe val e_p e cd st)
  show ?case (is "?LHS ⟶ ?RHS")
  proof
    assume 0: "fmlookup e_p STR ''Victim'' = Some (victim, SKIP)"
    show ?RHS (is "∀st6'. ?RHS st6'")
    proof
      fix st'
      show "?RHS st'" (is "?LHS ⟶ ?RHS")
      proof
        assume *: "stmt (EXTERNAL ad i xe val) e_p e cd st = Normal ((), st')"
        show ?RHS (is "?LHS ∧ ?RHS")
        proof
          show "?LHS"
          proof
            assume asm: "address e ≠ STR ''Victim''"
            show "∀bal. frame bal st ⟶ frame bal st'"
            proof
              fix bal
              show "frame bal st ⟶ frame bal st'"
              proof
                assume ff: "frame bal st"
                moreover from * have a1: "(applyf (costs (EXTERNAL ad i xe val) e_p e cd) ⪢= (λg.
assert Gas (λst. gas st ≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st = Normal ((), st⦇gas := gas
st - costs (EXTERNAL ad i xe val) e_p e cd st⦈)" by auto
                moreover from * obtain adv st'' where **: "expr ad e_p e cd (st⦇gas:=gas st - (costs
(EXTERNAL ad i xe val) e_p e cd st)⦈) = Normal ((KValue adv, Value TAddr), st'')"
                  by (auto split:if_split_asm result.split_asm Stackvalue.split_asm Type.split_asm
Types.split_asm)
                moreover from * ** ff have f1: "frame bal st''" using asm 0 42(1) by (simp
add:frame_def split:if_split_asm)
                moreover from * ** obtain ct fb where ***: "fmlookup e_p adv = Some (ct, fb)"
                  by (auto split:if_split_asm option.split_asm)
                moreover from * ** *** f1 obtain v t st''' where ****: "expr val e_p e cd st'' =
Normal ((KValue v, Value t), st''')"
                  by (auto split:if_split_asm result.split_asm Stackvalue.split_asm Type.split_asm)
                moreover from **** f1 have "frame bal st'''" using asm 42(2)[OF a1 ** _ _ _ _ ***] 0
by (simp split:if_split_asm)
                then have f2: "frame bal (st'''⦇stack := emptyStore, memory := emptyStore⦈)" by (simp
add:frame_def)
                show "frame bal st'"
                proof (cases "fmlookup ct i")
                  case None
                  with * ** *** **** obtain acc where trans: "Accounts.transfer (address e) adv v
(accounts st''') = Some acc" by (auto split:if_split_asm option.split_asm)
```

```
                with * ** *** **** None obtain st'''' where *****: "stmt fb e_p (emptyEnv adv
(address e) v) cd (st'''⦇accounts := acc,stack:=emptyStore, memory:=emptyStore⦈) = Normal ((), st'''')"
                  by (auto split:if_split_asm result.split_asm)
                moreover have f4: "frame bal (st''''⦇stack:=stack st'', memory := memory st''⦈)"
                proof (cases "adv = STR ''Victim''")
                  case True
                  with 0 *** have "fb = SKIP" by simp
                  moreover from f2 have "frame bal (st'''⦇accounts := acc,stack:=emptyStore,
memory:=emptyStore⦈)" using transfer_frame[OF trans] asm by (simp add:frame_def)
                  ultimately show ?thesis using ***** by (auto split:if_split_asm simp
add:frame_def)
                next
                  case False
                  moreover from f2 have "frame bal (st'''⦇accounts := acc,stack:=emptyStore,
memory:=emptyStore⦈)" using transfer_frame[OF trans] asm by (simp add:frame_def)
                  then have "frame bal st''''" using f2 0 42(5)[OF a1 ** _ _ _ _ *** _ **** _ _ _
None _ trans, of "KValue adv" "Value TAddr" TAddr fb "KValue v" "Value t" t st''' st''' st'''] asm *****
False by (auto simp add:frame_def)
                  then show ?thesis by (simp add:frame_def)
                qed
                ultimately show "frame bal st'" using a1 * ** *** **** None trans by (auto simp
add:frame_def)
              next
                case (Some a)
                with * ** *** **** obtain fp f where *****: "fmlookup ct i = Some (Method (fp, f,
None))"
                  by (auto split:if_split_asm option.split_asm Member.split_asm)
                moreover define e' where e'_def: "e' = ffold_init ct (emptyEnv adv (address e) v)
(fmdom ct)"
                moreover from * ** *** ***** **** obtain e'' cd' st'''' st''''' where *******:
"load True fp xe e_p e' emptyStore (st'''⦇stack:=emptyStore, memory:=emptyStore⦈) e cd st''' = Normal
((e'', cd', st''''), st''''')"
                  using e'_def by (auto split:if_split_asm result.split_asm option.split_asm)
                moreover from e'_def have ad2: "address e' = adv" and send2: "sender e' = address
e" and sval2: "svalue e' = v" using ffold_init_ad_same[of ct "(emptyEnv adv (address e) v)" "(fmdom
ct)" e'] by auto
                moreover from * ** *** ***** **** ******* e'_def obtain acc where trans:
"Accounts.transfer (address e) adv v (accounts st'''') = Some acc" by (auto split:if_split_asm
option.split_asm)
                then have ******: "Accounts.transfer (address e) adv v (accounts st'''') = Some acc"
by (auto split:if_split_asm option.split_asm)
                moreover from * ** *** ***** **** ****** ******* obtain st'''''' where ********:
"stmt f e_p e'' cd' (st''''⦇accounts := acc⦈) = Normal ((), st'''''')"
                  using e'_def by (auto split:if_split_asm result.split_asm)
                moreover have f4: "frame bal st''''''"
                proof (cases "adv = STR ''Victim''")
                  case True
                  with 0 *** have ct_def: "ct = victim" by simp

                  moreover have
                    "(∀ (ev::Environment) cda st st' bal.
                            local.load True fp xe e_p e' emptyStore (st'''⦇stack := emptyStore, memory
:= emptyStore⦈) e cd st''' = Normal ((ev, cda, st), st') ⟶
                            (frame bal (st'''⦇stack := emptyStore, memory := emptyStore⦈) ⟶ frame
bal st) ∧
                            (frame bal st''' ⟶ frame bal st') ∧ address e' = address ev ∧ sender e'
= sender ev ∧ svalue e' = svalue ev)"
                    using 0 42(3)[OF a1 ** _ _ _ _ *** _ **** _ _ _ ***** _ _ _ _ e'_def] asm by simp
                  with f2 ******* have f3: "frame bal st''''" and ad1: "address e' = address e''"
and send1: "sender e' = sender e''" and sval1: "svalue e' = svalue e''" by auto

                  from ct_def ***** consider (withdraw) "i = STR ''withdraw''" | (deposit) "i = STR
''deposit''" using victim_def fmap_of_list_SomeD[of "[(STR ''balance'', Var (STMap TAddr (STValue
(TUInt 256)))), (STR ''deposit'', Method ([], deposit, None)), (STR ''withdraw'', Method ([], keep,
```

```
None))]"] by auto
                    then show ?thesis
                    proof cases
                      case withdraw
                      moreover have "fmlookup victim (STR ''withdraw'') = Some (Method ([], keep,
None))" using victim_def by eval
                      ultimately have "f=keep" and "fp = []" using *** ***** True 0 by auto
                      with ******** have *********: "stmt keep e_p e'' cd' (st''''⦇accounts := acc⦈) =
Normal ((), st'''''')" by simp
                      have "fmlookup (denvalue e'') STR ''balance'' = Some (Storage (STMap TAddr
(STValue (TUInt 256))), Storeloc STR ''balance'')"
                      proof -
                        from victim_def have some: "fmlookup victim (STR ''balance'') = Some (Var
(STMap TAddr (STValue (TUInt 256))))" by eval
                        with ct_def have "fmlookup ct (STR ''balance'') = Some (Var (STMap TAddr
(STValue (TUInt 256))))" by simp
                        moreover have "STR ''balance'' |∉| fmdom (denvalue (emptyEnv adv (address e)
v))" by simp
                        moreover from ct_def some have "STR ''balance'' |∈| fmdom ct" using fmdomI
by simp
                        ultimately have "fmlookup (denvalue e') STR ''balance'' = Some (Storage (STMap
TAddr (STValue (TUInt 256))), Storeloc STR ''balance'')" using e'_def ffold_init_fmap[of ct "STR
''balance''" "(STMap TAddr (STValue (TUInt 256)))" "(emptyEnv adv (address e) v)" "(fmdom ct)"] by
simp
                        moreover have "e'' = e'"
                        proof (cases "xe=[]")
                          case True
                          with ******* ‘fp = []‘ show ?thesis by simp
                        next
                          case False
                          then obtain xx xe' where "xe = xx # xe'" using list.exhaust by auto
                          with ******* ‘fp = []‘ show ?thesis by simp
                        qed
                        ultimately show ?thesis by simp
                      qed

                      moreover from ad1 ad2 True have ad: "address e'' = STR ''Victim''" by simp
                      moreover from ad send1 send2 asm have "sender e'' ≠ address e''" by simp
                      moreover from f3 have f4: "frame bal (st''''⦇accounts := acc⦈)" using
transfer_frame[OF ******] asm by simp
                      then obtain s'''' where "fmlookup (storage (st''''⦇accounts := acc⦈))
(STR ''Victim'') = Some s'''' ∧ ReadL_int (accessBalance (accounts (st''''⦇accounts := acc⦈)) (STR
''Victim'')) - (SUMM s'''') ≥ bal ∧ bal ≥ 0 ∧ POS s''''" by (auto simp add:frame_def)
                      ultimately have "(∃s'. fmlookup (storage st'''''') (STR ''Victim'') = Some s'
                      ∧ ReadL_int (accessBalance (accounts st'''''') (STR ''Victim'')) - (SUMM s') ≥ bal
∧ bal ≥ 0 ∧ POS s')"
                        using 0 ******** ‘f=keep‘ 42(4)[OF a1 ** _ _ _ _ *** _ **** _ _ _ *****  _ _ _
_ e'_def _ ******* _ _ _ trans, of "KValue adv" "Value TAddr" TAddr fb "KValue v" "Value t" t "(fp, f,
None)" "(f, None)" f None e'' "(cd', st'''')" cd' st''''' st''''' "()" "st''''⦇accounts := acc⦈"] apply
safe by auto
                      then show ?thesis by (simp add:frame_def)
                    next
                      case deposit
                      moreover from f2 have "frame bal (st'''⦇stack:=emptyStore,
memory:=emptyStore⦈)" using transfer_frame[OF ******] asm by simp
                      moreover have "fmlookup victim (STR ''deposit'') = Some (Method ([], deposit,
None))" using victim_def by eval
                      ultimately have "f=deposit" and "fp = []" using *** ***** True 0 by auto
                      with ******** have *: "stmt deposit e_p e'' cd' (st''''⦇accounts := acc⦈) = Normal
((), st'''''')" by simp

                      have f4: "frame (bal + ReadL_int v) (st''''⦇accounts := acc⦈)" and ad1: "address
e' = address e''" and send1: "sender e' = sender e''" and sval1: "svalue e' = svalue e''"
                      proof -
```

```
          have "ReadL_int (accessBalance acc (STR ''Victim'')) = ReadL_int (accessBalance
(accounts st'''') (STR ''Victim'')) + ReadL_int v" using transfer_add[OF ******] asm True by simp
          moreover have "ReadL_int v ≥ 0" using transfer_val1[OF  ******] by simp
          ultimately have "frame (bal + ReadL_int v) (st''''⦇accounts := acc⦈)" using f3
by (auto simp add:frame_def)
          then show "frame (bal + ReadL_int v) (st''''⦇accounts := acc⦈)" and "address e'
= address e''" and "sender e' = sender e''" and "svalue e' = svalue e''" using f2 0 42(3)[OF a1 ** _ _
_ _ *** _ **** _ _ _ ***** _ _ _ _ e'_def, of "KValue adv" "Value TAddr" TAddr fb "KValue v" "Value t"]
asm ******* by (auto simp add:frame_def)
        qed
        moreover from sval1 sval2 have "v = svalue e''" by simp
        ultimately have "frame (bal + ReadL_int (svalue e'')) (st''''⦇accounts := acc⦈)"
by simp
        then obtain s''''' where II: "INV (st''''⦇accounts := acc⦈) s''''' (SUMM s''''')
(bal + ReadL_int (svalue e''))" and III:"POS s'''''" by (auto simp add:frame_def)
        then have s'''''_def: "fmlookup (storage (st''''⦇accounts := acc⦈)) STR
''Victim'' = Some s'''''" by simp

        have yyy: "fmlookup (denvalue e'') STR ''balance'' = Some (Storage (STMap TAddr
(STValue (TUInt 256))), Storeloc STR ''balance'')"
        proof -
          from victim_def have some: "fmlookup victim (STR ''balance'') = Some (Var
(STMap TAddr (STValue (TUInt 256))))" by eval
          with ct_def have "fmlookup ct (STR ''balance'') = Some (Var (STMap TAddr
(STValue (TUInt 256))))" by simp
          moreover have "STR ''balance'' |∉| fmdom (denvalue (emptyEnv adv (address e)
v))" by simp
          moreover from ct_def some have "STR ''balance'' |∈| fmdom ct" using fmdomI
by simp
          ultimately have "fmlookup (denvalue e') STR ''balance'' = Some (Storage (STMap
TAddr (STValue (TUInt 256))), Storeloc STR ''balance'')" using e'_def ffold_init_fmap[of ct "STR
''balance''" "(STMap TAddr (STValue (TUInt 256)))" "(emptyEnv adv (address e) v)" "(fmdom ct)"] by
simp
          moreover have "e'' = e'"
          proof (cases "xe=[]")
            case True
            with ******* `fp = []` show ?thesis by simp
          next
            case False
            then obtain xx xe' where "xe = xx # xe'" using list.exhaust by auto
            with ******* `fp = []` show ?thesis by simp
          qed
          ultimately show ?thesis by simp
        qed

        from asm True have "address e ≠ (STR ''Victim'')" by simp
        then have  "ReadL_int (accessBalance (accounts st'''')  (STR ''Victim'')) +
ReadL_int v < 2^256" using transfer_val2[OF  ******] True by simp
        moreover from `address e ≠ (STR ''Victim'')` have "ReadL_int (accessBalance acc
(STR ''Victim'')) = ReadL_int (accessBalance (accounts st'''')  (STR ''Victim'')) + ReadL_int v" using
transfer_add[OF ******]  True by simp
        ultimately have abc: "ReadL_int (accessBalance (accounts (st''''⦇accounts :=
acc⦈))  (STR ''Victim'')) < 2^256" by simp

        from II have "fmlookup (storage (st''''⦇accounts := acc⦈)) (STR ''Victim'') =
Some s'''''  ∧ ReadL_int (accessBalance (accounts (st''''⦇accounts := acc⦈)) (STR ''Victim'')) - (SUMM
s''''') ≥ bal + ReadL_int  (svalue e'') ∧ bal + ReadL_int  (svalue e'') ≥ 0" by (auto)
        moreover have "ReadL_int (accessStorage (TUInt 256) (sender e'' + (STR ''.'' +
STR ''balance'')) s''''')  + ⌈svalue e''⌉ < 2 ^ 256 ∧
            ReadL_int (accessStorage (TUInt 256) (sender e'' + (STR ''.'' + STR
''balance'')) s''''')  + ⌈svalue e''⌉ ≥ 0"
        proof (cases "fmlookup s''''' (sender e'' + (STR ''.'' + STR ''balance'')) =
None")
          case True
```

```
                        hence "(accessStorage (TUInt 256) (sender e'' + (STR ''.'' + STR ''balance''))
s''''') = ShowL_int 0" by simp
                        moreover have "(⌈svalue e''⌉::int) < 2 ^ 256"
                        proof -
                          from II have "bal + ⌈svalue e''⌉ + SUMM s''''' ≤ ReadL_int (accessBalance
(accounts (st''''⦇accounts := acc⦈)) (STR ''Victim''))" by simp
                          moreover have "0 ≤ SUMM s'''''"
                          using III sum_nonneg[of "{(ad,x). fmlookup s''''' (ad + (STR ''.'' + STR
''balance'')) = Some x}" "λ(ad,x). ReadL_int x"] by auto
                          ultimately have "bal + ⌈svalue e''⌉ ≤ ReadL_int (accessBalance (accounts
(st''''⦇accounts := acc⦈)) (STR ''Victim''))" by simp
                          moreover from ff have "bal≥0" by (auto simp add:frame_def)
                          ultimately show "(⌈svalue e''⌉::int) < 2 ^ 256" using abc by simp
                        qed
                        moreover have "ReadL_int v ≥ 0" using transfer_val1[OF ******] by simp
                        with `svalue e' = v` sval1 have "(⌈svalue e''⌉::int) ≥ 0" by simp
                        ultimately show ?thesis using Read_ShowL_id by simp
                      next
                        case False
                        then obtain x where x_def: "fmlookup s'''''  (sender e'' + (STR ''.'' + STR
''balance'')) = Some x" by auto
                        moreover from II have "bal + ⌈svalue e''⌉ + SUMM s''''' ≤ ReadL_int
(accessBalance (accounts (st''''⦇accounts := acc⦈)) (STR ''Victim''))" by simp
                        moreover have "(case (sender e'', x) of (ad, x) ⇒ ⌈x⌉) ≤ (∑(ad, x)∈{(ad,
x). fmlookup s''''' (ad + (STR ''.'' + STR ''balance'')) = Some x}. ReadL_int x)"
                        proof (cases rule: member_le_sum[of "(sender e'',x)" "{(ad,x). fmlookup s'''''
(ad + (STR ''.'' + STR ''balance'')) = Some x}" "λ(ad,x). ReadL_int x"])
                          case 1
                          then show ?case using x_def by simp
                        next
                          case (2 x)
                          with III show ?case by auto
                        next
                          case 3
                          have "inj_on (λ(ad, x). (ad + (STR ''.'' + STR ''balance''), x)) {(ad, x).
(fmlookup s''''' ∘ (λad. ad + (STR ''.'' + STR ''balance''))) ad = Some x}" using balance_inj by simp
                          then have "finite {(ad, x). (fmlookup s''''' ∘ (λad. ad + (STR ''.'' + STR
''balance''))) ad = Some x}" using fmlookup_finite[of "λad. (ad + (STR ''.'' + STR ''balance''))"
s'''''] by simp
                          then show ?case using finite_subset[of "{(ad, x). fmlookup s''''' (ad + (STR
''.'' + STR ''balance'')) = Some x}" "{(ad, x). fmlookup s''''' (ad + (STR ''.'' + STR ''balance'')) =
Some x}"] by auto
                        qed
                        then have "ReadL_int x ≤ SUMM s'''''" by simp
                        ultimately have "bal + ⌈svalue e''⌉ + ReadL_int x ≤ ReadL_int (accessBalance
(accounts (st''''⦇accounts := acc⦈)) (STR ''Victim''))" by simp
                        moreover from ff have "bal≥0" by (auto simp add:frame_def)
                        ultimately have "⌈svalue e''⌉ + ReadL_int x ≤ ReadL_int (accessBalance (accounts
(st''''⦇accounts := acc⦈)) (STR ''Victim''))" by simp
                        with abc have "⌈svalue e''⌉ + ReadL_int x < 2^256" by simp
                        moreover have "fmlookup s''''' (sender e' + (STR ''.'' + STR ''balance'')) =
fmlookup s'''''  (sender e'' + (STR ''.'' + STR ''balance''))" using send1 by simp
                        ultimately have "bal + ⌈svalue e''⌉ ≤ ⌈accessBalance (accounts (st''''⦇accounts
:= acc⦈)) STR ''Victim''⌉ - SUMM s'''''" and lck: "fmlookup s''''' (sender e'' + (STR ''.'' + STR
''balance'')) = Some x"  and "ReadL_int x + ⌈svalue e''⌉ < 2 ^ 256" using ad1 ad2 True II x_def by simp+
                        moreover from lck have "(accessStorage (TUInt 256) (sender e'' + (STR ''.'' +
STR ''balance'')) s''''') = x" by simp
                        moreover have "⌈svalue e''⌉ + ReadL_int x ≥ 0"
                        proof -
                          have "ReadL_int v ≥ 0" using transfer_val1[OF ******] by simp
                          with `svalue e' = v` sval1 have "(⌈svalue e''⌉::int) ≥ 0" by simp
                          moreover from III have "ReadL_int x ≥ 0" using x_def by simp
                          ultimately show ?thesis by simp
                        qed
```

```
                    ultimately show ?thesis using Read_ShowL_id by simp
                  qed
                  moreover have "address e'' = STR ''Victim''" using ad1 ad2 True by simp
                  ultimately obtain s'''''' where "fmlookup (storage st'''''') (STR ''Victim'') =
Some s''''''" and "ReadL_int (accessBalance (accounts st'''''') (STR ''Victim'')) - SUMM s'''''' ≥ bal"
and "POS s''''''"
                  using deposit_frame[OF * s'''''_def _ yyy] III by auto
                  then show ?thesis using ff by (auto simp add:frame_def)
                qed
              next
                case False
                moreover from f2 have "frame bal (st'''⦇stack:=emptyStore, memory:=emptyStore⦈)"
using transfer_frame[OF ******] asm by simp
                then have "frame bal st''''" and ad1: "address e' = address e''" using f2 0
42(3)[OF a1 ** _ _ _ _ *** _ **** _ _ _ ***** _ _ _ _ e'_def, of "KValue adv" "Value TAddr" TAddr fb
"KValue v" "Value t"] asm ******* by (auto simp add:frame_def)
                then have f4: "frame bal (st''''⦇accounts := acc⦈)" using transfer_frame[OF
******] asm by simp

                moreover from ad1 ad2 have "address e'' ≠ STR ''Victim'' ∧ fmlookup e_p (STR
''Victim'') = Some (victim, SKIP)" using 0 False by simp
                then have "∀st6' bal.
                frame bal (st''''⦇accounts := acc⦈) ∧
                local.stmt f e_p e'' cd' (st''''⦇accounts := acc⦈) = Normal ((), st6') ⟶
                frame bal st6'" using 42(4)[OF a1 ** _ _ _ _ *** _ **** _ _ _ ***** _ _ _ _ e'_def
_ ******* _ _ _ ******] False asm by (auto simp add:frame_def)
                ultimately show ?thesis using ******** by blast
              qed
              ultimately show "frame bal st'" using a1 * ** *** **** by (auto simp add:frame_def)
            qed
          qed
        qed
      qed
    next
      show "?RHS" (is "?LHS ⟶ ?RHS")
      proof
        assume "address e = STR ''Victim''"
        show ?RHS (is "?A ∧ (?B ∧ ?C)")
        proof (rule conj3)
          show ?A (is "∀s val bal x. ?LHS s val bal x")
          proof (rule allI[OF allI[OF allI[OF allI]]])
            fix s val bal x
            show "?LHS s val bal x" (is "?LHS ⟶ ?RHS")
            proof
              assume ?LHS
              then show ?RHS by simp
            qed
          qed
        next
          show ?B (is "∀s bal x. ?LHS s bal x")
          proof (rule allI[OF allI[OF allI]])
            fix s bal x
            show "?LHS s bal x" (is "?LHS ⟶ ?RHS")
            proof
              assume ?LHS
              then show ?RHS by simp
            qed
          qed
        next
          show ?C (is "∀s bal. ?LHS s bal")
          proof (rule allI[OF allI])
            fix s bal
            show "?LHS s bal" (is "?LHS ⟶ ?RHS")
            proof
```

```
                  assume ?LHS
                  then show ?RHS by simp
                qed
              qed
            qed
          qed
        qed
      qed
    qed
  qed
next
  case (43 ad ex e_p e cd st)
  show ?case (is "?LHS ⟶ ?RHS")
  proof
    assume 0: "fmlookup e_p STR ''Victim'' = Some (victim, SKIP)"
    show ?RHS (is "∀st6'. ?RHS st6'")
    proof
      fix st'
      show "?RHS st'" (is "?LHS ⟶ ?RHS")
      proof
        assume *: "stmt (TRANSFER ad ex) e_p e cd st = Normal ((), st')"
        show ?RHS (is "?LHS ∧ ?RHS")
        proof
          show "?LHS"
          proof
            assume asm: "address e ≠ STR ''Victim''"
            show "∀bal. frame bal st ⟶ frame bal st'"
            proof
              fix bal
              show "frame bal st ⟶ frame bal st'"
              proof
                assume ff: "frame bal st"
                from * have a1: "(applyf (costs (TRANSFER ad ex) e_p e cd) ≫= (λg. assert Gas (λst.
gas st ≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st = Normal ((), st⦇gas := gas st - costs
(TRANSFER ad ex) e_p e cd st⦈)" by auto
                from * obtain v t st'' where **: "expr ex e_p e cd (st⦇gas:=gas st - (costs (TRANSFER
ad ex) e_p e cd st)⦈) = Normal ((KValue v, Value t), st'')"
                  by (auto split:if_split_asm result.split_asm Stackvalue.split_asm Type.split_asm)
                from asm ff * ** have f1: "frame bal st''" using 43(1)[OF a1] 0 by (simp
add:frame_def)
                from * ** obtain adv st''' where ***: "expr ad e_p e cd st'' = Normal ((KValue adv,
Value TAddr), st''')"
                  by (auto split:if_split_asm result.split_asm Stackvalue.split_asm Type.split_asm
Types.split_asm)
                from asm * *** f1 have f2: "frame bal st'''" using asm 43(2)[OF a1 **] 0 by (simp
add:frame_def)
                from * ** *** obtain acc where *****: "Accounts.transfer (address e) adv v (accounts
st''') = Some acc" by (auto split:if_split_asm option.split_asm)
                from f2 have f3: "frame bal (st'''⦇accounts := acc⦈)" using transfer_frame[OF *****]
asm by simp
                show "frame bal st'"
                proof (cases "fmlookup e_p adv")
                  case None
                  with a1 * ** *** ***** show ?thesis using f3 by auto
                next
                  case (Some a)
                  then show ?thesis
                  proof (cases a)
                    case (Pair ct f)
                    moreover define e' where "e' = ffold_init ct (emptyEnv adv (address e) v) (fmdom
ct)"
                    moreover from * ** *** Some Pair ***** e'_def obtain st'''' where ******: "stmt f
e_p e' emptyStore (st'''⦇accounts := acc, stack:=emptyStore, memory:=emptyStore⦈) = Normal ((), st'''')"
by (auto split:if_split_asm option.split_asm result.split_asm)
```

```
                    moreover from e'_def have ad: "adv = address e'" using ffold_init_ad_same[of ct
"(emptyEnv adv (address e) v)" "(fmdom ct)" e'] by simp

                    moreover have f4: "frame bal st''''"
                    proof (cases "adv = STR ''Victim''")
                      case True
                      with 0 ** *** Some Pair have "f = SKIP" using victim_def by simp
                      with ****** have "st''''=  st'''
                      (|accounts := acc, stack := emptyStore, memory := emptyStore,
                      gas := gas st''' - costs SKIP e_p e' emptyStore (st'''(|accounts := acc, stack :=
emptyStore, memory := emptyStore|))|)" by (simp split:if_split_asm)
                      with f3 show ?thesis by (simp add:frame_def)
                    next
                      case False
                      with asm ad have "∀st6' bal.
                      frame bal (st'''(|accounts := acc|)) ∧
                      local.stmt f e_p e' emptyStore (st'''(|accounts := acc, stack := emptyStore, memory
:= emptyStore|)) = Normal ((), st6') ⟶
                      frame bal st6'" using asm Some Pair 43(3)[OF a1 ** _ _ _ *** _ _ _ _ _ *****
_ _ e'_def, where s'e = "st'''(|accounts := acc, stack:=emptyStore, memory:=emptyStore|)", of "KValue
v" "Value t" t "KValue adv" "Value TAddr" TAddr st''' _ f st''' st''' "()"] using 0 by (simp
add:frame_def)
                      with f3 ****** show ?thesis by blast
                    qed
                    moreover from * ** *** Some Pair ***** ****** e'_def have st'_def: "st' =
st''''(|stack:=stack st''', memory := memory st'''|)" by (simp split:if_split_asm)
                    ultimately show "frame bal st'" apply safe by (simp_all add:frame_def)
                  qed
                qed
              qed
            qed
          qed
        next
          show "?RHS" (is "?LHS ⟶ ?RHS")
          proof
            assume ad: "address e = STR ''Victim''"
            show ?RHS (is "?A ∧ ?B ∧ ?C")
            proof (rule conj3)
              show ?A (is "∀s val bal x. ?LHS s val bal x")
              proof (rule allI[OF allI[OF allI[OF allI]]])
                fix s val bal x
                show "?LHS s val bal x" (is "?LHS ⟶ ?RHS")
                proof
                  assume ?LHS (is "?A1 ∧ ?A2 ∧ ?A3 ∧ ?A4 ∧ ?A5")
                  then have ?A1 and ?A2 and ?A3 and ?A4 and ?A5 by auto
                  define st'' where "st'' = st(|gas := gas st - costs transfer e_p e cd st|)"
                  define st''' where "st''' = st''(|gas := gas st'' - costs_e (LVAL (Id (STR ''bal'')))
e_p e cd st''|)"
                  define st'''' where "st'''' = st'''(|gas := gas st''' - costs_e SENDER e_p e cd st'''|)"
                  from `?A1` * have c1: "gas st > costs transfer e_p e cd st" by (auto
split:if_split_asm)
                  have c2: "gas st'' > costs_e (LVAL (Id (STR ''bal''))) e_p e cd st''"
                  proof (rule ccontr)
                    assume "¬ costs_e (LVAL (Id (STR ''bal''))) e_p e cd st'' < gas st''"
                    with c1 show False using `?A1` * st''_def st'''_def by auto
                  qed
                  with `?A4` `?A5` have 00:"expr (LVAL (Id (STR ''bal''))) e_p e cd st'' = Normal
((KValue val, Value (TUInt 256)), st''')" using st'''_def st''_def by simp
                  moreover have "gas st'''>costs_e SENDER e_p e cd st'''"
                  proof (rule ccontr)
                    assume "¬ costs_e SENDER e_p e cd st''' < gas st'''"
                    with c1 c2 show False using `?A1` `?A4` `?A5` * st''_def st'''_def by auto
                  qed
                  then have **:"expr SENDER e_p e cd st''' = Normal ((KValue (sender e), Value TAddr),
```

```
st'''')" using st''''_def by simp
                then obtain acc where ***:"Accounts.transfer (address e) (sender e) val (accounts
st'''') = Some acc"
                  and ****: "ReadL_int (accessBalance acc (STR ''Victim'')) = ReadL_int (accessBalance
(accounts st'''') (STR ''Victim'')) - (ReadL_int val)"
                proof -
                  from ‘?A1‘ * c1 00 ** obtain acc where acc_def: "Accounts.transfer (address e)
(sender e) val (accounts st'''') = Some acc" using st''''_def st'''_def st''_def by (auto split:
option.split_asm)
                  with ad obtain acc' where *: "subBalance (STR ''Victim'') val (accounts st'''') =
Some acc'"
                    and "addBalance (sender e) val acc' = Some acc" by (simp split:
option.split_asm)
                  moreover from * have "acc' = updateBalance(STR ''Victim'') (ShowL_int (ReadL_int
(accessBalance (accounts st'''') (STR ''Victim'')) - ReadL_int val)) (accounts st'''')" by (simp split:
if_split_asm)
                  then have "ReadL_int (accessBalance acc' (STR ''Victim'')) = ReadL_int (accessBalance
(accounts st'''') (STR ''Victim'')) - (ReadL_int val)" using Read_ShowL_id by simp
                  moreover from ‘?A5‘ ad have "sender e ≠ (STR ''Victim'')" by simp
                  ultimately have "ReadL_int (accessBalance acc (STR ''Victim'')) = ReadL_int
(accessBalance (accounts st'''') (STR ''Victim'')) - (ReadL_int val)" using addBalance_eq[of "sender
e" val acc' acc " STR ''Victim''"] by simp
                  with acc_def show ?thesis using that by simp
                qed
                show ?RHS
                proof (cases "fmlookup e_p (sender e)")
                  case None
                  with ‘?A1‘ 00 * ** *** have "st' = st''''(|accounts := acc|)" using c1 st''_def by
auto
                  moreover from ‘?A2‘ have "fmlookup (storage st'''') (STR ''Victim'') = Some s"
using st''_def st'''_def st''''_def by simp
                  moreover from ‘?A2‘ have "ReadL_int (accessBalance (accounts st'''') (STR
''Victim'')) - (SUMM s + ReadL_int val) ≥ bal ∧ bal ≥ 0"  using st''_def st'''_def st''''_def by simp
                  with **** have "ReadL_int (accessBalance acc (STR ''Victim'')) - SUMM s ≥ bal ∧ bal
≥ 0" by simp
                  then have "ReadL_int (accessBalance (accounts (st''''(|accounts := acc|))) (STR
''Victim'')) - SUMM s ≥ bal ∧ bal ≥ 0" by simp
                  ultimately show ?thesis using ‘?A3‘ by (simp add:frame_def)
                next
                  case (Some a)
                  then show ?thesis
                  proof (cases a)
                    case (Pair ct f)
                    moreover define e' where e'_def: "e'=ffold_init ct (emptyEnv (sender e) (address
e) val) (fmdom ct)"
                    ultimately obtain st''''' where *****: "stmt f e_p e' emptyStore (st''''(|accounts
:= acc, stack:=emptyStore, memory:=emptyStore|)) = Normal ((), st''''')"
                      and ******: "st' = st'''''(|stack:=stack st'''', memory := memory st''''|)"  us-
ing ‘?A1‘ 00 ** *** Some *  stmt.simps(8)[of SENDER "(LVAL (Id (STR ''bal'')))" e_p e cd st] c1 st''_def
st'''_def st''''_def by (auto split: result.split_asm unit.split_asm)
                    from ‘?A1‘ * have x1: "(applyf (costs (TRANSFER ad ex) e_p e cd) ≫= (λg. assert
Gas
                                                                              (λst. gas st ≤ g)
                                                                              (modify (λst. st(|gas := gas
st - g|))))
                     st =
                    Normal ((), st'')" using st''_def by (simp split:if_split_asm)
                    from 00 ‘?A1‘ have x2: "expr ex e_p e cd st'' = Normal ((KValue val, Value (TUInt
256)), st''')" by simp
                    have x3: "(KValue val, Value (TUInt 256)) = (KValue val, Value (TUInt 256))" by
simp
                    have x4: "KValue val = KValue val" by simp
                    have  x5: "Value (TUInt 256) = Value (TUInt 256)" by simp
                    from  ** ‘?A1‘ have x6: "expr ad e_p e cd st''' = Normal ((KValue (sender e),
```

```
Value TAddr), st'''')" by simp
           have x7: "(KValue (sender e), Value TAddr) = (KValue (sender e), Value TAddr)" by
simp
           have x8: "KValue (sender e) = KValue (sender e)" by simp
           have x9: "Value TAddr = Value TAddr" by simp
           have x10: "TAddr = TAddr" by simp
           have x11: "applyf accounts st'''' = Normal (accounts st'''', st'''')" by simp
           from *** have x12: "Accounts.transfer (address e) (sender e) val (accounts
st'''') = Some acc" by simp
           from Some Pair have x13: "fmlookup e_p (sender e) = Some (ct,f)" by simp
           have x14: "(ct, f) = (ct, f)" by simp
           from e'_def have x15: "e' = ffold_init ct (emptyEnv (sender e) (address e) val)
(fmdom ct)" by simp
           have x16: "get st'''' = Normal (st'''', st'''')" by simp
           have x17: "modify (λst. st⦇accounts := acc, stack := emptyStore, memory :=
emptyStore⦈) st'''' = Normal ((), st''''⦇accounts := acc, stack := emptyStore, memory := emptyStore⦈)"
by simp

           from ‘?A2‘ have "fmlookup (storage st'''') (STR ''Victim'') = Some s" using
st''_def st'''_def st''''_def by simp
           moreover from ‘?A2‘ have "ReadL_int (accessBalance (accounts st'''') (STR
''Victim'')) - (SUMM s + ReadL_int val) ≥ bal ∧ bal ≥ 0"  using st''_def st'''_def st''''_def by simp
           with **** have "ReadL_int (accessBalance acc (STR ''Victim'')) - SUMM s ≥ bal ∧
bal ≥ 0" by simp
           then have "ReadL_int (accessBalance (accounts (st''''⦇accounts := acc,
stack:=emptyStore, memory:=emptyStore⦈)) (STR ''Victim'')) - SUMM s ≥ bal ∧ bal ≥ 0" by simp
           moreover from ‘?A5‘ ad have "sender e ≠ (STR ''Victim'')" by simp
           with e'_def have "address e' ≠ STR ''Victim''" using ffold_init_ad_same[of ct
"(emptyEnv (sender e) (address e) val)" "(fmdom ct)" e'] by simp
           ultimately have "frame bal st'''''" using 0 ***** 43(3)[OF x1 x2 x3 x4 x5 x6 x7
x8 x9 x10 x11 x12 x13 x14 x15 x16 x17] ‘?A3‘ apply safe by (auto simp add:frame_def)
           with "******" show ?RHS by (auto simp add:frame_def)
          qed
         qed
        qed
       qed
      next
       show ?B (is "∀s bal x. ?LHS s bal x")
       proof (rule allI[OF allI[OF allI]])
        fix s bal x
        show "?LHS s bal x" (is "?LHS ⟶ ?RHS")
        proof
         assume ?LHS
         then show ?RHS by simp
        qed
       qed
      next
       show ?C (is "∀s bal. ?LHS s bal")
       proof (rule allI[OF allI])
        fix s bal
        show "?LHS s bal" (is "?LHS ⟶ ?RHS")
        proof
         assume ?LHS
         then show ?RHS by simp
        qed
       qed
      qed
     qed
    qed
   qed
  qed
 qed
next
 case (44 id0 tp ex smt e_p e_v cd st)
```

```
  show ?case (is "?LHS ⟶ ?RHS")
  proof
    assume 0: "fmlookup e_p STR ''Victim'' = Some (victim, SKIP)"
    show ?RHS (is "∀st6'. ?RHS st6'")
    proof
      fix st6'
      show "?RHS st6'" (is "?LHS ⟶ ?RHS")
      proof
        assume *: "stmt (BLOCK ((id0, tp), ex) smt) e_p e_v cd st = Normal ((), st6')"
        show ?RHS (is "?LHS ∧ ?RHS")
        proof
          show "?LHS"
          proof
            assume asm: "address e_v ≠ STR ''Victim''"
            show "∀bal. frame bal st ⟶ frame bal st6'"
            proof
              fix bal
              show "frame bal st ⟶ frame bal st6'"
              proof
                assume ff: "frame bal st"
                with * have a1: "(applyf (costs(BLOCK ((id0, tp), ex) smt) e_p e_v cd) ≫= (λg. assert Gas (λst. gas st ≤ g) (modify (λst. st⦇gas := gas st - g⦈)))) st = Normal ((), st⦇gas := gas st - costs (BLOCK ((id0, tp), ex) smt) e_p e_v cd st⦈)" by auto
                from * ff have f1: "frame bal (st⦇gas := gas st - costs (BLOCK ((id0, tp), ex) smt) e_p e_v cd st⦈)" by (simp add:frame_def)

                show "frame bal st6'"
                proof (cases ex)
                  case None
                  with * obtain cd' e' st' where **:"decl id0 tp None False cd (memory (st⦇gas := gas st - costs (BLOCK ((id0, tp), ex) smt) e_p e_v cd st⦈)) cd e_v (st⦇gas := gas st - costs (BLOCK ((id0, tp), ex) smt) e_p e_v cd st⦈) = Normal ((cd', e'), st')" by (auto split:result.split_asm if_split_asm)
                  with * have f2: "frame bal st'" using decl_frame[OF f1 **] by simp
                  moreover from * None ** obtain st'' where ***: "stmt smt e_p e' cd' st' = Normal ((), st'')" by (simp split:if_split_asm)
                  moreover from ** have ad: "address e' = address e_v" using decl_gas_address by simp
                  moreover from *** asm f2 ad 0 have "frame bal st''" using 44(3)[OF a1 None _ **, of cd' e'] by (simp add:frame_def)
                  moreover from * None ** *** have "st6' = st''" by (auto split:if_split_asm)
                  ultimately show ?thesis using f1 asm by auto
                next
                  case (Some ex')
                  with * obtain v t st' where **:"expr ex' e_p e_v cd (st⦇gas := gas st - costs (BLOCK ((id0, tp), ex) smt) e_p e_v cd st⦈) = Normal ((v, t), st')" by (auto split:result.split_asm if_split_asm)
                  with * f1 Some have f2: "frame bal st'" using 44(1)[OF a1 Some] asm 0 by simp
                  moreover from Some * ** obtain cd' e' st'' where ***:"decl id0 tp (Some (v,t)) False cd (memory st') cd e_v st' = Normal ((cd', e'), st'')" by (auto split:result.split_asm if_split_asm)
                  with * have f3: "frame bal st''" using decl_frame[OF f2 ***] by simp
                  moreover from ** *** have ad: "address e' = address e_v" using decl_gas_address by simp
                  moreover from * Some ** *** obtain st''' where ****: "stmt smt e_p e' cd' st'' = Normal ((), st''')" by (simp split:if_split_asm)
                  moreover from **** asm f3 ad have "frame bal st'''" using 44(2)[OF a1 Some ** _ _ ***] 0 by (simp add:frame_def)
                  moreover from * Some ** *** **** have "st6' = st'''" by (auto split:if_split_asm)
                  ultimately show ?thesis using f1 asm by auto
                qed
              qed
            qed
          qed
        next
          show "?RHS" (is "?LHS ⟶ ?RHS")
```

```
      proof
        assume ad: "address e_v = STR ''Victim''"
        show ?RHS (is "?A ∧ (?B ∧ ?C)")
        proof (rule conj3)
          show ?A (is "∀s val bal x. ?LHS s val bal x")
          proof (rule allI[OF allI[OF allI[OF allI]]])
            fix s val bal x
            show "?LHS s val bal x" (is "?LHS ⟶ ?RHS")
            proof
              assume ?LHS
              then show ?RHS by simp
            qed
          qed
        next
          show ?B (is "∀s bal x. ?LHS s bal x")
          proof (rule allI[OF allI[OF allI]])
            fix s bal x
            show "?LHS s bal x" (is "?LHS ⟶ ?RHS")
            proof
              assume ?LHS
              then show ?RHS by simp
            qed
          qed
        next
          show ?C (is "∀s bal. ?LHS s bal")
          proof (rule allI[OF allI])
            fix s bal
            show "?LHS s bal" (is "?LHS ⟶ ?RHS")
            proof
              assume ?LHS (is "?A1 ∧ ?A2 ∧ ?A3 ∧ ?A4 ∧ ?A5")
              then have ?A1 and ?A2 and ?A3 and ?A4 and ?A5 by auto

              define st'' where "st'' = st⦇gas := gas st - costs keep e_p e_v cd st⦈"
              with `?A2` `?A3` have 00: "fmlookup (storage st'') (STR ''Victim'') = Some s"
                and **: "ReadL_int (accessBalance (accounts st'') (STR ''Victim'')) - (SUMM s) ≥
bal ∧ bal ≥ 0 ∧ POS s" by simp+

              from `?A1` * st''_def obtain v t st''' cd' e' st'''' st'''''
                where ***: "expr mylval e_p e_v cd st'' = Normal ((v,t), st''')"
                  and ****: "decl (STR ''bal'') (Value (TUInt 256)) (Some (v, t)) False cd (memory
st''') cd e_v st''' = Normal ((cd', e'),st'''')"
                  and *****: "stmt comp e_p e' cd' st'''' = Normal ((), st''''')"
                  and "st6' = st'''''"
                by (auto split:if_split_asm result.split_asm)

              obtain s''' where
                  f1: "fmlookup (storage st''') (STR ''Victim'') = Some s'''"
                  and v_def: "v = KValue (accessStorage (TUInt 256) (sender e_v + (STR ''.'' + STR
''balance'')) s''')"
                  and t_def: "t = Value (TUInt 256)"
                  and f2: "ReadL_int (accessBalance (accounts st''') (STR ''Victim'')) - (SUMM s''')
≥ bal ∧ bal ≥ 0 ∧ POS s'''"
                using securelval[OF *** `?A4` 00 ** ad] by auto

              with **** obtain s'''' where
                *******: "fmlookup (storage st'''') (STR ''Victim'') = Some s''''"
                and bbal: "ReadL_int (accessBalance (accounts st'''') (STR ''Victim'')) - (SUMM
s'''') ≥ bal ∧ bal ≥ 0 ∧ POS s''''" using decl_frame frame_def by auto

              from ad `?A5` have
                ad2: "address e' = STR ''Victim''"
                and ss: "sender e'≠address e'" using decl_gas_address[OF ****] by auto

              then obtain x where
```

```
                  ******: "fmlookup (denvalue e') (STR ''bal'') = Some (Value (TUInt 256), (Stackloc
x))"
                  and lkup: "fmlookup (denvalue e') STR ''balance'' = Some (Storage (STMap TAddr
(STValue (TUInt 256))), Storeloc STR ''balance'')"
                  and "accessStore x (stack st'''') = Some (KValue (accessStorage (TUInt 256) (sender
e_v + (STR ''.'' + STR ''balance'')) s''''))"
                proof -
                  have "Valuetypes.convert (TUInt 256) (TUInt 256) (accessStorage (TUInt 256) (sender
e_v + (STR ''.'' + STR ''balance'')) s''') = Some (accessStorage (TUInt 256) (sender e_v + (STR ''.'' +
STR ''balance'')) s''', TUInt 256)" by simp
                  with **** v_def t_def have "append (STR ''bal'') (Value (TUInt 256)) (KValue
(accessStorage (TUInt 256) (sender e_v + (STR ''.'' + STR ''balance'')) s''')) cd e_v st''' = Normal
((cd', e'),st'''')" by simp
                  with f1 v_def t_def have st''''_def: "st'''' = st'''(|stack := push v (stack
st''')|)" and "e' = updateEnv (STR ''bal'') t (Stackloc (ShowL_nat (toploc (stack st''')))) e_v" by auto
                  moreover from ******* f1 st''''_def have "Some (KValue (accessStorage (TUInt 256)
(sender e_v + (STR ''.'' + STR ''balance'')) s''')) = Some (KValue (accessStorage (TUInt 256) (sender e_v
+ (STR ''.'' + STR ''balance'')) s''''))" by simp
                  ultimately show ?thesis using t_def v_def `?A4` that by simp
                qed
                with decl_gas_address **** have sck: "accessStore x (stack st'''') = Some (KValue
(accessStorage (TUInt 256) (sender e' + (STR ''.'' + STR ''balance'')) s''''))" by simp

                from * have a1: "(applyf (costs(BLOCK ((id0, tp), ex) smt) e_p e_v cd) ≫= (λg. assert
Gas (λst. gas st ≤ g) (modify (λst. st(|gas := gas st - g|)))) st = Normal ((), st(|gas := gas st -
costs (BLOCK ((id0, tp), ex) smt) e_p e_v cd st|))" by auto
                from `?A1` have a2: "ex = Some (LVAL (Ref STR ''balance'' [SENDER]))" by simp
                from `?A1` *** have a3: "local.expr (LVAL (Ref STR ''balance'' [SENDER])) e_p e_v cd
                     (st(|gas := gas st - costs (BLOCK ((id0, tp), ex) smt) e_p e_v cd st|)) =
                   Normal ((v, t), st''')" using st''_def by simp
                from `?A1` **** have a4: "decl id0 tp (Some (v, t)) False cd (memory st''') cd e_v
st''' = Normal ((cd', e'), st'''')" by simp
                from `?A1` ***** `st6' = st'''''` have a5: "local.stmt smt e_p e' cd' st'''' = Normal
((), st6')" by simp
                show "(∃s'. fmlookup (storage st6') STR ''Victim'' = Some s' ∧
                        ReadL_int (accessBalance (accounts st6') (STR ''Victim'')) - (SUMM s') ≥ bal ∧
bal ≥ 0 ∧ POS s')"
                  using 44(2)[OF a1 a2 a3 _ _ a4, of cd' e'] 0 a5 ad2 `?A1` ******* bbal ****** lkup
sck `?A4` ss apply safe by auto
              qed
            qed
          qed
        qed
      qed
    qed
  qed
 qed
qed

corollary final1:
  assumes "fmlookup ep (STR ''Victim'') = Some (victim, SKIP)"
      and "stmt (EXTERNAL (ADDRESS (STR ''Victim'')) (STR ''withdraw'') [] val) ep env cd st =
Normal((), st')"
      and "address env ≠ (STR ''Victim'')"
      and "frame bal st"
      shows "frame bal st'"
  using assms secure(7)[of ep "(EXTERNAL (ADDRESS (STR ''Victim'')) (STR ''withdraw'') [] val)" env cd
st] by simp

corollary final2:
  assumes "fmlookup ep (STR ''Victim'') = Some (victim, SKIP)"
      and "stmt (EXTERNAL (ADDRESS (STR ''Victim'')) (STR ''deposit'') [] val) ep env cd st =
Normal((), st')"
      and "address env ≠ (STR ''Victim'')"
```

```
      and "frame bal st"
      shows "frame bal st'"
  using assms secure(7)[of ep "(EXTERNAL (ADDRESS (STR ''Victim'')) (STR ''deposit'') [] val)" env cd
st] by simp

end

end
```

Bibliography

[1] The Bitcon market capitalisation. URL https://coinmarketcap.com/currencies/bitcoin/. Last checked on 2021-05-04.

[2] D. Marmsoler and A. D. Brucker. A denotational semantics of Solidity in Isabelle/HOL. In R. Calinescu and C. Pasareanu, editors, *Software Engineering and Formal Methods (SEFM)*, Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2021. ISBN 3-540-25109-X. URL https://www.brucker.ch/bibliography/abstract/marmsoler.ea-solidity-semantics-2021.

[3] D. Marmsoler and A. D. Brucker. Conformance testing of formal semantics using grammar-based fuzzing. In L. Kovacs and K. Meinke, editors, *TAP 2022: Tests And Proofs*, Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2022. ISBN 978-3-642-38915-3. URL https://www.brucker.ch/bibliography/abstract/marmsoler.ea-conformance-2022.

[4] Online. Solidity documentation. https://solidity.readthedocs.io/en/latest.

[5] D. Perez and B. Livshits. Smart contract vulnerabilities: Vulnerable does not imply exploited. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021. URL https://www.usenix.org/conference/usenixsecurity21/presentation/perez.

[6] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger, 2022. Berlin Version 3078285 – 2022-07-13. https://ethereum.github.io/yellowpaper/paper.pdf.