RIPEMD-160 - Verification of a SPARK/ADA Implementation

Fabian Immler

September 19, 2015

Abstract

This work presents a verification of an implementation in SPARK/ADA [1] of the cryptographic hash-function RIPEMD-160. A functional specification of RIPEMD-160 [2] is given in Isabelle/HOL [3]. Proofs for the verification conditions generated by the static-analysis toolset of SPARK certify the functional correctness of the implementation. The verification conditions are translated to Isabelle/HOL with a modified version of Victor-0.8.0 [4].

Contents

1 Introduction 2

2 Specification of RIPEMD-160 3

3 Global Specifications 6
   3.1 Specification of Bit-Operations . . . . . . . . . . . . . . . . . . . . 6
   3.2 Conversions for proof functions . . . . . . . . . . . . . . . . . . . . 7

4 Verification of $f$ 7

5 Verification of $k_l$ 9

6 Verification of $k_r$ 10

7 Arrays in SPARK vs Lists in Isabelle 12
   7.1 Functions vs Lists . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
   7.2 Maximum Element of Lists . . . . . . . . . . . . . . . . . . . . . . 13

8 Verification of $r_l$ 13

9 Verification of $r_r$ 15

10 Verification of $s_l$ 17
1 Introduction

The directory ada contains the sourcecode which has been verified against its specification in Isabelle/HOL (close to its pseudocode definition from [2]) in the following. The SPARK-code contains annotations with so called proof functions. The following proof functions (declared in ada/rmd.ads) are specified in Isabelle/HOL:

- bit_and
- bit_or
- bit_xor
- wordops_rotate_left
- f
- k_l
- k_r
- r_l
- r_r
- s_l
- s_r
- steps
- round
- rounds
- rmd_hash

From the annotations in the SPARK-code, verification conditions were generated using SPARK-GPL-2010 (http://libre.adacore.com/libre/download/):

$spark -vcg -rules=lazy ada/shadow/interfaces.ads ada/wordops.ads ada/rmd.ads ada/rmd.adb$

A slightly modified Version of VICTOR [4] translated these verification conditions to Isabelle (the results can be found in the theories ending with
Obligation and Declaration). Definitions for the roof-functions are given in the theories with the suffix Specification and the proofs are given in the theories ending in User.

2 Specification of RIPEMD-160

theory RMD
imports ~~/src/HOL/Word/Word
begin

type-synonym word32 = 32 word
type-synonym byte = 8 word
type-synonym perm = nat => nat
type-synonym chain = word32 * word32 * word32 * word32 * word32
type-synonym block = nat => word32
type-synonym message = nat => block

definition f::[nat, word32, word32, word32] => word32
where
  f j x y z =
  (if ( 0 <= j & j <= 15) then x XOR y XOR z 
  else if (16 <= j & j <= 31) then (x AND y) OR (NOT x AND z) 
  else if (32 <= j & j <= 47) then (x OR NOT y) XOR z 
  else if (48 <= j & j <= 63) then (x AND z) OR (y AND NOT z) 
  else if (64 <= j & j <= 79) then x XOR (y OR NOT z) 
  else 0)

definition K::nat => word32
where
  K j =
  (if ( 0 <= j & j <= 15) then 0x00000000 
  else if (16 <= j & j <= 31) then 0x5A827999 
  else if (32 <= j & j <= 47) then 0x6ED9EBA1 
  else if (48 <= j & j <= 63) then 0x8F1BBCDC 
  else if (64 <= j & j <= 79) then 0xA953FD4E 
  else 0)

definition K':::nat => word32
where
  K' j =
  (if ( 0 <= j & j <= 15) then 0x50A28BE6 
  else if (16 <= j & j <= 31) then 0x5C4DD124 
  else if (32 <= j & j <= 47) then 0x6D703EF3 
  else if (48 <= j & j <= 63) then 0x7A6D76E9 
  else if (64 <= j & j <= 79) then 0x00000000

There are some slight superficial differences between the original translated files and the ones included here, in order to conform to current Isabelle practice.
\text{else } 0\) \)

\textbf{definition} \( r\text{-list} :: \text{nat list} \)

\textbf{where} \( r\text{-list} = [\)
\( 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 7, 4, 13, 1, 10, 6, 15, 3, 12, 0, 9, 5, 2, 14, 11, 8, 3, 10, 14, 4, 9, 15, 8, 1, 2, 7, 0, 6, 13, 11, 5, 12, 1, 9, 11, 10, 0, 8, 12, 4, 13, 3, 7, 15, 14, 5, 6, 2, 4, 0, 5, 9, 7, 12, 2, 10, 14, 1, 3, 8, 11, 6, 15, 13 \)

\textbf{definition} \( r'\text{-list} :: \text{nat list} \)

\textbf{where} \( r'\text{-list} = [\)
\( 5, 14, 7, 0, 9, 2, 11, 4, 13, 6, 15, 8, 1, 10, 3, 12, 6, 11, 3, 7, 0, 13, 5, 10, 14, 15, 8, 12, 4, 9, 1, 2, 15, 5, 1, 3, 7, 14, 6, 9, 11, 8, 12, 2, 10, 0, 4, 13, 8, 6, 4, 1, 3, 11, 15, 0, 5, 12, 2, 13, 9, 7, 10, 14, 12, 15, 10, 4, 1, 5, 8, 7, 6, 2, 13, 14, 0, 3, 9, 11 \)

\textbf{definition} \( r :: \text{perm} \)

\textbf{where} \( r \ j = r\text{-list} ! j \)

\textbf{definition} \( r' :: \text{perm} \)

\textbf{where} \( r' \ j = r'\text{-list} ! j \)

\textbf{definition} \( s\text{-list} :: \text{nat list} \)

\textbf{where} \( s\text{-list} = [\)
\( 11, 14, 15, 12, 5, 8, 7, 9, 11, 13, 14, 15, 6, 7, 9, 8, 7, 6, 8, 13, 11, 9, 7, 15, 7, 12, 15, 9, 11, 7, 13, 12, 11, 13, 6, 7, 14, 9, 13, 15, 14, 8, 13, 6, 5, 12, 7, 5, 11, 12, 14, 15, 14, 15, 9, 8, 9, 14, 5, 6, 8, 6, 5, 12, 9, 15, 5, 11, 6, 8, 13, 12, 5, 12, 13, 14, 11, 8, 5, 6 \)

\textbf{definition} \( s'\text{-list} :: \text{nat list} \)

\textbf{where} \( s'\text{-list} = [\)
\( 8, 9, 9, 11, 13, 15, 15, 5, 7, 7, 8, 11, 14, 14, 12, 6, 9, 13, 15, 7, 12, 8, 9, 11, 7, 7, 12, 7, 6, 15, 13, 11, 9, 7, 15, 11, 8, 6, 6, 14, 12, 13, 5, 14, 13, 13, 7, 5, 15, 5, 8, 11, 14, 14, 6, 14, 6, 9, 12, 9, 12, 5, 15, 8, 8, 5, 12, 9, 12, 5, 14, 6, 8, 13, 6, 5, 15, 13, 11, 11 \)

\textbf{definition} \( s :: \text{perm} \)

\textbf{where} \( s \ j = s\text{-list} ! j \)

\textbf{definition} \( s' :: \text{perm} \)

\textbf{where} \( s' \ j = s'\text{-list} ! j \)

4
definition h0-0::word32 where h0-0 = 0x67452301
definition h1-0::word32 where h1-0 = 0xEFCDAB89
definition h2-0::word32 where h2-0 = 0x98BADCFE
definition h3-0::word32 where h3-0 = 0x10325476
definition h4-0::word32 where h4-0 = 0xC3D2E1F0
definition h-0::chain where
  h-0 = (h0-0, h1-0, h2-0, h3-0, h4-0)
definition step-l :: [block, chain, nat] => chain
  where
    step-l X c j = (let (A, B, C, D, E) = c in
                     ((* A *) E,
                      (* B *) word-rotl (s j) (A + f j B C D + X (r j) + K j) + E,
                      (* C *) B,
                      (* D *) word-rotl 10 C,
                      (* E *) D))
definition step-r :: [block, chain, nat] => chain
  where
    step-r X c' j = (let (A', B', C', D', E') = c' in
                     ((* A' *) E',
                      (* B' *) word-rotl (s' j) (A' + f (79 - j) B' C' D' + X (r' j) + K' j) + E',
                      (* C' *) B',
                      (* D' *) word-rotl 10 C',
                      (* E' *) D'))
definition step-both :: [block, chain * chain, nat] => chain * chain
  where
    step-both X cc j = (case cc of (c, c') =>
                         (step-l X c j, step-r X c' j))
definition steps::[block, chain * chain, nat] => chain * chain
  where
    steps X cc i = foldl (step-both X) cc [0..<i]
definition round::[block, chain] => chain
  where
    round X h = (let (h0, h1, h2, h3, h4) = h in
                 h-0 = (h0, h1, h2, h3, h4))
let \((A, B, C, D, E, A', B', C', D', E')\) = steps \((h, h)\) 80 in
\((\ast h0 \ast) h1 + C + D',
(\ast h1 \ast) h2 + D + E',
(\ast h2 \ast) h3 + E + A',
(\ast h3 \ast) h4 + A + B',
(\ast h4 \ast) h0 + B + C')\)

**definition** rmd-body::[message, chain, nat] => chain
where
rmd-body X h i = round (X i) h

**definition** rounds::message => chain => nat => chain
where
rounds X h i = foldl (rmd-body X) h-0 [0..<i]

**definition** rmd :: message => nat => chain
where
rmd X len = rounds X h-0 len

end

## 3 Global Specifications

**theory** Global-Specification
**imports** RMD

**begin**

SPARK has only one integer-type, therefore type-conversions are needed in order to specify the proof-functions in Isabelle.

### 3.1 Specification of Bit-Operations

The proof-functions for SPARK's bit-operations are specified with HOL-Word

**abbreviation** bit--and' :: int => int => int where
bit--and' m n == uint ((word-of-int m::word32) AND word-of-int n)

**abbreviation** bit--or' :: int => int => int where
bit--or' m n == uint ((word-of-int m::word32) OR word-of-int n)

**abbreviation** bit--xor' :: int => int => int where
bit--xor' m n == uint ((word-of-int m::word32) XOR word-of-int n)

**abbreviation** rotate-left' :: int => int => int where
rotate-left' i w == uint (word-rotl (nat i) (word-of-int w::word32))

This is how SPARK treats the bitwise not
\textbf{lemma} bit-not-spark-def[simp]:
\[(\text{word-of-int } (4294967295 - x) :: \text{word32}) = \text{NOT} \ (\text{word-of-int } x)\]
\langle\text{proof}\rangle

3.2 Conversions for proof functions

Here, the proof-functions declared in the SPARK-Annotations are mapped to the corresponding parts of the Isabelle-Specification.

abbreviation \(k-l'\) :: \(\text{int} \Rightarrow \text{int}\) where
\(k-l' \ j \ = \ \text{uint} \ (K \ (\text{nat} \ j))\)

abbreviation \(k-r'\) :: \(\text{int} \Rightarrow \text{int}\) where
\(k-r' \ j \ = \ \text{uint} \ (K' \ (\text{nat} \ j))\)

abbreviation \(r-l'\) :: \(\text{int} \Rightarrow \text{int}\) where
\(r-l' \ j \ = \ \text{int} \ (r \ (\text{nat} \ j))\)

abbreviation \(r-r'\) :: \(\text{int} \Rightarrow \text{int}\) where
\(r-r' \ j \ = \ \text{int} \ (r' \ (\text{nat} \ j))\)

abbreviation \(s-l'\) :: \(\text{int} \Rightarrow \text{int}\) where
\(s-l' \ j \ = \ \text{int} \ (s \ (\text{nat} \ j))\)

abbreviation \(s-r'\) :: \(\text{int} \Rightarrow \text{int}\) where
\(s-r' \ j \ = \ \text{int} \ (s' \ (\text{nat} \ j))\)

abbreviation \(f'\) :: \(\text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int}\) where
\(f' \ j \ x \ y \ z \ = \end{lemma}

4 Verification of \(f\)

\textbf{theory} F-Spark-Specification
\textbf{imports} F-Spark-Declaration Global-Specification

begin

abbreviation \(\text{bit--and}'\) :: \([\text{int}, \text{int}] \Rightarrow \text{int}\) where
\(\text{bit--and}' \ = \ \text{Global-Specification.bit--and}'\)

abbreviation \(\text{bit--or}'\) :: \([\text{int}, \text{int}] \Rightarrow \text{int}\) where
\(\text{bit--or}' \ = \ \text{Global-Specification.bit--or}'\)

abbreviation \(\text{bit--xor}'\) :: \([\text{int}, \text{int}] \Rightarrow \text{int}\) where
\(\text{bit--xor}' \ = \ \text{Global-Specification.bit--xor}'\)

abbreviation \(f'\) :: \(\text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow \text{int}\) where
\(f' \ = \ \text{Global-Specification.f}'\)

end
theory F-Spark-User
imports F-Spark-Specification F-Spark-Declaration
begin

lemma goal2':
  shows 0 <= bit-or' (bit-and' x'' y'') (bit-and' (4294967295 - x'') z'')
  ⟨proof⟩

lemma goal2'2:
  shows bit-or' (bit-and' x'' y'') (bit-and' (4294967295 - x'') z'') <= 4294967295
  ⟨proof⟩

lemma goal3'1:
  shows 0 <= bit-xor' (bit-or' x'' (4294967295 - y'')) z''
  ⟨proof⟩

lemma goal3'2:
  shows bit-xor' (bit-or' x'' (4294967295 - y'')) z'' <= 4294967295
  ⟨proof⟩

lemma goal4'1:
  shows 0 <= bit-or' (bit-and' x'' z'') (bit-and' y'' (4294967295 - z''))
  ⟨proof⟩

lemma goal4'2:
  shows bit-or' (bit-and' x'' z'') (bit-and' y'' (4294967295 - z'')) <= 4294967295
  ⟨proof⟩

lemma goal5'1:
  shows 0 <= bit-xor' x'' (bit-or' y'' (4294967295 - z''))
  ⟨proof⟩

lemma goal5'2:
  shows bit-xor' x'' (bit-or' y'' (4294967295 - z'')) <= 4294967295
  ⟨proof⟩

lemma goal6'1:
  assumes H8: j'' <= (15 :: int)
  shows bit-xor' x'' (bit-xor' y'' z'') = f' j'' x'' y'' z''
  ⟨proof⟩

lemma goal7'1:
  assumes H7: (16 :: int) <= j''
  assumes H8: j'' <= (31 :: int)
  shows bit-or' (bit-and' x'' y'') (bit-and' (4294967295 - x'') z'') = f' j'' x'' y'' z''
  ⟨proof⟩
lemma goal8'1:
  assumes H7: 32 <= j''
  assumes H8: j'' <= 47
  shows \text{bit-xor}' (\text{bit-or}' x'' (4294967295 \ - y'')) z'' = f' j'' x'' y'' z''
⟨proof⟩

lemma goal9'1:
  assumes H7: 48 <= j''
  assumes H8: j'' <= 63
  shows \text{bit-or}' (\text{bit-and}' x'' z'') (\text{bit-and}' y'' (4294967295 \ - z'')) = f' j'' x'' y'' z''
⟨proof⟩

lemma goal10'1:
  assumes H2: j'' <= 70
  assumes H12: 63 < j''
  shows \text{bit-xor}' x'' (\text{bit-or}' y'' (4294967295 \ - z'')) = f' j'' x'' y'' z''
⟨proof⟩

lemmas userlemmas =
goal2'1
goal2'2
goal3'1
goal3'2
goal4'1
goal4'2
goal5'1
goal5'2
goal6'1
goal7'1
goal8'1
goal9'1
goal10'1

end

5 Verification of $k_l$

theory K-L-Spark-Specification
imports K-L-Spark-Declaration Global-Specification

begin

abbreviation k-l' :: int => int where
  k-l' == Global-Specification.k-l'

end

theory K-L-Spark-User
imports K-L-Spark-Specification K-L-Spark-Declaration
lemma goal6'1:
  fixes j::int
  assumes H1: \( 0 \leq j \)
  assumes H2: \( j \leq 15 \)
  shows \( 0 = k-l' j \)
  ⟨proof⟩

lemma goal7'1:
  fixes j::int
  assumes H1: \( 16 \leq j \)
  assumes H2: \( j \leq 31 \)
  shows \( 1518500249 = k-l' j \)
  ⟨proof⟩

lemma goal8'1:
  assumes H1: \( 32 \leq j'' \)
  assumes H2: \( j'' \leq 47 \)
  shows \( 1859775393 = k-l' j'' \)
  ⟨proof⟩

lemma goal9'1:
  assumes H1: \( 48 \leq j'' \)
  assumes H2: \( j'' \leq 63 \)
  shows \( 2400959708 = k-l' j'' \) (is ?C1)
  ⟨proof⟩

lemma goal10'1:
  assumes H2: \( j'' \leq 79 \)
  assumes H6: \( 63 \leq j'' \)
  shows \( 2840853838 = k-l' j'' \) (is ?C1)
  ⟨proof⟩

lemmas userlemmas =
  goal6'1
  goal7'1
  goal8'1
  goal9'1
  goal10'1
end

6 Verification of \( k_r \)

theory K-R-Spark-Specification
imports K-R-Spark-Declaration Global-Specification
begin

abbreviation k-r' :: int => int where
  k-r' == Global-Specification.k-r'

end

theory K-R-Spark-User
imports K-R-Spark-Specification K-R-Spark-Declaration
begin

lemma goal6'1:
  assumes H1: (0 :: int) <= j''
  assumes H2: j'' <= (15 :: int)
  shows (1352829926 :: int) = k-r' j'' (is ?C1)
  ⟨proof⟩

lemma goal7'1:
  assumes H1: (16 :: int) <= j''
  assumes H2: j'' <= (31 :: int)
  shows (1548603684 :: int) = k-r' j'' (is ?C1)
  ⟨proof⟩

lemma goal8'1:
  assumes H1: (32 :: int) <= j''
  assumes H2: j'' <= (47 :: int)
  shows (1836072691 :: int) = k-r' j'' (is ?C1)
  ⟨proof⟩

lemma goal9'1:
  assumes H1: (48 :: int) <= j''
  assumes H2: j'' <= (63 :: int)
  shows (2053994217 :: int) = k-r' j'' (is ?C1)
  ⟨proof⟩

lemma goal10'1:
  assumes H2: j'' <= (79 :: int)
  assumes H6: (63 :: int) < j''
  shows (0 :: int) = k-r' j'' (is ?C1)
  ⟨proof⟩

lemmas userlemmas =
  goal6'1

11
Arrays in SPARK vs Lists in Isabelle

7.1 Functions vs Lists

Arrays defined in SPARK are represented as functions in Isabelle. In the specification, it is more convenient to use lists. Therefore it is a common task to prove equivalences like \( \forall i \leq \text{length } l. \ l! i = f\ i\ ), where \( l \) is the list specified in Isabelle and \( f \) the function corresponding to the array defined in SPARK.

Constructing a function from a list makes things easier for the simplifier, otherwise the definition of the list would need to be unfolded (\( \text{length } l \)) times what yields to efficiency-problems.

**Definition:**

\[
\text{list-to-fun} \quad \text{where} \\
\text{list-to-fun} \; \text{[]} = f \\
\text{list-to-fun} \; (a \# \text{xs}) \; i \; f = (\text{list-to-fun} \; \text{xs} \; (i + 1) \; f) \; (i := (\text{int} \; a))
\]

**Lemma:** \( \text{nth-list-to-fun-eq-aux} \):

\[
\text{assumes } i-0 \leq i \text{ and } i < \text{length } l + i-0 \\
\text{shows } \text{int} \; (l! (i - i-0)) = (\text{list-to-fun} \; l \; (\text{int} \; i-0) \; f) \; (\text{int } i)
\]

**Lemma:** \( \text{nth-list-to-fun-eq} \):

\[
\text{assumes } 0 \leq i \text{ and } i < \text{length } l \\
\text{shows } \text{int} \; (l! i) = (\text{list-to-fun} \; l \; 0 \; f) \; (\text{int } i)
\]

A tail-recursive definition makes it even more efficient.

**Definition:**

\[
\text{list-to-fun-eff} \quad \text{where} \\
\text{list-to-fun-eff} \; \text{[]} = (f::\text{int} \Rightarrow \text{int}) = f \\
\text{list-to-fun-eff} \; (a \# \text{xs}) \; i \; f = (\text{list-to-fun-eff} \; \text{xs} \; (i + 1) \; f) \; (f(i := (\text{int} \; a)))
\]

**Lemma:** \( \text{list-to-fun-id} \):

\[
\text{assumes } i-0 > i \\
\text{shows } \text{list-to-fun-eff} \; l \; (\text{int} \; i-0) \; f \; (\text{int } i) = f \; (\text{int } i)
\]

**Lemma:** \( \text{nth-list-to-fun-eff-eq-aux} \):
assumes $i-0 \leq i$ and $i < \text{length } l + i-0$
shows $\text{int } (l ! (i - i-0)) = (\text{list-to-fun-eff } l (\text{int } i-0) f) (\text{int } i)$
(proof)

lemma nth-list-to-fun-eff-eq:
assumes $0 \leq i$ and $i < \text{length } l$
shows $\text{int } (l ! i) = (\text{list-to-fun-eff } l 0 f) (\text{int } i)$
(proof)

7.2 Maximum Element of Lists

The following lemmas help the simplifier to prove properties about maximal elements of a list. It is easier to calculate the maximum element of a list in an efficient way (using fold) and prove the correctness of this calculation.

lemma fold-max-leq:
fixes $i, j :: \text{nat}$
assumes $i \leq j$
shows $\text{foldl } \text{max } i l \leq \text{foldl } \text{max } j l$
(proof)

lemma fold-max-lower:
fixes $i :: \text{nat}$
shows $i \leq \text{foldl } \text{max } i l$
(proof)

lemma list-max:
fixes $l :: \text{nat list}$
fixes $i :: \text{nat}$
assumes $0 \leq l ! i$
assumes $0 \leq i$
assumes $i < \text{length } l$
shows $l ! i \leq \text{foldl } \text{max } 0 l$
(proof)

lemma list-max-int:
assumes $l ! \text{nat } j \leq \text{foldl } \text{max } 0 l$
assumes $\text{foldl } \text{max } 0 l = \text{nat } U$
assumes $0 \leq j$
assumes $0 \leq U$
shows $\text{int } (l ! \text{nat } j) \leq U$
(proof)

end

8 Verification of $r_l$

theory R-L-Spark-Specification
imports Global-Specification R-L-Spark-Declaration

begin

abbreviation r-l' :: int => int where
r-l' == Global-Specification.r-l'
end

theory R-L-Spark-User
imports
R-L-Spark-Specification
R-L-Spark-Specification
Global-User
begin

lemma goal1':
assumes 0 <= j''
assumes j'' <= 79
shows (block-permutation-default-arr''
(0 := 0, 1 := 1, 2 := 2, 3 := 3, 4 := 4, 5 := 5, 6 := 6, 7 := 7,
8 := 8, 9 := 9, 10 := 10, 11 := 11, 12 := 12, 13 := 13, 14 := 14,
15 := 15, 16 := 17, 17 := 18, 18 := 19, 19 := 20, 20 := 21 := 6,
22 := 15, 23 := 24 := 12, 25 := 0, 26 := 9, 27 := 5, 28 := 2,
29 := 14, 30 := 31 := 8, 32 := 33 := 10, 34 := 14, 35 := 4,
36 := 9, 37 := 38 := 8, 39 := 1, 40 := 2, 41 := 7, 42 := 0,
43 := 6, 44 := 13, 45 := 11, 46 := 5, 47 := 12, 48 := 1, 49 := 9,
50 := 11, 51 := 12, 52 := 0, 53 := 8, 54 := 12, 55 := 4, 56 := 13,
57 := 3, 58 := 7, 59 := 15, 60 := 14, 61 := 5, 62 := 6, 63 := 2,
64 := 4, 65 := 0, 66 := 5, 67 := 9, 68 := 7, 69 := 12, 70 := 2,
71 := 10, 72 := 14, 73 := 1, 74 := 3, 75 := 8, 76 := 11, 77 := 6,
78 := 15, 79 := 13))

j'' = R-L-Spark-Specification.r-l' j''
⟨proof⟩

lemma goal2':
assumes 0 <= j''
assumes j'' <= 79
shows 0 <= (block-permutation-default-arr''
(0 := 0, 1 := 1, 2 := 2, 3 := 3, 4 := 4, 5 := 5, 6 := 6, 7 := 7,
8 := 8, 9 := 9, 10 := 10, 11 := 11, 12 := 12, 13 := 13, 14 := 14,
15 := 15, 16 := 17, 17 := 18, 18 := 19, 19 := 20, 20 := 21 := 6,
22 := 15, 23 := 24 := 12, 25 := 0, 26 := 9, 27 := 5, 28 := 2,
29 := 14, 30 := 31 := 8, 32 := 33 := 10, 34 := 14, 35 := 4,
36 := 9, 37 := 38 := 8, 39 := 1, 40 := 2, 41 := 7, 42 := 0,
43 := 6, 44 := 13, 45 := 11, 46 := 5, 47 := 12, 48 := 1, 49 := 9,
50 := 11, 51 := 12, 52 := 0, 53 := 8, 54 := 12, 55 := 4, 56 := 13,
57 := 3, 58 := 7, 59 := 15, 60 := 14, 61 := 5, 62 := 6, 63 := 2,
lemma goal2'3:
  assumes 0 <= j''
  assumes j'' <= 79
  shows (block-permutation---default-arr''
     (0 := 0, 1 := 1, 2 := 2, 3 := 3, 4 := 4, 5 := 5, 6 := 6, 7 := 7,
      8 := 8, 9 := 9, 10 := 10, 11 := 11, 12 := 12, 13 := 13, 14 := 14,
      15 := 15, 16 := 7, 17 := 4, 18 := 13, 19 := 1, 20 := 10, 21 := 6,
      22 := 15, 23 := 3, 24 := 12, 25 := 0, 26 := 9, 27 := 5, 28 := 2,
      29 := 14, 30 := 11, 31 := 8, 32 := 3, 33 := 10, 34 := 14, 35 := 4,
      36 := 9, 37 := 15, 38 := 8, 39 := 1, 40 := 2, 41 := 7, 42 := 0,
      43 := 6, 44 := 13, 45 := 11, 46 := 5, 47 := 12, 48 := 1, 49 := 9,
      50 := 11, 51 := 10, 52 := 0, 53 := 8, 54 := 12, 55 := 4, 56 := 13,
      57 := 3, 58 := 7, 59 := 15, 60 := 14, 61 := 5, 62 := 6, 63 := 2,
      64 := 4, 65 := 0, 66 := 5, 67 := 9, 68 := 7, 69 := 12, 70 := 2,
      71 := 10, 72 := 14, 73 := 1, 74 := 3, 75 := 8, 76 := 11, 77 := 6,
      78 := 15, 79 := 13))
  j'' 
  \leq 15
  \langle proof \rangle

lemmas userlemmas = goal2'1 goal2'2 goal2'3
end

9 Verification of \( r_r \)

theory R-R-Spark-Specification
imports Global-Specification R-R-Spark-Declaration
begin

abbreviation r-r' where
  r-r' == Global-Specification.r-r'
end

theory R-R-Spark-User
imports
  R-R-Spark-Specification
  R-R-Spark-Declaration
  Global-User
begin
lemma goal2'1:
assumes \( 0 \leq j'' \)
assumes \( j'' \leq 79 \)
shows \( \text{block-permutation---default-arr}'' \)
\[
(0 := 5, 1 := 14, 2 := 7, 3 := 0, 4 := 9, 5 := 2, 6 := 11, 7 := 4, \\
8 := 13, 9 := 6, 10 := 15, 11 := 8, 12 := 1, 13 := 10, 14 := 3, \\
15 := 12, 16 := 6, 17 := 11, 18 := 3, 19 := 7, 20 := 0, 21 := 13, \\
22 := 5, 23 := 10, 24 := 14, 25 := 15, 26 := 8, 27 := 12, 28 := 4, \\
29 := 9, 30 := 1, 31 := 2, 32 := 15, 33 := 5, 34 := 1, 35 := 3, \\
36 := 7, 37 := 14, 38 := 6, 39 := 9, 40 := 11, 41 := 8, 42 := 12, \\
43 := 2, 44 := 10, 45 := 0, 46 := 4, 47 := 13, 48 := 8, 49 := 6, \\
50 := 4, 51 := 1, 52 := 3, 53 := 11, 54 := 15, 55 := 0, 56 := 5, \\
57 := 12, 58 := 2, 59 := 13, 60 := 9, 61 := 7, 62 := 10, 63 := 14, \\
64 := 12, 65 := 15, 66 := 10, 67 := 4, 68 := 1, 69 := 5, 70 := 8, \\
71 := 7, 72 := 6, 73 := 2, 74 := 13, 75 := 14, 76 := 0, 77 := 3, \\
78 := 9, 79 := 11)
\]
\[j'' = \text{R-R-Spark-Specification.r-r}'j''\]
{proof}

lemma goal2'2:
assumes \( 0 \leq j'' \)
assumes \( j'' \leq 79 \)
shows \( \text{block-permutation---default-arr}'' \)
\[
(0 := 5, 1 := 14, 2 := 7, 3 := 0, 4 := 9, 5 := 2, 6 := 11, 7 := 4, \\
8 := 13, 9 := 6, 10 := 15, 11 := 8, 12 := 1, 13 := 10, 14 := 3, \\
15 := 12, 16 := 6, 17 := 11, 18 := 3, 19 := 7, 20 := 0, 21 := 13, \\
22 := 5, 23 := 10, 24 := 14, 25 := 15, 26 := 8, 27 := 12, 28 := 4, \\
29 := 9, 30 := 1, 31 := 2, 32 := 15, 33 := 5, 34 := 1, 35 := 3, \\
36 := 7, 37 := 14, 38 := 6, 39 := 9, 40 := 11, 41 := 8, 42 := 12, \\
43 := 2, 44 := 10, 45 := 0, 46 := 4, 47 := 13, 48 := 8, 49 := 6, \\
50 := 4, 51 := 1, 52 := 3, 53 := 11, 54 := 15, 55 := 0, 56 := 5, \\
57 := 12, 58 := 2, 59 := 13, 60 := 9, 61 := 7, 62 := 10, 63 := 14, \\
64 := 12, 65 := 15, 66 := 10, 67 := 4, 68 := 1, 69 := 5, 70 := 8, \\
71 := 7, 72 := 6, 73 := 2, 74 := 13, 75 := 14, 76 := 0, 77 := 3, \\
78 := 9, 79 := 11)
\]
\[j'' \]
{proof}

lemma goal2'3:
assumes \( 0 \leq j'' \)
assumes \( j'' \leq 79 \)
shows \( \text{block-permutation---default-arr}'' \)
\[
(0 := 5, 1 := 14, 2 := 7, 3 := 0, 4 := 9, 5 := 2, 6 := 11, 7 := 4, \\
8 := 13, 9 := 6, 10 := 15, 11 := 8, 12 := 1, 13 := 10, 14 := 3, \\
15 := 12, 16 := 6, 17 := 11, 18 := 3, 19 := 7, 20 := 0, 21 := 13, \\
22 := 5, 23 := 10, 24 := 14, 25 := 15, 26 := 8, 27 := 12, 28 := 4, \\
)
10 Verification of \( s_l \)

theory \( S-L\text{-}Spark\text{-}Specification \)

 imports Global\text{-}Specification S-L\text{-}Spark\text{-}Declaration

begin

abbreviation \( s-l' \) :: \( \text{int} \Rightarrow \text{int} \) where
\( s-l' = \text{Global\text{-}Specification.s-l'} \)

end

theory \( S-L\text{-}Spark\text{-}User \)

 imports
S-L\text{-}Spark\text{-}Specification
S-L\text{-}Spark\text{-}Declaration
Global\text{-}User

begin

lemma \( \text{goal2'1} \):
assumes \( 0 <= j'' \)
assumes \( j'' <= 79 \)
shows (rotate-definition---default-arr'')
\[
\]

\( j'' \leq 15 \)

(proof)

lemmas userlemmas = \( \text{goal2'2 goal2'3 goal2'1} \)

end
lemma \texttt{goal2'2}:
assumes 0 <= \texttt{j''}
assumes \texttt{j''} <= 79
shows 0 <= (rotate-definition---default-arr''
(0 := 11, 1 := 14, 2 := 15, 3 := 12, 4 := 5, 5 := 8, 6 := 7,
7 := 9, 8 := 11, 9 := 13, 10 := 14, 11 := 15, 12 := 6, 13 := 7,
14 := 9, 15 := 8, 16 := 7, 17 := 6, 18 := 8, 19 := 13, 20 := 11,
21 := 9, 22 := 7, 23 := 15, 24 := 7, 25 := 12, 26 := 15, 27 := 9,
28 := 11, 29 := 7, 30 := 13, 31 := 12, 32 := 11, 33 := 13,
34 := 6, 35 := 7, 36 := 14, 37 := 9, 38 := 13, 39 := 15, 40 := 14,
41 := 8, 42 := 13, 43 := 6, 44 := 5, 45 := 12, 46 := 7, 47 := 5,
48 := 11, 49 := 12, 50 := 14, 51 := 15, 52 := 14, 53 := 15,
54 := 9, 55 := 8, 56 := 9, 57 := 14, 58 := 5, 59 := 6, 60 := 8,
61 := 6, 62 := 5, 63 := 12, 64 := 9, 65 := 15, 66 := 5, 67 := 11,
68 := 6, 69 := 8, 70 := 13, 71 := 12, 72 := 5, 73 := 12, 74 := 13,
75 := 14, 76 := 11, 77 := 8, 78 := 5, 79 := 6))
\texttt{j''}

\langle proof \rangle

lemma \texttt{goal2'3}:
assumes 0 <= \texttt{j''}
assumes \texttt{j''} <= 79
shows (rotate-definition---default-arr''
(0 := 11, 1 := 14, 2 := 15, 3 := 12, 4 := 5, 5 := 8, 6 := 7, 7 := 9,
8 := 11, 9 := 13, 10 := 14, 11 := 15, 12 := 6, 13 := 7, 14 := 9,
15 := 8, 16 := 7, 17 := 6, 18 := 8, 19 := 13, 20 := 11, 21 := 9,
22 := 7, 23 := 15, 24 := 7, 25 := 12, 26 := 15, 27 := 9, 28 := 11,
29 := 7, 30 := 13, 31 := 12, 32 := 11, 33 := 13, 34 := 6, 35 := 7,
36 := 14, 37 := 9, 38 := 13, 39 := 15, 40 := 14, 41 := 8, 42 := 13,
43 := 6, 44 := 5, 45 := 12, 46 := 7, 47 := 5, 48 := 11, 49 := 12,
50 := 14, 51 := 15, 52 := 14, 53 := 15, 54 := 9, 55 := 8, 56 := 9,
57 := 14, 58 := 5, 59 := 6, 60 := 8, 61 := 6, 62 := 5, 63 := 12,
64 := 9, 65 := 15, 66 := 5, 67 := 11, 68 := 6, 69 := 8, 70 := 13,
71 := 12, 72 := 5, 73 := 12, 74 := 13, 75 := 14, 76 := 11, 77 := 8,
78 := 5, 79 := 6))
\texttt{j''}

\leq 15
\langle proof \rangle

lemmas userlemmas = \texttt{goal2'2 goal2'3 goal2'1}
11 Verification of \( s_r \)

theory S-R-Spark-Specification
imports Global-Specification S-R-Spark-Declaration

begin

abbreviation \( s_r' :: \text{int} \Rightarrow \text{int} \) where
\( s_r' = \text{Global-Specification.s-r'} \)

end

theory S-R-Spark-User
imports
S-R-Spark-Specification
S-R-Spark-Declaration
Global-User

begin

lemma goal2'1:
assumes 0 <= \( j'' \)
assumes \( j'' <= 79 \)
shows (rotate-definition---default-arr'"
(0 := 8, 1 := 9, 2 := 9, 3 := 11, 4 := 13, 5 := 15, 6 := 15, 7 := 5,
8 := 7, 9 := 7, 10 := 8, 11 := 11, 12 := 14, 13 := 14, 14 := 12,
15 := 6, 16 := 9, 17 := 13, 18 := 15, 19 := 7, 20 := 12, 21 := 8,
29 := 15, 30 := 13, 31 := 11, 32 := 9, 33 := 0, 34 := 7, 35 := 11,
36 := 8, 37 := 6, 38 := 6, 39 := 14, 40 := 12, 41 := 13, 42 := 5,
43 := 14, 44 := 13, 45 := 13, 46 := 7, 47 := 5, 48 := 15, 49 := 5,
50 := 8, 51 := 11, 52 := 14, 53 := 14, 54 := 6, 55 := 14, 56 := 6,
57 := 9, 58 := 12, 59 := 9, 60 := 12, 61 := 5, 62 := 15, 63 := 8,
64 := 8, 65 := 5, 66 := 12, 67 := 9, 68 := 12, 69 := 5, 70 := 14,
71 := 6, 72 := 8, 73 := 13, 74 := 6, 75 := 5, 76 := 15, 77 := 13,
78 := 11, 79 := 11))
\( j'' = \)
S-R-Spark-Specification.s-r' \( j'' \)

(proof)

lemma goal2'2:
assumes 0 <= \( j'' \)
assumes \( j'' <= 79 \)
shows 0 <= (rotate-definition---default-arr'"
(0 := 8, 1 := 9, 2 := 9, 3 := 11, 4 := 13, 5 := 15, 6 := 15, 7 := 5,
8 := 7, 9 := 7, 10 := 8, 11 := 11, 12 := 14, 13 := 14, 14 := 12,
15 := 6, 16 := 9, 17 := 13, 18 := 15, 19 := 7, 20 := 12, 21 := 8,
lemma goal2'3:
  assumes 0 <= j''
  assumes j'' <= 79
  shows (rotate-definition---default-arr'')
    (0 := 8, 1 := 9, 2 := 9, 3 := 11, 4 := 13, 5 := 15, 6 := 15, 7 := 5,
     8 := 7, 9 := 7, 10 := 8, 11 := 11, 12 := 14, 13 := 14, 14 := 12,
     15 := 6, 16 := 9, 17 := 13, 18 := 15, 19 := 7, 20 := 12, 21 := 8,
     29 := 15, 30 := 13, 31 := 11, 32 := 9, 33 := 7, 34 := 15, 35 := 11,
     36 := 8, 37 := 6, 38 := 6, 39 := 14, 40 := 12, 41 := 13, 42 := 5,
     43 := 14, 44 := 13, 45 := 13, 46 := 7, 47 := 5, 48 := 15, 49 := 5,
     50 := 8, 51 := 11, 52 := 14, 53 := 14, 54 := 6, 55 := 14, 56 := 6,
     57 := 9, 58 := 12, 59 := 9, 60 := 12, 61 := 5, 62 := 15, 63 := 8,
     64 := 8, 65 := 5, 66 := 12, 67 := 9, 68 := 12, 69 := 5, 70 := 14,
     71 := 6, 72 := 8, 73 := 13, 74 := 6, 75 := 5, 76 := 15, 77 := 13,
     78 := 11, 79 := 11)
  j''
  <= 15
  (proof)

lemmas userlemmas = goal2'2 goal2'3 goal2'1

end

12 Verification of round

theory Round-Specification
imports Global-Specification Round-Declaration

begin

abbreviation bit--and' :: [ int , int ] => int where
  bit--and' == Global-Specification.bit--and'
abbreviation bit--or' :: [ int , int ] => int where
  bit--or' == Global-Specification.bit--or'
abbreviation bit--xor' :: [ int , int ] => int where
bitxor' == Global-Specification.bitxor'

abbreviation f' :: [int, int, int, int] => int where
f' == Global-Specification.f'

abbreviation k-l' :: int => int where
k-l' == Global-Specification.k_l'

abbreviation k-r' :: int => int where
k-r' == Global-Specification.k_r'

abbreviation r-l' :: int => int where
r-l' == Global-Specification.r_l'

abbreviation r-r' :: int => int where
r-r' == Global-Specification.r_r'

abbreviation wordops-rotate-left' :: [int, int] => int where
wordops-rotate-left' == Global-Specification.rotate_left'

abbreviation s-l' :: int => int where
s-l' == Global-Specification.s_l'

abbreviation s-r' :: int => int where
s-r' == Global-Specification.s_r'

abbreviation from-chain :: chain' => chain where
from-chain c == (word-of-int (h0'chain c), word-of-int (h1'chain c), word-of-int (h2'chain c), word-of-int (h3'chain c), word-of-int (h4'chain c))

abbreviation from-chain-pair :: chain-pair' => chain * chain where
from-chain-pair cc == (from-chain (left'chain-pair cc), from-chain (right'chain-pair cc))

abbreviation to-chain :: chain => chain' where
to-chain c == (let (h0, h1, h2, h3, h4) = c in
chain---default-rcd''
(h0'chain := uint h0, h1'chain := uint h1, h2'chain := uint h2, h3'chain := uint h3, h4'chain := uint h4))

abbreviation to-chain-pair :: chain * chain => chain-pair' where
to-chain-pair c == (let (c1, c2) = c in
(left'chain-pair = to-chain c1, right'chain-pair = to-chain c2))

abbreviation steps' :: [chain-pair', int, block'] => chain-pair' where
steps' cc i b == to-chain-pair (steps
abbreviation round' :: [ chain' , block' ] => chain' where
round' c b == to-chain (round (%n. word-of-int (b (int n)))) (from-chain c)

end

theory Round-User

imports Round-Specification Round-Declaration

begin

lemma uint-word-of-int-id:
  assumes 0 <= (x::int)
  assumes x <= 4294967295
  shows uint(word-of-int x::word32) = x
  ⟨proof⟩

lemma steps-step: steps X cc (Suc i) = step-both X (steps X cc i) i
  ⟨proof⟩

lemma from-to-id: from-chain-pair (to-chain-pair CC) = CC
  ⟨proof⟩

lemma steps-to-steps':
  F A (steps X cc i) B =
  F A (from-chain-pair (to-chain-pair (steps X cc i))) B
  ⟨proof⟩

lemma steps'-step:
  assumes 0 <= i
  shows steps' cc (i + 1) X = to-chain-pair (step-both
  (λn. word-of-int (X (int n)))
  (from-chain-pair (steps' cc i X))
  (nat i))
  ⟨proof⟩

lemma step-from-hyp:
  fixes a b c d e
  fixes a' b' c' d' e'
  fixes a-0 b-0 c-0 d-0 e-0
  fixes x
  fixes j
  assumes
  step-hyp:
chain-pair---default-rcd"
\{left'chain-pair := chain---default-rcd"
  \{h'0\chain := a, h'1\chain := b, h'2\chain := c, h'3\chain := d, h'4\chain := e\},
right'chain-pair := chain---default-rcd"
  \{h''0\chain := a', h''1\chain := b', h''2\chain := c', h''3\chain := d', h''4\chain := e'\} =
steps'
  (chain-pair---default-rcd"
\{left'chain-pair := chain---default-rcd"
  \{h'0\chain := a-0, h'1\chain := b-0, h'2\chain := c-0, h'3\chain := d-0, h'4\chain := e-0\},
right'chain-pair := chain---default-rcd"
  \{h''0\chain := a-0, h''1\chain := b-0, h''2\chain := c-0, h''3\chain := d-0, h''4\chain := e-0\}⟩}
\ j x
assumes a-borders: 0 <= a <= 4294967295 (is - <= \?M)
assumes b-borders: 0 <= b b <= \?M
assumes c-borders: 0 <= c c <= \?M
assumes d-borders: 0 <= d d <= \?M
assumes e-borders: 0 <= e e <= \?M
assumes a'-borders: 0 <= a' a' <= \?M
assumes b'-borders: 0 <= b' b' <= \?M
assumes c'-borders: 0 <= c' c' <= \?M
assumes d'-borders: 0 <= d' d' <= \?M
assumes e'-borders: 0 <= e' e' <= \?M
assumes x-borders: 0 <= x (r-l') x (r-l') <= \?M
  0 <= x (r-r') x (r-r') <= \?M
assumes j-borders: 0 <= j j <= 79
shows
chain-pair---default-rcd"
\{left'chain-pair := chain---default-rcd"
  \{h''0\chain := e,
     h''1\chain :=
       (wordops--rotate-left' (s-l' j)
         (((((a + f' j b c d) mod 4294967296 + x (r-l' j)) mod 4294967296 + k-l' j) mod 4294967296) +
         e) mod 4294967296,
     h''2\chain := b, h''3\chain := wordops--rotate-left' 10 c, h''4\chain := d\},
right'chain-pair := chain---default-rcd"
  \{h''0\chain := e',
     h''1\chain :=
       (wordops--rotate-left' (s-r' j)
         (((((a' + f' (79 - j) b c d') mod 4294967296 + x (r-r' j)) mod 4294967296 + k-r' j) mod 4294967296) +
         e') mod 4294967296,
     h''2\chain := b, h''3\chain := wordops--rotate-left' 10 c, h''4\chain := d\}
\[4294967296 +
\]
\[x (r-r' j)) \mod
\]
\[4294967296 +
\]
\[k-r' j) \mod
\]
\[4294967296 +
\]
\[e') \mod
\]
\[4294967296,
\]
\[h2' \text{chain} := b', h3' \text{chain} := \text{wordops--rotate-left} ' 10 c',
\]
\[h4' \text{chain} := d'[]
\]
steps'
\]
\[(\text{chain-pair---default-red}''
\]
\[h0' \text{chain} := a-0, h1' \text{chain} := b-0, h2' \text{chain} := c-0,
\]
\[h3' \text{chain} := d-0, h4' \text{chain} := e-0]
\]
\[(j + 1) x
\]
\[\langle\text{proof}\rangle
\]

**abbreviation**

\[f-0\text{-result} == (((ca'' + f\text{-spark} ' 0 cb'' cc'' cd'') \mod 4294967296 +
\]
\[x'' (r-l\text{-spark} ' 0)) \mod 4294967296 + k-l\text{-spark} ' 0) \mod 4294967296
\]

**abbreviation**

\[f-79\text{-result} == (((ca'' + f\text{-spark} ' 79 cb'' cc'' cd'') \mod 4294967296 +
\]
\[x'' (r-r\text{-spark} ' 0)) \mod 4294967296 + k-r\text{-spark} ' 0) \mod 4294967296
\]

**lemma** goal61':

- assumes ca-borders: 0 <= ca'' ca' <= 4294967295 (is - <= ?M)
- assumes cb-borders: 0 <= cb'' cb' <= ?M
- assumes cc-borders: 0 <= cc'' cc' <= ?M
- assumes cd-borders: 0 <= cd'' cd' <= ?M
- assumes ce-borders: 0 <= ce'' ce' <= ?M
- assumes r-l\text{-0\-borders}: 0 <= r-l\text{-spark} ' 0 r-l\text{-spark} ' 0 <= 15
- assumes r-r\text{-0\-borders}: 0 <= r-r\text{-spark} ' 0 r-r\text{-spark} ' 0 <= 15
- assumes returns:
  - wordops--rotate' (s-l\text{-spark} ' 0) f-0\text{-result} =
  - wordops--rotate-left' (s-l\text{-spark} ' 0) f-0\text{-result}
  - wordops--rotate' (s-r\text{-spark} ' 0) f-79\text{-result} =
  - wordops--rotate-left' (s-r\text{-spark} ' 0) f-79\text{-result}
  - wordops--rotate' 10 cc'' = wordops--rotate-left' 10 cc''
  - f\text{-spark} ' 0 cb'' cc'' cd'' = f' 0 cb'' cc'' cd''
  - f\text{-spark} ' 79 cb'' cc'' cd'' = f' 79 cb'' cc'' cd''
  - k-l\text{-spark} ' 0 = k-l' 0
  - k-r\text{-spark} ' 0 = k-r' 0
  - r-l\text{-spark} ' 0 = r-l' 0
  - r-r\text{-spark} ' 0 = r-r' 0
  - s-l\text{-spark} ' 0 = s-l' 0
  - s-r\text{-spark} ' 0 = s-r' 0

24
assumes $x$-borders: $\forall i. \ 0 \leq i \land i \leq 15 \rightarrow 0 \leq x'' \land x'' i \leq ?M$

shows \texttt{chain-pair---default-rcd}$''$
\begin{align*}
&\langle \text{left} \rangle' \text{chain-pair} := \text{chain---default-rcd}$''$
\langle \text{h0} \rangle' \text{chain} := \text{ce}$''$,
\langle \text{h1} \rangle' \text{chain} := \\
&\hspace{1em}(\text{wordops--rotate'} (s-l-spark$' \hspace{1em} 0)
\langle (((\text{ca改良�} + f-spark') \hspace{1em} 0 \hspace{1em} \text{cb改良�} \hspace{1em} \text{cc改良�} \hspace{1em} \text{cd改良�}) \hspace{1em} \text{mod} \hspace{1em} 4294967296 + x'' (r-l-spark') 0) \hspace{1em} \text{mod}
\hspace{1em} 4294967296 +
\hspace{1em} k-l-spark') 0 \hspace{1em} \text{mod}
\hspace{1em} 4294967296 +
\hspace{1em} \text{ce改良�}) \hspace{1em} \text{mod}
\hspace{1em} 4294967296,
\langle \text{h2} \rangle' \text{chain} := \text{cb改良�}, \langle \text{h3} \rangle' \text{chain} := \text{wordops--rotate'} 10 \hspace{1em} \text{cc改良�},
\langle \text{h4} \rangle' \text{chain} := \text{cd改良�}$]
\langle \text{right} \rangle' \text{chain-pair} := \text{chain---default-rcd}$''$
\langle \text{h0} \rangle' \text{chain} := \text{ce}$''$,
\langle \text{h1} \rangle' \text{chain} := \\
&\hspace{1em}(\text{wordops--rotate'} (s-r-spark$' \hspace{1em} 0)
\langle (((\text{ca改良�} + f-spark') \hspace{1em} 79 \hspace{1em} \text{cb改良�} \hspace{1em} \text{cc改良�} \hspace{1em} \text{cd改良�}) \hspace{1em} \text{mod} \hspace{1em} 4294967296 + x'' (r-r-spark') 0) \hspace{1em} \text{mod}
\hspace{1em} 4294967296 +
\hspace{1em} k-r-spark') 0 \hspace{1em} \text{mod}
\hspace{1em} 4294967296 +
\hspace{1em} \text{ce改良�}) \hspace{1em} \text{mod}
\hspace{1em} 4294967296,
\langle \text{h2} \rangle' \text{chain} := \text{cb改良�}, \langle \text{h3} \rangle' \text{chain} := \text{wordops--rotate'} 10 \hspace{1em} \text{cc改良�},
\langle \text{h4} \rangle' \text{chain} := \text{cd改良�}$]
\end{align*}

steps$'$
\begin{align*}
&\langle \text{chain-pair---default-rcd}$''$
\langle \text{left} \rangle' \text{chain-pair} := \text{chain---default-rcd}$''$
\langle \text{h0} \rangle' \text{chain} := \text{ca改良�}, \langle \text{h1} \rangle' \text{chain} := \text{cb改良�}, \langle \text{h2} \rangle' \text{chain} := \text{ce改良�},
\langle \text{h3} \rangle' \text{chain} := \text{cd改良�}, \langle \text{h4} \rangle' \text{chain} := \text{cd改良�}$]
\langle \text{right} \rangle' \text{chain-pair} := \text{chain---default-rcd}$''$
\langle \text{h0} \rangle' \text{chain} := \text{ca改良�}, \langle \text{h1} \rangle' \text{chain} := \text{cb改良�}, \langle \text{h2} \rangle' \text{chain} := \text{ce改良�},
\langle \text{h3} \rangle' \text{chain} := \text{cd改良�}, \langle \text{h4} \rangle' \text{chain} := \text{ce改良�}$]
\end{align*}

\textit{1} \hspace{1em} x''

\langle \text{proof} \rangle

\textbf{abbreviation} \texttt{rotate-arg-l} \hspace{1em} \langle (((\text{ca改良�} + f-spark') \hspace{1em} \text{loop--1-j改良�} + 1) \hspace{1em} \text{cb改良�} \hspace{1em} \text{cle改良�} \hspace{1em} \text{cd改良�}) \hspace{1em} \text{mod} \hspace{1em} 4294967296 + x'' (r-l-spark') (\text{loop--1-j改良�} + 1)) \hspace{1em} \text{mod} \hspace{1em} 4294967296 + k-l-spark') (\text{loop--1-j改良�} + 1)) \hspace{1em} \text{mod} \hspace{1em} 4294967296)

\textbf{abbreviation} \texttt{rotate-arg-r} \hspace{1em} \langle (((\text{ca改良�} + f-spark') (79 - (\text{loop--1-j改良�} + 1)) \hspace{1em} \text{crb改良�} \hspace{1em} \text{crc改良�} \hspace{1em} \text{crd改良�}) \hspace{1em} \text{mod} \hspace{1em} 4294967296 + x'' (r-r-spark') (\text{loop--1-j改良�} + 1)) \hspace{1em} \text{mod} \hspace{1em} 4294967296 + k-r-spark') (\text{loop--1-j改良�} + 1)) \hspace{1em} \text{mod} \hspace{1em} 4294967296)
lemma goal62'1:
assumes cla-borders: 0 <= cla'' <= 4294967295 (is <= ?M)
assumes clb-borders: 0 <= clb'' clb'' <= ?M
assumes cle-borders: 0 <= cle'' cle'' <= ?M
assumes cld-borders: 0 <= cld'' cld'' <= ?M
assumes cle-borders: 0 <= cle'' cle'' <= ?M
assumes cra-borders: 0 <= cra'' cra'' <= ?M
assumes crb-borders: 0 <= crb'' crb'' <= ?M
assumes crc-borders: 0 <= crc'' crc'' <= ?M
assumes crd-borders: 0 <= crd'' crd'' <= ?M
assumes cre-borders: 0 <= cre'' cre'' <= ?M
assumes step-hyp:
chain-pair---default-rcd''
  (loop--1--j'' + 1) x''
assumes returns:
wordops--rotate' (s-l-spark' (loop--1-j'' + 1)) rotate-arg-l =
wordops--rotate-left' (s-l-spark' (loop--1-j'' + 1)) rotate-arg-l
wordops--rotate' (s-r-spark' (loop--1-j'' + 1)) rotate-arg-r =
wordops--rotate-left' (s-r-spark' (loop--1-j'' + 1)) rotate-arg-r
f-spark' (loop--1-j'' + 1) clb'' clc'' cld'' =
f' (loop--1-j'' + 1) clb'' clc'' cld''
assumes x-borders: \forall i. 0 <= i ∧ i <= 15 → 0 <= x'' i ∧ x'' i <= ?M
assumes r-l-borders:
0 <= r-l-spark' (loop--1-j'' + 1) r-l-spark' (loop--1-j'' + 1) <= 15
assumes r-r-borders:
0 <= r-r-spark' (loop--1-j'' + 1) r-r-spark' (loop--1-j'' + 1) <= 15
assumes j-loop-1-borders: 0 <= loop--1-j'' loop--1-j'' <= 78
shows chain-pair---default-rcd''
  |left'chain-pair := chain---default-rcd''
  |h0'chain := cle'',
  |h1'chain :=
  |  (wordops--rotate' (s-l-spark' (loop--1-j'' + 1))
  |    (((cl'' + f-spark' (loop--1-j'' + 1) clb'' clc'' cld'') mod
  |      4294967296 +
  |      x'' (r-l-spark' (loop--1-j'' + 1))) mod
  |      4294967296 +
  |      k-l-spark' (loop--1-j'' + 1)) mod
  |      4294967296) +
  |      cle'') mod
  |      4294967296,
  |h2'chain := clb'', h3'chain := wordops--rotate' 10 clc'',
  |h4'chain := cld''),
right'chain-pair := chain---default-rcd''
  |h0'chain := cre'',
  |h1'chain :=
  |  (wordops--rotate' (s-r-spark' (loop--1-j'' + 1))
  |    (((cra'' +
  |      f-spark' (79 - (loop--1-j'' + 1)) crb'' crc''
  |      crd'') mod
  |      4294967296 +
  |      x'' (r-r-spark' (loop--1-j'' + 1))) mod
  |      4294967296 +
  |      k-r-spark' (loop--1-j'' + 1)) mod
  |      4294967296) +
  |      cre'') mod
  |      4294967296,
  |h2'chain := crb'', h3'chain := wordops--rotate' 10 crc'',
  |h4'chain := crd'') |
steps''
  |chain-pair---default-rcd''
  |left'chain-pair := chain---default-rcd''
  |h0'chain := ca---init'', h1'chain := cb---init'',
  |h2'chain := cc---init'', h3'chain := cd---init'',
  |h4'chain := cc---init''),
right'chain-pair := chain---default-rcd''
  |h0'chain := ca---init'', h1'chain := cb---init'',
  |h2'chain := cc---init'', h3'chain := cd---init'',
  |h4'chain := cc---init'' |)
(\langle proof \rangle
abbreviation INIT-CHAIN == chain---default-rcd''
\( h0'\text{chain} := \text{ca\text{-}init}'', \ h1'\text{chain} := \text{cb\text{-}init}'', \\
\ h2'\text{chain} := \text{cc\text{-}init}'', \ h3'\text{chain} := \text{cd\text{-}init}'', \\
\ h4'\text{chain} := \text{cc\text{-}init}'') \)

**Lemma** goal76'1:

**Assumes** cla-borders: 0 <= cla'' cla'' <= 4294967295 (is - <= ?M)  
**Assumes** clb-borders: 0 <= clb'' clb'' <= ?M  
**Assumes** clc-borders: 0 <= clc'' clc'' <= ?M  
**Assumes** cld-borders: 0 <= cld'' cld'' <= ?M  
**Assumes** cle-borders: 0 <= cle'' cle'' <= ?M  
**Assumes** cra-borders: 0 <= cra'' cra'' <= ?M  
**Assumes** crb-borders: 0 <= crb'' crb'' <= ?M  
**Assumes** crc-borders: 0 <= crc'' crc'' <= ?M  
**Assumes** crd-borders: 0 <= crd'' crd'' <= ?M  
**Assumes** cre-borders: 0 <= cre'' cre'' <= ?M  
**Assumes** ca-init-borders: 0 <= ca---init'' ca---init'' <= ?M  
**Assumes** cb-init-borders: 0 <= cb---init'' cb---init'' <= ?M  
**Assumes** cc-init-borders: 0 <= cc---init'' cc---init'' <= ?M  
**Assumes** cd-init-borders: 0 <= cd---init'' cd---init'' <= ?M  
**Assumes** ce-init-borders: 0 <= ce---init'' ce---init'' <= ?M

**Assumes** step-hyp:

\( (\text{chain-pair\text{-}default-rcd}'') \)

\( \text{\{left\text{'}chain-pair := chain\text{-}default-rcd''} \)

\( \text{\{h0'\text{chain} := cla'', \ h1'\text{chain} := clb'', \ h2'\text{chain} := clc'', \ h3'\text{chain} := cld'',} \)

\( h4'\text{chain} := \text{cle''} \}

\( \text{\{right\text{'}chain-pair := chain\text{-}default-rcd''} \)

\( \text{\{h0'\text{chain} := cra'', \ h1'\text{chain} := crb'', \ h2'\text{chain} := crc'', \ h3'\text{chain} := crd'',} \)

\( h4'\text{chain} := \text{cre''} \}) \)

**Steps**

\( (\text{chain-pair\text{-}default-rcd}'') \)

\( \text{\{left\text{'}chain-pair := chain\text{-}default-rcd''} \)

\( \text{\{h0'\text{chain} := ca---init'', \ h1'\text{chain} := cb---init'', \ h2'\text{chain} := cc---init'',} \)

\( h3'\text{chain} := \text{cd---init''}, \)

\( h4'\text{chain} := \text{ce---init''} \}

\( \text{\{right\text{'}chain-pair := chain\text{-}default-rcd''} \)

\( \text{\{h0'\text{chain} := ca---init'', \ h1'\text{chain} := cb---init'', \ h2'\text{chain} := cc---init'',} \)

\( h3'\text{chain} := \text{cd---init''}, \)

\( h4'\text{chain} := \text{ce---init''} \}) \)

80 x''

**Shows** chain\text{-}default-rcd''

\( \text{\{h0'\text{chain} := ((\text{cb---init'' + clc''}) \mod 4294967296 + \text{crd''}) \mod 4294967296,} \)

\( h1'\text{chain} := ((\text{cc---init'' + clb''}) \mod 4294967296 + \text{cre''}) \mod 4294967296,} \)

\( h2'\text{chain} := ((\text{cd---init'' + cle''}) \mod 4294967296 + \text{crb''}) \mod 4294967296,} \)

\( h3'\text{chain} := ((\text{ce---init'' + cla''}) \mod 4294967296 + \text{cra''}) \mod 4294967296,} \)

\( h4'\text{chain} := ((\text{ca---init'' + clc''}) \mod 4294967296 + \text{crc''}) \mod 4294967296} \)

= round'

\( (\text{chain\text{-}default-rcd''} \)

28
lemma userlemmas = goal61'1 goal62'1 goal76'1

end

13 Verification of hash

theory Hash-Specification
imports Hash-Declaration Global-Specification

begin

abbreviation from-chain :: chain' => chain where
  from-chain c == (word-of-int h0'chain c, word-of-int h1'chain c, word-of-int h2'chain c, word-of-int h3'chain c, word-of-int h4'chain c)

abbreviation to-chain :: chain => chain' where
to-chain c == (let (h0, h1, h2, h3, h4) = c in chain---default-rcd''(|h0'chain := uint h0, h1'chain := uint h1, h2'chain := uint h2, h3'chain := uint h3, h4'chain := uint h4|))

abbreviation round' :: [chain', block'] => chain' where
  round' c b == to-chain (round (%n. word-of-int (b (int n))) (from-chain c))

abbreviation rounds' :: [chain', int, message'] => chain' where
  rounds' h i X ==
    to-chain (rounds (λn. λm. word-of-int (X (int n) (int m))) (from-chain h) (nat i))

abbreviation rmd-hash' :: [message', int] => chain' where
  rmd-hash' X i == to-chain (rmd (λn. λm. word-of-int (X (int n) (int m))) (nat i))

end
theory Hash-User
imports Hash-Specification Hash-Declaration
begin

lemma goal12':1:
assumes H1: x--index--subtype--1--first'' = (0 :: int)

assumes H6:
chain---default-rcd''
( | h0'chain
  := ca--1''
  | )
( | h1'chain
  := cb--1''
  | )
( | h2'chain
  := cc--1''
  | )
( | h3'chain
  := cd--1''
  | )
( | h4'chain
  := ce--1''
  | )
 = round'
  ( chain---default-rcd''
    ( | h0'chain
      := (1732584193 :: int)
      | )
    ( | h1'chain
      := (4023233417 :: int)
      | )
    ( | h2'chain
      := (2562383102 :: int)
      | )
    ( | h3'chain
      := (271733878 :: int)
      | )
    ( | h4'chain
      := (3285377520 :: int)
      | )
  )
( x'' x--index--subtype--1--first'' )
end
shows chain---default-rcd''
  ((| h0'chain
    := ca-1''
  )|)
  ((| h1'chain
    := cb-1''
  )|)
  ((| h2'chain
    := cc-1''
  )|)
  ((| h3'chain
    := cd-1''
  )|)
  ((| h4'chain
    := ce-1''
  )|)
  (rounds'''
    ( chain---default-rcd''
      ((| h0'chain
          := (1732584193 :: int)
        )|)
      ((| h1'chain
          := (4023233417 :: int)
        )|)
      ((| h2'chain
          := (2562383102 :: int)
        )|)
      ((| h3'chain
          := (271733878 :: int)
        )|)
      ((| h4'chain
          := (3285377520 :: int)
        )|)
    )
      (x--index--subtype--1--first'' + (1 :: int)
      )
    )''
  (is ?C1)
  (proof)

lemma rounds-step:
  assumes 0 <= i
  shows rounds X b (Suc i) = round (X i) (rounds X b i)
  (proof)

lemma from-to-id: from-chain (to-chain C) = C
  (proof)

lemma steps-to-steps':

31
round X (foldl a b c) = round X (from-chain (to-chain (foldl a b c)))
⟨proof⟩

**lemma** rounds′-step:
assumes 0 <= i
shows rounds′ c (i + 1) x = round′ (rounds′ c i x) (x i)
⟨proof⟩

**lemma** goal13′1:
assumes 0 <= loop--1--i″
assumes H1:
chain---default-rcd″
( | h0′chain := ca″ |
  | h1′chain := cb″ |
  | h2′chain := cc″ |
  | h3′chain := cd″ |
  | h4′chain := ce″ |
  )
= rounds′
  ( chain---default-rcd″
    ( | h0′chain := (1732584193 :: int) |
      | h1′chain := (4023233417 :: int) |
      | h2′chain := (2562383102 :: int) |
      | h3′chain := (271733878 :: int) |
      | h4′chain := (3285377520 :: int) |
      )
      )
  )
( loop--1--i″ + (1 :: int) )

x″

assumes H18:
\[ \begin{align*}
    \text{chain---default-rcd''} \quad & \\
    ( | & h0' \text{chain} \\
    & := \text{ca--1''} \\
    | ) \\
    ( | & h1' \text{chain} \\
    & := \text{cb--1''} \\
    | ) \\
    ( | & h2' \text{chain} \\
    & := \text{cc--1''} \\
    | ) \\
    ( | & h3' \text{chain} \\
    & := \text{cd--1''} \\
    | ) \\
    ( | & h4' \text{chain} \\
    & := \text{ce--1''} \\
    | ) \\

    = \text{round'} \\
    \quad ( \text{chain---default-rcd''} \\
    ( | & h0' \text{chain} \\
    & := \text{ca''} \\
    | ) \\
    ( | & h1' \text{chain} \\
    & := \text{cb''} \\
    | ) \\
    ( | & h2' \text{chain} \\
    & := \text{cc''} \\
    | ) \\
    ( | & h3' \text{chain} \\
    & := \text{cd''} \\
    | ) \\
    ( | & h4' \text{chain} \\
    & := \text{ce''} \\
    | ) \\

    ) \\
    ( x'' ( \text{loop--1''} + (1 :: \text{int}) ) ) \\
\end{align*} \]

\textbf{shows} \text{chain---default-rcd''}

\[ \begin{align*}
    ( | & h0' \text{chain} \\
    & := \text{ca--1''} \\
    | ) \\
    ( | & h1' \text{chain} \\
    & := \text{cb--1''} \\
    | ) \\
    ( | & h2' \text{chain} \\
    & := \text{cc--1''} \\
    | ) \\
    ( | & h3' \text{chain} \\
    & := \text{cd--1''} \\
    | ) \\
\end{align*} \]
\[ h_4' \text{chain} \quad := \quad c e' - 1'' \]
\[ = \quad \text{rounds}' \]
\[ ( \text{chain}---\text{default-rcd}'' \]
\[ ( h_0' \text{chain} \quad := \quad (1732584193 :: \text{int}) \]
\[ ) \]
\[ ( h_1' \text{chain} \quad := \quad (4023233417 :: \text{int}) \]
\[ ) \]
\[ ( h_2' \text{chain} \quad := \quad (2562383102 :: \text{int}) \]
\[ ) \]
\[ ( h_3' \text{chain} \quad := \quad (271733878 :: \text{int}) \]
\[ ) \]
\[ ( h_4' \text{chain} \quad := \quad (3285377520 :: \text{int}) \]
\[ ) \]
\[ ( \text{loop}--1--i'' + (2 :: \text{int}) \]
\[ x'' \]
\[ \langle \text{proof} \rangle \]

**lemma** goal17'1:

**assumes** H1:

\[ \text{chain}---\text{default-rcd}'' \]
\[ ( h_0' \text{chain} \quad := \quad c a'' \]
\[ ) \]
\[ ( h_1' \text{chain} \quad := \quad c b'' \]
\[ ) \]
\[ ( h_2' \text{chain} \quad := \quad c c'' \]
\[ ) \]
\[ ( h_3' \text{chain} \quad := \quad c d'' \]
\[ ) \]
\[ ( h_4' \text{chain} \quad := \quad c e'' \]
\[ ) \]
\[ = \quad \text{rounds}' \]
\[ ( \text{chain}---\text{default-rcd}'' \]
\[ ( h_0' \text{chain} \]
shows  chain---default-rcd''
  ( | h0'chain
      := ca''
  )
  ( | h1'chain
      := cb''
  )
  ( | h2'chain
      := cc''
  )
  ( | h3'chain
      := cd''
  )
  ( | h4'chain
      := ce''
  )
= rmd-hash'
  x''
    ( x--index--subtype--1--last'' + (1 :: int) )
