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 \( k_l \) 10

6 Verification of \( k_r \) 11

7 Arrays in SPARK vs Lists in Isabelle 13
   7.1 Functions vs Lists 13
   7.2 Maximum Element of Lists 15

8 Verification of \( r_l \) 17

9 Verification of \( r_r \) 19

10 Verification of \( s_l \) 21
11 Verification of \( s_r \)

12 Verification of \( round \)

13 Verification of \( hash \)

1 Introduction

The directory \texttt{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 \textit{proof functions}. The following proof functions (declared in \texttt{ada/rmd.ads}) are specified in Isabelle/HOL:

- \texttt{bit\_and}
- \texttt{bit\_or}
- \texttt{bit\_xor}
- \texttt{wordops\_rotate\_left}
- \texttt{f}
- \texttt{k_l}
- \texttt{k_r}
- \texttt{r_l}
- \texttt{r_r}
- \texttt{s_l}
- \texttt{s_r}
- \texttt{steps}
- \texttt{round}
- \texttt{rounds}
- \texttt{rmd\_hash}

From the annotations in the SPARK-code, verification conditions were generated using \textit{SPARK-GPL-2010}(http://libre.adacore.com/libre/download/):\n\$\texttt{spark -vcg -rules=lazy ada/shadow/interfaces.ads ada/wordops.ads ada/rmd.ads ada/rmd.adb}\n
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 0x55A5DD12
    else if (32 <= j & j <= 47) then 0x6D703EF3
    else if (48 <= j & j <= 63) then 0x7A6D76E9
    else if (64 <= j & j <= 79) then 0x00000000

1There 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)\\
\text{definition } r\text{-list }:: \text{ nat list}\\
\text{where } r\text{-list }= [\text{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}\\
\text{definition } r'\text{-list }:: \text{ nat list}\\
\text{where } r'\text{-list } = [\text{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}\\
\text{definition } r :: \text{ perm}\\
\text{where } r\ j = r\text{-list }!\ j\\
\text{definition } r' :: \text{ perm}\\
\text{where } r'\ j = r'\text{-list }!\ j\\
\text{definition } s\text{-list }:: \text{ nat list}\\
\text{where } s\text{-list } = [\text{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}\\
\text{definition } s'\text{-list }:: \text{ nat list}\\
\text{where } s'\text{-list } = [\text{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}\\
\text{definition } s :: \text{ perm}\\
\text{where } s\ j = s\text{-list }!\ j\\
\text{definition } s' :: \text{ perm}\\
\text{where } s'\ j = s'\text{-list }!\ j
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
let ((A, B, C, D, E), (A', B', C', D', E')) = steps X (h, h) 80 in
((∗ h0 ∗) h1 + C' + D',
(∗ h1 ∗) h2 + D + E',
(∗ h2 ∗) h3 + E + A',
(∗ h3 ∗) h4 + A + B',
(∗ h4 ∗) 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-opertations 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
lemma bit-not-spark-def[simp]:
(word-of-int (4294967295 − x)::word32) = \text{NOT} (word-of-int x)
proof
  have word-of-int x + (word-of-int (4294967295 − x)::word32) =
    word-of-int x + \text{NOT} (word-of-int x)
    by (simp only: bwsimps bin-add-not Min-def) simp
thus \text{?thesis} by (simp only: add-left-imp-eq)
qed

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' :: int => int where
  k-l' j == uint (K (nat j))
abbreviation k-r' :: int => int where
  k-r' j == uint (K' (nat j))
abbreviation r-l' :: int => int where
  r-l' j == int (r (nat j))
abbreviation r-r' :: int => int where
  r-r' j == int (r' (nat j))
abbreviation s-l' :: int => int where
  s-l' j == int (s (nat j))
abbreviation s-r' :: int => int where
  s-r' j == int (s' (nat j))
abbreviation f' :: int => int => int => int => int where
  f' j x y z ==
    uint (f (nat j) (word-of-int x::word32) (word-of-int y) (word-of-int z))

end

4 Verification of \( f \)

theory F-Spark-Specification
imports F-Spark-Declaration Global-Specification
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
  bit--xor' == Global-Specification.bit--xor'


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

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

lemma goal2'1:
  shows $0 <\Rightarrow (\text{bit--or'} (\text{bit--and'} x''' y''') (\text{bit--and'} (4294967295 - x''') z''))$
  by (rule Word.uint-0)

lemma goal2'2:
  shows $\text{bit--or'} (\text{bit--and'} x''' y''') (\text{bit--and'} (4294967295 - x''') z'')) <= 4294967295$
  by (simp add: bwsimps int-word-uint)

lemma goal3'1:
  shows $0 <\Rightarrow (\text{bit--xor'} (\text{bit--or'} x''') (4294967295 - y''')) z''$
  by (rule Word.uint-0)

lemma goal3'2:
  shows $\text{bit--xor'} (\text{bit--or'} x''') (4294967295 - y''')) z'' <= 4294967295$
  by (simp add: bwsimps int-word-uint)

lemma goal4'1:
  shows $0 <\Rightarrow (\text{bit--or'} (\text{bit--and'} x'''' z''') (\text{bit--and'} y''') (4294967295 - z'''))$
  by simp

lemma goal4'2:
  shows $\text{bit--or'} (\text{bit--and'} x'''' z''') (\text{bit--and'} y''') (4294967295 - z''')) <= 4294967295$
  by (simp add: bwsimps int-word-uint)

lemma goal5'1:
  shows $0 <\Rightarrow (\text{bit--xor'} x''' (\text{bit--or'} y''' (4294967295 - z'''))$
  by simp

lemma goal5'2:
  shows $\text{bit--xor'} x''' (\text{bit--or'} y''' (4294967295 - z''')) <= 4294967295$
  by (simp add: bwsimps int-word-uint)

lemma goal6'1:
  assumes H8: $j'' <= (15 :: int)$
  shows $\text{bit--xor'} x''' (\text{bit--xor'} y''') z''') = f' j'' x''' y''' z''$
proof
  from H8 have $\text{nat j'' <= 15}$ by simp
  thus $?\text{thesis}$
  by (simp add: f-def)
lemma goal7':
  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 -
  from H7 have 16 <= nat j'' by simp
  moreover from H8 have nat j'' <= 31 by simp
  ultimately show ?thesis
    by (simp add: f-def)
qed

lemma goal8':
  assumes H7: 32 <= j''
  assumes H8: j'' <= 47
  shows "bit-xor' (bit-or' x'' (4294967295 - y'')) z'' = f' j'' x'' y'' z''"
proof -
  from H7 have 32 <= nat j'' by simp
  moreover from H8 have nat j'' <= 47 by simp
  ultimately show ?thesis by (simp add: f-def)
qed

lemma goal9':
  assumes H7: 48 <= j''
  assumes H8: j'' <= 63
  shows "bit-or' (bit-and' x'' z'') (bit-and' y'' (4294967295 - z'')) = f' j'' x'' y'' z''"
proof -
  from H7 have 48 <= nat j'' by simp
  moreover from H8 have nat j'' <= 63 by simp
  ultimately show ?thesis by (simp add: f-def)
qed

lemma goal10':
  assumes H2: j'' <= 79
  assumes H12: 63 < j''
  shows "bit-xor' x'' (bit-or' y'' (4294967295 - z'')) = f' j'' x'' y'' z''"
proof -
  from H2 have nat j'' <= 79 by simp
  moreover from H12 have 64 <= nat j'' by simp
  ultimately show ?thesis by (simp add: f-def)
qed

lemmas userlemmas =
goal2'1
goal2'2
goal3'1
5 Verification of $k_l$

theory $K\text{-}L\text{-}Spark\text{-}Specification$
imports $K\text{-}L\text{-}Spark\text{-}Declaration$ $Global\text{-}Specification$

begin

abbreviation $k\text{-}l'$ :: int => int where $k\text{-}l' = Global\text{-}Specification.k\text{-}l'$

end

theory $K\text{-}L\text{-}Spark\text{-}User$
imports $K\text{-}L\text{-}Spark\text{-}Specification$ $K\text{-}L\text{-}Spark\text{-}Declaration$

begin

lemma $goal6'$:
  fixes $j$ :: int
  assumes $H1$: $0 \leq j$
  assumes $H2$: $j \leq 15$
  shows $0 = k\text{-}l' j$
  using assms by (simp add: $K$-def)

lemma $goal7'$:
  fixes $j$ :: int
  assumes $H1$: $16 \leq j$
  assumes $H2$: $j \leq 31$
  shows $1518500249 = k\text{-}l' j$

proof
  from $H1$ have $16 \leq \text{nat } j$ by simp
  moreover from $H2$ have $\text{nat } j \leq 31$ by simp
  ultimately show $\text{thesis}$ by (simp add: $K$-def)
qed
lemma goal8'1:
  assumes H1: \((32 :: \text{int}) \leq j''\)
  assumes H2: \(j'' \leq (47 :: \text{int})\)
  shows \((1859775393 :: \text{int}) = k-l' j''\)
proof
  from H1 have \(32 \leq \text{nat } j''\) by simp
  moreover from H2 have \(\text{nat } j'' \leq 47\) by simp
  ultimately show \(?\text{thesis}\) by (simp add: K-def)
qed

lemma goal9'1:
  assumes H1: \((48 :: \text{int}) \leq j''\)
  assumes H2: \(j'' \leq (63 :: \text{int})\)
  shows \((2400959708 :: \text{int}) = k-l' j''\) (is \(?C1\))
proof
  from H1 have \(48 \leq \text{nat } j''\) by simp
  moreover from H2 have \(\text{nat } j'' \leq 63\) by simp
  ultimately show \(?\text{thesis}\) by (simp add: K-def)
qed

lemma goal10'1:
  assumes H2: \(j'' \leq (79 :: \text{int})\)
  assumes H6: \((63 :: \text{int}) \leq j''\)
  shows \((2840853838 :: \text{int}) = k-l' j''\) (is \(?C1\))
proof
  from H6 have \(64 \leq \text{nat } j''\) by simp
  moreover from H2 have \(\text{nat } j'' \leq 79\) by simp
  ultimately show \(?\text{thesis}\) by (simp add: K-def)
qed

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 \(k-r' :: \text{int} \Rightarrow \text{int}\) where
  \(k-r' = \text{Global-Specification.k-r'}\)
end
theory K-R-Spark-User
imports K-R-Spark-Specification K-R-Spark-Declaration
begin

lemma goal6':
  assumes H1: (0 :: int) <= j''
  assumes H2: j'' <= (15 :: int)
  shows (1352829926 :: int) = k-r' j'' (is ?C1)
  using assms by (simp add: K'-def)

lemma goal7':
  assumes H1: (16 :: int) <= j''
  assumes H2: j'' <= (31 :: int)
  shows (1548603684 :: int) = k-r' j'' (is ?C1)
  proof -
    from H1 have 16 <= nat j'' by simp
    moreover from H2 have nat j'' <= 31 by simp
    ultimately show ?thesis by (simp add: K'-def)
  qed

lemma goal8':
  assumes H1: (32 :: int) <= j''
  assumes H2: j'' <= (47 :: int)
  shows (1836072691 :: int) = k-r' j'' (is ?C1)
  proof -
    from H1 have 32 <= nat j'' by simp
    moreover from H2 have nat j'' <= 47 by simp
    ultimately show ?thesis by (simp add: K'-def)
  qed

lemma goal9':
  assumes H1: (48 :: int) <= j''
  assumes H2: j'' <= (63 :: int)
  shows (2053994217 :: int) = k-r' j'' (is ?C1)
  proof -
    from H1 have 48 <= nat j'' by simp
    moreover from H2 have nat j'' <= 63 by simp
    ultimately show ?thesis by (simp add: K'-def)
  qed

lemma goal10':
  assumes H2: j'' <= (79 :: int)
  assumes H6: (63 :: int) < j''
  shows (0 :: int) = k-r' j'' (is ?C1)
proof
  from H6 have 6 < nat j by simp
moreover from H2 have nat j <= 79 by simp
ultimately show thesis by (simp add: K-def)
qed

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

7 Arrays in SPARK vs Lists in Isabelle

theory Global-User
imports Main
begin

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.

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

lemma nth-list-to-fun-eq-aux:
  assumes i-0 <= i and i < length l + i-0
  shows int (l ! (i - i-0)) = (list-to-fun l (int i-0) f) (int i)
  using assms
proof (induct l arbitrary: i i-0)
  case Nil
  thus ?case by simp
next
  case (Cons a xs)
  moreover have aux: 1 + int i-0 = int i-0 + 1 by simp
  ultimately show ?case by (simp add: nth-Cons' aux)
qed
lemma nth-list-to-fun-eq:
  assumes \(0 \leq i\) and \(i < \text{length } l\)
  shows \(\text{int}(l!i) = (\text{list-to-fun } l \ 0\ f)\ (\text{int } i)\)
proof –
  have \(\text{int}(l! (i - 0)) = \)
    \((\text{list-to-fun } l \ (\text{int } 0)\ f)\ (\text{int } i)\)
    by (rule nth-list-to-fun-eq-aux) (simp-all add: assms)
  thus \(?thesis by simp\)
qed

A tail-recursive definition makes it even more efficient.

primrec list-to-fun-eff where
  list-to-fun-eff [] = (f :: int ⇒ int) = f
| list-to-fun-eff (a # xs) i f = list-to-fun-eff xs (i + 1) (f(i := (int a)))

lemma list-to-fun-id:
  assumes \(i-0 > i\)
  shows list-to-fun-eff l (int i-0) f (int i) = f (int i)
using assms
proof (induct l arbitrary: i-0 f)
  case Nil
  thus \(?case by simp\)
next
case (Cons a xs)
  have I: \(\text{int } i-0 + 1 = \text{int } (i-0 + 1)\) by simp
  from Cons(2) have L: \(i < i-0 + 1\) by simp
  with Cons have
    list-to-fun-eff xs (int i-0 + 1) (f(int i-0 := int a)) (int i) = f (int i)
    unfolding I Cons(1)[OF L] by simp
  thus \(?case by simp\)
qed

lemma nth-list-to-fun-eff-eq-aux:
  assumes \(i-0 \leq i\) and \(i < \text{length } l + i-0\)
  shows \(\text{int}(l! (i - i-0)) = (\text{list-to-fun-eff } l \ (\text{int } i-0)\ f)\ (\text{int } i)\)
using assms
proof (induct l arbitrary: i f i-0)
  case Nil
  thus \(?case by simp\)
next
case (Cons a xs)
  have I: \(\text{int } i-0 + 1 = \text{int } (i-0 + 1)\) by simp
  { 
    assume i = i-0
    moreover
    have i-0 + 1 > i-0 by simp
    have \(\text{int } a = \text{list-to-fun-eff } xs\ (\text{int } i-0 + 1)\ (f(int i-0 := int a))\ (\text{int } i-0)\)
unfolding \( \text{I list-to-fun-id[](OF \ i-0 + 1 > i-0]} \) by simp 
ultimately have ?case by (simp add: nth-Cons)

moreover
{
  assume \( i \neq i-0 \)
moreover
  hence \( \text{H}: i-0 + 1 \leq i \) using Cons by simp
have \( \text{H'}: i < \text{length xs} + (i-0 + 1) \) using Cons (3) by simp
have \( \text{int (xs ! (i - Suc i-0))} = \)
list-to-fun-eff xs (int i-0 + 1) (f(int i-0 := int a)) (int i)
unfolding I Cons(1)[OF H H', symmetric] by simp
ultimately have ?case using Cons(2) by (simp add: nth-Cons)
}
ultimately show ?case by blast

qed

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
  have \( \text{int (l ! (i - 0))} = \)
  (list-to-fun-eff l (int 0) f) (int i)
  by (rule nth-list-to-fun-eff-eq-aux) (simp-all add: assms)
  thus ?thesis by simp
qed

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 \( i \) :: nat
  assumes \( i \leq j \)
  shows foldl max i l \( \leq \) foldl max j l
using assms
by (induct l arbitrary: i j) simp-all

lemma fold-max-lower:
  fixes \( i \) :: nat
  shows \( i \leq \) foldl max i l
proof (induct l arbitrary: i)
case Nil
  thus ?case by simp
next
case (Cons x xs)
  show ?case
  proof (cases \( i \leq x \))
case True
  moreover have \( x \leq \text{foldl max} \ x \ \text{xs using Cons} \).
ultimately show \(?\text{thesis}\) by simp
next
case False
  thus \(?\text{thesis}\) using Cons by (simp add: max-def)
qed
qed

lemma list-max:
  fixes \( l :: \text{nat list} \)
  fixes \( i :: \text{nat} \)
  assumes 0 < = \( l \! \! i \)
  assumes 0 < = i
  assumes i < length \( l \)
  shows \( l \! \! i \leq \text{foldl max} \ 0 \ l \)
  using assms
proof (induct \( l \) arbitrary: i)
case Nil
  thus \(?\text{case}\) by simp
next
case (Cons \( x \ \text{xs} \))
  show \(?\text{case}\)
  proof (cases i)
  case (Suc \( j \))
  note \( \text{Cons(1)} \)
  moreover have 0 < = \( \text{xs} \! \! (i - 1) \) using Suc Cons by simp
  moreover have 0 < = i - 1 using Cons by simp
  moreover have i - 1 < length \( \text{xs} \) using Suc Cons by simp
  ultimately have \( \text{xs} \! \! (i - 1) \leq \text{foldl max} \ 0 \ \text{xs} \).
  moreover have \( \text{(x#xs)} \! \! i \ = \ \text{xs} \! \! (i - 1) \)
  using Suc Cons by simp
  moreover have \( \text{foldl max} \ 0 \ \text{xs} \leq \text{foldl max} \ (\text{max} \ 0 \ x) \ \text{xs} \)
  by (rule fold-max-leq) simp
  ultimately show \(?\text{thesis}\) by simp
next
case 0
  moreover have \( H : (\text{max} \ 0 \ x) \leq \text{foldl max} \ (\text{max} \ 0 \ x) \ \text{xs} \) using fold-max-lower
  by simp
  ultimately show \(?\text{thesis}\)
  by (cases 0 < = x) simp-all
qed
qed

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

8 Verification of $r_l$

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

abbreviation $r-l' :: \text{int} \Rightarrow \text{int}$ where
$r-l' == \text{Global-Specification}.r-l'$

end

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

lemma goal2'1:
  assumes $0 \leq j''$
  assumes $j'' \leq 79$
  shows $(\text{block-permutation---default-arr''})$
j'' =
$R-L$-Spark-Specification.r-l' j''

proof
  note nth-list-to-fun-off-eq
  moreover have $0 \leq \text{nat} j''$ by simp
  moreover from $ij'' \leq 79$, have $\text{nat} j'' < \text{length} r$-list
unfolding \( r \)-list-def by simp
ultimately have conversion:
\[
\begin{align*}
int (r \cdot ! \cdot nat j''') &= \\
list-to-fun-eff r \cdot list 0 block-permutation---default-arr'' (int (nat j''')) .
\end{align*}
\]
show thesis
unfolding \( r \)-def conversion
unfolding \( r \)-list-def
using \((0 \leq j''') (j'' \leq 79)\)
by simp
qed

lemma goal2'2:
assumes \(0 \leq j''\)
assumes \(j'' \leq 79\)
shows \(0 \leq (block-permutation---default-arr''\)
\(j''\)
unfolding goal2'1[OF assms]
by simp

lemma goal2'3:
assumes \(0 \leq j''\)
assumes \(j'' \leq 79\)
shows \((block-permutation---default-arr''\)
\[ j'' \leq 15 \]

**proof**

- have \( r\text{-list} \ni \text{nat } j'' \leq \text{foldl max } 0 \text{ r\text{-list}} \)
  - by (insert assms, rule list-max) (simp-all add: r-list-def)

**thus** \( \text{thesis unfolding } \text{goal2'}[\text{OF } \text{assms } \text{r-def} \]
  - by (rule list-max-int) (simp-all add: assms r-list-def)

**qed**

**lemmas** userlemmas = \( \text{goal2'}[1 \text{ goal2'}[2 \text{ goal2'}[3 \text { end}}

### 9 Verification of \( r_r \)

**theory** \( \text{R-R-Spark-Specification} \)

**imports** \( \text{Global-Specification R-R-Spark-Declaration} \)

**begin**

**abbreviation** \( r\text{-r'} \text{ where} \)

\[ r\text{-r'} = \text{Global-Specification}\_r\text{-r'} \]

**end**

**theory** \( \text{R-R-Spark-User} \)

**imports**

- \( \text{R-R-Spark-Specification} \)
- \( \text{R-R-Spark-Declaration} \)
- \( \text{Global-User} \)

**begin**

**lemma** \( \text{goal2'}[1] \)

- **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 := 15, 75 := 14, 76 := 0, 77 := 3, \\
78 := 9, 79 := 11))\)

\[ j'' = \]

\( \text{R-R-Spark-Specification}\_r\text{-r'} \ j'' \)

19
proof

note nth-list-to-fun-eff.eq
moreover have 0 <= nat j'' by simp
moreover from j'' <= 79: have nat j'' < length r'-list
  unfolding r'-list-def by simp
ultimately have conversion:
  int (r'-list ! nat j'') =
  list-to-fun-eff
  r'-list 0 block-permutation---default-arr'' (int (nat j'')).
show thesis unfolding r'-def conversion
  unfolding r'-list-def using (0 <= j'; j'' <= 79)
by simp
qed

lemma goal2'2:
  assumes 0 <= j''
  assumes j'' <= 79
  shows 0 <= (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 := 0, 69 := 7, 70 := 8,
     71 := 7, 72 := 6, 73 := 2, 74 := 13, 75 := 14, 76 := 0, 77 := 3,
     78 := 9, 79 := 11))
  j''
  unfolding goal2'1[OF assms]
by simp

lemma goal2'3:
  assumes 0 <= j''
  assumes j'' <= 79
  shows (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 := 0, 69 := 7, 70 := 8,
     71 := 7, 72 := 6, 73 := 2, 74 := 13, 75 := 14, 76 := 0, 77 := 3,
     78 := 9, 79 := 11, 80 := 3, 81 := 11, 82 := 14, 83 := 0, 84 := 4,
     85 := 6, 86 := 9, 87 := 11, 88 := 15, 89 := 8, 90 := 10, 91 := 7,
     92 := 63, 93 := 13, 94 := 2, 95 := 10, 96 := 7, 97 := 4, 98 := 15,
     99 := 14, 100 := 3, 101 := 11, 102 := 0, 103 := 10, 104 := 8, 105 := 7,
     112 := 3, 113 := 11, 114 := 2, 115 := 10, 116 := 7, 117 := 4,
proof -
  have \( \prod \leq \text{foldl max 0 } \prod \)\( \text{list} \)
    by (insert assms, rule list-max) (simp-all add: \( \prod \text{list-def} \))
  thus \(?thesis\) unfolding goal2'1 OF assms \( \prod \text{list-def} \)
    by (rule list-max-int) (simp-all add: assms \( \prod \text{list-def} \))
qed

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

end

10 Verification of \( s_l \)

theory S-L-Spark-Specification
imports Global-Specification S-L-Spark-Declaration
begin

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

end

theory S-L-Spark-User
imports
  S-L-Spark-Specification
  S-L-Spark-Declaration
  Global-User
begin

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

\[ \text{lemma } \text{goal2'}: \]
\[ \text{assumes } 0 \leq j'' \]
\[ \text{assumes } j'' \leq 79 \]
\[ \text{shows } 0 \leq (\text{rotate-definition---default-arr}''(0 \leq j'')) \]
\[ j'' \]
\[ \text{unfolding } \text{goal2'}[\text{OF assms}] \]
\[ \text{by simp} \]

\[ \text{lemma } \text{goal2'}3: \]
\[ \text{assumes } 0 \leq j'' \]
\[ \text{assumes } j'' \leq 79 \]
\[ \text{shows } (\text{rotate-definition---default-arr}''(0 \leq j'')) \]
\[ j'' \]
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)

\text{j''} \leq 15

\text{proof –}

\text{have s-list ! nat} \ j'' \leq \text{foldl max} \ 0 \ s-list

\text{by (insert assms, rule list-max) (simp-all add: s-list-def)}

\text{thus} \ \text{?thesis unfolding goal2'1[OF assms] s-def}

\text{by (rule list-max-int) (simp-all add: assms s-list-def)}

\text{qed}

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

\text{end}

11 Verification of \text{s_r}

\text{theory S-R-Spark-Specification}

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

\text{begin}

\text{abbreviation} \ \mathit{s-r'} :: \text{int} \Rightarrow \text{int} \ \text{where}

\mathit{s-r'} \ = \ \text{Global-Specification.s-r'}

\text{end}

\text{theory S-R-Spark-User}

\text{imports}

\text{S-R-Spark-Specification}

\text{S-R-Spark-Declaration}

\text{Global-User}

\text{begin}

\text{lemma goal2'1:}

\text{assumes 0 <= j''}

\text{assumes j'' <= 79}

\text{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, 5,
 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)

23
\[ j'' = \]
\[ S-R-Spark-Specification.s-r^\prime j'' \]

**proof**

```latex
text{note nth-list-to-fun-eff eq}
moreover have 0 \leq nat j'' by simp
moreover from \( j'' \leq 79 \) have nat \( j'' \) < length s'\text{-}list
  unfolding s'\text{-}list-def by simp
ultimately have conversion:
  int (s'\text{-}list ! nat j'') =
  list-to-fun-eff
  s'\text{-}list 0 \text{ rotate-definition---default-arr''} (int (nat j'')).
show \(?thesis unfolding s'\text{-}def conversion
  unfolding s'\text{-}list-def using (0 \leq j'' \& \ j'' \leq 79)
  by simp
qed
```

**lemma** goal2'2:

assumes 0 \leq j''
assumes \( j'' \leq 79 \)
shows \( 0 \leq (\text{rotate-definition---default-arr''} \)
  \( \langle 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) \rangle \]
\( j'' \)

unfolding goal2'1[of assms]
by simp

**lemma** goal2'3:

assumes 0 \leq j''
assumes \( j'' \leq 79 \)
shows (\text{rotate-definition---default-arr''} \)
  \( \langle 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) \rangle \]
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'' \leq 15
proof
  have s'-list ! nat j'' \leq foldl max 0 s'-list
    by (insert assms, rule list-max) (simp-all add: s'-list-def)
  thus ?thesis unfolding goal2'1[OF assms] s'-def
    by (rule list-max-int) (simp-all add: assms s'-list-def)
qed

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
  bit-xor' == Global-Specification.bit--xor'
abbreviation f' :: [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'

25
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
(\%n. word-of-int (b (int n)))
(from-chain-pair cc)
(nat i))

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
  unfolding int-word-uint
  using assms
  by (simp add:int-mod-eq')

lemma steps-step: steps X cc (Suc i) = step-both X (steps X cc i) i
  unfolding steps-def
  by (induct i) simp-all

lemma from-to-id: from-chain-pair (to-chain-pair CC) = CC
proof (cases CC)
  fix a::chain
  fix b c d e f::word32
  assume CC = (a, b, c, d, e, f)
  thus ?thesis by (cases a) simp
qed

lemma steps'-step:
  assumes 0 <= i
  shows steps' cc (i + 1) X = to-chain-pair (step-both
    (\n. word-of-int (X (int n)))
    (from-chain-pair (steps' cc i X))
    (nat i))
proof
  have nat (i + 1) = Suc (nat i) using assms by simp
  show ?thesis
    unfolding (nat (i + 1) = Suc (nat i)): steps-step steps-to-steps'
  ..
qed

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 c-0
  fixes x
  fixes j
  assumes
  step-hyp:
chain-pair---default-rcd"
\[\text{left'chain-pair := chain---default-rcd}\]
\[\text{\{h0'chain := a, h1'chain := b, h2'chain := c, h3'chain := d,}
\text{ h4'chain := e\}\},
\text{right'chain-pair := chain---default-rcd}\]
\[\text{\{h0'chain := a', h1'chain := b', h2'chain := c', h3'chain := d',}
\text{ h4'chain := e'\}\} =
\text{steps'}
\text{(chain-pair---default-rcd')}
\[\text{\{left'chain-pair := chain---default-rcd\}}
\text{\{h0'chain := a-0, h1'chain := b-0, h2'chain := c-0,}
\text{ h3'chain := d-0, h4'chain := e-0\}},
\text{right'chain-pair := chain---default-rcd\}}
\[\text{\{h0'chain := a-0, h1'chain := b-0, h2'chain := c-0,}
\text{ h3'chain := d-0, h4'chain := e-0\}}\}
\text{j x}
\text{assumes a-borders: 0 <= a a <= 4294967295 (is - <= ?M)}
\text{assumes b-borders: 0 <= b b <= ?M}
\text{assumes c-borders: 0 <= c c <= ?M}
\text{assumes d-borders: 0 <= d d <= ?M}
\text{assumes e-borders: 0 <= e e <= ?M}
\text{assumes a'-borders: 0 <= a' a' <= ?M}
\text{assumes b'-borders: 0 <= b' b' <= ?M}
\text{assumes c'-borders: 0 <= c' c' <= ?M}
\text{assumes d'-borders: 0 <= d' d' <= ?M}
\text{assumes e'-borders: 0 <= e' e' <= ?M}
\text{assumes x-borders: 0 <= x (r-l' j) x (r-l' j) <= ?M}
\text{0 <= x (r-r' j) x (r-r' j) <= ?M}
\text{assumes j-borders: 0 <= j j <= 79}
\text{shows}
\text{chain-pair---default-rcd''}
\[\text{\{left'chain-pair := chain---default-rcd\}'}
\text{\{h0'chain := e,}
\text{ h1'chain :=}
\text{ (wordops--rotate-left' (s-l' j))}
\text{ (((a + f' j) b c d) mod 4294967296 +}
\text{ x (r-l' j)) mod}
\text{ 4294967296 +}
\text{ k-l' j) mod}
\text{ 4294967296) +}
\text{ e) mod}
\text{ 4294967296,}
\text{ h2'chain := b, h3'chain := wordops--rotate-left' 10 c,}
\text{ h4'chain := d\},
\text{right'chain-pair := chain---default-rcd''}
\[\text{\{h0'chain := e',}
\text{ h1'chain :=}
\text{ (wordops--rotate-left' (s-r' j))}
\text{ (((a' + f' (79 - j) b' c' d') mod}
\text{ 4294967296 +}
\text{ k-r' j) mod}
\text{ 4294967296) +}
\text{ e') mod}
\text{ 4294967296,}
\text{ h2'chain := b, h3'chain := wordops--rotate-left' 10 c,}
\text{ h4'chain := d\}}\}
\[4294967296 + x (r-r') \mod 4294967296 + \]
\[4294967296 + k-r' \mod 4294967296 + e' \mod 4294967296,\]
\[h_2' \text{chain} := b', h_3' \text{chain} := \text{wordops--rotate-left'} 10 c',\]
\[h_4' \text{chain} := d'[0] =\]

steps'

(chain-pair---default-red''

\[\langle h_0' \text{chain} := a-0, h_1' \text{chain} := b-0, h_2' \text{chain} := c-0,\]
\[h_3' \text{chain} := d-0, h_4' \text{chain} := e-0]\])

(j + 1) x

proof

let \(\text{MM} = 4294967296\)

have \(AL\): \(\text{uint}(\text{word-of-int } e::\text{word32}) = e\)

by \(\text{rule} \ \text{uint-word-of-int-id}[OF : 0 <= e \ (e <= \ ?M[:])]\)

have \(CL\): \(\text{uint}(\text{word-of-int } b::\text{word32}) = b\)

by \(\text{rule} \ \text{uint-word-of-int-id}[OF : 0 <= b \ (b <= \ ?M[:])]\)

have \(DL\): True ..

have \(EL\): \(\text{uint}(\text{word-of-int } d::\text{word32}) = d\)

by \(\text{rule} \ \text{uint-word-of-int-id}[OF : 0 <= d \ (d <= \ ?M[:])]\)

have \(AR\): \(\text{uint}(\text{word-of-int } e'::\text{word32}) = e'\)

by \(\text{rule} \ \text{uint-word-of-int-id}[OF : 0 <= e' \ (e' <= \ ?M[:])]\)

have \(CR\): \(\text{uint}(\text{word-of-int } b'::\text{word32}) = b'\)

by \(\text{rule} \ \text{uint-word-of-int-id}[OF : 0 <= b' \ (b' <= \ ?M[:])]\)

have \(DR\): True ..

have \(ER\): \(\text{uint}(\text{word-of-int } d'::\text{word32}) = d'\)

by \(\text{rule} \ \text{uint-word-of-int-id}[OF : 0 <= d' \ (d' <= \ ?M[:])]\)

have \(BL\): \(\langle \text{uint} \ (\text{word-rotl} \ (s \ (\text{nat } j))\))\)

\((\text{word-of-int::int} \Rightarrow \text{word32})\)

\(((\text{word-of-int a} + f \ (\text{nat } j) \ (\text{word-of-int } b) \ (\text{word-of-int } c) \ (\text{word-of-int } d) + \text{word-of-int} \ (x \ (r-l' j)) + \)

\[4294967296 = \]

\[\text{uint} \ (\text{word-rotl} \ (s \ (\text{nat } j))) \]

\(\text{word-of-int a} + f \ (\text{nat } j) \ (\text{word-of-int } b) \ (\text{word-of-int } c) \ (\text{word-of-int } d) + \text{word-of-int} \ (x \ (r-l' j)) + \)
\[ K \text{(nat } j) \] + \\
\text{word-of-int } e \] \\
(is \text{(uint \text{word-rotl } - (\text{(((}\text{mod } + \text{?X} \text{ mod } - ) \text{ mod } - )) + -) mod -= - )}) \\
\text{proof} - \\
\text{have a mod ?MM = a using } \langle 0 <= a \rangle \langle a <= ?M \rangle \] \\
\text{by (simp add: \text{int-mod-eq}')} \\
\text{have \text{?X mod ?MM = ?X using } \langle 0 <= \text{?X} \rangle \langle \text{?X <= ?M} \rangle} \\
\text{by (simp add: \text{int-mod-eq}')} \\
\text{have e mod ?MM = e using } \langle 0 <= e \rangle \langle e <= \text{?M} \rangle \\
\text{by (simp add: \text{int-mod-eq}'')} \\
\text{have (?MM::int) = 2 ^ len-of \text{TYPE}(32) by simp} \\
\text{show ?thesis unfolding} \\
\text{word-add-def} \\
\text{uint-word-of-int-id[OF \langle 0 <= a \rangle \langle a <= ?M \rangle]} \\
\text{uint-word-of-int-id[OF \langle 0 <= \text{?X} \rangle \langle \text{?X <= ?M} \rangle]} \\
\text{int-word-uint unfolding (?MM = 2 ^ len-of \text{TYPE}(32))} \\
\text{unfolding word-uint.Abs-norm by (simp add: \langle a mod ?MM = a \rangle \langle e mod ?MM = e \rangle \langle \text{?X mod ?MM = ?X} \rangle)} \\
\text{qed} \\
\text{have \text{BR: (uint \text{word-rotl } (\text{nat } j) \} \} \text{word-of-int } a' \] + \\
\text{(\text{word-of-int::int} => \text{word32} \} \text{(((}a' + f' (79 - j) b' c' d') mod 4294967296 + \\
x (r-r' j) mod 4294967296 + \\
k-r' j) mod 4294967296)) + \\
e') mod 4294967296 = } \\
\text{uint} \\
\text{(\text{word-rotl } (\text{nat } j) \} \text{word-of-int } a' \] + \\
f (79 - \text{nat } j) \text{\ (\text{word-of-int } b') \text{\ (\text{word-of-int } c') \text{\ (\text{word-of-int } d') \} \} \text{\ \text{word-of-int } (x \text{\ (r-r' j) + \text{\ K' (nat } j) + \text{\ word-of-int } e') \} \} \text{\ (is \text{(uint \text{word-rotl } - (\text{(((\text{mod } + \text{?F} \text{ mod } - ) \text{ mod } - ) \text{ mod } - )) + -) mod -= - )})}} + -) \\
\text{mod -= - )} \\
\text{proof -} \\
\text{have a' mod ?MM = a' using } \langle 0 <= a' \rangle \langle a' <= ?M \rangle \\
\text{by (simp add: \text{int-mod-eq}'')} \]
have \( ?X \mod \ ?MM = ?X \) using \( 0 \leq ?X \) \( ?X \leq \ ?M \)
by (simp add: int-mod-eq)

have \( e' \mod \ ?MM = e' \) using \( 0 \leq e' \) \( e' \leq \ ?M \)
by (simp add: int-mod-eq)

have \( (?MM::int) = 2 \ast \text{len-of TYPE(32)} \) by simp

have nat-transfer: \( 79 - nat \ j = nat \ (79 - j) \)
using nat-diff-distrib \( 0 \leq j \) \( j \leq 79 \)
by simp

show \( ?thesis \)

unfolding
word-add-def

\[ \text{uint-word-of-int-id} (OF \langle 0 \leq a \rangle \langle a' \leq \ ?M \rangle) \]
\[ \text{uint-word-of-int-id} (OF \langle 0 \leq ?X \rangle \langle ?X \leq \ ?M \rangle) \]
\[ \text{int-word-uint} \]
\[ \text{nat-transfer} \]

unfolding \( (?MM = 2 \ast \text{len-of TYPE(32)}) \)

unfolding word-uint.Abs-norm

by (simp add:
\( \langle a' \mod \ ?MM = a \rangle \)
\( \langle e' \mod \ ?MM = e' \rangle \)
\( \langle ?X \mod \ ?MM = ?X \rangle \)

qed

show \( ?thesis \)

unfolding steps'-step[OF \( 0 \leq j \)] \[ \text{step-hyp}[\text{symmetric}] \]
\[ \text{step-both-def} \text{ step-r-def} \text{ step-l-def} \]

by (simp add: AL BL CL DL EL AR BR CR DR ER)

qed

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

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

lemma goal61':
assumes ca-borders: \( 0 \leq ca'' ca''' \leq 4294967295 \) (is - <= \ ?M )
assumes cb-borders: \( 0 \leq cb'' cb''' \leq \ ?M \)
assumes cc-borders: \( 0 \leq cc'' cc''' \leq \ ?M \)
assumes cd-borders: \( 0 \leq cd'' cd''' \leq \ ?M \)
assumes ce-borders: \( 0 \leq ce'' ce''' \leq \ ?M \)
assumes r-l-borders: \( 0 \leq r-l-spark' \ 0 r-l-spark' 0 \leq 15 \)
assumes r-r-borders: \( 0 \leq r-r-spark' \ 0 r-r-spark' 0 \leq 15 \)
assumes returns:
wordops--rotate' (s-l-spark' 0 ) \ f-0\text{-result} =
wordops--rotate-left' (s-l-spark' 0 ) \ f-0\text{-result} \nwordops--rotate' (s-r-spark' 0 ) \ f-79\text{-result} =
wordops--rotate-left' (s-r-spark' 0 ) \ f-79\text{-result}
\[ \text{wordops--rotate}' 10 \text{ cc}'' = \text{wordops--rotate-left}' 10 \text{ cc}'' \]
\[ f\text{-spark}' 0 \text{ cb}'' \text{ cc}'' \text{ cd}'' = f' 0 \text{ cb}'' \text{ cc}'' \text{ cd}'' \]
\[ f\text{-spark}' 79 \text{ cb}'' \text{ cc}'' \text{ cd}'' = f' 79 \text{ cb}'' \text{ cc}'' \text{ cd}'' \]
\[ k\text{-l-spark}' 0 = k\text{-l}' 0 \]
\[ k\text{-r-spark}' 0 = k\text{-r}' 0 \]
\[ r\text{-l-spark}' 0 = r\text{-l}' 0 \]
\[ r\text{-r-spark}' 0 = r\text{-r}' 0 \]
\[ s\text{-l-spark}' 0 = s\text{-l}' 0 \]
\[ s\text{-r-spark}' 0 = s\text{-r}' 0 \]
\[ \text{assumes x-borders: } \forall i. \ 0 \leq i \wedge i \leq 15 \rightarrow 0 \leq x'' i \wedge x'' i \leq 15 \]
\[ \text{shows chain-pair---default-rcd}'' \]
\[ \left\{ \begin{array}{l}
\text{left}'\text{chain-pair} := \text{chain---default-rcd}'' \\
\{ h0'\text{chain} := \text{cc}'' , \\
h1'\text{chain} := \text{(wordops--rotate}' (s\text{-l-spark}' 0) \\
 (((ca'' + f\text{-spark}' 0 cb'' cc'' cd'')) \mod 4294967296 + \\
 x'' (r\text{-l-spark}' 0)) \mod \\
 4294967296 + \\
k\text{-l-spark}' 0) \mod \\
 4294967296 + \\
 cc'') \mod \\
 4294967296, \\
h2'\text{chain} := cb'' , h3'\text{chain} := \text{wordops--rotate}' 10 \text{ cc}'' , \\
h4'\text{chain} := cd'' ]
\end{array} \right. \]
\[ \text{right}'\text{chain-pair} := \text{chain---default-rcd}'' \\
\left\{ \begin{array}{l}
\{ h0'\text{chain} := \text{cc}'' , \\
h1'\text{chain} := \text{(wordops--rotate}' (s\text{-r-spark}' 0) \\
 (((ca'' + f\text{-spark}' 79 cb'' cc'' cd'')) \mod 4294967296 + \\
 x'' (r\text{-r-spark}' 0)) \mod \\
 4294967296 + \\
k\text{-r-spark}' 0) \mod \\
 4294967296 + \\
 cc'') \mod \\
 4294967296, \\
h2'\text{chain} := cb'' , h3'\text{chain} := \text{wordops--rotate}' 10 \text{ cc}'' , \\
h4'\text{chain} := cd'' ]
\end{array} \right. = \]
\[ \text{steps}' \\
\left\{ \begin{array}{l}
\text{left}'\text{chain-pair} := \text{chain---default-rcd}'' \\
\{ h0'\text{chain} := ca''' , h1'\text{chain} := cb''' , h2'\text{chain} := cc''' , \\
h3'\text{chain} := cd''' , h4'\text{chain} := ce''' ]
\end{array} \right. \]
\[ \text{right}'\text{chain-pair} := \text{chain---default-rcd}'' \\
\left\{ \begin{array}{l}
\{ h0'\text{chain} := ca''' , h1'\text{chain} := cb''' , h2'\text{chain} := cc''' , \\
h3'\text{chain} := cd''' , h4'\text{chain} := ce''' ]
\end{array} \right. \\
1 x'' \]
\[ \text{proof} – \\
\text{have step-hyp:} \\
\text{chain-pair---default-rcd}'' \]
\( \text{left'}\text{chain-pair} := \text{chain---default-rcd}'' \)
\( \{ h0'\text{chain} := \text{ca}'', h1'\text{chain} := \text{cb}'', h2'\text{chain} := \text{cc}'', h3'\text{chain} := \text{cd}'', h4'\text{chain} := \text{ce}'\} \)

\( \text{right'}\text{chain-pair} := \text{chain---default-rcd}'' \)
\( \{ h0'\text{chain} := \text{ca}'', h1'\text{chain} := \text{cb}'', h2'\text{chain} := \text{cc}'', h3'\text{chain} := \text{cd}'', h4'\text{chain} := \text{ce}'\} \)

\( \text{steps'} \)
\( \{ \text{chain-pair---default-rcd}'' \)
\( \{ h0'\text{chain} := \text{ca}'', h1'\text{chain} := \text{cb}'', h2'\text{chain} := \text{cc}'', h3'\text{chain} := \text{cd}'', h4'\text{chain} := \text{ce}'\} \)

\( \text{0} \times'' \)

unfolding steps-def 
by \{ simp add: \}
  | \text{uint-word-of-int-id[OF ca-borders]}
  | \text{uint-word-of-int-id[OF cb-borders]}
  | \text{uint-word-of-int-id[OF cc-borders]}
  | \text{uint-word-of-int-id[OF cd-borders]}
  | \text{uint-word-of-int-id[OF ce-borders]} \}

from r-l-0-borders x-borders
have \( 0 \leq x'' (r-l'\text{spark}' 0) \) by blast
hence \( x'' (r-l'\text{spark}' 0) \leq x'' (r-l' 0) \) unfolding returns .

from r-l-0-borders x-borders x-borders
have \( x'' (r-l'\text{spark}' 0) \leq ?M \) by blast
hence \( x'' (r-l'\text{spark}' 0) \leq ?M \) unfolding returns .

from r-r-0-borders x-borders
have \( 0 \leq x'' (r-r'\text{spark}' 0) \) by blast
hence \( x'' (r-r'\text{spark}' 0) \leq x'' (r-r' 0) \) unfolding returns .

from r-r-0-borders x-borders
have \( 0 \leq x'' (r-r'\text{spark}' 0) \) by blast
hence \( x'' (r-r'\text{spark}' 0) \leq ?M \) by blast
hence \( x'' (r-r'\text{spark}' 0) \leq ?M \) unfolding returns .

have \( 0 \leq (0 :: \text{int}) \) by simp
have \( 0 \leq (79 :: \text{int}) \) by simp
note step-from-hyp \[ OF \]
  | step-hyp
  \text{ca-borders} cb-borders cc-borders cd-borders ce-borders
cb-borders cc-borders cd-borders ce-borders
  \]
note this[OF x-lower x-upper x-lower' x-upper' \( (0 \leq 0) \) \( (0 \leq 79) \)]
thus \?thesis unfolding returns(1) returns(2) unfolding returns
by simp
**Qed**

**Abbreviation** `rotate-arg-l` =

\[
(((\text{cla}'' + f\text{-spark}' (\text{loop}--1--j'' + 1)) \text{clb}'' \text{cle}'' \text{cld}'') \mod 4294967296 + x'' (r-l-spark' (\text{loop}--1--j'' + 1))) \mod 4294967296 + k-l-spark' (\text{loop}--1--j'' + 1)) \mod 4294967296
\]

**Abbreviation** `rotate-arg-r` =

\[
(((\text{crc}'' + f\text{-spark}' (79 - (\text{loop}--1--j'' + 1)) \text{crb}'' \text{crc}'' \text{crd}'') \mod 4294967296 + x'' (r-r-spark' (\text{loop}--1--j'' + 1))) \mod 4294967296 + k-r-spark' (\text{loop}--1--j'' + 1)) \mod 4294967296
\]

**Lemma** `goal62'1`:

**Assumptions**
- `cla-borders`: `0 <= \text{cla}'' \text{cla}'' <= 4294967295` (is `<= ?M`)
- `clb-borders`: `0 <= \text{clb}'' \text{clb}'' <= ?M`
- `clc-borders`: `0 <= \text{clc}'' \text{clc}'' <= ?M`
- `cld-borders`: `0 <= \text{cld}'' \text{cld}'' <= ?M`
- `cle-borders`: `0 <= \text{cle}'' \text{cle}'' <= ?M`
- `cra-borders`: `0 <= \text{cra}'' \text{cra}'' <= ?M`
- `crb-borders`: `0 <= \text{crb}'' \text{crb}'' <= ?M`
- `cra-borders`: `0 <= \text{cre}'' \text{cre}'' <= ?M`
- `crd-borders`: `0 <= \text{crd}'' \text{crd}'' <= ?M`
- `cra-borders`: `0 <= \text{clb}'' \text{clb}'' <= ?M`

**Step Hyp:**

\[
\text{chain-pair}--\text{default-rcd}''
\]

\[
(\text{left}'\text{chain-pair} := \text{chain}--\text{default-rcd}''
\hspace{2em}(h0'\text{chain} := \text{cla}''', h1'\text{chain} := \text{clb}'', h2'\text{chain} := \text{cle}'',
\hspace{2em}h3'\text{chain} := \text{cld}'', h4'\text{chain} := \text{cle}''))
\hspace{2em}\text{right}'\text{chain-pair} := \text{chain}--\text{default-rcd}''
\]

\[
(\text{left}'\text{chain-pair} := \text{chain}--\text{default-rcd}''
\hspace{2em}(h0'\text{chain} := \text{ca}--\text{init}', h1'\text{chain} := \text{cb}--\text{init}'',
\hspace{2em}h2'\text{chain} := \text{cc}--\text{init}'', h3'\text{chain} := \text{cd}--\text{init}'',
\hspace{2em}h4'\text{chain} := \text{ce}--\text{init}''))
\hspace{2em}\text{right}'\text{chain-pair} := \text{chain}--\text{default-rcd}''
\]

\[
(\text{left}'\text{chain-pair} := \text{chain}--\text{default-rcd}''
\hspace{2em}(h0'\text{chain} := \text{ca}--\text{init}', h1'\text{chain} := \text{cb}--\text{init}'',
\hspace{2em}h2'\text{chain} := \text{cc}--\text{init}'', h3'\text{chain} := \text{cd}--\text{init}'',
\hspace{2em}h4'\text{chain} := \text{ce}--\text{init}''))
\]

\[
(\text{loop}--1--j'' + 1) x''
\]

**Returns:**

\[
\text{wordops--rotate}' (s-l-spark' (\text{loop}--1--j'' + 1)) \text{rotate-arg-r} =
\hspace{2em}\text{wordops--rotate-left}' (s-l-spark' (\text{loop}--1--j'' + 1)) \text{rotate-arg-l}
\]

\[
\text{wordops--rotate}' (s-r-spark' (\text{loop}--1--j'' + 1)) \text{rotate-arg-r} =
\hspace{2em}\text{wordops--rotate-left}' (s-r-spark' (\text{loop}--1--j'' + 1)) \text{rotate-arg-l}
\]

\[
(f' (\text{loop}--1--j'' + 1) \text{clb}'' \text{cle}'' \text{cld}'') =
\]

34
\[f'(78 - \text{loop}--1-j^\prime) \text{ crb'' crc'' crd''} =\]
\[f'(78 - \text{loop}--1-j^\prime) \text{ crb'' crc'' crd''}\]

Assumes \text{x-borders:}\]
\[\forall i, \: 0 \leq i \land i \leq 15 \implies 0 \leq x'' \land x'' i \leq ?M\]

Assumes \text{x-borders:}\]
\[\forall i, \: 0 \leq i \land i \leq 15 \implies 0 \leq x'' \land x'' i \leq ?M\]

Shows \text{chain-pair---default-rcd}'
\[\{\text{left'}chain-pair := \text{chain---default-rcd'}\} \]
\[\{\text{h0'}chain := \text{cle''}, \]
\[\text{h1'}chain :=\]
\[((\text{wordops--rotate'} (s-l-spark' (\text{loop}--1-j'' + 1))) \]
\[(((\text{clb''} + f-spark' (\text{loop}--1-j'' + 1)) \text{ cle'' crd''}) \text{ mod}\]
\[4294967296 +\]
\[x'' (r-l-spark' (\text{loop}--1-j'' + 1))) \text{ mod}\]
\[4294967296 +\]
\[k-l-spark' (\text{loop}--1-j'' + 1)) \text{ mod}\]
\[4294967296) +\]
\[\text{cle''}) \text{ mod}\]
\[4294967296, \]
\[h2'chain := \text{clb''}, h3'chain := \text{wordops--rotate'} 10 \text{ cle''}, \]
\[h4'chain := \text{cle'}}\}\]

Right \text{chain-pair := chain---default-rcd'}
\[\{\text{h0'}chain := \text{cre''}, \]
\[\text{h1'}chain :=\]
\[((\text{wordops--rotate'} (s-r-spark' (\text{loop}--1-j'' + 1))) \]
\[(((\text{cre''} +\]
\[f-spark' (79 - \text{loop}--1-j'' + 1)) \text{ crb'' crc''} \text{ crd''}) \text{ mod}\]
\[4294967296 +\]
\[x'' (r-r-spark' (\text{loop}--1-j'' + 1))) \text{ mod}\]
\[4294967296 +\]
\[k-r-spark' (\text{loop}--1-j'' + 1)) \text{ mod}\]
\[4294967296) +\]
\[\text{cre''}) \text{ mod}\]
\[4294967296, \]
\[h2'chain := \text{crb''}, h3'chain := \text{wordops--rotate'} 10 \text{ crc''}, \]
\[h4'chain := \text{crd''}]\} = \]

Steps'
\[(\text{chain-pair---default-rcd''} \]
\( \text{left-chain-pair} := \text{chain---default-rcd}'' \)
\( \text{h0'} \text{chain} := \text{ca---init''}, \text{h1'} \text{chain} := \text{cb---init''}, \text{h2'} \text{chain} := \text{cc---init''}, \text{h3'} \text{chain} := \text{cd---init''}, \text{h4'} \text{chain} := \text{ce---init''} \),
\( \text{right-chain-pair} := \text{chain---default-rcd}'' \)
\( \text{h0'} \text{chain} := \text{ca---init''}, \text{h1'} \text{chain} := \text{cb---init''}, \text{h2'} \text{chain} := \text{cc---init''}, \text{h3'} \text{chain} := \text{cd---init''}, \text{h4'} \text{chain} := \text{ce---init''} \)}
(\( \text{loop--1--j'' + 2} \) \( x'' \))

**proof**

have \( s: 78 = \text{loop--1--j''} = (79 - (\text{loop--1--j'' + 1})) \) by simp

from \( \text{r-l-borders} \ \text{x-borders} \)
have \( 0 \leq x'' (\text{r-l-spark'} (\text{loop--1--j'' + 1})) \) by blast
hence \( x'' \text{lower}: 0 <= x'' (\text{r-l'} (\text{loop--1--j'' + 1})) \) unfolding returns .

from \( \text{r-l-borders} \ \text{x-borders} \)
have \( x'' (\text{r-l-spark'} (\text{loop--1--j'' + 1})) <= ?M \) by blast
hence \( x'' \text{upper}: x'' (\text{r-l'} (\text{loop--1--j'' + 1})) <= ?M \) unfolding returns .

from \( \text{r-r-borders} \ \text{x-borders} \)
have \( 0 \leq x'' (\text{r-r-spark'} (\text{loop--1--j'' + 1})) \) by blast
hence \( x'' \text{lower'}: 0 <= x'' (\text{r-r'} (\text{loop--1--j'' + 1})) \) unfolding returns .

from \( \text{r-r-borders} \ \text{x-borders} \)
have \( x'' (\text{r-r-spark'} (\text{loop--1--j'' + 1})) <= ?M \) by blast
hence \( x'' \text{upper'}: x'' (\text{r-r'} (\text{loop--1--j'' + 1})) <= ?M \) unfolding returns .

from \( \text{j-loop-1-borders} \)
have \( 0 <= \text{loop--1--j'' + 1} \) by simp
from \( \text{j-loop-1-borders} \)
have \( \text{loop--1--j'' + 1} <= 79 \) by simp

have \( \text{loop--1--j'''' + 1 + 1} = \text{loop--1--j'''' + 2} \) by simp

have \( f' (79 - (\text{loop--1--j'''' + 1})) \text{ crb'' crc'' crd''} = \text{f-spark'} (79 - (\text{loop--1--j'''' + 1})) \text{ crb'' crc'' crd''} \)
using returns by simp

**note** returns = returns this

**note** step-from-hyp[OF step-hyp]
cla-borders
clb-borders
clc-borders
cld-borders
cle-borders
cra-borders
crb-borders
crc-borders
crd-borders
crc-borders]
from this OF
  x-lower x-upper x-lower' x-upper'
\(0 <= \text{loop}--1--j'' + 1\) \(\text{loop}--1--j'' + 1 <= 79\)
show ?thesis unfolding \(\text{loop}--1--j'' + 1 + 1 = \text{loop}--1--j'' + 2\)
  unfolding returns(1) returns(2) unfolding returns
by simp
qed

abbreviation INIT-CHAIN == chain--default-rcd''
  \(\langle h0'\text{chain} := ca---\text{init}''\rangle, h1'\text{chain} := cb---\text{init}''\),
  h2'\text{chain} := cc---\text{init}''\), h3'\text{chain} := cd---\text{init}''\),
  h4'\text{chain} := ce---\text{init}''\)

lemma goal76'1:
  assumes cla-borders: \(0 <= \text{cla}''\) \(\text{cla}'' <= 4294967295\) (is - <= ?M)
  assumes clb-borders: \(0 <= \text{clb}''\) \(\text{clb}'' <= ?M\)
  assumes cle-borders: \(0 <= \text{cle}''\) \(\text{cle}'' <= ?M\)
  assumes clc-borders: \(0 <= \text{clc}''\) \(\text{clc}'' <= ?M\)
  assumes cld-borders: \(0 <= \text{cld}''\) \(\text{cld}'' <= ?M\)
  assumes cle-borders: \(0 <= \text{cle}''\) \(\text{cle}'' <= ?M\)
  assumes cra-borders: \(0 <= \text{cra}''\) \(\text{cra}'' <= ?M\)
  assumes crb-borders: \(0 <= \text{crb}''\) \(\text{crb}'' <= ?M\)
  assumes crc-borders: \(0 <= \text{crc}''\) \(\text{crc}'' <= ?M\)
  assumes crd-borders: \(0 <= \text{crd}''\) \(\text{crd}'' <= ?M\)
  assumes cla-borders: \(0 <= \text{cla}''\) \(\text{cla}'' <= ?M\)
  assumes cre-borders: \(0 <= \text{cre}''\) \(\text{cre}'' <= ?M\)
  assumes ca-init-borders: \(0 <= \text{ca---init}''\) \(\text{ca---init}'' <= ?M\)
  assumes cb-init-borders: \(0 <= \text{cb---init}''\) \(\text{cb---init}'' <= ?M\)
  assumes cc-init-borders: \(0 <= \text{cc---init}''\) \(\text{cc---init}'' <= ?M\)
  assumes cd-init-borders: \(0 <= \text{cd---init}''\) \(\text{cd---init}'' <= ?M\)
  assumes ce-init-borders: \(0 <= \text{ce---init}''\) \(\text{ce---init}'' <= ?M\)
  assumes left'chain-pair := chain--default-rcd''
    \(\langle h0'\text{chain} := \text{cla}''\), h1'\text{chain} := \text{clb}''\), h2'\text{chain} := \text{clc}''\)
    h3'\text{chain} := \text{cld}''\),
    h4'\text{chain} := \text{cle}''\),
    right'chain-pair := chain--default-rcd''
      \(\langle h0'\text{chain} := \text{cra}''\), h1'\text{chain} := \text{crb}''\), h2'\text{chain} := \text{crc}''\)
      h3'\text{chain} := \text{crd}''\)
steps'
  (chain-pair---default-rcd''
    \(\langle h0'\text{chain} := \text{ca---init}''\), h1'\text{chain} := \text{cb---init}''\)
    h2'\text{chain} := \text{cc---init}''\),
    h3'\text{chain} := \text{cd---init}''\)
    h4'\text{chain} := \text{ce---init}''\)
    right'chain-pair := chain--default-rcd''
      \(\langle h0'\text{chain} := \text{ca---init}''\), h1'\text{chain} := \text{cb---init}''\), h2'\text{chain} := \text{cc---init}''\)
      h3'\text{chain} := \text{cd---init}''\)
      h4'\text{chain} := \text{ce---init}''\)
80 \(x''\)
shows chain--default-rcd''
\( \langle h_0 \rangle_{\text{chain}} := ((cb - \text{init}'' + clc'') \mod 4294967296, \) 
\( h_1 \text{'}_{\text{chain}} := ((cc - \text{init}'' + cld'') \mod 4294967296 + cre'') \mod 4294967296, \) 
\( h_2 \text{'}_{\text{chain}} := ((cd - \text{init}'' + cld'') \mod 4294967296 \mod 4294967296, \) 
\( h_3 \text{'}_{\text{chain}} := ((ce - \text{init}'' + cla'') \mod 4294967296 + crb'') \mod 4294967296, \) 
\( h_4 \text{'}_{\text{chain}} := ((ca - \text{init}'' + clb'') \mod 4294967296 + crc'') \mod 4294967296) \)

\[ \text{round}' \]
\[(\text{chain-\text{default-rcd}''} \]
\( \langle h_0 \rangle_{\text{chain}} := ca - \text{init}'' , h_1 \text{'}_{\text{chain}} := cb - \text{init}'' , h_2 \text{'}_{\text{chain}} := cc - \text{init}'' , \)
\( h_3 \text{'}_{\text{chain}} := cd - \text{init}'' , h_4 \text{'}_{\text{chain}} := ce - \text{init}'' \rangle \)

\[ x'' \]

**proof** –

**have** steps-to-steps:’
steps
\((\lambda n::\text{nat}. \text{word-of-int} (x'' (\text{int } n))) \)
(from-chain INIT-CHAIN, from-chain INIT-CHAIN)
\(80 = \)
from-chain-pair (steps’
\( (\text{chain-pair-\text{default-rcd}''} \]
\( \langle \text{left}'\text{chain-pair} := \text{INIT-CHAIN , right}'\text{chain-pair} := \text{INIT-CHAIN} \rangle) \)
\(80 \)
\( x'' \)

**unfolding** from-to-id by simp
**show** ?thesis
**unfolding** round-def
**unfolding** steps-to-steps’
**unfolding** step-hyp[\text{symmetric}]
by (simp add: \text{uint-word-ariths}(1) \text{ rdmods}
\text{uint-word-of-int-\text{id}(OF ca-init-borders]} \)
\text{uint-word-of-int-\text{id}(OF cb-init-borders]} \)
\text{uint-word-of-int-\text{id}(OF cc-init-borders]} \)
\text{uint-word-of-int-\text{id}(OF cd-init-borders]} \)
\text{uint-word-of-int-\text{id}(OF ce-init-borders]} \)
\text{uint-word-of-int-\text{id}(OF cla-borders]} \)
\text{uint-word-of-int-\text{id}(OF clb-borders]} \)
\text{uint-word-of-int-\text{id}(OF cle-borders]} \)
\text{uint-word-of-int-\text{id}(OF crd-borders]} \)
\text{uint-word-of-int-\text{id}(OF cre-borders]} \)
\text{uint-word-of-int-\text{id}(OF crb-borders]} \)
\text{uint-word-of-int-\text{id}(OF crc-borders]} \)
\text{uint-word-of-int-\text{id}(OF cre-borders]} \)

\[ \text{qed} \]

**lemmas** userlemmas = goal61'1 goal62'1 goal76'1
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-red''
     (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 \textit{goal12'1}:
assumes \textit{H1}: \textit{x--index--subtype--1--first''} = (0 :: int)

assumes \textit{H6}:
chain---default-rcd''
( ( h0'chain
    := \textit{ca--1}''
) )
( ( h1'chain
    := \textit{cb--1}''
) )
( ( h2'chain
    := \textit{cc--1}''
) )
( ( h3'chain
    := \textit{cd--1}''
) )
( ( h4'chain
    := \textit{ce--1}''
) )
= \textit{round'}
  ( chain---default-rcd''
( ( h0'chain
    := \textit{1732584193 :: int}
) )
( ( h1'chain
    := \textit{4023233417 :: int}
) )
( ( h2'chain
    := \textit{2562383102 :: int}
) )
( ( h3'chain
    := \textit{271733878 :: int}
) )
( ( h4'chain
    := \textit{3285377520 :: int}
) )
) ( \textit{x'' x--index--subtype--1--first''} )

shows chain---default-rcd''
( ( h0'chain
    := \textit{ca--1}''
) )
( ( h1'chain
    := \textit{cb--1}''
) )
( ( h2'chain
    := \textit{cc--1}''
) )
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{rounds''}(\text{chain---default-rcd''}) = \text{rounds'} \]
unfolding from-to-id ..

lemma rounds'-step:
    assumes 0 <= i
    shows rounds' c (i + 1) x = round' (rounds' c i x) (x i)
proof -
    have makesuc: nat (i + 1) = Suc (nat i) using assms by simp
    show ?thesis using assms
      by (simp add: makesuc rounds-def rmd-body-def steps-to-steps')
qed

lemma goal13'1:
    assumes 0 <= loop--1--i''
    assumes H1:
      chain---default-rcd''
      (| h0'chain := ca''
          |
          | h1'chain := cb''
          |
          | h2'chain := ca''
          |
          | 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 \( H18: \)

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

\[
= \text{round'} \\
\quad ( \text{chain---default-rcd}'' \\
(1) \ h0'\text{chain} \\
\quad := ca'' \\
(1) \ h1'\text{chain} \\
\quad := cb'' \\
(1) \ h2'\text{chain} \\
\quad := cc'' \\
(1) \ h3'\text{chain} \\
\quad := cd'' \\
(1) \ h4'\text{chain} \\
\quad := ce'' \\
\quad ) \\
\quad ( x'' ( \text{loop---1}'' + (1 :: \text{int}) ) \\
\quad )
\]

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

\[
\begin{align*}
(1) \ h0'\text{chain} \\
\quad := ca-1'' \\
(1) \ h1'\text{chain} \\
\quad := cb-1'' \\
(1) \ h2'\text{chain} \\
\quad := cc-1'' \\
\end{align*}
\]
proof
have loop-suc: loop--1--i'' + 2 = (loop--1--i'' + 1) + 1 by simp
have 0 <= loop--1--i'' + 1 using ⟨0 <= loop--1--i''⟩ by simp
show thesis
  unfolding loop-suc
  unfolding rounds''-step[OF ⟨0 <= loop--1--i''⟩]
  unfolding H1[ symmetric]
  unfolding H18 ..
qed

lemma goal17'1:
  assumes H1:
  chain---default-rcd''
  (| h0' chain := cd'' |
  )
  (| h1' chain := ce'' |
  )
  (| h3' chain := cd--1'' |
  )
  (| h4' chain := ce--1'' |
  )
  = rounds'
    ( chain---default-rcd''
      (| h0' chain := (1732584193 :: int) |
      )
      (| h1' chain := (402323417 :: int) |
      )
      (| h2' chain := (2562383102 :: int) |
      )
      (| h3' chain := (271733878 :: int) |
      )
      (| h4' chain := (3285377520 :: int) |
      )
    )
  loop--1--i'' + (2 :: int) )

x''
\[
\begin{align*}
&:= cc'' \\
&\mid \ \\
&\mid h3'\text{chain} \\
&\quad := cd'' \\
&\mid \ \\
&\mid h4'\text{chain} \\
&\quad := ce'' \\
&\mid \ \\
&\mid = \text{rounds'} \\
&\quad ( \text{chain---default-rcd''} \\
&\quad \mid \mid h0'\text{chain} \\
&\quad \quad := (1732584193 :: \text{int}) \\
&\quad \mid \ \\
&\quad \mid h1'\text{chain} \\
&\quad \quad := (4023233417 :: \text{int}) \\
&\quad \mid \ \\
&\quad \mid h2'\text{chain} \\
&\quad \quad := (2562383102 :: \text{int}) \\
&\quad \mid \ \\
&\quad \mid h3'\text{chain} \\
&\quad \quad := (271733878 :: \text{int}) \\
&\quad \mid \ \\
&\quad \mid h4'\text{chain} \\
&\quad \quad := (3285377520 :: \text{int}) \\
&\quad \mid \\
&\quad ) \\
&\quad ( x--\text{index--subtype--1--last''} + (1 :: \text{int}) ) \\
&x'' \\
\end{align*}
\]

shows \text{chain---default-rcd''} \\
\quad ( \mid \mid h0'\text{chain} \\
\quad \quad := ca'' \\
\quad \mid \ \\
\quad \mid h1'\text{chain} \\
\quad \quad := cb'' \\
\quad \mid \ \\
\quad \mid h2'\text{chain} \\
\quad \quad := cc'' \\
\quad \mid \ \\
\quad \mid h3'\text{chain} \\
\quad \quad := cd'' \\
\quad \mid \ \\
\quad \mid h4'\text{chain} \\
\quad \quad := ce'' \\
\quad \mid \\
\quad ) \\
\quad = \text{rmd-hash'} \\
\quad x'' \\
\quad ( x--\text{index--subtype--1--last''} + (1 :: \text{int}) )
unfolding \texttt{rnd-def H1 rounds-def} ..

\texttt{lemmas userlemmas = goal12'1 goal13'1 goal17'1}
\texttt{end}

References


