RIPEMD-160 - Verification of a SPARK/ADA Implementation

Fabian Immler

August 28, 2014

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 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
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 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)

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
\textbf{definition} h0-0::\texttt{word32} where h0-0 = 0x67452301
\textbf{definition} h1-0::\texttt{word32} where h1-0 = 0xEFCDAB89
\textbf{definition} h2-0::\texttt{word32} where h2-0 = 0x98BADCFE
\textbf{definition} h3-0::\texttt{word32} where h3-0 = 0x10325476
\textbf{definition} h4-0::\texttt{word32} where h4-0 = 0xC3D2E1F0
\textbf{definition} h-0 ::\texttt{chain} where
h-0 = (h0-0, h1-0, h2-0, h3-0, h4-0)

\textbf{definition} step-l ::
[\texttt{block},
 \texttt{chain},
 \texttt{nat}]
\Rightarrow \texttt{chain}
\textbf{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))

\textbf{definition} step-r ::
[\texttt{block},
 \texttt{chain},
 \texttt{nat}]
\Rightarrow \texttt{chain}
\textbf{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'))

\textbf{definition} step-both ::
[\texttt{block}, \texttt{chain} * \texttt{chain}, \texttt{nat}]
\Rightarrow \texttt{chain} * \texttt{chain}
\textbf{where}
step-both \ X \ cc \ j = (case cc of (c, c') =>
(step-l \ X \ c \ j, step-r \ X \ c' \ j))

\textbf{definition} steps::[\texttt{block}, \texttt{chain} * \texttt{chain}, \texttt{nat}]
\Rightarrow \texttt{chain} * \texttt{chain}
\textbf{where}
steps \ X \ cc \ i = foldl (step-both \ X) \ cc [0..<i]

\textbf{definition} round::[\texttt{block}, \texttt{chain}]
\Rightarrow \texttt{chain}
\textbf{where}
round \ X \ h =
(let (h0, h1, h2, h3, h4) = h in
let \((A, B, C, D, E), (A', B', C', D', E')\) = steps \((h, h)\) 80 in
\[
\begin{aligned}
&((\ast h0 \ast) h1 + C + D', \\
&\ast h1 \ast) h2 + D + E', \\
&\ast h2 \ast) h3 + E + A', \\
&\ast h3 \ast) h4 + A + B', \\
&\ast h4 \ast) h0 + B + C'))
\end{aligned}
\]

definition \texttt{rmd-body}::\([\text{message, chain, nat}] \Rightarrow \text{chain}\)
where
\texttt{rmd-body X h i} = \text{round} ((X i) h)

definition \texttt{rounds}::\([\text{message} \Rightarrow \text{chain}] \Rightarrow \text{nat} \Rightarrow \text{chain}\)
where
\texttt{rounds X h i} = \text{foldl} (\texttt{rmd-body X}) h-0 [0..<i]

definition \texttt{rmd} :: \([\text{message} \Rightarrow \text{nat}] \Rightarrow \text{chain}\)
where
\texttt{rmd X len} = \texttt{rounds X h-0 len}

end

3 Global Specifications

theory \texttt{Global-Specification}
imports \texttt{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 \texttt{HOL-Word}

abbreviation \texttt{bit-and'}::\([\text{int} \Rightarrow \text{int}] \Rightarrow \text{int}\)
where
\texttt{bit-and'} m n == \text{uint} (((\text{word-of-int} m::\text{word32}) \text{ AND} \text{word-of-int} n))

abbreviation \texttt{bit-or'}::\([\text{int} \Rightarrow \text{int}] \Rightarrow \text{int}\)
where
\texttt{bit-or'} m n == \text{uint} (((\text{word-of-int} m::\text{word32}) \text{ OR} \text{word-of-int} n))

abbreviation \texttt{bit-xor'}::\([\text{int} \Rightarrow \text{int}] \Rightarrow \text{int}\)
where
\texttt{bit-xor'} m n == \text{uint} (((\text{word-of-int} m::\text{word32}) \text{ XOR} \text{word-of-int} n))

abbreviation \texttt{rotate-left'}::\([\text{int} \Rightarrow \text{int}] \Rightarrow \text{int}\)
where
\texttt{rotate-left'} i w == \text{uint} (\text{word-rotl} (\text{nat} i) (\text{word-of-int} w::\text{word32}))

This is how SPARK treats the bitwise not
lemma bit-not-spark-def [simp]:
(word-of-int $(4294967295 - x)\text{::}\text{word32}$) = NOT (word-of-int $x$)
proof 
  have word-of-int $x + (word-of-int (4294967295 - x)\text{::}\text{word32}) =$
    word-of-int $x + NOT (word-of-int x)$
    by (simp only: bin-add-not Min-def simp)
  thus 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
  $k-l' j\ =\ \text{uint} (K (\text{nat} j))$
abbreviation $k-r'$ :: int => int where
  $k-r' j\ =\ \text{uint} (K' (\text{nat} j))$
abbreviation $r-l'$ :: int => int where
  $r-l' j\ =\ \text{int} (r (\text{nat} j))$
abbreviation $r-r'$ :: int => int where
  $r-r' j\ =\ \text{int} (r' (\text{nat} j))$
abbreviation $s-l'$ :: int => int where
  $s-l' j\ =\ \text{int} (s (\text{nat} j))$
abbreviation $s-r'$ :: int => int where
  $s-r' j\ =\ \text{int} (s' (\text{nat} j))$
abbreviation $f'$ :: int => int => int => int => int where
  $f' j\ x\ y\ z\ =\ $
    \text{uint} (f (\text{nat} j) (\text{word-of-int} x\text{::}\text{word32})\ (\text{word-of-int} y)\ (\text{word-of-int} z))

end

4 Verification of $f$

theory F-Spark-Specification
imports F-Spark-Declaration Global-Specification

begin

abbreviation $bit--\text{and}' [\text{int}, \text{int}] => \text{int} where$
  $bit--\text{and}'\ =\ \text{Global-Specification.bit--\text{and}'}$
abbreviation $bit--\text{or}' [\text{int}, \text{int}] => \text{int} where$
  $bit--\text{or}'\ =\ \text{Global-Specification.bit--\text{or}'}$
abbreviation $bit--\text{xor}' [\text{int}, \text{int}] => \text{int} where$
  $bit--\text{xor}'\ =\ \text{Global-Specification.bit--\text{xor}'}$

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

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

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

lemma goal2'2:
shows \( bit-or'(bit-and' x'' y'') (bit-and' (4294967295 - x'') z'') \leq 4294967295 \)
by (simp add: bwsimps int-word-uint)

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

lemma goal3'2:
shows \( bit-xor'(bit-or' x'' (4294967295 - y'')) z'' \leq 4294967295 \)
by (simp add: bwsimps int-word-uint)

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

lemma goal4'2:
shows \( bit-or'(bit-and' x'' z'') (bit-and' y'' (4294967295 - z'')) \leq 4294967295 \)
by (simp add: bwsimps int-word-uint)

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

lemma goal5'2:
shows \( bit-xor' x'' (bit-or' y'' (4294967295 - z'')) \leq 4294967295 \)
by (simp add: bwsimps int-word-uint)

lemma goal6'1:
assumes \( H8: j'' \leq (15 :: \text{int}) \)
shows \( bit-xor' x'' (bit-xor' y'' z'') = f' j'' x'' y'' z'' \)
proof
  from \( H8 \) have \( \text{nat j''} \leq 15 \) by simp
  thus \( \text{thesis} \)
  by (simp add: f-def)
qed

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
  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 goal8'1:
  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 goal9'1:
  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 goal10'1:
  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 \textit{K-L-Spark-Specification}  
imports \textit{K-L-Spark-Declaration Global-Specification}  

begin

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

end

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

begin

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

lemma \textit{goal7'}:  
fixes \( j :: \text{int} \)  
assumes \( H1: 16 <= j \)  
assumes \( H2: j <= 31 \)  
shows \( 1518500249 = k-l' j \)  
proof -  
from \( H1 \) have \( 16 <= \text{nat} j \) by simp  
moreover from \( H2 \) have \( \text{nat} j <= 31 \) by simp  
ultimately show \( \text{thesis} \) by (simp add: \textit{K-def})

qed
lemma goal8':1:
  assumes H1: (32 :: int) <= j''
  assumes H2: j'' <= (47 :: int)
  shows (1859775393 :: int) = k-l' j''
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':1:
  assumes H1: (48 :: int) <= j''
  assumes H2: j'' <= (63 :: int)
  shows (2400959708 :: int) = k-l' 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':1:
  assumes H2: j'' <= (79 :: int)
  assumes H6: (63 :: int) < j''
  shows (2840853898 :: int) = k-l' j'' (is ?C1)
proof -
  from H6 have 64 <= nat j'' by simp
  moreover from H2 have nat j'' <= 79 by simp
  ultimately show ?thesis by (simp add: K-def)
qed

lemmas userlemmas =
gold6'1
gold7'1
gold8'1
gold9'1
gold10'1
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':
  assumes H1: \((0 :: \text{int}) \leq j''\)
  assumes H2: \(j'' \leq (15 :: \text{int})\)
  shows \((1352829926 :: \text{int}) = k-r' j''\) (is \(?C1\))
  using assms by (simp add: K'-def)

lemma goal7':
  assumes H1: \((16 :: \text{int}) \leq j''\)
  assumes H2: \(j'' < (31 :: \text{int})\)
  shows \((1548603684 :: \text{int}) = k-r' j''\) (is \(?C1\))

proof
  from H1 have \(16 \leq \text{nat} j''\) by simp
  moreover from H2 have \(\text{nat} j'' < 31\) by simp
  ultimately show \(?thesis\) by (simp add: K'-def)
qed

lemma goal8':
  assumes H1: \((32 :: \text{int}) \leq j''\)
  assumes H2: \(j'' < (47 :: \text{int})\)
  shows \((1836072691 :: \text{int}) = k-r' j''\) (is \(?C1\))

proof
  from H1 have \(32 \leq \text{nat} j''\) by simp
  moreover from H2 have \(\text{nat} j'' < 47\) by simp
  ultimately show \(?thesis\) by (simp add: K'-def)
qed

lemma goal9':
  assumes H1: \((48 :: \text{int}) \leq j''\)
  assumes H2: \(j'' < (63 :: \text{int})\)
  shows \((2053994217 :: \text{int}) = k-r' j''\) (is \(?C1\))

proof
  from H1 have \(48 \leq \text{nat} j''\) by simp
  moreover from H2 have \(\text{nat} j'' < 63\) by simp
  ultimately show \(?thesis\) by (simp add: K'-def)
qed

lemma goal10':
  assumes H2: \(j'' < (79 :: \text{int})\)
  assumes H6: \((63 :: \text{int}) < j''\)
  shows \((0 :: \text{int}) = k-r' j''\) (is \(?C1\))
proof
from H6 have $64 < \text{nat } j''$ by simp
moreover from H2 have $\text{nat } j'' < 79$ by simp
ultimately show ?thesis by (simp add: $K'$-def)
qed

lemmas userlemmas =
goal6'
goal7'
goal8'
goal9'
goal10'
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 [] - (f :: int ⇒ int) = f
| list-to-fun (a # xs) i f = (list-to-fun xs (i + 1) f) (i := (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: $1 + \text{int } i-0 = \text{int } i-0 + 1$ by simp
  ultimately show ?case by (simp add: nth-Cons' aux)
qed
lemma nth-list-to-fun-eq:
  assumes 0 < i and i < length l
  shows int (l ! i) = (list-to-fun l 0 f) (int i)
proof –
  have int (l ! (i - 0)) =
    (list-to-fun l (int 0) f) (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 f i-0)
  case Nil
  thus ?case by simp
next
  case (Cons a xs)
  have I: int i-0 + 1 = 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 <= i and i < length l + i-0
  shows int (l ! (i - i-0)) = (list-to-fun-eff l (int i-0) f) (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: int i-0 + 1 = int (i-0 + 1) by simp
  { assume i = i-0
    moreover
    have i-0 + 1 > i-0 by simp
    have int a = list-to-fun-eff xs (int i-0 + 1) (f(int i-0 := int a)) (int i-0)
  }
unfolding \textit{l-list-to-fun-id}(OF \ i-0 + 1 > i-0) by simp
ultimately have \textit{?case} by (simp add: nth-Cons')

moreover
{
  assume \( i \neq i-0 \)
moreover
hence \( H: i-0 + 1 \leq i \) using Cons by simp

have \( \text{int} (xs ! (i - \text{Suc} i-0)) = \)
  \textit{list-to-fun-eff} \( xs \) \( (\text{int} i-0 + 1) \) \( (f(\text{int} i-0 := \text{int} a)) \) \( (\text{int} i) \)
unfolding \( I \text{Cons}(1)(OF H H', \text{symmetric}) \) by simp
ultimately have \textit{?case} using Cons(2) by (simp add: nth-Cons')

}
ultimately show \textit{?case} by blast

qed

\textbf{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 foldl) and prove the correctness of this calculation.

\textbf{lemma fold-max-leq:}
\begin{verbatim}
  fixes i j :: nat
  assumes i <= j
  shows foldl max i l <= foldl max j l
  using assms
  by (induct l arbitrary: i j) simp-all
\end{verbatim}

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

lemma \textit{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 $\text{int}(l \cdot \text{nat} \cdot j) \leq U$
using assms by simp

end

8 Verification of $r_l$

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

begin

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

proof —
note nth-list-to-fun-off-eq
moreover have $0 \leq \text{nat} \cdot j''$ by simp
moreover from $j'' \leq 79$ have $\text{nat} \cdot j'' \leq \text{length} \cdot r-list$
unfolding $r$-list-def by simp
ultimately have conversion:
\[
\text{int} \ (r \text{-} ! \text{nat } j'') =
\text{list-to-fun-eff}
\]
r-list 0 block-permutation---default-arr'' \ (\text{int} \ (\text{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 $(\text{block-permutation---default-arr''})$
\[
(0 := 0, 1 := 1, 2 := 2, 3 := 3, 4 := 4, 5 := 5, 6 := 6, 7 := 7,
8 := 8, 9 := 9, 10 := 10, 11 := 11, 12 := 12, 13 := 13, 14 := 14,
15 := 15, 16 := 7, 17 := 4, 18 := 13, 19 := 1, 20 := 10, 21 := 6,
22 := 15, 23 := 3, 24 := 12, 25 := 0, 26 := 9, 27 := 5, 28 := 2,
29 := 14, 30 := 11, 31 := 8, 32 := 3, 33 := 10, 34 := 14, 35 := 4,
36 := 9, 37 := 15, 38 := 8, 39 := 1, 40 := 2, 41 := 7, 42 := 0,
43 := 6, 44 := 13, 45 := 11, 46 := 5, 47 := 12, 48 := 1, 49 := 9,
50 := 11, 51 := 10, 52 := 0, 53 := 8, 54 := 12, 55 := 4, 56 := 13,
57 := 3, 58 := 7, 59 := 15, 60 := 14, 61 := 5, 62 := 6, 63 := 2,
64 := 4, 65 := 0, 66 := 5, 67 := 9, 68 := 7, 69 := 12, 70 := 2,
71 := 10, 72 := 14, 73 := 1, 74 := 3, 75 := 8, 76 := 11, 77 := 6,
78 := 15, 79 := 13))
\]
j''
unfolding goal2'1[OF assms]
by simp

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

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 \( r\text{-}r' \) where
\( r\text{-}r' \equiv \text{Global-Specification} \cdot r\text{-}r' \)

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

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'\text{-list}
  unfolding r'\text{-list-def} by simp
ultimately have conversion:
  int (r'\text{-list} ! nat j'') =
  list-to-fun-eff
  r'\text{-list} 0 block-permutation-def\text{-arr''} (int (nat j'')) .

show ?thesis unfolding r'\text{-def conversion}
  unfolding r'\text{-list-def} using (0 <= j'' \& j'' <= 79)
  by simp
qed

lemma goal2'2:
  assumes 0 <= j''
  assumes j'' <= 79
  shows 0 <= (block-permutation-def\text{-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''

unfolding goal2'1 [OF assms]
by simp

lemma goal2'3:
  assumes 0 <= j''
  assumes j'' <= 79
  shows (block-permutation-def\text{-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,
\[
\begin{align*}
64 := 12, & \quad 65 := 15, & \quad 66 := 10, & \quad 67 := 4, & \quad 68 := 1, & \quad 69 := 5, & \quad 70 := 8, \\
71 := 7, & \quad 72 := 6, & \quad 73 := 2, & \quad 74 := 13, & \quad 75 := 14, & \quad 76 := 0, & \quad 77 := 3, \\
78 := 9, & \quad 79 := 11)
\end{align*}
\]
\[j'' \leq 15\]

proof
have \(r'\)-list ! nat \(j'' \leq \) foldl max 0 r'-list
  by (insert assms, rule list-max) (simp-all add: r'-list-def)
thus \(?thesis unfolding goal2'1[of assms r'-def]
  by (rule list-max-int) (simp-all add: assms r'-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 \Rightarrow 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 \(0 <= j''\)
  assumes \(j'' <= 79\)
  shows \(\text{rotate-definition---default-arr}''\)
  \((0 := 11, 1 := 14, 2 := 15, 3 := 12, 4 := 5, 5 := 8, 6 := 7, 7 := 9, \\
8 := 11, 9 := 13, 10 := 14, 11 := 15, 12 := 6, 13 := 7, 14 := 9, \\
15 := 8, 16 := 7, 17 := 6, 18 := 8, 19 := 13, 20 := 11, 21 := 9, \\
29 := 7, 30 := 13, 31 := 12, 32 := 11, 33 := 13, 34 := 6, 35 := 7, \\
36 := 14, 37 := 9, 38 := 13, 39 := 15, 40 := 14, 41 := 8, 42 := 13, \\
43 := 6, 44 := 5, 45 := 12, 46 := 7, 47 := 5, 48 := 11, 49 := 12, \\
50 := 14, 51 := 15, 52 := 14, 53 := 15, 54 := 9, 55 := 8, 56 := 9, \\
57 := 14, 58 := 5, 59 := 6, 60 := 8, 61 := 6, 62 := 5, 63 := 12, \\
64 := 9, 65 := 15, 66 := 5, 67 := 11, 68 := 6, 69 := 8, 70 := 13, \\
71 := 7, 72 := 6, 73 := 2, 74 := 13, 75 := 14, 76 := 0, 77 := 3, \\
78 := 9, 79 := 11)\)}
\[ j' = \text{S-L-Spark-Specification.s-l'} \]

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

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

lemma \( \text{goal2'3:} \)
assumes \( 0 <= j'' \)
assumes \( j'' <= 79 \)
shows \( \text{rotate-definition---default-arr''} \)
  \( (0 := 11, 1 := 14, 2 := 15, 3 := 12, 4 := 5, 5 := 8, 6 := 7, 7 := 9, \)
  \( 8 := 11, 9 := 13, 10 := 14, 11 := 15, 12 := 6, 13 := 7, 14 := 9, \)
  \( 15 := 8, 16 := 7, 17 := 6, 18 := 8, 19 := 13, 20 := 11, 21 := 9, \)
  \( 22 := 7, 23 := 15, 24 := 7, 25 := 12, 26 := 15, 27 := 9, 28 := 11, \)
  \( 29 := 7, 30 := 13, 31 := 12, 32 := 11, 33 := 13, 34 := 6, 35 := 7, \)
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

11 Verification of s_r 

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

begin 

abbreviation s-r' :: int => int where 
  s-r' == 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 := 7, 34 := 15, 35 := 11, 
  36 := 8, 37 := 6, 38 := 6, 39 := 14, 40 := 12, 41 := 13, 42 := 5, 

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

**proof**

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

**qed**

**lemma** \texttt{goal2'2:}

- **assumes** \( 0 \leq j'' \)
- **assumes** \( j'' \leq 79 \)
- **shows** \( 0 \leq (\text{rotate-definition---default-arr''}) \)
  - \texttt{j''}

**unfolding** \texttt{goal2'1[OF assms]}

by \texttt{simp}

**lemma** \texttt{goal2'3:}

- **assumes** \( 0 \leq j'' \)
- **assumes** \( j'' \leq 79 \)
- **shows** \( (\text{rotate-definition---default-arr''}) \)
  - \texttt{j''}
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'

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

end

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)
abbreviation \texttt{s-l': :: int => int} where
\texttt{s-l' == Global-Specification.s-l'}

abbreviation \texttt{s-r' :: int => int} where
\texttt{s-r' == Global-Specification.s-r'}

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

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

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

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

abbreviation \texttt{steps' :: [chain-pair', int, block] => chain-pair'} where
\texttt{steps' cc i b == to-chain-pair (steps}
\texttt{(\%n. word-of-int (b (int n)))}
\texttt{(from-chain-pair cc)
  (nat i))}

abbreviation \texttt{round' :: [chain', block'] => chain'} where
\texttt{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 \(0 \leq (x::\text{int})\)
  assumes \(x \leq 4294967295\)
  shows\(\text{uint}(\text{word-of-int } x::\text{word32}) = x\)
  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 \(a::\text{chain}\)
  fix \(b\ c\ d\ e\ f::\text{word32}\)
  assume \(CC = (a, b, c, d, e, f)\)
  thus \(?\text{thesis}\) by (cases \(a\)) simp
qed

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

lemma step-from-hyp:
  fixes \(a\ b\ c\ d\ e\)
  fixes \(a'\ b'\ c'\ d'\ e'\)
  fixes \(a-0\ b-0\ c-0\ d-0\ e-0\)
  fixes \(x\)
  fixes \(j\)
  assumes \step-hyp:
chain-pair---default-rcd
\( \text{left chain-pair := chain---default-rcd} \)
\( \{h0\text{\'chain := } a, h1\text{\'chain := } b, h2\text{\'chain := } c, h3\text{\'chain := } d, \)
\( h4\text{\'chain := } e\} \)
\( \text{right chain-pair := chain---default-rcd} \)
\( \{h0\text{\'chain := } a', h1\text{\'chain := } b', h2\text{\'chain := } c', h3\text{\'chain := } d', \)
\( h4\text{\'chain := } e'\} \)

steps
\( \text{chain-pair---default-rcd} \)
\( \{\text{left chain-pair := chain---default-rcd} \}
\( \{h0\text{\'chain := } a-0, h1\text{\'chain := } b-0, h2\text{\'chain := } c-0, \)
\( h3\text{\'chain := } d-0, h4\text{\'chain := } e-0\} \)
\( \text{right chain-pair := chain---default-rcd} \)
\( \{h0\text{\'chain := } a-0, h1\text{\'chain := } b-0, h2\text{\'chain := } c-0, \)
\( h3\text{\'chain := } d-0, h4\text{\'chain := } e-0\} \)

\( j \times x \)

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

28
4294967296 + 
\times (r' \cdot j)) \mod
4294967296 + 
\times (k' \cdot j) \mod
4294967296 + 
e^{' \mod
4294967296,
h_2' \cdot \text{chain} := b', h_3' \cdot \text{chain} := \text{wordops--rotate-left'} 10 c',
h_4' \cdot \text{chain} := d'\rangle =
\begin{aligned}
\text{steps'} \\
(\text{chain-pair---default-red'} \\
(\text{left'} \cdot \text{chain-pair} := \text{chain---default-rcd} \\
(\text{right'} \cdot \text{chain-pair} := \text{chain---default-rcd})
\end{aligned}
\begin{aligned}
(j + 1) \times 
\text{proof --}
\text{let } \mathbf{MM} = 4294967296
\text{have } A L: \text{uint(word-of-int e::word32)} = e \\
\quad \text{by } (\text{rule uint-word-of-int-id}[OF : 0 \le e \le \mathbf{M}])
\text{have } C L: \text{uint(word-of-int b::word32)} = b \\
\quad \text{by } (\text{rule uint-word-of-int-id}[OF : 0 \le b \le \mathbf{M}])
\text{have } D L: \text{True} ..
\text{have } E L: \text{uint(word-of-int d::word32)} = d \\
\quad \text{by } (\text{rule uint-word-of-int-id}[OF : 0 \le d \le \mathbf{M}])
\text{have } A R: \text{uint(word-of-int e'::word32)} = e' \\
\quad \text{by } (\text{rule uint-word-of-int-id}[OF : 0 \le e' \le \mathbf{M}])
\text{have } C R: \text{uint(word-of-int b'::word32)} = b' \\
\quad \text{by } (\text{rule uint-word-of-int-id}[OF : 0 \le b' \le \mathbf{M}])
\text{have } D R: \text{True} ..
\text{have } E R: \text{uint(word-of-int d'::word32)} = d' \\
\quad \text{by } (\text{rule uint-word-of-int-id}[OF : 0 \le d' \le \mathbf{M}])
\text{have } B L: (\text{uint}
\begin{aligned}
(\text{word-rotl (s (nat j))} \\
((\text{word-of-int::int} \Rightarrow \text{word32}) \\
(((a + f \cdot j \cdot b \cdot c \cdot d) \mod 4294967296 + \\
\times (r-l' \cdot j) \mod
4294967296 + \\
k-l' \cdot j) \mod
4294967296 + \\
e')) \mod
4294967296 =
\text{uint}
\begin{aligned}
(\text{word-rotl (s (nat j))} \\
(\text{word-of-int a} + \\
f (nat j) (\text{word-of-int b}) (\text{word-of-int c}) (\text{word-of-int d}) + \\
\text{word-of-int (x (r-l' \cdot j))} + 
\end{aligned}
\end{aligned}
\[ K \ (\text{nat} \ j) \ + \ \text{word-of-int} \ e \]
\[(\text{is} \ (\text{uint} \ (\text{word-rotl} \ - \ (((\ (- \ + \ F) \ \text{mod} \ - \ + \ X) \ \text{mod} \ - \ + \ -) \ \text{mod}) -)) \ + -) \ \text{mod} \ - \ -)\]

**proof**

**have** a \(\mod\) ?MM = a using \(\theta \leq a\) \(\langle a \leq \ ?M\rangle\)

**by** (simp add: int-mod-eq)

**have** ?X \(\mod\) ?MM = ?X using \(\theta \leq ?X\) \(\langle ?X \leq \ ?M\rangle\)

**by** (simp add: int-mod-eq)

**have** e \(\mod\) ?MM = e using \(\theta \leq e\) \(\langle e \leq \ ?M\rangle\)

**by** (simp add: int-mod-eq)

**have** (?MM::int) = 2 \^\ len-of TYPE(32) by simp

**show** ?thesis

**unfolding**

word-add-def

uint-word-of-int-id[OF \(\theta \leq a\) \(\langle a \leq \ ?M\rangle\)]

uint-word-of-int-id[OF \(\theta \leq ?X\) \(\langle ?X \leq \ ?M\rangle\)]

int-word-uint

**unfolding** (?MM = 2 \^\ 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**

**have** BR: (uint (word-rotl \(s'\) (nat \ j))

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

\(((\langle a' + f'(79 - j) \ b' \ c' \ d'\rangle \ \text{mod} \ 4294967296 +

x \ (r-r' j) \ \text{mod} \ 4294967296 +

k-r' j) \ \text{mod} \ 4294967296)) +

e' \ \text{mod}

4294967296 =

uint

(word-rotl \(s'\) (nat \ j))

(word-of-int \ a' +

f (79 - nat \ j) \ (\text{word-of-int} \ b') \ (\text{word-of-int} \ c') \ (\text{word-of-int} \ d') +

\text{word-of-int} \ (x \ (r-r' j)) +

K' (nat \ j)) +

\text{word-of-int} \ e'\)

\((\text{is} \ (\text{uint} \ (\text{word-rotl} \ - \ (((\ (- \ + \ F) \ \text{mod} \ - \ + \ X) \ \text{mod} \ - \ + \ -) \ \text{mod}) -)) \ + -) \ \text{mod} \ - \ -)\)

**proof**

**have** a' \(\mod\) ?MM = a' using \(\theta \leq a'\) \(\langle a' \leq \ ?M\rangle\)

**by** (simp add: int-mod-eq)
have ?X mod ?MM = ?X using ⟨0 <= ?X⟩ ⟨?X <= ?M⟩
  by (simp add: int-mod-eq)
have e′ mod ?MM = e′ using ⟨0 <= e′⟩ ⟨e′ <= ?M⟩
  by (simp add: int-mod-eq)
have (?MM::int) = 2 ^ len-of TYPE(32) by simp
have nat-transfer: 79 - nat j = nat (79 - j)
  using nat-diff-distrib ⟨0 <= j⟩ ⟨j <= 79⟩
  by simp
show ?thesis
  unfolding word-add-def
  int-word-uint
  nat-transfer
  unfolding ⟨?MM = 2 ^ len-of TYPE(32)⟩
  unfolding word-uint.Abs-norm
  by (simp add: ⟨a′ mod ?MM = a′⟩ ⟨e′ mod ?MM = e′⟩ ⟨?X mod ?MM = ?X⟩)
qed

show ?thesis
  unfolding steps'-step[OF ⟨0 <= j⟩] step-hyp[symmetric]
  step-both-def step-r-def step-l-def
  by (simp add: AL BL CL DL EL AR BR CR DR ER)
qed

abbreviation f-0-result == (((ca'' + f-spark' 0 cb'' cc'' cd'') mod 4294967296 +
  x'' (r-l-spark' 0)) mod 4294967296 + k-l-spark' 0) mod 4294967296
abbreviation f-79-result == (((ca'' + f-spark' 79 cb'' cc'' cd'') mod 4294967296 +
  x'' (r-r-spark' 0)) mod 4294967296 + k-r-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-0-borders: 0 <= r-l-spark' 0 r-l-spark' 0 <= 15
  assumes r-r-0-borders: 0 <= r-r-spark' 0 r-r-spark' 0 <= 15
  assumes returns:
  wordops--rotate'(s-l-spark' 0) f-0-result =
  wordops--rotate-left'(s-l-spark' 0) f-0-result
  wordops--rotate'(s-r-spark' 0) f-79-result =
  wordops--rotate-left'(s-r-spark' 0) f-79-result
\begin{align*}
\text{wordops--rotate}' 10 cc'' &= \text{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\text{-spark}' 0 &= k\text{-l}' 0 \\
k\text{-spark}' 0 &= k\text{-r}' 0 \\
r\text{-spark}' 0 &= r\text{-l}' 0 \\
r\text{-spark}' 0 &= r\text{-r}' 0 \\
s\text{-spark}' 0 &= s\text{-l}' 0 \\
s\text{-spark}' 0 &= s\text{-r}' 0 \\
\text{assumes} \quad x\text{-borders: } \forall i. \; 0 \leq i \land i \leq 15 \implies 0 \leq x'' i \land x'' i \leq ?M \\
\text{shows} \quad \text{chain-pair---default-rcd}'' \\
\langle\text{left}'\text{chain-pair} := \text{chain---default-rcd}''\rangle \\
\langle h0'\text{chain} := cc''\rangle, \\
h1'\text{chain} := \\
\quad \langle\text{wordops--rotate}' (s\text{-spark}' 0) \\
\quad (((ca'' + f\text{-spark}' 0 cb'' cc'' cd'') \mod 4294967296 + x'' (r\text{-l}\text{-spark}' 0)) \mod 4294967296 + k\text{-l}\text{-spark}' 0) \mod 4294967296 + cc'') \mod 4294967296, \\
h2'\text{chain} := cb'', h3'\text{chain} := \text{wordops--rotate}' 10 cc'', \\
h4'\text{chain} := cd''\rangle, \\
\text{right}'\text{chain-pair} := \text{chain---default-rcd}'' \\
\langle h0'\text{chain} := cc''\rangle, \\
h1'\text{chain} := \\
\quad \langle\text{wordops--rotate}' (s\text{-r}\text{-spark}' 0) \\
\quad (((ca'' + f\text{-spark}' 79 cb'' cc'' cd'') \mod 4294967296 + x'' (r\text{-r}\text{-spark}' 0)) \mod 4294967296 + k\text{-r}\text{-spark}' 0) \mod 4294967296 + cc'') \mod 4294967296, \\
h2'\text{chain} := cb'', h3'\text{chain} := \text{wordops--rotate}' 10 cc'', \\
h4'\text{chain} := cd''\rangle = \\
\text{steps}' \\
\langle\text{chain-pair---default-rcd}''\rangle \\
\langle\text{left}'\text{chain-pair} := \text{chain---default-rcd}''\rangle \\
\langle h0'\text{chain} := ca'', h1'\text{chain} := cb'', h2'\text{chain} := cc'', \\
h3'\text{chain} := cd'', h4'\text{chain} := ce''\rangle, \\
\text{right}'\text{chain-pair} := \text{chain---default-rcd}'' \\
\langle h0'\text{chain} := ca'', h1'\text{chain} := cb'', h2'\text{chain} := cc'', \\
h3'\text{chain} := cd'', h4'\text{chain} := ce''\rangle\rangle \\
1 x'' \\
\text{proof} \\
\text{have step-hyp:} \\
\text{chain-pair---default-rcd}''
\end{align*}
\( \langle \text{left\-chain\-pair} := \text{chain---default-rcd}' \rangle \)
\( \langle \text{h0\-'chain} := \text{ca}'', \text{h1\-'chain} := \text{cb}''', \text{h2\-'chain} := \text{cc}'', \text{h3\-'chain} := \text{cd}''', \text{h4\-'chain} := \text{ce}''' \rangle, \)
\( \text{right\-chain\-pair} := \text{chain---default-rcd}'' \)
\( \langle \text{h0\-'chain} := \text{ca}'', \text{h1\-'chain} := \text{cb}''', \text{h2\-'chain} := \text{cc}'', \text{h3\-'chain} := \text{cd}''', \text{h4\-'chain} := \text{ce}''' \rangle \)
steps'
\( \langle \text{chain\-pair---default-rcd}' \rangle \)
\( \langle \text{h0\-'chain} := \text{ca}'', \text{h1\-'chain} := \text{cb}''', \text{h2\-'chain} := \text{cc}'', \text{h3\-'chain} := \text{cd}''', \text{h4\-'chain} := \text{ce}''' \rangle \)
\( 0 \ x'' \)
unfolding steps-def
by \( \text{simp add:} \)
\( \text{uint-word-of-int-id[OF ca-borders]} \)
\( \text{uint-word-of-int-id[OF cb-borders]} \)
\( \text{uint-word-of-int-id[OF cc-borders]} \)
\( \text{uint-word-of-int-id[OF cd-borders]} \)
\( \text{uint-word-of-int-id[OF ce-borders]} \)

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

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

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

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

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

abbreviation rotate-arg-l ==
(((((cla'' + f-spark') (loop--1--j'' + 1)) clb'' cle'' cld'')) mod 4294967296 + 
x'' (r-l-spark' (loop--1--j'' + 1))) mod 4294967296 +
k-l-spark' (loop--1--j'' + 1)) mod 4294967296

abbreviation rotate-arg-r == (((((cr' + f-spark') (79 - (loop--1--j'' + 1)) crb'')
cre'' crd'')) mod 
4294967296 + x'' (r-r-spark' (loop--1--j'' + 1))) mod 4294967296 +
k-r-spark' (loop--1--j'' + 1)) mod 4294967296

lemma goal62'1:
assumes cla-borders: 0 <= cla'' cla'' <= 4294967295 (is - <= ?M)
assumes clb-borders: 0 <= clb'' clb'' <= ?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 steps:
chain-pair---default-rcd''
| left\'chain-pair := chain---default-rcd''
| \{ h0\'chain := cla'', h1\'chain := clb'', h2\'chain := cle'',
| h3\'chain := cld'', h4\'chain := cle''\}
| right\'chain-pair := chain---default-rcd''
| \{ h0\'chain := cra'', h1\'chain := crb'', h2\'chain := crc'',
| h3\'chain := crc'', h4\'chain := cre''\}

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

assumes returns:
wordops--rotate' (s-l-spark' (loop--1--j'' + 1)) rotate-arg-rl =
wordops--rotate-left' (s-l-spark' (loop--1--j'' + 1)) rotate-arg-rl
wordops--rotate' (s-r-spark' (loop--1--j'' + 1)) rotate-arg-rr =
wordops--rotate-left' (s-r-spark' (loop--1--j'' + 1)) rotate-arg-rr
f-spark' (loop--1--j'' + 1) clb'' cle'' cld'' =
f' (loop--1--j'' + 1) clb'' cle'' cld'"
f-spark' (78 - loop--1-j'') crb'' crc'' crd'' =
f' (78 - loop--1-j'') crb'' crc'' crd''
wordops--rotate' 10 cle'' = wordops--rotate-left' 10 cle''
wordops--rotate' 10 crc'' = wordops--rotate-left' 10 crc''
k-l-spark'' (loop--1-j'' + 1) = k-l' (loop--1-j'' + 1)
k-r-spark'' (loop--1-j'' + 1) = k-r' (loop--1-j'' + 1)
r-l-spark'' (loop--1-j'' + 1) = r-l' (loop--1-j'' + 1)
r-r-spark'' (loop--1-j'' + 1) = r-r' (loop--1-j'' + 1)
s-l-spark'' (loop--1-j'' + 1) = s-l' (loop--1-j'' + 1)
s-r-spark'' (loop--1-j'' + 1) = s-r' (loop--1-j'' + 1)
assumes x-borders: ∀ 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))
(((((cle'' + f-spark'' (loop--1-j'' + 1) clb'' cle'' 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 cle'',
\ h4'chain := cld''],)
right'chain-pair := chain---default-rcd''
(\h0'chain := cre''),
\ h1'chain :=
(\wordops--rotate' (s-r-spark'' (loop--1-j'' + 1))
(((((cre'' +
\ f-spark'' (79 - (loop--1-j'' + 1)) crb'' crc''
\ cld'')) 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 cre'',
\ h4'chain := crd'']) =
steps'
(chain-pair---default-rcd'')
\[ \text{left\ primal-chain-pair := chain---default-rcd} \]
\[ \text{h0\ primal-chain := ca---init'', h1\ primal-chain := cb---init'',} \]
\[ \text{h2\ primal-chain := cc---init'', h3\ primal-chain := cd---init'',} \]
\[ \text{h4\ primal-chain := ce---init''} \]
\[ \text{right\ primal-chain-pair := chain---default-rcd} \]
\[ \text{h0\ primal-chain := ca---init'', h1\ primal-chain := cb---init'',} \]
\[ \text{h2\ primal-chain := cc---init'', h3\ primal-chain := cd---init'',} \]
\[ \text{h4\ primal-chain := ce---init''} \]
\[ \text{(loop--1--j'' + 2) x''} \]

**proof**

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

**from** \( r-l\)-borders \( x\)-borders

**have** \( 0 \leq x'' (r-l\-spark' (\text{loop--1--j''} + 1)) \) by blast

**hence** \( x\)-lower: \( 0 \leq x'' (r-l' (\text{loop--1--j''} + 1)) \) unfolding returns.

**from** \( r-l\)-borders \( x\)-borders

**have** \( x'' (r-l\-spark' (\text{loop--1--j''} + 1)) \leq ?M \) by blast

**hence** \( x\)-upper: \( x'' (r-l' (\text{loop--1--j''} + 1)) \leq ?M \) unfolding returns.

**from** \( r-r\)-borders \( x\)-borders

**have** \( 0 \leq x'' (r-r\-spark' (\text{loop--1--j''} + 1)) \) by blast

**hence** \( x\)-lower': \( 0 \leq x'' (r-r' (\text{loop--1--j''} + 1)) \) unfolding returns.

**from** \( r-r\)-borders \( x\)-borders

**have** \( x'' (r-r\-spark' (\text{loop--1--j''} + 1)) \leq ?M \) by blast

**hence** \( x\)-upper': \( x'' (r-r' (\text{loop--1--j''} + 1)) \leq ?M \) unfolding returns.

**from** \( j\)-loop-1-borders **have** \( 0 \leq \text{loop--1--j''} + 1 \) by simp

**from** \( j\)-loop-1-borders **have** \( \text{loop--1--j''} + 1 \leq 79 \) by simp

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

**have** \( f' (79 - (\text{loop--1--j''} + 1)) \) crb'' crc'' crd'' = f-spark' (79 - (\text{loop--1--j''} + 1)) 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
  cdr-borders
  cre-borders

36
from this OF
  \( x \leq x \) x-upper \( x \) x-upper
  \( 0 \leq \text{loop--} 1-j'' + 1 \) \( \text{loop--} 1-j'' + 1 \leq 79 \)
show \( \text{thesis unfolding} \) \( \text{loop--} 1-j'' + 1 + 1 = \text{loop--} 1-j'' + 2 \)
  unfolding returns(1) returns(2) unfolding returns
  by simp
qed

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

lemma goal76'1:
  assumes cla-borders: \( 0 \leq cla'' cla'' \leq 4294967295 \) (is - \( \leq \) M)
  assumes clb-borders: \( 0 \leq clb'' clb'' \leq M \)
  assumes clc-borders: \( 0 \leq clc'' clc'' \leq M \)
  assumes cld-borders: \( 0 \leq cld'' cld'' \leq M \)
  assumes cle-borders: \( 0 \leq cle''cle'' \leq M \)
  assumes cra-borders: \( 0 \leq cra'' cra'' \leq M \)
  assumes crb-borders: \( 0 \leq crb'' crb'' \leq M \)
  assumes crc-borders: \( 0 \leq crc'' crc'' \leq M \)
  assumes crd-borders: \( 0 \leq crd'' crd'' \leq M \)
  assumes cld-borders: \( 0 \leq cld'' cld'' \leq M \)
  assumes cla-init-borders: \( 0 \leq ca---\text{init}'' ca---\text{init}'' \leq M \)
  assumes cb-init-borders: \( 0 \leq cb---\text{init}'' cb---\text{init}'' \leq M \)
  assumes cc-init-borders: \( 0 \leq cc---\text{init}'' cc---\text{init}'' \leq M \)
  assumes cd-init-borders: \( 0 \leq cd---\text{init}'' cd---\text{init}'' \leq M \)
  assumes ce-init-borders: \( 0 \leq ce---\text{init}'' ce---\text{init}'' \leq M \)
  assumes step-hyp:
    chain-pair---default-rcd''
    \( \{ left' \text{chain-pair} := chain---\text{default-rcd}'' \)
    \( \{ h0' \text{chain} := cla'', h1' \text{chain} := clb'', h2'chain := clc'', h3'chain := cld'',
      h4'chain := cle'' \),
    right'chain-pair := chain---default-rcd''
    \( \{ h0' \text{chain} := cra'', h1'chain := crb'', h2'chain := crc'', h3'chain := crd'',
      h4'chain := cre'' \} \)
    steps'
    (chain-pair---default-rcd''
    \( \{ left' \text{chain-pair} := chain---\text{default-rcd}'' \)
    \( \{ h0' \text{chain} := ca---\text{init}'' h1'chain := cb---\text{init}'' h2'chain := cc---\text{init}''
      h3'chain := cd---\text{init}'' h4'chain := ce---\text{init}'' \),
    right'chain-pair := chain---default-rcd''
    \( \{ h0' \text{chain} := ca---\text{init}'' h1'chain := cb---\text{init}'' h2'chain := cc---\text{init}''
      h3'chain := cd---\text{init}''
      h4'chain := ce---\text{init}'' \} \) \}
  80 x''
shows chain---default-rcd''
\[ \begin{align*}
&h_0' \text{chain} := ((cb-\text{init}'' + clc'') \mod 4294967296 + crd'') \mod 4294967296, \\
&h_1' \text{chain} := ((cc-\text{init}'' + cld'') \mod 4294967296 + cre'') \mod 4294967296, \\
&h_2' \text{chain} := ((cd-\text{init}'' + cle'') \mod 4294967296 + cra'') \mod 4294967296, \\
&h_3' \text{chain} := ((ce-\text{init}'' + cla'') \mod 4294967296 + crb'') \mod 4294967296, \\
&h_4' \text{chain} := ((ca-\text{init}'' + clb'') \mod 4294967296 + crc'') \mod 4294967296 \\
\end{align*} \]

\[ \begin{align*}
&= \text{round'} \\
&(\text{chain-\text{default-rcd}''} \\
& ((h_0' \text{chain} := ca-\text{init}'', h_1' \text{chain} := cb-\text{init}'', h_2' \text{chain} := cc-\text{init}'', \\
& h_3' \text{chain} := cd-\text{init}'', \\
& h_4' \text{chain} := ce-\text{init}'')) \\
&x'' \\
\end{align*} \]

**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'  
(chain-pair-\text{default-rcd}'' 
 ((left'chain-pair := INIT-CHAIN, right'chain-pair := INIT-CHAIN)) 
80 
\))  

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

**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
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''
)

shows chain---default-rcd''
( | h0'chain := ca--1''
 |)
( | h1'chain := cb--1''
 |)
( | h2'chain
\[ \text{rounds'} = \text{rounds''} \]
\[ \text{chain''} \]
\[ \text{h0'} \text{chain} := (1732584193 :: \text{int}) \]
\[ \text{h1'} \text{chain} := (4023233417 :: \text{int}) \]
\[ \text{h2'} \text{chain} := (2562383102 :: \text{int}) \]
\[ \text{h3'} \text{chain} := (271733878 :: \text{int}) \]
\[ \text{h4'} \text{chain} := (3285377520 :: \text{int}) \]
\[ \begin{aligned} \text{x -- index -- subtype -- 1 -- first'' + (1 :: \text{int}) } \\
\end{aligned} \]
\[ x'' \]
\[ \text{(is ?C1)} \]
\[ \text{using H1 H6} \]
\[ \text{by (simp add: rounds-def rmd-body-def round-def} \]
\[ h-0-def h0-0-def h1-0-def h2-0-def h3-0-def h4-0-def) \]

\textbf{lemma rounds-step:}
\begin{itemize}
\item \textbf{assumes} \( 0 \leq i \)
\item \textbf{shows} \( \text{rounds} X b \ (\text{Suc} \ i) = \text{round} \ (X \ i) \ (\text{rounds} X b i) \)
\end{itemize}
\textbf{by (simp add: rounds-def rmd-body-def)}

\textbf{lemma from-to-id: from-chain (to-chain C) = C}
\textbf{proof (cases C)}
\begin{itemize}
\item \textbf{fix} \( a \ b \ c \ d \ e \ f :: \text{word32} \)
\item \textbf{assume} \( C = (a, b, c, d, e) \)
\item \textbf{thus} \( ?\text{thesis} \) \textbf{by (cases a) simp}
\end{itemize}
\textbf{qed}

\textbf{lemma steps-to-steps':}
\[ \text{round} X \ (\text{foldl} a b c) = \text{round} X \ (\text{from-chain (to-chain (foldl a b c)))} \]

41
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--I-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--I-i'' + (1 :: int) )
\[ x'' \]

**assumes** \( H18: \)

\[
\begin{align*}
\text{chain---default-rcd}'' \\
( | h0'\text{chain} \\
\quad := ca--1'' \\
| ) \\
( | h1'\text{chain} \\
\quad := cb--1'' \\
| ) \\
( | h2'\text{chain} \\
\quad := cc--1'' \\
| ) \\
( | h3'\text{chain} \\
\quad := cd--1'' \\
| ) \\
( | h4'\text{chain} \\
\quad := ce--1'' \\
| ) \\
) = \text{round'} \\
( \text{chain---default-rcd}'' \\
( | h0'\text{chain} \\
\quad := ca'' \\
| ) \\
( | h1'\text{chain} \\
\quad := cb'' \\
| ) \\
( | h2'\text{chain} \\
\quad := cc'' \\
| ) \\
( | h3'\text{chain} \\
\quad := cd'' \\
| ) \\
( | h4'\text{chain} \\
\quad := ce'' \\
| ) \\
) \\
( x'' ( \text{loop--1--i''} + (1 \triangleright int) ) \\
) \\
\]
\begin{align*}
&= \text{rounds}' \\
&\quad (\text{chain---default-rcd}'') \\
&\quad (\{h0'\text{chain} := (1732584193 :: \text{int})\}) \\
&\quad (\{h1'\text{chain} := (402323417 :: \text{int})\}) \\
&\quad (\{h2'\text{chain} := (2562383102 :: \text{int})\}) \\
&\quad (\{h3'\text{chain} := (271733878 :: \text{int})\}) \\
&\quad (\{h4'\text{chain} := (3285377520 :: \text{int})\}) \\
&\quad (\text{loop--1--i}'' + (2 :: \text{int})) \\
&\text{proof} - \\
&\text{have loop-suc: loop--1--i}'' + 2 = (\text{loop--1--i}'' + 1) + 1 \text{ by simp} \\
&\text{have } 0 <= \text{loop--1--i}'' + 1 \text{ using } (0 <= \text{loop--1--i}'') \text{ by simp} \\
&\text{show } ?\text{thesis} \\
&\quad \text{unfolding loop-suc} \\
&\quad \text{unfolding rounds}''\text{-step}[\text{OF } (0 <= \text{loop--1--i}'')] \\
&\quad \text{unfolding } H1[\text{symmetric}] \\
&\quad \text{unfolding } H18 .. \\
&\text{qed} \\
\end{align*}
shows chain---default-rcd''

| h0'chain := (1732584193 :: int) |
| h1'chain := (4023233417 :: int) |
| h2'chain := (2562383102 :: int) |
| h3'chain := (271733878 :: int) |
| h4'chain := (3285377520 :: int) |

= rmd-hash' 

( x--index--subtype--1--last'' + (1 :: int) )
unfolding rmd-def H1 rounds-def ..

lemmas userlemmas = goal12'1 goal13'1 goal17'1
end

References


