RIPEMD-160 - Verification of a SPARK/ADA Implementation

Fabian Immler

May 28, 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 ada contains the sourcecode which has been verified against its specification in Isabelle/HOL (close to its pseudocode definition from [2]) in the following. The SPARK-code contains annotations with so called proof functions. The following proof functions (declared in ada/rmd.ads) are specified in Isabelle/HOL:

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

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

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

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

2 Specification of RIPEMD-160

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

type-synonym word32 = 32 word

(type-synonym byte = 8 word)

(type-synonym perm = nat => nat)

(type-synonym chain = word32 * word32 * word32 * word32 * word32)

(type-synonym block = nat => word32)

(type-synonym message = nat => block)

definition f::[nat, word32, word32, word32] => word32

where

f j x y z =

(if ( 0 <= j & j <= 15) then x XOR y XOR z
else if (16 <= j & j <= 31) then (x AND y) OR (NOT x AND z)
else if (32 <= j & j <= 47) then (x OR NOT y) XOR z
else if (48 <= j & j <= 63) then (x AND z) OR (y AND NOT z)
else if (64 <= j & j <= 79) then x XOR (y OR NOT z)
else 0)

definition K::nat => word32

where

K j =

(if ( 0 <= j & j <= 15) then 0x00000000
else if (16 <= j & j <= 31) then 0x5A827999
else if (32 <= j & j <= 47) then 0x6ED9EBA1
else if (48 <= j & j <= 63) then 0x8F1BBCDC
else if (64 <= j & j <= 79) then 0xA953FD4E
else 0)

definition K'::nat => word32

where

K' j =

(if ( 0 <= j & j <= 15) then 0x50A28BE6
else if (16 <= j & j <= 31) then 0x5C4DD124
else if (32 <= j & j <= 47) then 0x6D703EF3
else if (48 <= j & j <= 63) then 0x7A6D76E9
else if (64 <= j & j <= 79) then 0x00000000

\footnote{There are some slight superficial differences between the original translated files and the ones included here, in order to conform to current Isabelle practice}
\begin{verbatim}
definition r-list :: nat list
  where r-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 ]

definition r'-list :: nat list
  where r'-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 ]

definition r :: perm
  where r j = r-list ! j

definition r' :: perm
  where r' j = r'-list ! j

definition s-list :: nat list
  where s-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 ]

definition s'-list :: nat list
  where s'-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 ]

definition s :: perm
  where s j = s-list ! j

definition s' :: perm
  where s' j = s'-list ! j
\end{verbatim}
definition \( h_{0-0} :: \text{word32} \) where \( h_{0-0} = 0x67452301 \)

definition \( h_{1-0} :: \text{word32} \) where \( h_{1-0} = 0xEFCDAB89 \)

definition \( h_{2-0} :: \text{word32} \) where \( h_{2-0} = 0x98BADCFE \)

definition \( h_{3-0} :: \text{word32} \) where \( h_{3-0} = 0x10325476 \)

definition \( h_{4-0} :: \text{word32} \) where \( h_{4-0} = 0xC3D2E1F0 \)

definition \( h_{-0} :: \text{chain} \) where

\[
h_{-0} = (h_{0-0}, h_{1-0}, h_{2-0}, h_{3-0}, h_{4-0})
\]

definition \( \text{step-l} :: \) block, chain, nat \( \Rightarrow \) chain where

\[
\text{step-l} X c j = (\text{let} (A, B, C, D, E) = c \text{ in}\
\quad ((A \times E),\
\quad (B \times \text{word-rotl} (s j) (A + f j B C D + X (r j) + K j) + E,\
\quad (C \times B),\
\quad (D \times \text{word-rotl} 10 C,\
\quad (E \times D)))
\]

definition \( \text{step-r} :: \) block, chain, nat \( \Rightarrow \) chain where

\[
\text{step-r} X c' j = (\text{let} (A', B', C', D', E') = c' \text{ in}\
\quad ((A' \times E'),\
\quad (B' \times \text{word-rotl} (s' j) (A' + f (79 - j) B' C' D' + X (r' j) + K' j) + E',\
\quad (C' \times B'),\
\quad (D' \times \text{word-rotl} 10 C',\
\quad (E' \times D'))
\]

definition \( \text{step-both} :: \) block, chain * chain, nat \( \Rightarrow \) chain * chain where

\[
\text{step-both} X cc j = (\text{case} cc \text{ of} (c, c') \Rightarrow\
\quad (\text{step-l} X c j, \text{step-r} X c' j))
\]

definition \( \text{steps} :: \) block, chain * chain, nat \( \Rightarrow \) chain * chain where

\[
\text{steps} X cc i = \text{foldl} (\text{step-both} X) cc [0..<i]
\]

definition \( \text{round} :: \) block, chain \( \Rightarrow \) chain where

\[
\text{round} X h = (\text{let} (h0, h1, h2, h3, h4) = h \text{ in}\
\quad (h0, h1, h2, h3, h4))
\]
let \( ((A, B, C, D, E), (A', B', C', D', E')) = \text{steps } (h, h) \) 80 in
\[
\begin{align*}
&((\ast h0 \ast) h1 + C + D', \\
&(\ast h1 \ast) h2 + D + E', \\
&(\ast h2 \ast) h3 + E + A', \\
&(\ast h3 \ast) h4 + A + B', \\
&(\ast h4 \ast) h0 + B + C'))
\end{align*}
\]

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

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

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

end

3 Global Specifications

theory \textit{Global-Specification} imports \textit{RMD} begin

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

3.1 Specification of Bit-Operations

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

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

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

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

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

This is how SPARK treats the bitwise not
lemma bit-not-spark-def[simp]:
(word-of-int (4294967295 - x)::word32) = NOT (word-of-int x)

proof -
  have word-of-int x + (word-of-int (4294967295 - x)::word32) =
    word-of-int x + NOT (word-of-int x)
  by (simp only: bwsimps 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 == 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'

end
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 $0 <= \text{bit-or' (bit-and' x'' y'') (bit-and' (4294967295 - x'') z'')}$
  by (rule Word.uint-0)

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

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

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

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

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

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

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

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

end
lemma goal7'1:
assumes H7: \((16 :: \text{int}) \leq j''\)
assumes H8: \(j'' \leq (31 :: \text{int})\)
shows \(\text{bit--or}'(\text{bit--and}' x'' y'') (\text{bit--and}'(4294967295 - x'') z'') = f' j'' x'' y'' z''\)
proof –
from H7 have \(16 \leq \text{nat} j''\) by simp
moreover from H8 have \(\text{nat} j'' \leq 31\) by simp
ultimately show \(?thesis\) by (simp add: f-def)
qed

lemma goal8'1:
assumes H7: \(32 \leq j''\)
assumes H8: \(j'' \leq 47\)
shows \(\text{bit--xor}'(\text{bit--or}' x'' (4294967295 - y'')) z'' = f' j'' x'' y'' z''\)
proof –
from H7 have \(32 \leq \text{nat} j''\) by simp
moreover from H8 have \(\text{nat} j'' \leq 47\) by simp
ultimately show \(?thesis\) by (simp add: f-def)
qed

lemma goal9'1:
assumes H7: \(48 \leq j''\)
assumes H8: \(j'' \leq 63\)
shows \(\text{bit--or}'(\text{bit--and}' x'' z'') (\text{bit--and}' y'' (4294967295 - z'')) = f' j'' x'' y'' z''\)
proof –
from H7 have \(48 \leq \text{nat} j''\) by simp
moreover from H8 have \(\text{nat} j'' \leq 63\) by simp
ultimately show \(?thesis\) by (simp add: f-def)
qed

lemma goal10'1:
assumes H2: \(j'' \leq 79\)
assumes H12: \(63 < j''\)
shows \(\text{bit--xor}' x'' (\text{bit--or}' y'' (4294967295 - z'')) = f' j'' x'' y'' z''\)
proof –
from H2 have \(\text{nat} j'' \leq 79\) by simp
moreover from H12 have \(64 \leq \text{nat} j''\) 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 goal6'1:
fixes $j$ :: $int$
assumes $H1$: $0 \leq j$
assumes $H2$: $j \leq 15$
shows $0 = k\text{-}l' \cdot j$
using assms by (simp add: $K\text{-}def$)

lemma goal7'1:
fixes $j$ :: $int$
assumes $H1$: $16 \leq j$
assumes $H2$: $j \leq 31$
shows $1518500249 = k\text{-}l' \cdot j$

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: $K\text{-}def$)
qed

end
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 \(?\text{C1}\))
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: \((63 :: \text{int}) < j''\)
  shows \((2840853838 :: \text{int}) = k-l' j''\) (is \(?\text{C1}\))
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: \((0 :: \text{int}) \leq j''\)
  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: \(j'' \leq (31 :: \text{int})\)
  shows \((1548603684 :: \text{int}) = k-r' j''\) (is \(?C1\))
  proof
    from H1 have \(16 \leq \text{nat } j''\) by simp
    moreover from H2 have \(\text{nat } j'' \leq 31\) by simp
    ultimately show \(?thesis\) by (simp add: K'-'def)
  qed

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

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

lemma goal10':
  assumes H2: \(j'' \leq (79 :: \text{int})\)
  assumes H6: \((63 :: \text{int}) < j''\)
  shows \((0 :: \text{int}) = k-r' j''\) (is \(?C1\))
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 =
good6'1
good7'1
good8'1
good9'1
good10'1
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 [] : (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 \( \text{int } (l ! (i - i-0)) = (\text{list-to-fun } l (\text{int } i-0) f) (\text{int } i) \)
using assms
proof (induct l arbitrary: i i-0)
case Nil
thus thesis by simp
next
case (Cons a xs)
moreover have aux: 1 + int i-0 = int i-0 + 1 by simp
ultimately show thesis 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-0 f)
  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 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[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 I list-to-fun-id[OF i-0 + 1 > i-0] by simp
ultimately have ?case by (simp add: nth-Cons)
\}
moreover
\{
assume i ≠ i-0
moreover
hence H: i-0 + 1 ≤ i using Cons by simp
have H': i < length xs + (i-0 + 1) using Cons (3) by simp
have 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-Const)
\}
ultimately show ?case by blast
qed

lemma nth-list-to-fun-eff-eq:
assumes 0 ≤ i and i < length l
shows int (l ! i) = (list-to-fun-eff l 0 f) (int i)
proof -
  have 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 j :: nat
assumes i ≤ j
shows foldl max i l ≤ foldl max j l
using assms
by (induct l arbitrary: i j) simp-all

lemma fold-max-lower:
fixes i :: nat
shows i ≤ 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 ≤ x)
case True
  moreover have \( x \leq \text{foldl} \ max \ x \ \text{xs using} \ \text{Cons} \).
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 \( 0 \leq l ! i \)
  assumes \( 0 \leq i \)
  assumes \( i < \text{length} \ l \)
  shows \( l ! i \leq \text{foldl} \ max \ 0 \ l \)
  using \text{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 (Suc \( j \))
    note \text{Cons}(1)
    moreover have \( 0 \leq xs ! (i - 1) \) using \text{Suc Cons by simp}
    moreover have \( 0 \leq i - 1 \) using \text{Cons by simp}
    moreover have \( i - 1 < \text{length} \ xs \) using \text{Suc Cons by simp}
    ultimately
    have \( xs ! (i - 1) \leq \text{foldl} \ max \ 0 \ xs \).
    moreover have \( (x\#xs) ! i = xs ! (i - 1) \)
      using \text{Suc Cons by simp}
    moreover have \( \text{foldl} \ max \ 0 \ xs \leq \text{foldl} \ max \ (\text{max} \ 0 \ x) \ xs \)
      by (rule \text{fold-max-leq}) \text{simp}
    ultimately
    show \(?\text{thesis by simp}\)
next
case \( \text{0} \)
  moreover have \( H: (\text{max} \ 0 \ x) \leq \text{foldl} \ max \ (\text{max} \ 0 \ x) \ xs \) using \text{fold-max-lower}
    by \text{simp}
  ultimately show \(?\text{thesis}\)
    by (cases \( 0 \leq x \)) \text{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 $\text{int}(l \downarrow \text{nat } j) \leq U$
using assms by simp

end

8 Verification of $r_l$

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

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

end

theory $R\text{-}L\text{-}Spark\text{-}User$
imports $R\text{-}L\text{-}Spark\text{-}Specification$
$R\text{-}L\text{-}Spark\text{-}Declaration$
$\text{Global}\text{-}User$
begin

lemma goal2':
assumes $0 \leq j''$
assumes $j'' \leq 79$
shows $(\text{block-permutation---default-arr''})$
\]
\[ j'' = \text{R}\text{-}L\text{-}Spark\text{-}Specification.r-l'.j'' \]
proof
  note nth-list-to-fun-off-eq
  moreover have $0 \leq \text{nat } j''$ by simp
  moreover from $j'' \leq 79$, have $\text{nat } j'' < \text{length } r\text{-}list$

end
unfolding \texttt{r-list-def} by \texttt{simp} \\
ultimately have conversion:
\begin{verbatim}
int (r-list ! nat j'') =
list-to-fun-eff
r-list 0 block-permutation---default-arr'' (int (nat j'')).
\end{verbatim}
show \texttt{thesis}
unfolding \texttt{r-def conversion}
unfolding \texttt{r-list-def}
using (0 \leq j'' (j'' \leq 79)
by \texttt{simp}
qed

\textbf{lemma goal2'2:}
assumes 0 \leq j''
assumes j'' \leq 79
\textbf{shows} 0 \leq (block-permutation---default-arr'')
\begin{verbatim}
(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))
\end{verbatim}
n''
unfolding \texttt{goal2'1[OF assms]}
by \texttt{simp}

\textbf{lemma goal2'3:}
assumes 0 \leq j''
assumes j'' \leq 79
\textbf{shows} (block-permutation---default-arr'')
\begin{verbatim}
(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))
\end{verbatim}
proof
  have r-list ! nat j'' ≤ 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 = goal2'1 goal2'2 goal2'3
end

9 Verification of $r_r$

theory R-R-Spark-Specification
imports Global-Specification R-R-Spark-Declaration
begin
abbreviation r-r' where
  r-r' == Global-Specification.r-r'
end
theory R-R-Spark-User
imports
  R-R-Spark-Specification
  R-R-Spark-Declaration
  Global-User
begin

lemma goal2'1:
  assumes 0 ≤ 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 := 1, 69 := 5, 70 := 8,
     71 := 7, 72 := 6, 73 := 2, 74 := 13, 75 := 14, 76 := 0, 77 := 3,
     78 := 9, 79 := 11))
  j'' =
  R-R-Spark-Specification.r-r' j"
proof 

note nth-list-to-fun-eff-eg
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 := 1, 69 := 5, 70 := 8,
     71 := 7, 72 := 6, 73 := 2, 74 := 13, 75 := 14, 76 := 0, 77 := 3,
     78 := 9, 79 := 11))

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,
\textbf{proof} \\
\quad \text{have } r'\text{-list} ! \text{ nat } j'' \leq \text{ foldl max 0 } r'\text{-list} \\
\quad \text{ by } (\text{insert assms, rule list-max}) (\text{simp-all add: } r'\text{-list-def}) \\
\quad \text{thus } ?\text{thesis unfolding goal2'1[OF assms } r'\text{-def]} \\
\quad \text{ by } (\text{rule list-max-int}) (\text{simp-all add: assms } r'\text{-list-def}) \\
\text{qed}

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

end

\textbf{10 Verification of s'1}

\textbf{theory S-L-Spark-Specification}
\textbf{imports} Global-Specification S-L-Spark-Declaration

begin

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

end

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

begin

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

**proof**

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

qed

**lemma** goal2'2:

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

\[ j'' \]

unfolding goal2'1[OF assms]

by simp

**lemma** goal2'3:

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

proof –
  have s-list ! nat j" ≤ 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 sr

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,
\begin{verbatim}
45 := 14, 44 := 13, 45 := 13, 46 := 7, 47 := 5, 48 := 15, 49 := 5,
50 := 8, 51 := 11, 52 := 14, 53 := 14, 54 := 6, 55 := 14, 56 := 6,
57 := 9, 58 := 12, 59 := 9, 60 := 12, 61 := 5, 62 := 15, 63 := 8,
64 := 8, 65 := 5, 66 := 12, 67 := 9, 68 := 12, 69 := 5, 70 := 14,
71 := 6, 72 := 8, 73 := 13, 74 := 6, 75 := 5, 76 := 15, 77 := 13,
78 := 11, 79 := 11)\)
j'' =
S-R-Spark-Specification.s-r' j''
proof −

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

qed

lemma goal2'2:
  assumes 0 <= j''
  assumes j'' <= 79
  shows 0 <= (rotate-definition---default-arr'')
    (0 := 8, 1 := 9, 2 := 9, 3 := 11, 4 := 13, 5 := 15, 6 := 15, 7 := 5,
     8 := 7, 9 := 7, 10 := 8, 11 := 11, 12 := 14, 13 := 14, 14 := 12,
     15 := 6, 16 := 9, 17 := 13, 18 := 15, 19 := 7, 20 := 12, 21 := 8,
     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''
unfolding goal2'2[OF assms]
by simp

lemma goal2'3:
  assumes 0 <= j''
  assumes j'' <= 79
  shows (rotate-definition---default-arr'')
    (0 := 8, 1 := 9, 2 := 9, 3 := 11, 4 := 13, 5 := 15, 6 := 15, 7 := 5,

\end{verbatim}

24
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 j' :: [ int , int , int , int ] => int where 
     j' == Global-Specification.j'

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'

end

12 Verification of round

theory Round-Specification

imports Global-Specification Round-Declaration

begin

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 \( s-l' :: \) int \( => \) int where
\( s-l' = \) Global-Specification \( s-l' \)

abbreviation \( s-r' :: \) int \( => \) int where
\( s-r' = \) Global-Specification \( s-r' \)

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

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

abbreviation \( \text{to-chain} :: \) chain \( => \) chain' where
\( \text{to-chain} c = \)
\( \quad \text{(let (h0, h1, h2, h3, h4) = c in} \)
\( \quad \text{chain---default-rcd''} \)
\( \quad \left([h0'chain := \text{uint h0}, \right. \)
\( \quad \left.h1'chain := \text{uint h1}, \right. \)
\( \quad \left.h2'chain := \text{uint h2}, \right. \)
\( \quad \left.h3'chain := \text{uint h3}, \right. \)
\( \quad \left.h4'chain := \text{uint h4}] \right) \)

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

abbreviation \( \text{steps' :: [chain-pair', int, block'] => chain-pair' where} \)
\( \text{steps' cc i b} = \)
\( \quad \text{to-chain-pair (steps} \)
\( \quad \text{(%n. word-of-int (b (int n)))} \)
\( \quad \text{(from-chain-pair cc) \)} \)
\( \quad \text{(not i)} \)

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

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
    (λn. 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 e-0
  fixes x
  fixes j
  assumes
  step-hyp:
\begin{align*}
\text{chain-pair} & \quad \text{---default-rcd} \\
& \begin{cases}
\langle \text{left\textquotesingle }\text{chain-pair} \rangle := \text{chain} \quad \text{---default-rcd} \\
\langle h_0\text{chain} := a, h_1\text{chain} := b, h_2\text{chain} := c, h_3\text{chain} := d, \\
h_4\text{chain} := e \rangle, \\
\text{right\textquotesingle }\text{chain-pair} := \text{chain} \quad \text{---default-rcd} \\
\langle h_0\text{chain} := a', h_1\text{chain} := b', h_2\text{chain} := c', h_3\text{chain} := d', \\
h_4\text{chain} := e' \rangle 
\end{cases} \\
\text{steps'} \\
\langle \text{chain-pair} \quad \text{---default-rcd} \rangle \\
& \begin{cases}
\langle \text{left\textquotesingle }\text{chain-pair} \rangle := \text{chain} \quad \text{---default-rcd} \\
\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 \rangle, \\
\text{right\textquotesingle }\text{chain-pair} := \text{chain} \quad \text{---default-rcd} \\
\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 \rangle 
\end{cases} \\
\end{align*}
\begin{align*}

\text{assumes a-borders: } & \quad 0 \leq a \leq 4294967295 \quad (\text{is } - \leq M) \\
\text{assumes b-borders: } & \quad 0 \leq b \leq M \\
\text{assumes c-borders: } & \quad 0 \leq c \leq M \\
\text{assumes d-borders: } & \quad 0 \leq d \leq M \\
\text{assumes e-borders: } & \quad 0 \leq e \leq M \\
\text{assumes a\textquotesingle-borders: } & \quad 0 \leq a' \leq M \\
\text{assumes b\textquotesingle-borders: } & \quad 0 \leq b' \leq M \\
\text{assumes c\textquotesingle-borders: } & \quad 0 \leq c' \leq M \\
\text{assumes d\textquotesingle-borders: } & \quad 0 \leq d' \leq M \\
\text{assumes e\textquotesingle-borders: } & \quad 0 \leq e' \leq M \\
\text{assumes x-borders: } & \quad 0 \leq x \quad (r-l' \ j) \quad x \quad (r-l' \ j) \leq M \\
& \quad 0 \leq x \quad (r-r' \ j) \quad x \quad (r-r' \ j) \leq M \\
\text{assumes j-borders: } & \quad 0 \leq j \leq 79 \\
\text{shows } \\
\text{chain-pair} \quad \text{---default-rcd} \\
& \begin{cases}
\langle \text{left\textquotesingle }\text{chain-pair} \rangle := \text{chain} \quad \text{---default-rcd} \\
\langle h_0\text{chain} := e, \\
h_1\text{chain} := \\
\quad (\text{wordops--rotate-left'} \ (s-l' \ j) \\
\quad (((a + f' \ j \ b \ c \ d) \mod 4294967296 + \\
\quad x \quad (r-l' \ j))) \mod \\
\quad 4294967296 + \\
\quad k-l' \ j) \mod \\
\quad 4294967296) + \\
\quad c) \mod \\
\quad 4294967296, \\
h_2\text{chain} := b, h_3\text{chain} := \text{wordops--rotate-left'} \ 10 \ c, \\
h_4\text{chain} := d \rangle, \\
\text{right\textquotesingle }\text{chain-pair} := \text{chain} \quad \text{---default-rcd} \\
\langle h_0\text{chain} := e', \\
h_1\text{chain} := \\
\quad (\text{wordops--rotate-left'} \ (s-r' \ j) \\
\quad (((a' + f' \ (79 \ - \ j) \ b \ c' \ d') \mod \\
\end{align*}
\[4294967296 + x (r-r') j \mod 4294967296 + k-r' j \mod 4294967296 + e' \mod 4294967296,\]
\[h2'\text{chain} := b', h3'\text{chain} := \text{wordops}-\text{rotate-left}' 10 e',\]
\[h4'\text{chain} := d'[0]\] steps'

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

(left\text{chain-pair} := \text{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),

\[\text{right}\text{chain-pair} := \text{chain}---default-rcd''\]
\[h0'\text{chain} := a-0, h1'\text{chain} := b-0, h2'\text{chain} := c-0, h3'\text{chain} := d-0, h4'\text{chain} := e-0))\]
\[(j + 1) x\]

proof –

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

have \(AL: \text{uint}(\text{word-of-int } e::\text{word32}) = e\)
by (rule uint-word-of-int-id[\(OF:0 <= e\) \(e <= \text{?M}])\)

have \(CL: \text{uint}(\text{word-of-int } b::\text{word32}) = b\)
by (rule uint-word-of-int-id[\(OF:0 <= b\) \(b <= \text{?M}])\)

have \(DL: \text{True} ..\)

have \(EL: \text{uint}(\text{word-of-int } d::\text{word32}) = d\)
by (rule uint-word-of-int-id[\(OF:0 <= d\) \(d <= \text{?M}])\)

have \(AR: \text{uint}(\text{word-of-int } e'::\text{word32}) = e'\)
by (rule uint-word-of-int-id[\(OF:0 <= e'\) \(e' <= \text{?M})\])

have \(CR: \text{uint}(\text{word-of-int } b'::\text{word32}) = b'\)
by (rule uint-word-of-int-id[\(OF:0 <= b'\) \(b' <= \text{?M})\])

have \(DR: \text{True} ..\)

have \(ER: \text{uint}(\text{word-of-int } d'::\text{word32}) = d'\)
by (rule uint-word-of-int-id[\(OF:0 <= d'\) \(d' <= \text{?M}])\)

have \(BL: (\text{uint}\n\(\text{word-rotl}\ (s\ (\text{nat } j))\n\quad ((\text{word-of-int}:\text{int} \Rightarrow \text{word32})\n\quad (((a + f' j b c d) \mod 4294967296 + x (r-l' j)) \mod 4294967296 + k-l' j) \mod 4294967296)) + e) \mod 4294967296 =\)
\(\text{uint}\n\(\text{word-rotl}\ (s\ (\text{nat } j))\n\quad (\text{word-of-int } a + f (\text{nat } j) (\text{word-of-int } b) (\text{word-of-int } c) (\text{word-of-int } d) + \text{word-of-int} (x (r-l' j)) +\)

29
\[ K (\text{nat } j)) + \]
word-of-int \( e \)
\( (\text{is } (\text{uint } (\text{word-rotl} - (- ((((- + \text{?F}) \mod + \text{?X}) \mod + -) \mod -)))) + -) \mod - = -) \]

proof
have a mod ?MM = a using \( \langle 0 <= a \rangle \langle a <= ?M \rangle \)
by (simp add: int-mod-eq')
have ?X mod ?MM = ?X using \( \langle 0 <= ?X \rangle \langle ?X <= ?M \rangle \)
by (simp add: int-mod-eq')
have e mod ?MM = e using \( \langle 0 <= e \rangle \langle e <= ?M \rangle \)
by (simp add: int-mod-eq')
have \( (?MM :: \text{int}) = 2 ^ \text{len-of TYPE} (32) \) by simp
show ?thesis
unfolding
word-add-def
uint-word-of-int-id [OF \( \langle 0 <= a \rangle \langle a <= ?M \rangle \)]
uint-word-of-int-id [OF \( \langle 0 <= ?X \rangle \langle ?X <= ?M \rangle \)]
int-word-uint
unfolding \( (?MM = 2 ^ \text{len-of TYPE} (32)) \)
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

have BR: (\text{uint} \( (\text{word-rotl} \ (s' \ (\text{nat } j))) \)
(\text{is} \ (\text{word-of-int} :: \text{int} \Rightarrow \text{word} (32)) \)
\( (((a' + f' (79 - j)) \ b' \ c' \ d') \mod 4294967296 + \)
\( x \ (r-r' j) \mod 4294967296 + \)
\( K' \ (\text{nat } j)) + \)
word-of-int \( e' \)
\( (\text{is} \ (\text{uint} \ (\text{word-rotl} - (- ((((- + \text{?F}) \mod + \text{?X}) \mod + -) \mod -)))) + -) \mod - = -) \)

proof
have a' mod ?MM = a' using \( \langle 0 <= a' \rangle \langle a' <= ?M \rangle \)
by (simp add: int-mod-eq')
have $\langle ?X \mod ?MM = ?X \rangle$ using $\langle 0 <= ?X \rangle$ 

by (simp add: int-mod-eq)

have $\langle e' \mod ?MM = e' \rangle$ using $\langle 0 <= e' \rangle$ 

by (simp add: int-mod-eq)

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

have nat-transfer: $79 - nat j = nat (79 - j)$ 

using nat-diff-distrib $\langle 0 <= j \rangle$ 

$j <= 79$

by simp

show $\langle \text{thesis} \rangle$

unfolding

word-add-def

uint-word-of-int-id[OF $\langle 0 <= a'' \rangle$ $(a'' <= ?M)$]

uint-word-of-int-id[OF $\langle 0 <= ?X \rangle$ $(?X <= ?M)$]

int-word-int

nat-transfer

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

unfolding word-uint.Abs-norm

by (simp add:

$\langle a' \mod ?MM = a' \rangle$

$\langle e' \mod ?MM = e' \rangle$

$\langle ?X \mod ?MM = ?X \rangle$

qd

show $\langle \text{thesis} \rangle$

unfolding steps'-step[OF $\langle 0 <= j \rangle$] 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)

qd

abbreviation

$f-0\text{-result} = (((ca'' + f\text{-spark'} 0 cb'' cc'' cd'') \mod 4294967296 +

x'' (r-l\text{-spark'} 0)) \mod 4294967296 + k-l\text{-spark'} 0) \mod 4294967296$

abbreviation

$f-79\text{-result} = (((ca'' + f\text{-spark'} 79 cb'' cc'' cd'') \mod 4294967296 +

x'' (r-r\text{-spark'} 0)) \mod 4294967296 + k-r\text{-spark'} 0) \mod 4294967296$

lemma goal61'1:

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 cc-borders: $0 <= cc'' cc'' <= ?M$

assumes r-l-borders: $0 <= r-l\text{-spark'} 0 r-l\text{-spark'} 0 <= 15$

assumes r-r-borders: $0 <= r-r\text{-spark'} 0 r-r\text{-spark'} 0 <= 15$

assumes returns:

wordops--rotate'(s-l\text{-spark'} 0) f-0\text{-result} =

wordops--rotate-left'(s-l\text{-spark'} 0) f-0\text{-result}

wordops--rotate'(s-r\text{-spark'} 0) f-79\text{-result} =

wordops--rotate-left'(s-r\text{-spark'} 0) f-79\text{-result}
\begin{align*}
\text{wordops--rotate}' 10 cc'' & = \text{wordops--rotate-left}' 10 cc'' \\
\text{f-spark}' 0 \text{ cb'' cc'' cd''} & = f' 0 \text{ cb'' cc'' cd''} \\
\text{f-spark}' 79 \text{ cb'' cc'' cd''} & = f' 79 \text{ cb'' cc'' cd''} \\
\text{k-l-spark}' 0 & = k-l' 0 \\
\text{k-r-spark}' 0 & = k-r' 0 \\
\text{r-l-spark}' 0 & = r-l' 0 \\
\text{r-r-spark}' 0 & = r-r' 0 \\
\text{s-l-spark}' 0 & = s-l' 0 \\
\text{s-r-spark}' 0 & = s-r' 0 \\
\text{assumes x-borders: } & \forall i. \ 0 \leq i \land i \leq 15 \implies 0 \leq x'' i \land x'' i \leq ?M \\
\text{shows chain-pair---default-rcd}'' & \\
\left\{ \begin{array}{l}
\text{left'chain-pair} := \text{chain---default-rcd}'' \\
\text{h0'chain} := \text{ce''}, \\
\text{h1'chain} := \\
\quad (\text{wordops--rotate}' (s-l-spark' 0)) \\
\quad (((ca'' + \text{f-spark}' 0 \text{ cb'' cc'' cd''}) \mod 4294967296 + x'' (r-l-spark' 0)) \mod 4294967296 + k-l-spark' 0) \mod 4294967296) + \text{ce''}) \mod 4294967296, \\
\text{h2'chain} := \text{cb''}, \text{h3'chain} := \text{ce''}, \\
\text{h4'chain} := \text{cd''} \\
\end{array} \right.
\end{align*}

\begin{align*}
\text{right'chain-pair} & := \text{chain---default-rcd}'' \\
\left\{ \begin{array}{l}
\text{h0'chain} := \text{ce''}, \\
\text{h1'chain} := \\
\quad (\text{wordops--rotate}' (s-r-spark' 0)) \\
\quad (((ca'' + \text{f-spark}' 79 \text{ cb'' cc'' cd''}) \mod 4294967296 + x'' (r-r-spark' 0)) \mod 4294967296 + k-r-spark' 0) \mod 4294967296) + \text{ce''}) \mod 4294967296, \\
\text{h2'chain} := \text{cb''}, \text{h3'chain} := \text{wordops--rotate}' 10 cc'', \\
\text{h4'chain} := \text{ce''} \\
\end{array} \right.
\end{align*}

\text{steps'} \\
\left\{ \begin{array}{l}
\text{left'chain-pair} := \text{chain---default-rcd}'' \\
\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''} \\
\end{array} \right.

\left\{ \begin{array}{l}
\text{right'chain-pair} := \text{chain---default-rcd}'' \\
\text{h0'chain} := \text{ca''}, \text{h1'chain} := \text{cb''}, \text{h2'chain} := \text{ce''}, \\
\text{h3'chain} := \text{cd''}, \text{h4'chain} := \text{ce''} \\
\end{array} \right.

\text{1 x''} \\
\text{proof} - \\
\text{have step-hyp:} \\
\text{chain-pair---default-rcd}''
left'\,chain-pair := chain---default-rcd''
right'\,chain-pair := chain---default-rcd''

\(\langle h_0'\,\text{chain} := ca'', h_1'\,\text{chain} := cb'', h_2'\,\text{chain} := cc'', h_3'\,\text{chain} := cd'', h_4'\,\text{chain} := ce''\rangle\)
\(\langle h_0'\,\text{chain} := ca'', h_1'\,\text{chain} := cb'', h_2'\,\text{chain} := cc'', h_3'\,\text{chain} := cd'', h_4'\,\text{chain} := ce''\rangle\)

\(\langle h_0'\,\text{chain} := ca'', h_1'\,\text{chain} := cb'', h_2'\,\text{chain} := cc'', h_3'\,\text{chain} := cd'', h_4'\,\text{chain} := ce''\rangle\)
\(\langle h_0'\,\text{chain} := ca'', h_1'\,\text{chain} := cb'', h_2'\,\text{chain} := cc'', h_3'\,\text{chain} := cd'', h_4'\,\text{chain} := ce''\rangle\)

\(0 \, x''\)

unfolding steps-def

by \{ simp add: \}
  \(\text{uint-word-of-int-id[OF caBorders]}\)
  \(\text{uint-word-of-int-id[OF cbBorders]}\)
  \(\text{uint-word-of-int-id[OF ccBorders]}\)
  \(\text{uint-word-of-int-id[OF cdBorders]}\)
  \(\text{uint-word-of-int-id[OF ceBorders]}\)

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

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

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

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

have \(0 \leq (0::\text{int})\) \text{by simp}
have \(0 \leq (79::\text{int})\) \text{by simp}

\text{note step-from-hyp [OF}
  \text{step-hyp}
  \(\text{caBorders cbBorders ccBorders cdBorders ceBorders}\)
  \(\text{caBorders cbBorders ccBorders cdBorders ceBorders}\)
\]

\text{note this[OF x-lower\ x-upper\ x-lower' x-upper' \(0 \leq 0\) \(0 \leq 79\)]}

\text{thus \?thesis unfolding returns(1) returns(2) unfolding returns}
  \text{by simp}

33
qed

abbreviation rotate-arg-l ==
 (\(((\text{cla}'' + f\text{-spark}')(\text{loop}--1--j'' + 1)\ \text{clb}''\ \text{clc}''\ \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{cra}'' + f\text{-spark}')(79 - (\text{loop}--1--j'' + 1))\ \text{crb}''
 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:
 assumes cl-a-borders: 0 <= \text{cla}'' \text{cla}'' <= 4294967295 (\text{is} - <= ?\text{M})
 assumes cl-b-borders: 0 <= \text{clb}'' \text{clb}'' <= ?\text{M}
 assumes cl-c-borders: 0 <= \text{clc}'' \text{clc}'' <= ?\text{M}
 assumes cl-d-borders: 0 <= \text{cld}'' \text{cld}'' <= ?\text{M}
 assumes cr-a-borders: 0 <= \text{cra}'' \text{cra}'' <= ?\text{M}
 assumes cr-b-borders: 0 <= \text{crb}'' \text{crb}'' <= ?\text{M}
 assumes cr-c-borders: 0 <= \text{crc}'' \text{crc}'' <= ?\text{M}
 assumes cr-d-borders: 0 <= \text{crd}'' \text{crd}'' <= ?\text{M}
 assumes ca-borders: 0 <= \text{cre}'' \text{cre}'' <= ?\text{M}

assumes step-hyp:

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

(left\text{chain-pair} := \text{chain-\text{---default-rcd}''}
 h0'\text{chain} := \text{cla}''', h1'\text{chain} := \text{clb}''', h2'\text{chain} := \text{clc}'',
 h3'\text{chain} := \text{cld}'', h4'\text{chain} := \text{clc}''\})

(right\text{chain-pair} := \text{chain-\text{---default-rcd}''}
 h0'\text{chain} := \text{cra}''', h1'\text{chain} := \text{crb}''', h2'\text{chain} := \text{crc}'',
 h3'\text{chain} := \text{crd}'', h4'\text{chain} := \text{cre}''\})

steps'

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

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

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

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

assumes \text{returns}:

wordops-\text{rotate}' (s-l-spark' (\text{loop}--1--j'' + 1)) \text{rotate-arg-l} =
wordops-\text{rotate-left}' (s-l-spark' (\text{loop}--1--j'' + 1)) \text{rotate-arg-l}

wordops-\text{rotate}' (s-r-spark' (\text{loop}--1--j'' + 1)) \text{rotate-arg-r} =
wordops-\text{rotate-left}' (s-r-spark' (\text{loop}--1--j'' + 1)) \text{rotate-arg-r}

f\text{-spark}' (\text{loop}--1--j'' + 1) \text{clb}'' \text{clc}'' \text{cld}'''' =
 f' (\text{loop}--1--j'' + 1) \text{clb}'' \text{clc}'' \text{cld}''''
f-spark' (78 \text{-} \text{loop}--1{-}j''') \text{ crb'' crc'' crd''} = \\
 f' (78 \text{-} \text{loop}--1{-}j''') \text{ crb'' crc'' crd''} \\
 \text{wordops--rotate' 10 cle'' = wordops--rotate-left' 10 cle''} \\
 \text{wordops--rotate' 10 crc'' = wordops--rotate-left' 10 crc''} \\
 k-l-spark' (\text{loop}--1{-}j'''' + 1) = k-l' (\text{loop}--1{-}j'''' + 1) \\
 k-r-spark' (\text{loop}--1{-}j'''' + 1) = k-r' (\text{loop}--1{-}j'''' + 1) \\
 r-l-spark' (\text{loop}--1{-}j'''' + 1) = r-l' (\text{loop}--1{-}j'''' + 1) \\
 r-r-spark' (\text{loop}--1{-}j'''' + 1) = r-r' (\text{loop}--1{-}j'''' + 1) \\
 s-l-spark' (\text{loop}--1{-}j'''' + 1) = s-l' (\text{loop}--1{-}j'''' + 1) \\
 s-r-spark' (\text{loop}--1{-}j'''' + 1) = s-r' (\text{loop}--1{-}j'''' + 1) \\
 \text{assumes x-thirds: } \forall i. 0 \leq i \land i \leq 15 \rightarrow 0 \leq x'' i \land x'' i \leq ?M \\
 \text{assumes r-l-thirds:} \\
 0 \leq r-l-spark' (\text{loop}--1{-}j'''' + 1) \leq 15 \\
 \text{assumes r-r-thirds:} \\
 0 \leq r-r-spark' (\text{loop}--1{-}j'''' + 1) \leq 15 \\
 \text{assumes j-loop-1-thirds: } 0 \leq \text{loop}--1{-}j'' \text{ loop}--1{-}j'' \leq 78 \\
 \text{shows chain-pair---default-rcd''} \\
 \langle \text{left'}\text{chain-pair} := \text{chain---default-rcd''} \\
 \langle h0'\text{chain} := \text{cle''}, \\
 h1'\text{chain} := \\
 (\text{wordops--rotate' (s-l-spark' (\text{loop}--1{-}j'''' + 1))}) \\
 (((\text{cle''} + f-spark' (\text{loop}--1{-}j'''' + 1)) \text{ clb'' cle'' cld''}) \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'\text{chain} := \text{clb''}, h3'\text{chain} := \text{wordops--rotate' 10 cle''}, \\
 h4'\text{chain} := \text{cld''}) \rangle, \\
 \text{right'}\text{chain-pair} := \text{chain---default-rcd''} \\
 \langle h0'\text{chain} := \text{cre''}, \\
 h1'\text{chain} := \\
 (\text{wordops--rotate' (s-r-spark' (\text{loop}--1{-}j'''' + 1))}) \\
 (((\text{cre''} + \\
 f-spark' (79 \text{-} \text{loop}--1{-}j'''' + 1)) \text{ crb'' crc'' 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'\text{chain} := \text{crb''}, h3'\text{chain} := \text{wordops--rotate' 10 cre''}, \\
 h4'\text{chain} := \text{crd''}) \rangle = \\
 \text{steps'} \\
 \langle \text{chain-pair---default-rcd''} \\
 \rangle
proof -

have s: 78 = (loop--1--j'' + 2) x'' by simp

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

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

from j-loop-1-borders have 0 <= loop--1--j'' + 1 by simp
from j-loop-1-borders have loop--1--j'' + 1 <= 79 by simp

have f' (79 - (loop--1--j'' + 1)) crb'' crc'' crd'' = f-spark' (79 - (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
crd-borders
cre-borders]
from this[OF
  x-lower x-upper x-lower' x-upper'
\langle 0 <= \text{loop--1-j''} + 1 \rangle (\text{loop--1-j''} + 1 <= 79)]
show ?thesis unfolding \langle \text{loop--1-j''} + 1 + 1 = \text{loop--1-j''} + 2 \rangle
  unfolding returns(1) returns(2) unfolding returns
  by simp
qed

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

lemma goal76'1:
  assumes cla-borders: 0 <= cla'' cla''' <= 4294967295 (is - <= ?M)
  assumes clb-borders: 0 <= clb'' clb''' <= ?M
  assumes clc-borders: 0 <= clc'' clc''' <= ?M
  assumes cld-borders: 0 <= cld'' cld''' <= ?M
  assumes cle-borders: 0 <= cle'' cle''' <= ?M
  assumes cra-borders: 0 <= cra'' cra''' <= ?M
  assumes crb-borders: 0 <= crb'' crb''' <= ?M
  assumes crc-borders: 0 <= crc'' crc''' <= ?M
  assumes crd-borders: 0 <= crd'' crd''' <= ?M
  assumes cre-borders: 0 <= cre'' cre''' <= ?M
  assumes ca-init-borders: 0 <= ca---init'' ca---init''' <= ?M
  assumes cb-init-borders: 0 <= cb---init'' cb---init''' <= ?M
  assumes cc-init-borders: 0 <= cc---init'' cc---init''' <= ?M
  assumes cd-init-borders: 0 <= cd---init'' cd---init''' <= ?M
  assumes ce-init-borders: 0 <= ce---init'' ce---init''' <= ?M
  assumes step-hyp:
    chain-pair---default-rcd''
      \langle left'chain-pair := chain---default-rcd''
        \langle h0'chain := cla'', h1'chain := clb'', h2'chain := clc'', h3'chain := cld'',
          h4'chain := cle''\rangle,
        right'chain-pair := chain--default-rcd''
          \langle h0'chain := cra'', h1'chain := crb'', h2'chain := crc'', h3'chain := crd'',
            h4'chain := cre''\rangle\rangle =
      steps'
    \langle chain-pair---default-rcd''
      \langle left'chain-pair := chain---default-rcd''
        \langle h0'chain := ca---init'', h1'chain := cb---init'', h2'chain := cc---init'',
          h3'chain := cd---init'',
          h4'chain := ce---init''\rangle,
        right'chain-pair := chain--default-rcd''
          \langle h0'chain := ca---init'', h1'chain := cb---init'', h2'chain := cc---init'',
            h3'chain := cd---init'',
            h4'chain := ce---init''\rangle\rangle \rangle
    80 x''
  shows chain---default-rcd''
\begin{align*}
\forall h^0_\text{chain} & := ((cb\text{-init}'' + clc'') \mod 4294967296 + crd'') \mod 4294967296, \\
\forall h^1_\text{chain} & := ((cc\text{-init}'' + cld'') \mod 4294967296 + cre'') \mod 4294967296, \\
\forall h^2_\text{chain} & := ((cd\text{-init}'' + cle'') \mod 4294967296 + cra'') \mod 4294967296, \\
\forall h^3_\text{chain} & := ((ce\text{-init}'' + cla'') \mod 4294967296 + crb'') \mod 4294967296, \\
\forall h^4_\text{chain} & := ((ca\text{-init}'' + clb'') \mod 4294967296 + crc'') \mod 4294967296)
\end{align*}

= round'
chain\text{-default-rcd''}
(\forall 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''

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

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 cle-borders]
uint-word-of-int-id[OF crb-borders]
uint-word-of-int-id[OF crc-borders]
uint-word-of-int-id[OF crc-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-rcd''
     (h0′chain := uint h0,
      h1′chain := uint h1,
      h2′chain := uint h2,
      h3′chain := uint h3,
      h4′chain := uint h4))

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

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

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

end

theory Hash-User
imports Hash-Specification Hash-Declaration

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

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

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

\[
\text{rounds'} = \text{rounds}''
\]

\[
\begin{align*}
\text{chain} & \triangleq \text{default-rcd''} \\
\text{h0'chain} & \triangleq (1732584193 :: \text{int}) \\
\text{h1'chain} & \triangleq (4023233417 :: \text{int}) \\
\text{h2'chain} & \triangleq (2562383102 :: \text{int}) \\
\text{h3'chain} & \triangleq (271733878 :: \text{int}) \\
\text{h4'chain} & \triangleq (3285377520 :: \text{int}) \\
\end{align*}
\]

\[
(x'-\text{index-\text{--}subtype-1-first''} + (1 :: \text{int}) ) \quad x''
\]

\text{is } ?C1

\text{using } H1 \; H6

\text{by } (\text{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})

\text{lemma } \text{rounds-step}:
\text{assumes } 0 <= i
\text{shows } \text{rounds X b (Suc i) = round (X i) (rounds X b i)}
\text{by } (\text{simp add: rounds-def rmd-body-def})

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

\text{lemma } \text{steps-to-steps'}:
\text{round X (foldl a b c) = round X (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--1--i''
  assumes H1: chain---default-rcd''
    (| h0'chain := ca'' |
    (| h1'chain := cb'' |
    (| h2'chain := ca'' |
    (| h3'chain := cd'' |
    (| h4'chain := ca'' |
    )
  )
  )
  = rounds'
    ( chain---default-rcd''
    (| h0'chain := (1732584193 :: int) |
    (| h1'chain := (4023233417 :: int) |
    (| h2'chain := (2562983102 :: int) |
    (| h3'chain := (271733878 :: int) |
    (| h4'chain := (3285377520 :: int) |
    )
  )
  ( loop--1--i'' + (1 :: int) )

42
assumes $H18$:

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
        := ca''
        |)
    ( | $h1'$chain
        := cb''
        |)
    ( | $h2'$chain
        := cc''
        |)
    ( | $h3'$chain
        := cd''
        |)
    ( | $h4'$chain
        := ce''
        |)
  )
( $x'' ( loop--1--i'' + (1 :: int) )$ )
)

shows  chain---default-rcd''
(| $h0'$chain
  := ca--1''
  |)
(| $h1'$chain
  := cb--1''
  |)
(| $h2'$chain
  := cc--1''
  |)
\begin{verbatim}
|)
| h3'\textit{chain} 
  ::= \textit{cd--1}''
|)
| h4'\textit{chain}  
  ::= \textit{ce--1}''
|)
= \textit{rounds'}
( \textit{chain---default-rcd}''
 (| h0'\textit{chain} 
   ::= (1732584193 :: \textit{int})
 (| h1'\textit{chain} 
   ::= (402323417 :: \textit{int})
 (| h2'\textit{chain} 
   ::= (2562383102 :: \textit{int})
 (| h3'\textit{chain} 
   ::= (271733878 :: \textit{int})
 (| h4'\textit{chain} 
   ::= (3285377520 :: \textit{int})
 |
 |
 |
 |
 |
 )
 ( \textit{loop--1--i}'' + (2 :: \textit{int}) )
)
\end{verbatim}

\textbf{proof} –
\begin{itemize}
  \item \textbf{have} \textit{loop-suc}: \textit{loop--1--i}'' + 2 = (\textit{loop--1--i}'' + 1) + 1 \textbf{by} simp
  \item \textbf{have} 0 <= \textit{loop--1--i}'' + 1 \textbf{using} \langle 0 <= \textit{loop--1--i}'' \rangle \textbf{by} simp
  \item \textbf{show} \textbf{?thesis}
    \textbf{unfolding} \textit{loop-suc}
    \textbf{unfolding} \textit{rounds'-step}[OF \langle 0 <= \textit{loop--1--i}'' \rangle]
    \textbf{unfolding} \textit{H1}[\textit{symmetric}]
    \textbf{unfolding} \textit{H18} ..
\end{itemize}
\textbf{qed}

\textbf{lemma} \textit{goal17'1}:
\begin{itemize}
  \item \textbf{assumes} \textit{H1}:
    \textit{chain---default-rcd}''
    (| h0'\textit{chain} 
      ::= \textit{ca}''
    (| h1'\textit{chain} 
      ::= \textit{cb}''
    (| h2'\textit{chain}
\end{itemize}
\[ \begin{align*}
&:= cc''
\end{align*} \]
\[\begin{align*}
&:= cd''
\end{align*} \]
\[\begin{align*}
&:= ce''
\end{align*} \]
\[\begin{align*}
&= \text{rounds'}
\end{align*} \]
\[\begin{align*}
\text{(chain---default-rcd')} &\begin{align*}
\text{|} h0' \text{chain} &:= (1732584193 :: \text{int}) \\
\text{|} h1' \text{chain} &:= (4023233417 :: \text{int}) \\
\text{|} h2' \text{chain} &:= (2562383102 :: \text{int}) \\
\text{|} h3' \text{chain} &:= (271733878 :: \text{int}) \\
\text{|} h4' \text{chain} &:= (3285377520 :: \text{int}) \\
\end{align*} \]
\[\begin{align*}
\text{x--index--subtype--1--last''} &+ (1 :: \text{int}) \end{align*} \]
\[\begin{align*}
\text{x''} &\text{x'' shows chain---default-rcd''} \\
\text{|} h0' \text{chain} &:= ca'' \\
\text{|} h1' \text{chain} &:= cb'' \\
\text{|} h2' \text{chain} &:= cc'' \\
\text{|} h3' \text{chain} &:= cd'' \\
\text{|} h4' \text{chain} &:= ce'' \\
\end{align*} \]
\[\begin{align*}
\text{rmd-hash'} &\begin{align*}
\text{x''} &\text{x'' (x--index--subtype--1--last''} + (1 :: \text{int}) \end{align*} \]
unfolding rmd-def H1 rounds-def ..

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

References


