

Mersenne primes and the Lucas–Lehmer test

Manuel Eberl

May 20, 2026

Abstract

This article provides formal proofs of basic properties of Mersenne numbers, i. e. numbers of the form $2^n - 1$, and especially of Mersenne primes. In particular, an efficient, verified, and executable version of the Lucas–Lehmer test is developed. This test decides primality for Mersenne numbers in time polynomial in n .

Contents

1	Auxiliary material	2
1.1	Auxiliary number-theoretic material	2
1.2	Auxiliary algebraic material	6
2	The Lucas–Lehmer test	9
2.1	General properties of Mersenne numbers and Mersenne primes	10
2.2	The Lucas–Lehmer sequence	14
2.3	The ring $\mathbb{Z}[\sqrt{3}]$	15
2.4	The ring $(\mathbb{Z}/m\mathbb{Z})[\sqrt{3}]$	16
2.5	$\mathbb{Z}[\sqrt{3}]$ as a subring of \mathbb{R}	21
2.6	The canonical homomorphism $\mathbb{Z}[\sqrt{3}] \rightarrow (\mathbb{Z}/m\mathbb{Z})[\sqrt{3}]$	23
2.7	Correctness of the Lucas–Lehmer test	25
2.8	A first executable version Lucas–Lehmer test	34
3	Efficient code for testing Mersenne primes	35
3.1	Efficient computation of remainders modulo a Mersenne number	36
3.2	Efficient code for the Lucas–Lehmer sequence	39
3.3	Code for the Lucas–Lehmer test	40
3.4	Examples	40

1 Auxiliary material

```
theory Lucas-Lehmer-Auxiliary
imports
  HOL-Algebra.Ring
  Probabilistic-Prime-Tests.Jacobi-Symbol
begin
```

1.1 Auxiliary number-theoretic material

```
lemma congD:  $[a = b] \pmod n \implies a \bmod n = b \bmod n$ 
  by (auto simp: cong-def)
```

```
lemma eval-coprime:
   $(b :: 'a :: euclidean-semiring-gcd) \neq 0 \implies \text{coprime } a \ b \longleftrightarrow \text{coprime } b \ (a \bmod b)$ 
  by (simp add: coprime-commute)
```

```
lemma two-power-odd-mod-12:
  assumes odd n n > 1
  shows  $[2^n = 8] \pmod{12}$ 
  using assms
proof (induction n rule: less-induct)
  case (less n)
  show ?case
  proof (cases n = 3)
  case False
  with less.prem1 have n > 3 by (auto elim!: oddE)
  hence  $[2^{n-2+2} = (8 * 4)] \pmod{12}$ 
  unfolding power-add using less.prem1 by (intro cong-mult less) auto
  also have  $n - 2 + 2 = n$ 
  using <n > 3 by simp
  finally show ?thesis by (simp add: cong-def)
qed auto
qed
```

```
lemma Legendre-3-right:
  fixes p :: nat
  assumes p: prime p p > 3
  shows  $p \bmod 12 \in \{1, 5, 7, 11\}$  and Legendre p 3 = (if p mod 12 ∈ {1, 7}
  then 1 else -1)
proof -
  have coprime p 2 using p prime-nat-not-dvd[of p 2]
  by (intro prime-imp-coprime) (auto dest: dvd-imp-le)
  moreover have coprime p 3 using p
  by (intro prime-imp-coprime) auto
  ultimately have coprime p (2 * 2 * 3)
  unfolding coprime-mult-right-iff by auto
  hence coprime 12 p
  by (simp add: coprime-commute)
```

```

hence  $p \bmod 12 \in \{p \in \{..11\}. \text{coprime } 12\ p\}$  by auto
also have  $\{p \in \{..11\}. \text{coprime } 12\ p\} = \{1::\text{nat}, 5, 7, 11\}$ 
  unfolding atMost-nat-numeral-pred-numeral-simps arith-simps
  by (auto simp del: coprime-imp-gcd-eq-1 simp: eval-coprime)
finally show  $p \bmod 12 \in \{1, 5, 7, 11\}$  by auto
hence  $p \bmod 12 = 1 \vee p \bmod 12 = 5 \vee p \bmod 12 = 7 \vee p \bmod 12 = 11$ 
  by auto
thus  $\text{Legendre } p\ 3 = (\text{if } p \bmod 12 \in \{1, 7\} \text{ then } 1 \text{ else } -1)$ 
proof safe
  assume  $p \bmod 12 = 1$ 
  have  $\text{Legendre } (\text{int } p)\ 3 = \text{Legendre } (\text{int } p \bmod 3)\ 3$ 
    by (intro Legendre-mod [symmetric]) auto
  also from  $\langle p \bmod 12 = 1 \rangle$  have  $p \bmod 12 \bmod 3 = 1$  by simp
  hence  $p \bmod 3 = 1$  by (simp add: mod-mod-cancel)
  hence  $\text{int } p \bmod 3 = 1$  by presburger
  finally have  $\text{Legendre } p\ 3 = 1$  by simp
  thus ?thesis using  $\langle p \bmod 12 = 1 \rangle$  by simp
next
  assume  $p \bmod 12 = 5$ 
  have  $\text{Legendre } (\text{int } p)\ 3 = \text{Legendre } (\text{int } p \bmod 3)\ 3$ 
    by (intro Legendre-mod [symmetric]) auto
  also from  $\langle p \bmod 12 = 5 \rangle$  have  $p \bmod 12 \bmod 3 = 2$  by simp
  hence  $p \bmod 3 = 2$  by (simp add: mod-mod-cancel)
  hence  $\text{int } p \bmod 3 = 2$  by presburger
  finally have  $\text{Legendre } p\ 3 = -1$  by (simp add: supplement2-Legendre)
  thus ?thesis using  $\langle p \bmod 12 = 5 \rangle$  by simp
next
  assume  $p \bmod 12 = 7$ 
  have  $\text{Legendre } (\text{int } p)\ 3 = \text{Legendre } (\text{int } p \bmod 3)\ 3$ 
    by (intro Legendre-mod [symmetric]) auto
  also from  $\langle p \bmod 12 = 7 \rangle$  have  $p \bmod 12 \bmod 3 = 1$  by simp
  hence  $p \bmod 3 = 1$  by (simp add: mod-mod-cancel)
  hence  $\text{int } p \bmod 3 = 1$  by presburger
  finally have  $\text{Legendre } p\ 3 = 1$  by simp
  thus ?thesis using  $\langle p \bmod 12 = 7 \rangle$  by simp
next
  assume  $p \bmod 12 = 11$ 
  have  $\text{Legendre } (\text{int } p)\ 3 = \text{Legendre } (\text{int } p \bmod 3)\ 3$ 
    by (intro Legendre-mod [symmetric]) auto
  also from  $\langle p \bmod 12 = 11 \rangle$  have  $p \bmod 12 \bmod 3 = 2$  by simp
  hence  $p \bmod 3 = 2$  by (simp add: mod-mod-cancel)
  hence  $\text{int } p \bmod 3 = 2$  by presburger
  finally have  $\text{Legendre } p\ 3 = -1$  by (simp add: supplement2-Legendre)
  thus ?thesis using  $\langle p \bmod 12 = 11 \rangle$  by simp
qed
qed

lemma Legendre-3-left:
  fixes  $p :: \text{nat}$ 

```

assumes p : prime $p > 3$
shows Legendre 3 $p =$ (if $p \bmod 12 \in \{1, 11\}$ then 1 else -1)
proof (cases $p \bmod 12 = 1 \vee p \bmod 12 = 5$)
 case True
 hence $p \bmod 12 \bmod 4 = 1$ **by** auto
 hence even $((p - \text{Suc } 0) \text{ div } 2)$
 by (intro even-mod-4-div-2) (auto simp: mod-mod-cancel)
 with Quadratic-Reciprocity[*of* p 3] Legendre-3-right(2)[*of* p] *assms* True **show**
 ?thesis
 by auto
next
 case False
 with Legendre-3-right(1)[*OF* *assms*] **have** *: $p \bmod 12 = 7 \vee p \bmod 12 = 11$
 by auto
 hence $p \bmod 12 \bmod 4 = 3$ **by** auto
 hence odd $((p - \text{Suc } 0) \text{ div } 2)$
 by (intro odd-mod-4-div-2) (auto simp: mod-mod-cancel)
 with Quadratic-Reciprocity[*of* p 3] Legendre-3-right(2)[*of* p] *assms* * **show** ?thesis
 by fastforce
qed

lemma supplement2-Legendre':
assumes prime p $p \neq 2$
shows Legendre 2 $p =$ (if $p \bmod 8 = 1 \vee p \bmod 8 = 7$ then 1 else -1)
proof -
 from *assms* **have** $p > 2$
 using prime-gt-1-int[*of* p] **by** auto
 moreover from *this* **and** *assms* **have** odd p
 by (auto simp: prime-odd-int)
 ultimately show ?thesis
 using supplement2-Jacobi'[*of* p] *assms* prime-odd-int[*of* p]
 by (simp add: prime-p-Jacobi-eq-Legendre)
qed

lemma little-Fermat-nat:
fixes $a :: \text{nat}$
assumes prime p $\neg p \text{ dvd } a$
shows $[a \wedge^p = a] \pmod{p}$
proof -
 have $p = \text{Suc } (p - 1)$
 using prime-gt-0-nat[*OF* *assms*(1)] **by** simp
 also have $p - 1 = \text{totient } p$
 using *assms* **by** (simp add: totient-prime)
 also have $a \wedge^{\text{Suc } \dots} = a * a \wedge^{\text{totient } p}$
 by simp
 also have $[\dots = a * 1] \pmod{p}$
 using prime-imp-coprime[*of* p a] *assms*
 by (intro cong-mult cong-refl euler-theorem) (auto simp: coprime-commute)
 finally show ?thesis **by** simp

qed

lemma *little-Fermat-int*:

fixes $a :: \text{int}$ **and** $p :: \text{nat}$
assumes $\text{prime } p \ \neg p \ \text{dvd } a$
shows $[a \wedge^p = a] \ (\text{mod } p)$

proof –

have $p > 1$ **using** *prime-gt-1-nat* **assms** **by** *simp*
have $\neg \text{int } p \ \text{dvd } a \ \text{mod } \text{int } p$
using *assms* **by** (*simp add: dvd-mod-iff*)
also from $\langle p > 1 \rangle$ **have** $a \ \text{mod } \text{int } p = \text{int } (\text{nat } (a \ \text{mod } \text{int } p))$
by *simp*
finally have *not-dvd*: $\neg p \ \text{dvd } \text{nat } (a \ \text{mod } \text{int } p)$
by (*subst (asm) int-dvd-int-iff*)

have $[a \wedge^p = (a \ \text{mod } p) \wedge^p] \ (\text{mod } p)$
by (*intro cong-pow*) (*auto simp: cong-def*)
also have $(a \ \text{mod } p) \wedge^p = (\text{int } (\text{nat } (a \ \text{mod } p))) \wedge^p$
using $\langle p > 1 \rangle$ **by** (*subst of-nat-nat*) *auto*
also have $\dots = \text{int } (\text{nat } (a \ \text{mod } p) \wedge^p)$
by *simp*
also have $[\dots = \text{int } (\text{nat } (a \ \text{mod } p))] \ (\text{mod } p)$
by (*subst cong-int-iff, rule little-Fermat-nat*) (*use assms not-dvd in auto*)
also have $\text{int } (\text{nat } (a \ \text{mod } p)) = a \ \text{mod } p$
using $\langle p > 1 \rangle$ **by** *simp*
also have $[a \ \text{mod } p = a] \ (\text{mod } p)$
by (*auto simp: cong-def*)
finally show *?thesis* .

qed

lemma *prime-dvd-choose*:

assumes $0 < k \ k < p \ \text{prime } p$
shows $p \ \text{dvd } (p \ \text{choose } k)$

proof –

have $k \leq p$ **using** $\langle k < p \rangle$ **by** *auto*

have $p \ \text{dvd } \text{fact } p$ **using** *assms* **by** (*simp add: prime-dvd-fact-iff*)

moreover have $\neg p \ \text{dvd } \text{fact } k * \text{fact } (p - k)$
unfolding *prime-dvd-mult-iff*[*OF assms*(3)] *prime-dvd-fact-iff*[*OF assms*(3)]
using *assms* **by** *simp*

ultimately show *?thesis*

unfolding *binomial-fact-lemma*[*OF* $\langle k \leq p \rangle$, *symmetric*]
using *assms* *prime-dvd-multD* **by** *blast*

qed

lemma *prime-natD*:

assumes $\text{prime } (p :: \text{nat}) \ a \ \text{dvd } p$

shows $a = 1 \vee a = p$
using *assms* **by** (*auto simp: prime-nat-iff*)

lemma *not-prime-imp-ex-prod-nat*:
assumes $m > 1 \neg \text{prime } (m::\text{nat})$
shows $\exists n k. m = n * k \wedge 1 < n \wedge n < m \wedge 1 < k \wedge k < m$
proof –

from *assms* **have** $\neg \text{Factorial-Ring.irreducible } m$
by (*simp flip: prime-elem-iff-irreducible*)

with *assms* **obtain** $n k$ **where** $nk: m = n * k \ n \neq 1 \ k \neq 1$
by (*auto simp: Factorial-Ring.irreducible-def*)

moreover from *this* *assms* **have** $n > 0 \ k > 0$
by *auto*

with nk **have** $n > 1 \ k > 1$ **by** *auto*

moreover {

from *assms nk* **have** $n \text{ dvd } m \ k \text{ dvd } m$ **by** *auto*

with *assms* **have** $n \leq m \ k \leq m$
by (*auto intro!: dvd-imp-le*)

moreover from $nk \langle n > 1 \rangle \langle k > 1 \rangle$ **have** $n \neq m \ k \neq m$
by *auto*

ultimately have $n < m \ k < m$ **by** *auto*

}

ultimately show *?thesis* **by** *blast*

qed

1.2 Auxiliary algebraic material

lemma (*in group*) *ord-eqI-prime-factors*:
assumes $\bigwedge p. p \in \text{prime-factors } n \implies x [\wedge] (n \text{ div } p) \neq 1$ **and** $x [\wedge] n = 1$
assumes $x \in \text{carrier } G \ n > 0$
shows $\text{group.ord } G \ x = n$
proof –

have $\text{group.ord } G \ x \text{ dvd } n$
using *assms* **by** (*subst pow-eq-id [symmetric]*) *auto*

then obtain k **where** $k: n = \text{group.ord } G \ x * k$
by *auto*

have $k = 1$

proof (*rule ccontr*)

assume $k \neq 1$

then obtain p **where** $p: \text{prime } p \ p \text{ dvd } k$
using *prime-factor-nat* **by** *blast*

have $x [\wedge] (\text{group.ord } G \ x * (k \text{ div } p)) = 1$
by (*subst pow-eq-id*) (*use assms in auto*)

also have $\text{group.ord } G \ x * (k \text{ div } p) = n \text{ div } p$
using p **by** (*auto simp: k*)

finally have $x [\wedge] (n \text{ div } p) = 1$.

moreover have $x [\wedge] (n \text{ div } p) \neq 1$
using $p \ k$ *assms* **by** (*intro assms*) (*auto simp: in-prime-factors-iff*)

ultimately show *False* **by** *contradiction*

qed
 with k show ?thesis by simp
 qed

lemma (in monoid) pow-nat-eq-1-imp-unit:

fixes $n :: nat$
 assumes $x [\wedge] n = \mathbf{1}$ and $n > 0$ and [simp]: $x \in carrier\ G$
 shows $x \in Units\ G$

proof –

from $\langle n > 0 \rangle$ have $x [\wedge] (1 :: nat) \otimes x [\wedge] (n - 1) = x [\wedge] n$
 by (subst nat-pow-mult) auto
 with assms have $x \otimes x [\wedge] (n - 1) = \mathbf{1}$
 by simp

moreover from $\langle n > 0 \rangle$ have $x [\wedge] (n - 1) \otimes x [\wedge] (1 :: nat) = x [\wedge] n$
 by (subst nat-pow-mult) auto
 with assms have $x [\wedge] (n - 1) \otimes x = \mathbf{1}$
 by simp

ultimately show ?thesis by (auto simp: Units-def)

qed

lemma (in cring) finsum-reindex-bij-betw:

assumes bij-betw $h\ S\ T\ g \in T \rightarrow carrier\ R$
 shows $finsum\ R\ (\lambda x. g\ (h\ x))\ S = finsum\ R\ g\ T$
 using assms by (auto simp: bij-betw-def finsum-reindex)

lemma (in cring) finsum-reindex-bij-witness:

assumes witness:
 $\bigwedge a. a \in S \implies i\ (j\ a) = a$
 $\bigwedge a. a \in S \implies j\ a \in T$
 $\bigwedge b. b \in T \implies j\ (i\ b) = b$
 $\bigwedge b. b \in T \implies i\ b \in S$
 $\bigwedge b. b \in S \implies g\ b \in carrier\ R$

assumes eq:

$\bigwedge a. a \in S \implies h\ (j\ a) = g\ a$

shows $finsum\ R\ g\ S = finsum\ R\ h\ T$

proof –

have bij: bij-betw $j\ S\ T$

using bij-betw-byWitness[where $A=S$ and $f=j$ and $f'=i$ and $A'=T$] witness

by auto

hence T-eq: $T = j\ ' S$ by (auto simp: bij-betw-def)

from assms have $h \in T \rightarrow carrier\ R$

by (subst T-eq) auto

moreover have $finsum\ R\ g\ S = finsum\ R\ (\lambda x. h\ (j\ x))\ S$

using assms by (intro finsum-cong) (auto simp: eq)

ultimately show ?thesis using assms(5)

using finsum-reindex-bij-betw[OF bij, of h] by simp

qed

lemma (in cring) binomial:

fixes $n :: \text{nat}$
assumes $[simp]: x \in \text{carrier } R \ y \in \text{carrier } R$
shows $(x \oplus y) [\wedge] n = (\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (n \text{ choose } i) \ (x [\wedge] i \otimes y [\wedge] (n - i)))$
proof (*induction n*)
case (*Suc n*)
have *binomial-Suc*: $\text{Suc } n \text{ choose } i = (n \text{ choose } (i - 1)) + (n \text{ choose } i)$ **if** $i \in \{1..n\}$ **for** i
using *that by (cases i) auto*
have *Suc-diff*: $\text{Suc } n - i = \text{Suc } (n - i)$ **if** $i \leq n$ **for** i
using *that by linarith*
have $(x \oplus y) [\wedge] \text{Suc } n =$
 $(\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (n \text{ choose } i) \ (x [\wedge] i \otimes y [\wedge] (n - i))) \otimes x \oplus$
 $(\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (n \text{ choose } i) \ (x [\wedge] i \otimes y [\wedge] (n - i))) \otimes y$
by (*simp add: semiring-simprules Suc*)
also have $(\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (n \text{ choose } i) \ (x [\wedge] i \otimes y [\wedge] (n - i))) \otimes x =$
 $(\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (n \text{ choose } i) \ (x [\wedge] \text{Suc } i \otimes y [\wedge] (n - i)))$
by (*subst finsum-ldistr*)
(auto simp: cring-simprules Suc add-pow-rdistr intro!: finsum-cong)
also have $\dots = (\bigoplus_{i \in \{1..\text{Suc } n\}}. \text{add-pow } R \ (n \text{ choose } (i - 1)) \ (x [\wedge] i \otimes y$
 $[\wedge] (\text{Suc } n - i)))$
by (*intro finsum-reindex-bij-witness[of - \lambda i. i - 1 Suc] auto*)
also have $\{1..\text{Suc } n\} = \text{insert } (\text{Suc } n) \ \{1..n\}$ **by** *auto*
also have $(\bigoplus_{i \in \dots} \text{add-pow } R \ (n \text{ choose } (i - 1)) \ (x [\wedge] i \otimes y [\wedge] (\text{Suc } n -$
 $i))) =$
 $x [\wedge] \text{Suc } n \oplus (\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (n \text{ choose } (i - 1)) \ (x [\wedge] i \otimes y$
 $[\wedge] (\text{Suc } n - i)))$
(is - = - \oplus ?S1) by (*subst finsum-insert auto*)
also have $(\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (n \text{ choose } i) \ (x [\wedge] i \otimes y [\wedge] (n - i))) \otimes y =$
 $(\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (n \text{ choose } i) \ (x [\wedge] i \otimes y [\wedge] (\text{Suc } n - i)))$
by (*subst finsum-ldistr*)
(auto simp: cring-simprules Suc add-pow-rdistr Suc-diff intro!: finsum-cong)
also have $\{1..n\} = \text{insert } 0 \ \{1..n\}$ **by** *auto*
also have $(\bigoplus_{i \in \dots} \text{add-pow } R \ (n \text{ choose } i) \ (x [\wedge] i \otimes y [\wedge] (\text{Suc } n - i))) =$
 $y [\wedge] \text{Suc } n \oplus (\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (n \text{ choose } i) \ (x [\wedge] i \otimes y [\wedge] (\text{Suc}$
 $n - i)))$
(is - = - \oplus ?S2) by (*subst finsum-insert auto*)
also have $(x [\wedge] \text{Suc } n \oplus ?S1) \oplus (y [\wedge] \text{Suc } n \oplus ?S2) =$
 $x [\wedge] \text{Suc } n \oplus y [\wedge] \text{Suc } n \oplus (?S1 \oplus ?S2)$
by (*simp add: cring-simprules*)
also have $?S1 \oplus ?S2 = (\bigoplus_{i \in \{1..n\}}. \text{add-pow } R \ (\text{Suc } n \text{ choose } i) \ (x [\wedge] i \otimes y$
 $[\wedge] (\text{Suc } n - i)))$
by (*subst finsum-addf [symmetric], simp, simp, rule finsum-cong'*)
(auto intro!: finsum-cong simp: binomial-Suc add.nat-pow-mult)
also have $x [\wedge] \text{Suc } n \oplus y [\wedge] \text{Suc } n \oplus \dots =$
 $(\bigoplus_{i \in \{0, \text{Suc } n\} \cup \{1..n\}}. \text{add-pow } R \ (\text{Suc } n \text{ choose } i) \ (x [\wedge] i \otimes y$
 $[\wedge] (\text{Suc } n - i)))$
by (*subst finsum-Un-disjoint (auto simp: cring-simprules)*)
also have $\{0, \text{Suc } n\} \cup \{1..n\} = \{..\text{Suc } n\}$ **by** *auto*

finally show ?case .
qed auto

lemma (in cring) binomial-finite-char:

fixes $p :: \text{nat}$
assumes [simp]: $x \in \text{carrier } R \ y \in \text{carrier } R$ **and** $\text{add-pow } R \ p \ \mathbf{1} = \mathbf{0}$ *prime* p
shows $(x \oplus y) [\wedge] p = x [\wedge] p \oplus y [\wedge] p$
proof –
have *: $\text{add-pow } R \ (p \ \text{choose } i) \ (x [\wedge] i \otimes y [\wedge] (p - i)) = \mathbf{0}$ **if** $i \in \{1..<p\}$ **for** i
proof –
have $p \ \text{dvd} \ (p \ \text{choose } i)$
by (rule prime-dvd-choose) (use that **assms** in auto)
then obtain k **where** [simp]: $(p \ \text{choose } i) = p * k$
by auto
have $\text{add-pow } R \ (p \ \text{choose } i) \ (x [\wedge] i \otimes y [\wedge] (p - i)) =$
 $\text{add-pow } R \ (p \ \text{choose } i) \ \mathbf{1} \otimes (x [\wedge] i \otimes y [\wedge] (p - i))$
by (simp add: add-pow-ldistr)
also have $\text{add-pow } R \ (p \ \text{choose } i) \ \mathbf{1} = \mathbf{0}$
using **assms** **by** (simp flip: add.nat-pow-pow)
finally show ?thesis **by** simp
qed

have $(x \oplus y) [\wedge] p = (\bigoplus_{i \in \{..p\}} \text{add-pow } R \ (p \ \text{choose } i) \ (x [\wedge] i \otimes y [\wedge] (p - i)))$
by (rule binomial) auto
also have $\dots = (\bigoplus_{i \in \{0, p\}} \text{add-pow } R \ (p \ \text{choose } i) \ (x [\wedge] i \otimes y [\wedge] (p - i)))$
using * **by** (intro add.finprod-mono-neutral-cong-right) auto
also have $\dots = x [\wedge] p \oplus y [\wedge] p$
using **assms** prime-gt-0-nat[of p] **by** (simp add: cring-simprules)
finally show ?thesis .
qed

lemma (in ring-hom-cring) hom-add-pow-nat:

$x \in \text{carrier } R \implies h \ (\text{add-pow } R \ (n :: \text{nat}) \ x) = \text{add-pow } S \ n \ (h \ x)$
by (induction n) auto

end

2 The Lucas–Lehmer test

theory Lucas-Lehmer

imports

Lucas-Lehmer-Auxiliary

HOL-Algebra.Ring

Probabilistic-Prime-Tests.Jacobi-Symbol

Pell.Pell

begin

2.1 General properties of Mersenne numbers and Mersenne primes

We mostly follow the proofs given on Wikipedia [4, 3] in the following sections.

We first show some basic and theorems about Mersenne numbers and Mersenne primes in general, beginning with this: Mersenne primes are the only primes of the form $a^n - 1$ for $n > 1$.

lemma *prime-power-minus-oneD*:

```

fixes  $a\ n :: nat$ 
assumes  $prime\ (a^n - 1)$ 
shows  $n = 1 \vee a = 2$ 
proof -
  from assms have  $n > 0$ 
    by (intro Nat.gr0I) auto
  have  $a \neq 0\ a \neq 1$ 
    by (rule notI, use  $\langle n > 0 \rangle$  assms in  $\langle simp\ add:\ zero-power \rangle$ ) +
  hence  $a > 1$  by auto
  have  $[a - 1 + 1 = 0 + 1] \pmod{a - 1}$ 
    by (rule cong-add) (auto simp: cong-def)
  hence  $[a = 1] \pmod{a - 1}$ 
    using  $\langle a > 1 \rangle$  by simp
  hence  $[a^n - 1 = 1^n - 1] \pmod{a - 1}$ 
    using  $\langle a > 1 \rangle$  by (intro cong-pow cong-diff-nat) auto
  hence  $(a - 1) \mid (a^n - 1)$ 
    by (simp add: cong-0-iff)
  have  $a - 1 = 1 \vee a - 1 = a^n - 1$ 
    using  $\langle prime\ (a^n - 1) \rangle$  and  $\langle (a - 1) \mid (a^n - 1) \rangle$  by (rule prime-natD)
  thus ?thesis
proof
  assume  $a - 1 = 1$ 
  hence  $a = 2$  by simp
  thus ?thesis by simp
next
  assume  $a - 1 = a^n - 1$ 
  hence  $a^n = a$ 
    using  $\langle a > 1 \rangle$  by (simp add: Nat.eq-diff-iff)
  hence  $n = 1$ 
    using  $\langle a > 1 \rangle$  by (subst (asm) power-inject-exp) auto
  thus ?thesis by simp
qed
qed

```

Next, we show that if a prime q divides a Mersenne number $2^p - 1$ with an odd prime exponent p , then q must be of the form $q = 1 + 2kp$ for some $k > 0$.

lemma *prime-dvd-mersenneD*:

```

fixes  $p\ q :: nat$ 

```

assumes *prime p p ≠ 2 prime q q dvd (2 ^ p - 1)*
shows $[q = 1] \pmod{(2 * p)}$

proof –

from *assms* **have** *odd p*
using *prime-gt-1-nat[of p]* **by** (*intro prime-odd-nat*) *auto*
have $q \neq 0 \ q \neq 1 \ q \neq 2$
using *assms* **by** (*auto intro!: Nat.gr0I*)
hence $q > 2$ **by** *simp*
with $\langle \text{prime } q \rangle$ **have** *odd q*
by (*simp add: prime-odd-nat*)

have $\text{ord } q \ 2 = p$

proof –

from *assms* **have** $[2 ^ p - 1 + 1 = 0 + 1] \pmod{q}$
by (*intro cong-add cong-refl*) (*auto simp: cong-0-iff*)
hence $[2 ^ p = 1] \pmod{q}$ **by** *simp*
hence $\text{ord } q \ 2 \text{ dvd } p$
by (*subst (asm) ord-divides*)
hence $\text{ord } q \ 2 = 1 \vee \text{ord } q \ 2 = p$
using $\langle \text{prime } p \rangle$ **and** *prime-natD* **by** *blast*
moreover **have** $\text{ord } q \ 2 \neq 1$
using *ord-works[of 2 q]* **and** $\langle \text{prime } q \rangle$ **by** (*auto simp: cong-altdef-nat*)
ultimately show $\text{ord } q \ 2 = p$ **by** *blast*

qed

have *q-dvd-iff: q dvd (2 ^ x - 1) ↔ p dvd x for x :: nat*

proof –

have $q \text{ dvd } (2 ^ x - 1) \longleftrightarrow [2 ^ x = 1] \pmod{q}$
by (*auto simp: cong-altdef-nat*)
also have $\dots \longleftrightarrow \text{ord } q \ 2 \text{ dvd } x$
by (*rule ord-divides*)
also note $\langle \text{ord } q \ 2 = p \rangle$
finally show *?thesis* .

qed

from $\langle q > 2 \rangle$ **and** *assms* **have** $\neg q \text{ dvd } 2$
using *primes-dvd-imp-eq two-is-prime-nat* **by** *blast*
hence $[2 ^{(q-1)} - 1 = 1 - 1] \pmod{q}$
using *assms* **by** (*intro fermat-theorem cong-diff-nat*) *auto*
hence $q \text{ dvd } (2 ^{(q-1)} - 1)$
by (*simp add: cong-0-iff*)
hence $p \text{ dvd } (q - 1)$
by (*subst (asm) q-dvd-iff*)
hence $[q = 1] \pmod{p}$
using $\langle q > 2 \rangle$ **by** (*auto simp: cong-altdef-nat prime-gt-1-nat*)

moreover **have** $[q = 1] \pmod{2}$

using $\langle \text{odd } q \rangle$ **by** (*auto simp: cong-def odd-iff-mod-2-eq-one*)
ultimately show $[q = 1] \pmod{(2 * p)}$

using $\langle \text{odd } p \rangle$ by (intro coprime-cong-mult-nat) auto
qed

lemma *prime-dvd-mersenneD'*:
 fixes $p\ q :: \text{nat}$
 assumes *prime* p $p \neq 2$ *prime* q $q \text{ dvd } (2 \wedge p - 1)$
 shows $\exists k > 0. q = 1 + 2 * k * p$
proof –
 have $q \neq 0$ $q \neq 1$ $q \neq 2$
 using *assms* by (auto intro!: Nat.gr0I)
 hence $q > 2$ by *simp*

 have $[q = 1] \pmod{(2 * p)}$
 by (rule *prime-dvd-mersenneD*) *fact+*
 hence $(2 * p) \text{ dvd } (q - 1)$
 using $\langle q > 2 \rangle$ by (auto *simp: cong-altdef-nat*)
 then obtain k where $k: q - 1 = (2 * p) * k$
 by *blast*
 hence $q = 1 + 2 * k * p$
 using $\langle q > 2 \rangle$ by (*simp add: algebra-simps*)
 moreover have $k > 0$
 using $\langle q > 2 \rangle$ and k by (intro *Nat.gr0I*) auto
 ultimately show *?thesis* by *blast*
 qed

A Mersenne number is any number of the form $2^p - 1$ for a natural number p . To make things a bit more pleasant, we additionally exclude $2^2 - 1$, i.e. we require $p > 2$. It can be shown that p is then always an odd prime.

locale *mersenne-prime* =
 fixes $p\ M :: \text{nat}$
 defines $M \equiv 2 \wedge p - 1$
 assumes *p-gt-2*: $p > 2$ and *prime*: *prime* M
begin

lemma *M-gt-6*: $M > 6$
proof –
 from *p-gt-2* have $2 \wedge p \geq (2 \wedge 3 :: \text{nat})$
 by (intro *power-increasing*) auto
 thus *?thesis* by (*simp add: M-def*)
 qed

lemma *M-odd*: *odd* M
 using *p-gt-2* by (auto *simp: M-def*)

theorem *p-prime*: *prime* p
proof (*rule ccontr*)
 assume $\neg \text{prime } p$
 then obtain $a\ b$ where $ab: p = a * b$ $a > 1$ $b > 1$
 using *p-gt-2* *not-prime-imp-ex-prod-nat[of p]* by auto

have *geometric-sum-aux*: $(x - (1 :: \text{int})) * (\sum k < a. x^k) = x^a - 1$ **for** x
by (*induction a*) (*auto simp: algebra-simps*)
have $(2^b - 1 :: \text{int}) * (\sum k < a. (2^b)^k) = (2^b)^a - 1$
by (*rule geometric-sum-aux*)
hence $2^{(a*b)} - 1 = (2^b - 1 :: \text{int}) * (\sum k < a. 2^{(k*b)})$
by (*simp flip: power-mult add: algebra-simps*)
hence $(2^b - 1) \text{ dvd } (2^{(a*b)} - 1 :: \text{int})$
by *simp*
hence $\text{int } (2^b - 1) \text{ dvd int } (2^{(a*b)} - 1)$
by (*subst of-nat-diff*) *simp-all*
hence $(2^b - 1) \text{ dvd } (2^{(a*b)} - 1 :: \text{nat})$
by (*subst (asm) int-dvd-int-iff*)
with prime **have** $2^b - 1 = (1 :: \text{nat}) \vee 2^b - 1 = (2^p - 1 :: \text{nat})$
unfolding *ab M-def* **by** (*intro prime-natD*) *auto*
moreover **have** $2^b > (2^1 :: \text{nat})$
using *ab* **by** (*intro power-strict-increasing*) *auto*
moreover **have** $2^b < (2^p :: \text{nat})$
using *ab* **by** (*intro power-strict-increasing*) *auto*
hence $2^b - 1 < (2^p - 1 :: \text{nat})$
by (*subst less-diff-iff*) *auto*
ultimately show *False* **by** *auto*
qed

lemma *p-odd*: *odd p*
using *p-prime p-gt-2 prime-odd-nat* **by** *auto*

We now first show a few more properties of Mersenne primes regarding congruences and the Legendre symbol.

lemma *M-cong-7-mod-12*: $[M = 7] \pmod{12}$
proof –
have $[M = 8 - 1] \pmod{12}$
using *p-gt-2 p-odd* **unfolding** *M-def* **by** (*intro cong-diff-nat two-power-odd-mod-12*)
auto
thus $[M = 7] \pmod{12}$ **by** *simp*
qed

lemma *Legendre-3-M*: *Legendre 3 M = -1*
using *prime M-cong-7-mod-12* **by** (*subst Legendre-3-left*) (*auto simp: cong-def*)

lemma *M-cong-7-mod-8*: $[M = 7] \pmod{8}$
proof –
have $2^3 \text{ dvd } (2^p :: \text{int})$
using *p-gt-2* **by** (*intro le-imp-power-dvd*) *auto*
hence $[2^p - 1 = 0 - 1] \pmod{8 :: \text{int}}$
by (*intro cong-diff*) (*auto simp: cong-def*)
also **have** $2^p - 1 = \text{int } M$
by (*simp add: M-def*)
finally **have** $\text{int } M \text{ mod int } 8 = 7$

by (*simp add: cong-def*)
thus $[M = 7] \pmod{8}$
by (*subst (asm) zmod-int [symmetric]*) (*auto simp: cong-def*)
qed

lemma *Legendre-2-M: Legendre 2 M = 1*
using *prime M-gt-6 M-cong-7-mod-8*
by (*subst supplement2-Legendre'*) (*auto simp: cong-def nat-mod-as-int*)

lemma *M-not-dvd-24: $\neg M \text{ dvd } 24$*
proof
assume *M dvd 24*
hence *M dvd 2 * 2 * 2 * 3*
by *simp*
also have *?this $\longleftrightarrow M \text{ dvd } 2 \vee M \text{ dvd } 3$*
using *prime* **by** (*simp only: prime-dvd-mult-iff*) *auto*
finally show *False* **using** *M-gt-6* **by** (*auto dest: dvd-imp-le*)
qed

end

2.2 The Lucas–Lehmer sequence

We now define the Lucas–Lehmer sequence $a_{n+1} = a_n^2 - 2$. The starting value we will always use is $a_0 = 4$.

primrec *gen-lucas-lehmer-sequence* :: *int \Rightarrow nat \Rightarrow int* **where**
gen-lucas-lehmer-sequence a 0 = a
| *gen-lucas-lehmer-sequence a (Suc n) = gen-lucas-lehmer-sequence a n ^ 2 - 2*

lemma *gen-lucas-lehmer-sequence-Suc'*:
gen-lucas-lehmer-sequence a (Suc n) = gen-lucas-lehmer-sequence (a ^ 2 - 2) n
by (*induction n arbitrary: a*) *auto*

lemma *gen-lucas-lehmer-code* [*code*]:
gen-lucas-lehmer-sequence a 0 = a
gen-lucas-lehmer-sequence a (Suc n) = gen-lucas-lehmer-sequence (a ^ 2 - 2) n
by (*fact gen-lucas-lehmer-sequence.simps gen-lucas-lehmer-sequence-Suc'*)**+**

For $a_0 = 4$, the recurrence has the closed form $a_{4,n} = \omega^{2^n} + \bar{\omega}^{2^n}$ with $\omega = 2 + \sqrt{3}$ and $\bar{\omega} = 2 - \sqrt{3}$.

lemma *gen-lucas-lehmer-sequence-4-closed-form1*:
real-of-int (gen-lucas-lehmer-sequence 4 n) = (2 + sqrt 3) ^ (2 ^ n) + (2 - sqrt 3) ^ (2 ^ n)
by (*induction n*)
(*auto simp: algebra-simps power2-eq-square power-mult simp flip: power-mult-distrib*)

lemma *gen-lucas-lehmer-sequence-4-closed-form2*:
gen-lucas-lehmer-sequence 4 n = round ((2 + sqrt 3) ^ (2 ^ n))

proof (*rule sym, rule round-unique'*)
have $5 / 3 < \text{sqrt } 3 \text{ :: real}$
by (*rule real-less-rsqrt*) (*auto simp: power2-eq-square*)
hence $(2 - \text{sqrt } 3) ^ (2 ^ n) < (1 / 3) ^ (2 ^ n)$
by (*intro power-strict-mono*) (*auto simp: real-le-lsqrt*)
also have $\dots \leq (1 / 3) ^ 1$
by (*intro power-decreasing*) *auto*
finally have $(2 - \text{sqrt } 3) ^ (2 ^ n) < 1 / 2$ **by** *simp*
moreover have $(2 - \text{sqrt } 3) ^ (2 ^ n) \geq 0$
by (*intro zero-le-power*) (*auto simp: real-le-lsqrt*)
ultimately show $|(2 + \text{sqrt } 3) ^ 2 ^ n - \text{real-of-int } (\text{gen-lucas-lehmer-sequence } 4 \ n)| < 1 / 2$
unfolding *gen-lucas-lehmer-sequence-4-closed-form1* **by** *linarith*
qed

lemma *gen-lucas-lehmer-sequence-4-closed-form3*:
 $\text{gen-lucas-lehmer-sequence } 4 \ n = \lceil (2 + \text{sqrt } 3) ^ (2 ^ n) \rceil$
proof (*rule sym, rule ceiling-unique*)
show $\text{real-of-int } (\text{gen-lucas-lehmer-sequence } 4 \ n) \geq (2 + \text{sqrt } 3) ^ 2 ^ n$
unfolding *gen-lucas-lehmer-sequence-4-closed-form1* **by** (*auto intro!: zero-le-power real-le-lsqrt*)
next
have $5 / 3 < \text{sqrt } 3 \text{ :: real}$
by (*rule real-less-rsqrt*) (*auto simp: power2-eq-square*)
hence $(2 - \text{sqrt } 3) ^ (2 ^ n) < (1 / 3) ^ (2 ^ n)$
by (*intro power-strict-mono*) (*auto simp: real-le-lsqrt*)
also have $\dots \leq (1 / 3) ^ 1$
by (*intro power-decreasing*) *auto*
finally have $(2 - \text{sqrt } 3) ^ (2 ^ n) < 1 / 2$ **by** *simp*
moreover have $(2 - \text{sqrt } 3) ^ (2 ^ n) \geq 0$
by (*intro zero-le-power*) (*auto simp: real-le-lsqrt*)
ultimately show $\text{real-of-int } (\text{gen-lucas-lehmer-sequence } 4 \ n) - 1 < (2 + \text{sqrt } 3) ^ 2 ^ n$
unfolding *gen-lucas-lehmer-sequence-4-closed-form1* **by** *linarith*
qed

2.3 The ring $\mathbb{Z}[\sqrt{3}]$

To relate this sequence to Mersenne primes, we now first need to define the ring $\mathbb{Z}[\sqrt{3}]$, which is a subring of \mathbb{R} . This ring can be seen as the lattice on \mathbb{R} that is freely generated by 1 and $\sqrt{3}$.

It is, however, more convenient to explicitly describe it as a ring structure over the set $\mathbb{Z} \times \mathbb{Z}$ with a corresponding injective homomorphism $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{R}$.

definition *lucas-lehmer-add'* :: $\text{int} \times \text{int} \Rightarrow \text{int} \times \text{int} \Rightarrow \text{int} \times \text{int}$ **where**
 $\text{lucas-lehmer-add}' = (\lambda(a,b) (c,d). (a + c, b + d))$

definition *lucas-lehmer-mult'* :: $\text{int} \times \text{int} \Rightarrow \text{int} \times \text{int} \Rightarrow \text{int} \times \text{int}$ **where**
 $\text{lucas-lehmer-mult}' = (\lambda(a,b) (c,d). (a * c + 3 * b * d, a * d + b * c))$

definition *lucas-lehmer-ring* :: (int × int) ring **where**

```

lucas-lehmer-ring =
  (|carrier = UNIV,
   monoid.mult = lucas-lehmer-mult',
   one = (1, 0),
   ring.zero = (0, 0),
   add = lucas-lehmer-add')

```

lemma *carrier-lucas-lehmer-ring* [simp]: carrier *lucas-lehmer-ring* = UNIV
by (simp add: *lucas-lehmer-ring-def*)

lemma *cring-lucas-lehmer-ring* [intro]: cring (*lucas-lehmer-ring*)

proof

```

have ∃ aa ba. lucas-lehmer-add' (aa, ba) (a, b) = (0, 0) ∧
  lucas-lehmer-add' (a, b) (aa, ba) = (0, 0) for a b

```

```

by (rule exI[of - -a], rule exI[of - -b]) (auto simp: lucas-lehmer-add'-def)

```

```

thus carrier (add-monoid lucas-lehmer-ring) ⊆ Units (add-monoid lucas-lehmer-ring)

```

```

by (auto simp: Units-def lucas-lehmer-ring-def)

```

```

qed (auto simp: lucas-lehmer-ring-def lucas-lehmer-add'-def lucas-lehmer-mult'-def
  algebra-simps)

```

2.4 The ring $(\mathbb{Z}/m\mathbb{Z})[\sqrt{3}]$

We shall also need the ring $(\mathbb{Z}/m\mathbb{Z})[\sqrt{3}]$, which is obtained from $\mathbb{Z}[\sqrt{3}]$ by reducing each component separately modulo m . This essentially identifies any two points that are a multiple of m apart and then all those that are a multiple of $m\sqrt{3}$ apart.

definition *lucas-lehmer-mult* :: nat ⇒ nat × nat ⇒ nat × nat ⇒ nat × nat **where**

```

lucas-lehmer-mult m = (λ(a,b) (c,d). ((a * c + 3 * b * d) mod m, (a * d + b *
c) mod m))

```

definition *lucas-lehmer-add* :: nat ⇒ nat × nat ⇒ nat × nat ⇒ nat × nat **where**

```

lucas-lehmer-add m = (λ(a,b) (c,d). ((a + c) mod m, (b + d) mod m))

```

definition *lucas-lehmer-ring-mod* :: nat ⇒ (nat × nat) ring **where**

```

lucas-lehmer-ring-mod m =
  (|carrier = {..m} × {..m},
   monoid.mult = lucas-lehmer-mult m,
   one = (1, 0),
   ring.zero = (0, 0),
   add = lucas-lehmer-add m)

```

lemma *lucas-lehmer-add-in-carrier*: $m > 0 \implies \text{lucas-lehmer-add } m \ x \ y \in \{..*m\} \times \{..*m\}**$

```

by (auto simp: lucas-lehmer-add-def split: prod.splits)

```

lemma *lucas-lehmer-mult-in-carrier*: $m > 0 \implies \text{lucas-lehmer-mult } m \ x \ y \in \{..<m\} \times \{..<m\}$

by (*auto simp: lucas-lehmer-mult-def split: prod.splits*)

lemma *lucas-lehmer-add-cong*:

[*fst (lucas-lehmer-add m x y) = fst x + fst y (mod m)*]

[*snd (lucas-lehmer-add m x y) = snd x + snd y (mod m)*]

by (*simp-all add: lucas-lehmer-add-def cong-def case-prod-unfold*)

lemma *lucas-lehmer-mult-cong*:

[*fst (lucas-lehmer-mult m x y) = fst x * fst y + 3 * snd x * snd y (mod m)*]

[*snd (lucas-lehmer-mult m x y) = fst x * snd y + snd x * fst y (mod m)*]

by (*simp-all add: lucas-lehmer-mult-def cong-def case-prod-unfold*)

lemma *lucas-lehmer-add-neutral* [*simp*]:

assumes *fst x < m snd x < m*

shows *lucas-lehmer-add m (0, 0) x = x*

and *lucas-lehmer-add m x (0, 0) = x*

using *assms* **by** (*auto simp: lucas-lehmer-add-def case-prod-unfold*)

lemma *lucas-lehmer-mult-neutral* [*simp*]:

assumes *fst x < m snd x < m*

shows *lucas-lehmer-mult m (Suc 0, 0) x = x*

and *lucas-lehmer-mult m x (Suc 0, 0) = x*

using *assms* **by** (*auto simp: lucas-lehmer-mult-def case-prod-unfold*)

lemma *lucas-lehmer-add-commute*: *lucas-lehmer-add m x y = lucas-lehmer-add m y x*

by (*simp add: lucas-lehmer-add-def algebra-simps case-prod-unfold*)

lemma *lucas-lehmer-mult-commute*: *lucas-lehmer-mult m x y = lucas-lehmer-mult m y x*

by (*simp add: lucas-lehmer-mult-def algebra-simps case-prod-unfold*)

lemma *lucas-lehmer-add-assoc*:

assumes *m: m > 0*

shows *lucas-lehmer-add m x (lucas-lehmer-add m y z) =*

lucas-lehmer-add m (lucas-lehmer-add m x y) z

proof (*rule prod-eqI*)

let *?add = lucas-lehmer-add m*

have [*fst (?add x (?add y z)) = fst x + (fst y + fst z) (mod m)*]

by (*rule lucas-lehmer-add-cong[THEN cong-trans] cong-add cong-mult cong-refl*)+

also have *fst x + (fst y + fst z) = (fst x + fst y) + fst z*

by (*simp add: add-ac*)

also have [*... = fst (?add (?add x y) z) (mod m)*]

by (*rule cong-sym, (rule lucas-lehmer-add-cong[THEN cong-trans] cong-add cong-mult cong-refl)*)+

finally show *fst (?add x (?add y z)) = fst (?add (?add x y) z)*

by (*rule cong-less-modulus-unique-nat*)

(use m in ⟨auto simp: lucas-lehmer-add-def case-prod-unfold⟩)

have $[snd (?add x (?add y z)) = snd x + (snd y + snd z)] \pmod m$
by (rule lucas-lehmer-add-cong[THEN cong-trans] cong-add cong-mult cong-refl)+
also have $snd x + (snd y + snd z) = (snd x + snd y) + snd z$
by (simp add: add-ac)
also have $[... = snd (?add (?add x y) z)] \pmod m$
by (rule cong-sym, (rule lucas-lehmer-add-cong[THEN cong-trans] cong-add cong-mult cong-refl)+)
finally show $snd (?add x (?add y z)) = snd (?add (?add x y) z)$
by (rule cong-less-modulus-unique-nat)
(use m in ⟨auto simp: lucas-lehmer-add-def case-prod-unfold⟩)

qed

lemma lucas-lehmer-mult-assoc:
assumes $m: m > 0$
shows $lucas-lehmer-mult\ m\ x\ (lucas-lehmer-mult\ m\ y\ z) = lucas-lehmer-mult\ m\ (lucas-lehmer-mult\ m\ x\ y)\ z$

proof (rule prod-eqI)
let $?mul = lucas-lehmer-mult\ m$
have $[fst (?mul\ x\ (?mul\ y\ z)) = fst\ x * (fst\ y * fst\ z + 3 * snd\ y * snd\ z) + 3 * snd\ x * (fst\ y * snd\ z + snd\ y * fst\ z)] \pmod m$
by (rule lucas-lehmer-mult-cong[THEN cong-trans] cong-add cong-mult cong-refl)+
also have $fst\ x * (fst\ y * fst\ z + 3 * snd\ y * snd\ z) + 3 * snd\ x * (fst\ y * snd\ z + snd\ y * fst\ z) = (fst\ x * fst\ y + 3 * snd\ x * snd\ y) * fst\ z + 3 * (fst\ x * snd\ y + snd\ x * fst\ y) * snd\ z$
by (simp add: algebra-simps)
also have $[... = fst (?mul\ (?mul\ x\ y)\ z)] \pmod m$
by (rule cong-sym, (rule lucas-lehmer-mult-cong[THEN cong-trans] cong-add cong-mult cong-refl)+)
finally show $fst (?mul\ x\ (?mul\ y\ z)) = fst (?mul\ (?mul\ x\ y)\ z)$
by (rule cong-less-modulus-unique-nat)
(use m in ⟨auto simp: lucas-lehmer-mult-def case-prod-unfold⟩)

have $[snd (?mul\ x\ (?mul\ y\ z)) = fst\ x * (fst\ y * snd\ z + snd\ y * fst\ z) + snd\ x * (fst\ y * fst\ z + 3 * snd\ y * snd\ z)] \pmod m$
by (rule lucas-lehmer-mult-cong[THEN cong-trans] cong-add cong-mult cong-refl)+
also have $fst\ x * (fst\ y * snd\ z + snd\ y * fst\ z) + snd\ x * (fst\ y * fst\ z + 3 * snd\ y * snd\ z) = (fst\ x * fst\ y + 3 * snd\ x * snd\ y) * snd\ z + (fst\ x * snd\ y + snd\ x * fst\ y) * fst\ z$
by (simp add: algebra-simps)
also have $[... = snd (?mul\ (?mul\ x\ y)\ z)] \pmod m$
by (rule cong-sym, (rule lucas-lehmer-mult-cong[THEN cong-trans] cong-add cong-mult cong-refl)+)
finally show $snd (?mul\ x\ (?mul\ y\ z)) = snd (?mul\ (?mul\ x\ y)\ z)$
by (rule cong-less-modulus-unique-nat)
(use m in ⟨auto simp: lucas-lehmer-mult-def case-prod-unfold⟩)

qed

lemma *lucas-lehmer-distrib-right*:

assumes $m: m > 1$

shows $\text{lucas-lehmer-mult } m (\text{lucas-lehmer-add } m x y) z =$
 $\text{lucas-lehmer-add } m (\text{lucas-lehmer-mult } m x z) (\text{lucas-lehmer-mult } m y z)$

proof (rule *prod-eqI*)

let $?mul = \text{lucas-lehmer-mult } m$ **and** $?add = \text{lucas-lehmer-add } m$

have $[\text{fst } (?mul (?add x y) z) = (\text{fst } x + \text{fst } y) * \text{fst } z + 3 * (\text{snd } x + \text{snd } y) * \text{snd } z] \text{ (mod } m)$

by (rule *lucas-lehmer-mult-cong*[*THEN cong-trans*] *lucas-lehmer-add-cong*[*THEN cong-trans*])

cong-add cong-mult cong-refl)+

also have $(\text{fst } x + \text{fst } y) * \text{fst } z + 3 * (\text{snd } x + \text{snd } y) * \text{snd } z =$
 $(\text{fst } x * \text{fst } z + 3 * \text{snd } x * \text{snd } z) + (\text{fst } y * \text{fst } z + 3 * \text{snd } y * \text{snd } z)$

by (*simp add: algebra-simps*)

also have $[\dots = \text{fst } (?add (?mul x z) (?mul y z))] \text{ (mod } m)$

by (rule *cong-sym*, (rule *lucas-lehmer-mult-cong*[*THEN cong-trans*])

lucas-lehmer-add-cong[*THEN cong-trans*] *cong-add cong-mult cong-refl*)+)

finally show $\text{fst } (?mul (?add x y) z) = \text{fst } (?add (?mul x z) (?mul y z))$

by (rule *cong-less-modulus-unique-nat*)

(*use m in <auto simp: lucas-lehmer-add-def lucas-lehmer-mult-def case-prod-unfold>*)

have $[\text{snd } (?mul (?add x y) z) = (\text{fst } x + \text{fst } y) * \text{snd } z + (\text{snd } x + \text{snd } y) * \text{fst } z] \text{ (mod } m)$

by (rule *lucas-lehmer-mult-cong*[*THEN cong-trans*] *lucas-lehmer-add-cong*[*THEN cong-trans*])

cong-add cong-mult cong-refl)+

also have $(\text{fst } x + \text{fst } y) * \text{snd } z + (\text{snd } x + \text{snd } y) * \text{fst } z =$
 $(\text{fst } x * \text{snd } z + \text{snd } x * \text{fst } z) + (\text{fst } y * \text{snd } z + \text{snd } y * \text{fst } z)$

by (*simp add: algebra-simps*)

also have $[\dots = \text{snd } (?add (?mul x z) (?mul y z))] \text{ (mod } m)$

by (rule *cong-sym*, (rule *lucas-lehmer-mult-cong*[*THEN cong-trans*])

lucas-lehmer-add-cong[*THEN cong-trans*] *cong-add cong-mult cong-refl*)+)

finally show $\text{snd } (?mul (?add x y) z) = \text{snd } (?add (?mul x z) (?mul y z))$

by (rule *cong-less-modulus-unique-nat*)

(*use m in <auto simp: lucas-lehmer-add-def lucas-lehmer-mult-def case-prod-unfold>*)

qed

lemma *lucas-lehmer-distrib-left*:

assumes $m > 1$

shows $\text{lucas-lehmer-mult } m z (\text{lucas-lehmer-add } m x y) =$
 $\text{lucas-lehmer-add } m (\text{lucas-lehmer-mult } m z x) (\text{lucas-lehmer-mult } m z y)$

using *lucas-lehmer-distrib-right*[*of m x y z*] *assms*

by (*simp add: lucas-lehmer-mult-commute*)

lemma *cring-lucas-lehmer-ring-mod* [*intro*]:

assumes $m > 1$

shows $\text{cring } (\text{lucas-lehmer-ring-mod } m)$

proof *unfold-locales*

let $?neg = \lambda x. \text{if } x = 0 \text{ then } 0 \text{ else } m - x$
have $\exists x \in \text{carrier } (\text{lucas-lehmer-ring-mod } m).$
 $x \oplus_{\text{lucas-lehmer-ring-mod } m} (a, b) = \mathbf{0}_{\text{lucas-lehmer-ring-mod } m} \wedge$
 $(a, b) \oplus_{\text{lucas-lehmer-ring-mod } m} x = \mathbf{0}_{\text{lucas-lehmer-ring-mod } m}$
if $(a, b) \in \text{carrier } (\text{lucas-lehmer-ring-mod } m)$ **for** a b
using *that assms*
by $(\text{intro } \text{bexI}[\text{of } - \text{ (?neg } a, \text{ ?neg } b)])$
 $(\text{auto simp: lucas-lehmer-ring-mod-def lucas-lehmer-add-def})$
thus $\text{carrier } (\text{add-monoid } (\text{lucas-lehmer-ring-mod } m)) \subseteq \text{Units } (\text{add-monoid } (\text{lucas-lehmer-ring-mod } m))$
by $(\text{auto simp: Units-def})$
qed $(\text{insert assms,}$
 $\text{auto simp: lucas-lehmer-ring-mod-def algebra-simps lucas-lehmer-mult-assoc}$
 $\text{lucas-lehmer-add-assoc lucas-lehmer-distrib-right lucas-lehmer-distrib-left}$
 $\text{intro: lucas-lehmer-mult-in-carrier lucas-lehmer-add-in-carrier}$
 $\text{lucas-lehmer-add-commute lucas-lehmer-mult-commute})$

Since 0 is clearly not a unit in the ring and its carrier has size m^2 , the number of units is strictly less than m^2 .

lemma *card-lucas-lehmer-Units:*

assumes $m > 1$
shows $\text{card } (\text{Units } (\text{lucas-lehmer-ring-mod } m)) < m^2$
proof –
interpret $\text{cring lucas-lehmer-ring-mod } m$
using *assms* **by** *auto*
have $m^2 > 0$
using *assms* **by** *auto*
from *assms* **have** $\text{card } (\text{Units } (\text{lucas-lehmer-ring-mod } m)) \leq \text{card } (\{..<m\} \times \{..<m\} - \{(0, 0)\})$
by (intro card-mono) $(\text{auto simp: Units-def lucas-lehmer-ring-mod-def lucas-lehmer-mult-def})$
also have $\dots = m^2 - 1$
using *assms* **by** $(\text{subst card-Diff-subset})$ $(\text{auto simp: power2-eq-square})$
finally show $?thesis$ **using** $\langle m^2 > 0 \rangle$ **by** *linarith*
qed

Consider now the case of a prime modulus m : Since $\mathbb{Z}/m\mathbb{Z} = \text{GF}(m)$ is a field, any element of $\mathbb{Z}/m\mathbb{Z}$ is a unit in $(\mathbb{Z}/m\mathbb{Z})[\sqrt{3}]$.

lemma *int-in-Units-lucas-lehmer-ring-mod:*

assumes *prime* p
assumes $x > 0$ $x < p$
shows $(x, 0) \in \text{Units } (\text{lucas-lehmer-ring-mod } p)$
proof –
define R **where** $R = \text{lucas-lehmer-ring-mod } p$
have $[x * (x^{p-2} \text{ mod } p) = x * x^{p-2}] \text{ (mod } p)$
by (intro cong-mult) $(\text{auto simp: cong-def})$
also have $x * x^{p-2} = x^{p-1}$
by $(\text{simp add: mult-ac})$
also have $\text{Suc } (p - 2) = p - 1$

using *prime-gt-1-nat*[of p] *assms* **by** *simp*
also have $[x \wedge (p - 1) = 1] \pmod{p}$
using *assms* **by** (*intro fermat-theorem*) (*auto dest: dvd-imp-le*)
finally have $(x, 0) \otimes_R (x \wedge (p - 2) \pmod{p}, 0) = \mathbf{1}_R$
 $(x \wedge (p - 2) \pmod{p}, 0) \otimes_R (x, 0) = \mathbf{1}_R$
 $(x \wedge (p - 2) \pmod{p}, 0) \in \text{carrier } R$
using *prime-gt-1-nat*[of p] *assms*
by (*auto simp: lucas-lehmer-mult-def cong-def lucas-lehmer-ring-mod-def mult-ac R-def*)
moreover from *assms* **have** $(x, 0) \in \text{carrier } R$
by (*auto simp: R-def lucas-lehmer-ring-mod-def*)
ultimately show *?thesis* **using** *assms*
by (*auto simp: Units-def R-def*)
qed

2.5 $\mathbb{Z}[\sqrt{3}]$ as a subring of \mathbb{R}

We now define the homomorphism from $\mathbb{Z}[\sqrt{3}]$ into the reals:

definition *lucas-lehmer-to-real* :: *int* \times *int* \Rightarrow *real* **where**
lucas-lehmer-to-real = $(\lambda(a,b). \text{real-of-int } a + \text{real-of-int } b * \text{sqrt } 3)$

context
begin

interpretation *cring lucas-lehmer-ring* ..

lemma *minus-lucas-lehmer-ring*: $\ominus_{\text{lucas-lehmer-ring}} x = (\text{case } x \text{ of } (a, b) \Rightarrow (-a, -b))$

by (*rule sym, rule sum-zero-eq-neg*)
(*auto simp: case-prod-unfold lucas-lehmer-ring-def lucas-lehmer-add'-def*)

lemma *lucas-lehmer-to-real-simps1*:

lucas-lehmer-to-real (a, b) = *of-int* a + *of-int* b * *sqrt* 3
lucas-lehmer-to-real (x $\oplus_{\text{lucas-lehmer-ring}}$ y) =
lucas-lehmer-to-real x + *lucas-lehmer-to-real* y
lucas-lehmer-to-real (x $\otimes_{\text{lucas-lehmer-ring}}$ y) =
lucas-lehmer-to-real x * *lucas-lehmer-to-real* y
lucas-lehmer-to-real ($\ominus_{\text{lucas-lehmer-ring}}$ x) = -*lucas-lehmer-to-real* x
lucas-lehmer-to-real ($\mathbf{0}_{\text{lucas-lehmer-ring}}$) = 0
lucas-lehmer-to-real ($\mathbf{1}_{\text{lucas-lehmer-ring}}$) = 1

using *minus-lucas-lehmer-ring*

by (*simp-all add: lucas-lehmer-to-real-def lucas-lehmer-add'-def lucas-lehmer-mult'-def case-prod-unfold algebra-simps lucas-lehmer-ring-def*)

lemma *lucas-lehmer-to-add-pow-nat*:

lucas-lehmer-to-real ([n] $\cdot_{\text{lucas-lehmer-ring}}$ x) = *of-nat* n * *lucas-lehmer-to-real* x
by (*induction n*) (*auto simp: lucas-lehmer-to-real-simps1 algebra-simps*)

lemma *lucas-lehmer-to-add-pow-int*:
 $lucas\text{-lehmer-to-real } ([n] \cdot lucas\text{-lehmer-ring } x) = of\text{-int } n * lucas\text{-lehmer-to-real } x$

proof (*cases* $n \geq 0$)
case *True*
hence $lucas\text{-lehmer-to-real } ([n] \cdot lucas\text{-lehmer-ring } x) =$
 $lucas\text{-lehmer-to-real } ([int (nat n)] \cdot lucas\text{-lehmer-ring } x)$
by *simp*
also have $\dots = lucas\text{-lehmer-to-real } ([nat n] \cdot lucas\text{-lehmer-ring } x)$
by (*simp add: add-pow-int-ge*)
also have $\dots = of\text{-int } n * lucas\text{-lehmer-to-real } x$ **using** *True*
by (*simp add: lucas-lehmer-to-add-pow-nat algebra-simps*)
finally show *?thesis* .

next
case *False*
hence $lucas\text{-lehmer-to-real } ([n] \cdot lucas\text{-lehmer-ring } x) =$
 $lucas\text{-lehmer-to-real } (add\text{-pow } lucas\text{-lehmer-ring } (-int (nat (-n)))) x$
by *simp*
also have $add\text{-pow } lucas\text{-lehmer-ring } (-int (nat (-n))) x =$
 $\ominus_{lucas\text{-lehmer-ring}} (add\text{-pow } lucas\text{-lehmer-ring } (nat (-n))) x$
using *False* **by** (*subst add.int-pow-neg-int*) (*auto simp: lucas-lehmer-ring-def*)
also have $lucas\text{-lehmer-to-real } \dots = of\text{-int } n * lucas\text{-lehmer-to-real } x$ **using** *False*
by (*simp add: lucas-lehmer-to-add-pow-nat lucas-lehmer-to-real-simps1 algebra-simps*)
finally show *?thesis* .

qed

lemma *lucas-lehmer-to-real-power*:
 $lucas\text{-lehmer-to-real } (x [\wedge]_{lucas\text{-lehmer-ring}} (n :: nat)) = lucas\text{-lehmer-to-real } x \wedge^n$

by (*induction n*) (*auto simp: lucas-lehmer-to-real-simps1*)

lemmas *lucas-lehmer-to-real-simps =*
 $lucas\text{-lehmer-to-real-simps1}$ *lucas-lehmer-to-real-power*
 $lucas\text{-lehmer-to-add-pow-nat}$ *lucas-lehmer-to-add-pow-int*

end

lemma *lucas-lehmer-to-real-inj*: *inj lucas-lehmer-to-real*

proof (*rule injI, clarify*)
fix $a b c d :: int$
assume $eq: lucas\text{-lehmer-to-real } (a, b) = lucas\text{-lehmer-to-real } (c, d)$
have $b = d$
proof (*rule ccontr*)
assume $b \neq d$
hence $\sqrt{3} = (c - a) / (b - d)$
using eq **by** (*simp add: lucas-lehmer-to-real-def field-simps*)
also have $\dots \in \mathbb{Q}$ **by** *auto*
finally have $\sqrt{3} \in \mathbb{Q}$.
moreover have $\sqrt{3} \notin \mathbb{Q}$

using *is-nth-power-prime-power-nat-iff*[of 3 2 1] *irrat-sqrt-nonsquare*[of 3] by
auto
 ultimately show *False* by *contradiction*
 qed
 moreover from *this* and *eq* have $a = c$
 by (*auto simp: lucas-lehmer-to-real-def*)
 ultimately show $a = c \wedge b = d$ by *blast*
 qed

2.6 The canonical homomorphism $\mathbb{Z}[\sqrt{3}] \rightarrow (\mathbb{Z}/m\mathbb{Z})[\sqrt{3}]$

Next, we show that reduction modulo m is indeed a homomorphism.

definition *lucas-lehmer-hom* :: $\text{nat} \Rightarrow (\text{int} \times \text{int}) \Rightarrow (\text{nat} \times \text{nat})$ where
lucas-lehmer-hom $m = (\lambda(x,y). (\text{nat } (x \bmod m), \text{nat } (y \bmod m)))$

lemma *lucas-lehmer-hom-cong*:

$[\text{fst } x = \text{fst } y] (\bmod \text{int } m) \Longrightarrow [\text{snd } x = \text{snd } y] (\bmod \text{int } m) \Longrightarrow$
 $\text{lucas-lehmer-hom } m \ x = \text{lucas-lehmer-hom } m \ y$
 by (*auto simp: lucas-lehmer-hom-def cong-def case-prod-unfold*)

lemma *lucas-lehmer-hom-cong'*:

$[a = b] (\bmod \text{int } m) \Longrightarrow [c = d] (\bmod \text{int } m) \Longrightarrow$
 $\text{lucas-lehmer-hom } m \ (a, c) = \text{lucas-lehmer-hom } m \ (b, d)$
 by (*auto simp: lucas-lehmer-hom-def cong-def*)

context

fixes $m :: \text{nat}$
 assumes $m: m > 1$

begin

lemma *lucas-lehmer-hom-in-carrier*: $\text{lucas-lehmer-hom } m \ x \in \{..<m\} \times \{..<m\}$
 using m *nat-less-iff* by (*auto simp: lucas-lehmer-hom-def case-prod-unfold*)

lemma *lucas-lehmer-hom-add*:

$\text{lucas-lehmer-hom } m \ (\text{lucas-lehmer-add}' \ x \ y) =$
 $\text{lucas-lehmer-add } m \ (\text{lucas-lehmer-hom } m \ x) \ (\text{lucas-lehmer-hom } m \ y)$

proof (*rule prod-eqI*)

let $?add1 = \text{lucas-lehmer-add}'$ and $?add2 = \text{lucas-lehmer-add } m$

let $?φ = \text{lucas-lehmer-hom } m$

have $\text{fst } (?φ \ (?add1 \ x \ y)) = \text{nat } ((\text{fst } x + \text{fst } y) \bmod \text{int } m)$

by (*simp add: lucas-lehmer-hom-def lucas-lehmer-add'-def case-prod-unfold*)

also have $(\text{fst } x + \text{fst } y) \bmod \text{int } m = ((\text{fst } x \bmod m) + (\text{fst } y \bmod m)) \bmod \text{int } m$

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

also have $\text{nat } \dots = (\text{nat } (\text{fst } x \bmod \text{int } m) + \text{nat } (\text{fst } y \bmod \text{int } m)) \bmod m$

using m *nat-add-distrib nat-mod-distrib* by *auto*

also have $\dots = \text{fst } (?add2 \ (?φ \ x) \ (?φ \ y))$

by (*auto simp: lucas-lehmer-hom-def lucas-lehmer-add-def case-prod-unfold*)

finally show $\text{fst } (?φ \ (?add1 \ x \ y)) = \text{fst } (?add2 \ (?φ \ x) \ (?φ \ y))$.

have $\text{snd } (? \varphi (? \text{add1 } x y)) = \text{nat } ((\text{snd } x + \text{snd } y) \text{ mod } \text{int } m)$
by (*simp add: lucas-lehmer-hom-def lucas-lehmer-add'-def case-prod-unfold*)
also have $(\text{snd } x + \text{snd } y) \text{ mod } \text{int } m = ((\text{snd } x \text{ mod } m) + (\text{snd } y \text{ mod } m)) \text{ mod } \text{int } m$
by (*simp add: mod-add-eq*)
also have $\text{nat } \dots = (\text{nat } (\text{snd } x \text{ mod } \text{int } m) + \text{nat } (\text{snd } y \text{ mod } \text{int } m)) \text{ mod } m$
using *m nat-add-distrib nat-mod-distrib by auto*
also have $\dots = \text{snd } (? \text{add2 } (? \varphi x) (? \varphi y))$
by (*auto simp: lucas-lehmer-hom-def lucas-lehmer-add-def case-prod-unfold*)
finally show $\text{snd } (? \varphi (? \text{add1 } x y)) = \text{snd } (? \text{add2 } (? \varphi x) (? \varphi y))$.
qed

lemma *lucas-lehmer-hom-mult:*

$\text{lucas-lehmer-hom } m (\text{lucas-lehmer-mult}' x y) =$
 $\text{lucas-lehmer-mult } m (\text{lucas-lehmer-hom } m x) (\text{lucas-lehmer-hom } m y)$

proof (*rule prod-eqI*)

let $? \text{mul1} = \text{lucas-lehmer-mult}'$ **and** $? \text{mul2} = \text{lucas-lehmer-mult } m$

let $? \varphi = \text{lucas-lehmer-hom } m$

have $\text{fst } (? \varphi (? \text{mul1 } x y)) = \text{nat } ((\text{fst } x * \text{fst } y + 3 * \text{snd } x * \text{snd } y) \text{ mod } \text{int } m)$

by (*simp add: lucas-lehmer-hom-def lucas-lehmer-mult'-def case-prod-unfold*)

also have $(\text{fst } x * \text{fst } y + 3 * \text{snd } x * \text{snd } y) \text{ mod } \text{int } m =$

$((\text{fst } x \text{ mod } \text{int } m) * (\text{fst } y \text{ mod } \text{int } m) +$
 $3 * (\text{snd } x \text{ mod } \text{int } m) * (\text{snd } y \text{ mod } \text{int } m)) \text{ mod } m$

by (*intro congD cong-mult cong-add cong-refl*) (*auto simp: cong-def*)

also have $\dots = \text{int } (\text{nat } (((\text{fst } x \text{ mod } \text{int } m) * (\text{fst } y \text{ mod } \text{int } m) +$

$3 * (\text{snd } x \text{ mod } \text{int } m) * (\text{snd } y \text{ mod } \text{int } m)) \text{ mod } m)$

using *m by (subst of-nat-nat) auto*

also have $\dots = \text{int } (\text{nat } (\text{fst } x \text{ mod } \text{int } m) * \text{nat } (\text{fst } y \text{ mod } \text{int } m) +$

$3 * (\text{nat } (\text{snd } x \text{ mod } \text{int } m) * \text{nat } (\text{snd } y \text{ mod } \text{int } m)) \text{ mod } m$

using *m by simp*

also have $\text{nat } \dots = (\text{nat } (\text{fst } x \text{ mod } \text{int } m) * \text{nat } (\text{fst } y \text{ mod } \text{int } m) +$

$3 * \text{nat } (\text{snd } x \text{ mod } \text{int } m) * \text{nat } (\text{snd } y \text{ mod } \text{int } m)) \text{ mod } m$

using *m by (metis nat-int zmod-int)*

also have $\dots = \text{fst } (? \text{mul2 } (? \varphi x) (? \varphi y))$

by (*simp add: lucas-lehmer-hom-def lucas-lehmer-mult-def case-prod-unfold*)

finally show $\text{fst } (? \varphi (? \text{mul1 } x y)) = \text{fst } (? \text{mul2 } (? \varphi x) (? \varphi y))$.

have $\text{snd } (? \varphi (? \text{mul1 } x y)) = \text{nat } ((\text{fst } x * \text{snd } y + \text{snd } x * \text{fst } y) \text{ mod } \text{int } m)$

by (*simp add: lucas-lehmer-hom-def lucas-lehmer-mult'-def case-prod-unfold*)

also have $(\text{fst } x * \text{snd } y + \text{snd } x * \text{fst } y) \text{ mod } \text{int } m =$

$((\text{fst } x \text{ mod } \text{int } m) * (\text{snd } y \text{ mod } \text{int } m) +$
 $(\text{snd } x \text{ mod } \text{int } m) * (\text{fst } y \text{ mod } \text{int } m)) \text{ mod } m$

by (*intro congD cong-mult cong-add cong-refl*) (*auto simp: cong-def*)

also have $\dots = \text{int } (\text{nat } (((\text{fst } x \text{ mod } \text{int } m) * (\text{snd } y \text{ mod } \text{int } m) +$

$(\text{snd } x \text{ mod } \text{int } m) * (\text{fst } y \text{ mod } \text{int } m)) \text{ mod } m)$

using *m by (subst of-nat-nat) auto*

also have $\dots = \text{int } (\text{nat } (\text{fst } x \text{ mod } \text{int } m) * \text{nat } (\text{snd } y \text{ mod } \text{int } m) +$

$(\text{nat } (\text{snd } x \text{ mod } \text{int } m) * \text{nat } (\text{fst } y \text{ mod } \text{int } m)) \text{ mod } m$

using m **by** *simp*
also have $\text{nat } \dots = (\text{nat } (\text{fst } x \text{ mod int } m) * \text{nat } (\text{snd } y \text{ mod int } m) +$
 $\text{nat } (\text{snd } x \text{ mod int } m) * \text{nat } (\text{fst } y \text{ mod int } m)) \text{ mod } m$
using m **by** (*metis nat-int zmod-int*)
also have $\dots = \text{snd } (?mul2 (?φ x) (?φ y))$
by (*simp add: lucas-lehmer-hom-def lucas-lehmer-mult-def case-prod-unfold*)
finally show $\text{snd } (?φ (?mul1 x y)) = \text{snd } (?mul2 (?φ x) (?φ y))$.
qed

lemma *lucas-lehmer-hom-1* [*simp*]: *lucas-lehmer-hom* m $(1, 0) = (1, 0)$
using m **by** (*simp add: lucas-lehmer-hom-def*)

lemma *ring-hom-lucas-lehmer-hom*:

lucas-lehmer-hom $m \in \text{ring-hom } \text{lucas-lehmer-ring } (\text{lucas-lehmer-ring-mod } m)$

proof –

interpret R : *cring lucas-lehmer-ring* ..

from m **interpret** S : *cring lucas-lehmer-ring-mod* m ..

show *?thesis*

unfolding *ring-hom-def* **using** *lucas-lehmer-hom-in-carrier* m

by (*auto simp: lucas-lehmer-ring-mod-def lucas-lehmer-hom-add*
lucas-lehmer-ring-def lucas-lehmer-hom-mult)

qed

end

2.7 Correctness of the Lucas–Lehmer test

In this section, we will prove that the Lucas–Lehmer test is both a necessary and sufficient condition for the primality of a Mersenne number of the form $2^p - 1$ for an odd prime p . The proof that shall be given here is rather explicit and heavily draws from the Wikipedia article on the Lucas–Lehmer test [3].

A shorter and more high-level proof of a more general statement can be obtained using more theory on finite fields (in particular the field $\text{GF}(q^2)$) (cf. e.g. Rödseth [2]).

definition *lucas-lehmer-test* **where**

lucas-lehmer-test $p = (p > 2 \wedge$
 $(2 \wedge p - 1) \text{ dvd } \text{gen-lucas-lehmer-sequence } 4 (p - 2))$

We can now prove that any Mersenne number $2^p - 1$ for p prime that passes the Lucas–Lehmer test is prime. We follow the simple argument given by Bruce [1], which is also given on Wikipedia [3].

theorem *lucas-lehmer-sufficient*:

assumes *prime* p *odd* p

assumes $(2 \wedge p - 1) \text{ dvd } \text{gen-lucas-lehmer-sequence } 4 (p - 2)$

shows *prime* $(2 \wedge p - 1 :: \text{nat})$

proof (*rule ccontr*)

```

assume not-prime:  $\neg \text{prime } (2^{\wedge} p - 1 :: \text{nat})$ 
from assms obtain  $k :: \text{int}$  where  $k$ : gen-lucas-lehmer-sequence 4  $(p - 2) = k$ 
*  $(2^{\wedge} p - 1)$ 
  by (elim dvdE) (auto simp: mult-ac)
from assms have  $p > 2$ 
  using odd-prime-gt-2-nat by blast
from  $\langle p > 2 \rangle$  have  $2^{\wedge} p \geq (2^{\wedge} 3 :: \text{nat})$  by (intro power-increasing) auto
hence  $2^{\wedge} p \geq (8 :: \text{nat})$  by simp

define  $q :: \text{nat}$  where  $q = \text{Min } (\text{prime-factors } (2^{\wedge} p - 1))$ 
have  $q \in \text{prime-factors } (2^{\wedge} p - 1)$  using  $\langle 2^{\wedge} p \geq 8 \rangle$ 
  unfolding q-def by (intro Min-in) (auto simp: prime-factorization-empty-iff)
hence  $q$ : prime  $q$   $q$  dvd  $(2^{\wedge} p - 1 :: \text{nat})$ 
  by (auto simp: in-prime-factors-iff)
have q-minimal:  $q \leq q'$  if  $q' \in \text{prime-factors } (2^{\wedge} p - 1)$  for  $q'$ 
  unfolding q-def by (rule Min-le) (use that in auto)

have  $2^{\wedge} p - 1 \geq q^{\wedge} 2$ 
proof -
  from  $q$  obtain  $k$  where  $k$ :  $2^{\wedge} p - 1 = q * k$  by auto
  have prime-factorization  $(2^{\wedge} p - 1 :: \text{nat}) \neq \{\#q\#$ 
  proof
    assume *: prime-factorization  $(2^{\wedge} p - 1 :: \text{nat}) = \{\#q\#$ 
    have  $2^{\wedge} p - 1 = \text{prod-mset } (\text{prime-factorization } (2^{\wedge} p - 1 :: \text{nat}))$ 
      using  $\langle 2^{\wedge} p \geq 8 \rangle$  by (subst prod-mset-prime-factorization-nat) auto
    also have  $\dots = q$  by (subst *) auto
    finally show False using not-prime  $q$  by simp
  qed
  hence prime-factorization  $k \neq \{\#\}$  using  $q$   $k$   $\langle 2^{\wedge} p \geq 8 \rangle$ 
    by (subst (asm) k, subst (asm) prime-factorization-mult)
    (auto intro!: Nat.grOI simp: prime-factorization-prime)
  hence  $k \neq 1$  by (auto simp: prime-factorization-empty-iff)
  then obtain  $q'$  where  $q'$ : prime  $q'$   $q'$  dvd  $k$ 
    using prime-factor-nat by blast
  from  $q'$   $k$   $\langle 2^{\wedge} p \geq 8 \rangle$  have  $q \leq q'$ 
    by (intro q-minimal) (auto simp: in-prime-factors-iff intro!: Nat.grOI)
  hence  $q^{\wedge} 2 \leq q * q'$ 
    unfolding power2-eq-square by (intro mult-mono) auto
  also have  $q * q' \leq 2^{\wedge} p - 1$ 
    using  $q$   $q'$   $k$   $\langle 2^{\wedge} p \geq 8 \rangle$  by (intro dvd-imp-le) (auto intro!: Nat.grOI)
  finally show  $2^{\wedge} p - 1 \geq q^{\wedge} 2$  .
qed

have  $q \neq 2$  using  $q$   $\langle p > 2 \rangle$  by auto
moreover from  $q$  have  $q \neq 0$   $q \neq 1$  by auto
ultimately have  $q > 2$  by auto

write lucas-lehmer-ring  $(\langle R \rangle)$ 
define  $S$  where  $S = \text{lucas-lehmer-ring-mod } q$ 

```

```

define  $S'$  where  $S' = \text{units-of } S$ 
define  $\varphi$  where  $\varphi = \text{lucas-lehmer-hom } q$ 

interpret  $R$ : cring  $R$  ..
interpret  $S$ : cring  $S$ 
  unfolding  $S$ -def by (rule cring-lucas-lehmer-ring-mod) (use  $\langle q \rangle$   $\langle 2 \rangle$  in auto)
interpret  $S'$ : comm-group  $S'$ 
  unfolding  $S'$ -def by (rule S.units-comm-group)
have  $\varphi \in \text{ring-hom } R S$ 
  unfolding  $\varphi$ -def  $S$ -def by (rule ring-hom-lucas-lehmer-hom) (use  $\langle q \rangle$   $\langle 2 \rangle$  in
auto)
interpret  $\varphi$ : ring-hom-cring  $R S \varphi$ 
  by standard fact

have  $(2 + \text{sqrt } 3) ^{(2 ^{(p-2)})} + (2 - \text{sqrt } 3) ^{(2 ^{(p-2)})} =$ 
  real-of-int (gen-lucas-lehmer-sequence 4 (p-2))
  unfolding gen-lucas-lehmer-sequence-4-closed-form1 ..
also have ... = real-of-int k * (2 ^ p - 1)
  by (simp add: k)
finally have *:  $(2 + \text{sqrt } 3) ^{(2 ^{(p-2)})} =$ 
  real-of-int k * (2 ^ p - 1) - (2 - \text{sqrt } 3) ^{(2 ^{(p-2)})}
  by (simp add: algebra-simps)
have  $((2 + \text{sqrt } 3) ^{(2 ^{(p-2)})}) ^ 2 =$ 
  real-of-int k * (2 ^ p - 1) * (2 + \text{sqrt } 3) ^{(2 ^{(p-2)})} -
  (2 - \text{sqrt } 3) ^{(2 ^{(p-2)})} * (2 + \text{sqrt } 3) ^{(2 ^{(p-2)})}
  unfolding power2-eq-square by (subst *) (simp add: algebra-simps)
also have  $((2 + \text{sqrt } 3) ^{(2 ^{(p-2)})}) ^ 2 = (2 + \text{sqrt } 3) ^{(2 * 2 ^{(p-2)})}$ 
  by (simp flip: power-mult add: mult-ac)
also have  $2 * 2 ^{(p-2)} = 2 ^{(\text{Suc } (p-2))}$ 
  by simp
also from  $\langle p \rangle$   $\langle 2 \rangle$  have  $\text{Suc } (p-2) = p-1$ 
  by linarith
also have  $(2 - \text{sqrt } 3) ^{(2 ^{(p-2)})} * (2 + \text{sqrt } 3) ^{(2 ^{(p-2)})} = 1$ 
  by (subst power-mult-distrib [symmetric]) (auto simp: algebra-simps)
finally have  $(2 + \text{sqrt } 3) ^{(2 ^{(p-1)})} =$ 
  real-of-int k * (2 ^ p - 1) * (2 + \text{sqrt } 3) ^{(2 ^{(p-2)})} - 1 .

also have  $(2 + \text{sqrt } 3) ^{(2 ^{(p-1)})} =$ 
  lucas-lehmer-to-real ((2, 1) [ ]_R (2 ^{(p-1)} :: nat))
  by (simp add: lucas-lehmer-to-real-simps)
also have real-of-int k * (2 ^ p - 1) * (2 + \text{sqrt } 3) ^{(2 ^{(p-2)})} - 1 =
  lucas-lehmer-to-real ((k * (2 ^ p - 1), 0) \otimes_R
  (2, 1) [ ]_R (2 ^{(p-2)} :: nat) \oplus_R \ominus_R \mathbf{1}_R)
  by (simp add: lucas-lehmer-to-real-simps)
finally have  $((2, 1) [ ]_R (2 ^{(p-1)} :: nat)) =$ 
   $((k * (2 ^ p - 1), 0) \otimes_R (2, 1) [ ]_R (2 ^{(p-2)} :: nat) \oplus_R \ominus_R \mathbf{1}_R)$ 
  by (rule injD[OF lucas-lehmer-to-real-inj])

```

hence $\varphi ((2, 1) [\frown]_R (2 \wedge (p - 1) :: \text{nat})) =$
 $\varphi ((k * (2 \wedge p - 1), 0) \otimes_R (2, 1) [\frown]_R (2 \wedge (p - 2) :: \text{nat}) \oplus_R \ominus_R \mathbf{1}_R)$
by (*simp only*:)
also have $\varphi ((2, 1) [\frown]_R (2 \wedge (p - 1) :: \text{nat})) = \varphi (2, 1) [\frown]_S (2 \wedge (p - 1) :: \text{nat})$
by *simp*
also {
have $\text{int } q \text{ dvd int } (2 \wedge p - 1)$
by (*subst int-dvd-int-iff*) (*use q in auto*)
also have $\text{int } (2 \wedge p - 1) = 2 \wedge p - 1$
by *simp*
finally have $\varphi (k * (2 \wedge p - 1), 0) = \mathbf{0}_S$
by (*simp add: φ -def lucas-lehmer-hom-def S-def lucas-lehmer-ring-mod-def*)
}

hence $\varphi ((k * (2 \wedge p - 1), 0) \otimes_R (2, 1) [\frown]_R (2 \wedge (p - 2) :: \text{nat}) \oplus_R \ominus_R \mathbf{1}_R)$
 $=$
 $\ominus_S \mathbf{1}_S$
by *simp*
finally have $\text{eq: } \varphi (2, 1) [\frown]_S (2 \wedge (p - 1) :: \text{nat}) = \ominus_S \mathbf{1}_S$.

have $\varphi (2, 1) [\frown]_S (2 \wedge p :: \text{nat}) = \varphi (2, 1) [\frown]_S (2 \wedge (p - 1) * 2 :: \text{nat})$
using $\langle p > 2 \rangle$ **by** (*cases p*) (*auto simp: mult-ac*)
also have $\dots = (\varphi (2, 1) [\frown]_S (2 \wedge (p - 1) :: \text{nat})) [\frown]_S (2 :: \text{nat})$
by (*subst S.nat-pow-pow*) *auto*
also have $\dots = \mathbf{1}_S$
by (*subst eq*) (*auto simp: numeral-2-eq-2 S.l-minus*)
finally have $\text{eq': } \varphi (2, 1) [\frown]_S (2 \wedge p :: \text{nat}) = \mathbf{1}_S$.

from eq' **have** *unit*: $\varphi (2, 1) \in \text{Units } S$
by (*rule S.pow-nat-eq-1-imp-unit*) *auto*

have *neg-one-not-one*: $\ominus_S \mathbf{1}_S \neq \mathbf{1}_S$
proof
assume $*$: $\ominus_S \mathbf{1}_S = \mathbf{1}_S$
have $(\ominus_S \mathbf{1}_S) \oplus_S \mathbf{1}_S = \mathbf{0}_S$
by (*rule S.l-neg*) *auto*
hence $\mathbf{1}_S \oplus_S \mathbf{1}_S = \mathbf{0}_S$
by (*simp only: **)
thus *False* **using** $\langle q > 2 \rangle$
by (*auto simp: S-def lucas-lehmer-ring-mod-def lucas-lehmer-add-def*)
qed

have *fin*: *finite* (*Units* S)
by (*rule finite-subset[*of* - *carrier* S]*) (*auto simp: Units-def S-def lucas-lehmer-ring-mod-def*)

have $\text{group.ord } S' (\varphi (2, 1)) = 2 \wedge p$
using $\langle p > 2 \rangle$ *eq eq' unit neg-one-not-one*
by (*intro S'.ord-eqI-prime-factors*)
(auto simp: prime-factors-power prime-factorization-prime)

```

      S'-def S.units-of-pow units-of-carrier units-of-one power-diff)
hence  $2^p = \text{group.ord } S' (\varphi (2, 1))$ 
  by simp
also have ... = card (generate S' { $\varphi (2, 1)$ })
  using unit fin
  by (intro S'.generate-pow-card) (auto simp: S'-def units-of-carrier)
also have ...  $\leq$  card (carrier S')
  using fin unit by (intro card-mono S'.generate-incl) (auto simp: S'-def units-of-carrier)
also have ...  $< q^2$ 
  unfolding S'-def S-def using card-lucas-lehmer-Units[of q]  $\langle q > 2 \rangle$ 
  by (auto simp: units-of-carrier)
also note  $\langle q^2 \leq 2^p - 1 \rangle$ 
finally show False by simp
qed

```

Next, we show that any Mersenne prime passes the Lucas–Lehmer test. We again follow the rather explicit proof outlined on Wikipedia [3], which is a simplified (but less general and less abstract) version of the proof by Rödseth [2].

theorem (in *mersenne-prime*) *lucas-lehmer-necessary*:
 $(2^p - 1) \text{ dvd } \text{gen-lucas-lehmer-sequence } 4 (p - 2)$

proof –

```

write lucas-lehmer-ring ( $\langle R \rangle$ )
define S where S = lucas-lehmer-ring-mod M
define S' where S' = units-of S
define  $\varphi$  where  $\varphi = \text{lucas-lehmer-hom } M$ 

```

interpret R: *cring* R ..

interpret S: *cring* S **unfolding** S-def

by (rule *cring-lucas-lehmer-ring-mod*) (use *M-gt-6* in auto)

interpret S': *comm-group* S'

unfolding S'-def by (rule S.units-comm-group)

have $\varphi \in \text{ring-hom } R S$ **unfolding** φ -def S-def

by (rule *ring-hom-lucas-lehmer-hom*) (use *M-gt-6* in auto)

interpret φ : *ring-hom-cring* R S φ

by *standard fact*

have *R-pow-int*: $(n, 0) [\cdot]_R m = (n^m, 0)$ **for** $n :: \text{int}$ **and** $m :: \text{nat}$

by (*induction m*; *simp*; *simp add: lucas-lehmer-ring-def lucas-lehmer-mult'-def*)

have *add-pow* R n $\mathbf{1}_R = (\text{int } n, 0)$ **for** n

by (*induction n*; *simp*; *simp add: lucas-lehmer-ring-def lucas-lehmer-add'-def*)

hence *add-pow* R M $\mathbf{1}_R = (\text{int } M, 0)$

by *simp*

also have $\varphi \dots = \mathbf{0}_S$

by (*simp add: φ -def S-def lucas-lehmer-ring-mod-def lucas-lehmer-hom-def*)

finally have *add-pow* S M $\mathbf{1}_S = \mathbf{0}_S$

by (*simp add: φ .hom-add-pow-nat*)

```

define  $\sigma :: \text{int} \times \text{int}$  where  $\sigma = (0, 2)$ 
have  $eq1: \varphi((6, 2) [\wedge]_R M) = \varphi(6, -2)$ 
proof -
  have  $(6, 2) = (6, 0) \oplus_R \sigma$ 
    by (simp add: lucas-lehmer-ring-def  $\sigma$ -def lucas-lehmer-add'-def)
  also have  $\varphi(\dots [\wedge]_R M) = \varphi((6, 0) [\wedge]_R M) \oplus_S \varphi(\sigma [\wedge]_R M)$ 
    using prime and  $\langle \text{add-pow } S \ M \ \mathbf{1}_S = \mathbf{0}_S \rangle$ 
    by (simp add: S.binomial-finite-char)
  also have  $(6, 0) [\wedge]_R M = (6 \wedge M, 0)$ 
    by (simp add: R-pow-int)
  also have  $[6 \wedge M = 6] \pmod{(\text{int } M)}$  using M-gt-6
    by (intro little-Fermat-int) (use prime in  $\langle \text{auto simp flip: dvd-nat-abs-iff} \rangle$ )
  hence  $\varphi(6 \wedge M, 0) = \varphi(6, 0)$ 
    unfolding  $\varphi$ -def by (intro lucas-lehmer-hom-cong) auto
  also have  $\sigma = (2, 0) \otimes_R (0, 1)$ 
    by (simp add:  $\sigma$ -def lucas-lehmer-ring-def lucas-lehmer-mult'-def)
  hence  $\varphi(\sigma [\wedge]_R M) = \varphi((2, 0) [\wedge]_R M \otimes_R (0, 1) [\wedge]_R M)$ 
    by (subst R.nat-pow-distrib [symmetric]) auto
  also have  $\dots = \varphi((2, 0) [\wedge]_R M) \otimes_S \varphi((0, 1) [\wedge]_R M)$ 
    by simp
  also have  $(2, 0) [\wedge]_R M = (2 \wedge M, 0)$ 
    by (simp add: R-pow-int)
  also have  $[2 \wedge M = 2] \pmod{\text{int } M}$  using M-gt-6 prime
    by (intro little-Fermat-int) (auto simp flip: dvd-nat-abs-iff dest: dvd-imp-le)
  hence  $\varphi(2 \wedge M, 0) = \varphi(2, 0)$ 
    unfolding  $\varphi$ -def by (intro lucas-lehmer-hom-cong) auto
  also have  $M\text{-eq}: M = \text{Suc}(2 * ((M - 1) \text{div } 2))$ 
    using M-odd by auto
  have  $(0, 1) [\wedge]_R M = (0, 1) \otimes_R ((0, 1) [\wedge]_R (2::\text{nat})) [\wedge]_R ((M - 1) \text{div } 2)$ 
    by (subst M-eq) (auto simp: R.nat-pow-mult R.nat-pow-pow R.cring-simprules)
  also have  $(0, 1) [\wedge]_R (2::\text{nat}) = (3, 0)$ 
    by (simp add: eval-nat-numeral) (simp add: lucas-lehmer-ring-def lucas-lehmer-mult'-def)
  also have  $\varphi((0, 1) \otimes_R (3, 0) [\wedge]_R ((M - 1) \text{div } 2)) =$ 
     $\varphi((3, 0) [\wedge]_R ((M - 1) \text{div } 2)) \otimes_S \varphi(0, 1)$ 
    by (simp add: S.cring-simprules)
  also have  $(3, 0) [\wedge]_R ((M - 1) \text{div } 2) = (3 \wedge ((M - 1) \text{div } 2), 0)$ 
    by (simp add: R-pow-int)
  also have  $\varphi(3 \wedge ((M - 1) \text{div } 2), 0) = \varphi(-1, 0)$ 
    unfolding  $\varphi$ -def
  proof (intro lucas-lehmer-hom-cong')
    have  $[3 \wedge ((M - 1) \text{div } 2) = \text{Legendre } 3 \ M] \pmod{\text{int } M}$ 
      by (rule cong-sym, rule euler-criterion) (use prime M-gt-6 in auto)
    thus  $[3 \wedge ((M - 1) \text{div } 2) = -1] \pmod{\text{int } M}$ 
      by (simp add: Legendre-3-M)
  qed auto
  also have  $\varphi(2, 0) \otimes_S (\varphi(-1, 0) \otimes_S \varphi(0, 1)) = \varphi((2, 0) \otimes_R (-1, 0) \otimes_R$ 
 $(0, 1))$ 
    by (simp add: R.cring-simprules S.cring-simprules)
  also have  $\varphi(6, 0) \oplus_S \varphi((2, 0) \otimes_R (-1, 0) \otimes_R (0, 1)) =$ 

```

$\varphi ((6, 0) \oplus_R (2, 0) \otimes_R (-1, 0) \otimes_R (0, 1))$
by simp
also have $\dots = \varphi (6, -2)$ **unfolding** φ -def
by (intro lucas-lehmer-hom-cong)
(auto simp: lucas-lehmer-ring-def lucas-lehmer-mult'-def lucas-lehmer-add'-def)
finally show $\varphi ((6, 2) [\frown]_R M) = \varphi (6, -2)$
by (simp add: R.cring-simprules S.cring-simprules)
qed

have eq2: $\varphi ((24, 0) [\frown]_R ((M - 1) \text{ div } 2)) = \ominus_S \mathbf{1}_S$
proof -
have $(24, 0) = (2, 0) [\frown]_R (3::\text{nat}) \otimes_R (3, 0)$
by (simp add: eval-nat-numeral) (auto simp: lucas-lehmer-ring-def lucas-lehmer-mult'-def)
also have $\dots [\frown]_R ((M - 1) \text{ div } 2) =$
 $(2, 0) [\frown]_R ((M - 1) \text{ div } 2)) [\frown]_R (3::\text{nat}) \otimes_R (3, 0) [\frown]_R ((M -$
1) div 2)
by (simp add: R.cring-simprules R.nat-pow-distrib R.nat-pow-pow mult-ac)
also have $\varphi \dots = (\varphi ((2, 0) [\frown]_R ((M - 1) \text{ div } 2))) [\frown]_S (3::\text{nat}) \otimes_S$
 $\varphi ((3, 0) [\frown]_R ((M - 1) \text{ div } 2))$ **by simp**
also have $(2, 0) [\frown]_R ((M - 1) \text{ div } 2) = (2 \wedge ((M - 1) \text{ div } 2), 0)$
by (simp add: R-pow-int)
also have $\varphi \dots = \varphi (1, 0)$
unfolding φ -def
proof (intro lucas-lehmer-hom-cong')
have $[2 \wedge ((M - 1) \text{ div } 2) = \text{Legendre } 2 \ M] \text{ (mod int } M)$
by (rule cong-sym, rule euler-criterion) (use prime M-gt-6 in auto)
thus $[2 \wedge ((M - 1) \text{ div } 2) = 1] \text{ (mod int } M)$
using Legendre-2-M **by simp**
qed auto
also have $(1, 0) = \mathbf{1}_R$
by (simp add: lucas-lehmer-ring-def)
also have $(3, 0) [\frown]_R ((M - 1) \text{ div } 2) = (3 \wedge ((M - 1) \text{ div } 2), 0)$
by (simp add: R-pow-int)
also have $\varphi \dots = \varphi (-1, 0)$
unfolding φ -def
proof (intro lucas-lehmer-hom-cong')
have $[3 \wedge ((M - 1) \text{ div } 2) = \text{Legendre } 3 \ M] \text{ (mod int } M)$
by (rule cong-sym, rule euler-criterion) (use prime M-gt-6 in auto)
thus $[3 \wedge ((M - 1) \text{ div } 2) = -1] \text{ (mod int } M)$
using Legendre-3-M **by simp**
qed auto
also have $(-1, 0) = \ominus_R \mathbf{1}_R$
using minus-lucas-lehmer-ring **by** (simp add: lucas-lehmer-ring-def)
finally show $\varphi ((24, 0) [\frown]_R ((M - 1) \text{ div } 2)) = \ominus_S \mathbf{1}_S$
by simp
qed

define $\omega \ \omega' :: \text{int} \times \text{int}$ **where** $\omega = (2, 1)$ **and** $\omega' = (2, -1)$
have eq3: $\varphi (\omega [\frown]_R ((M + 1) \text{ div } 2)) = \ominus_S \mathbf{1}_S$

proof –
have $(M + 1) \text{ div } 2 = \text{Suc } ((M - 1) \text{ div } 2)$
using *M-odd M-gt-6* **by** *(auto elim!: oddE)*
have $*$: $\varphi ((24, 0) \otimes_R \omega) = \varphi ((6, 2) [\uparrow]_R (2 :: \text{nat}))$ **unfolding** $\varphi\text{-def}$
by *(intro lucas-lehmer-hom-cong)*
(simp-all add: eval-nat-numeral,
auto simp: lucas-lehmer-ring-def lucas-lehmer-mult'-def $\omega\text{-def}$)
have $\varphi (\ominus_R \mathbf{1}_R) \otimes_S \varphi ((24, 0) \otimes_R \omega) [\uparrow]_S ((M + 1) \text{ div } 2) =$
 $\varphi (\ominus_R \mathbf{1}_R) \otimes_S \varphi ((6, 2) [\uparrow]_R (2 :: \text{nat})) [\uparrow]_S ((M + 1) \text{ div } 2)$
by *(subst *) auto*
hence $\varphi (\ominus_R \mathbf{1}_R) \otimes_S \varphi ((24, 0) [\uparrow]_R ((M + 1) \text{ div } 2)) \otimes_S \varphi (\omega [\uparrow]_R ((M + 1) \text{ div } 2)) =$
 $\varphi (\ominus_R \mathbf{1}_R) \otimes_S \varphi ((6, 2) [\uparrow]_R (2 * ((M + 1) \text{ div } 2)))$
by *(simp add: R.nat-pow-distrib S.nat-pow-distrib R.nat-pow-pow*
S.nat-pow-pow R.cring-simprules S.cring-simprules)
also have $2 * ((M + 1) \text{ div } 2) = M + 1$
using *M-odd* **by** *auto*
finally have $\varphi (24, 0) \otimes_S (\varphi (\ominus_R \mathbf{1}_R) \otimes_S \varphi ((24, 0) [\uparrow]_R ((M - 1) \text{ div } 2)))$
 \otimes_S
 $\varphi (\omega [\uparrow]_R ((M + 1) \text{ div } 2)) =$
 $\varphi (\ominus_R \mathbf{1}_R) \otimes_S (\varphi (6, 2) \otimes_S \varphi ((6, 2) [\uparrow]_R M))$
by *(subst (asm) $\langle (M + 1) \text{ div } 2 = \cdot \rangle$) (simp add: S.cring-simprules R.cring-simprules)*

also have $\varphi ((24, 0) [\uparrow]_R ((M - 1) \text{ div } 2)) = \ominus_S \mathbf{1}_S$
by *(subst eq2) auto*
also have $(\varphi (\ominus_R \mathbf{1}_R) \otimes_S \ominus_S \mathbf{1}_S) = \mathbf{1}_S$
by *(simp add: S.cring-simprules)*
also have $\varphi ((6, 2) [\uparrow]_R M) = \varphi (6, -2)$
by *(subst eq1) auto*
also have $\varphi (6, 2) \otimes_S \varphi (6, -2) = \varphi ((6, 2) \otimes_R (6, -2))$
by *simp*
also have $\dots = \varphi (24, 0)$ **unfolding** $\varphi\text{-def}$
by *(intro lucas-lehmer-hom-cong) (auto simp: lucas-lehmer-ring-def lucas-lehmer-mult'-def)*
finally have $\varphi (24, 0) \otimes_S (\varphi (\omega [\uparrow]_R ((M + 1) \text{ div } 2))) =$
 $\varphi (24, 0) \otimes_S \varphi (\ominus_R \mathbf{1}_R)$
by *(simp add: S.cring-simprules)*
also have $\varphi (24, 0) = (24 \bmod M, 0)$
by *(simp add: $\varphi\text{-def}$ lucas-lehmer-hom-def nat-mod-as-int)*
finally have $(24 \bmod M, 0) \otimes_S (\varphi (\omega [\uparrow]_R ((M + 1) \text{ div } 2))) =$
 $(24 \bmod M, 0) \otimes_S \varphi (\ominus_R \mathbf{1}_R)$
moreover have $(24 \bmod M, 0) \in \text{Units } S$
unfolding *S-def* **using** *M-gt-6 prime M-not-dvd-24*
by *(intro int-in-Units-lucas-lehmer-ring-mod) (auto simp: dvd-mod-iff intro!:*
Nat.grOI)
ultimately show $\varphi (\omega [\uparrow]_R ((M + 1) \text{ div } 2)) = \ominus_S \mathbf{1}_S$
by *(subst (asm) S.Units-l-cancel) auto*
qed

have *eq4*: $\varphi (\omega [\uparrow]_R (2 \wedge (p - 2) :: \text{nat}) \oplus_R \omega' [\uparrow]_R (2 \wedge (p - 2) :: \text{nat})) = \mathbf{0}_S$

(is φ ?lhs = -)
proof -
 have $\varphi (\omega [\frown]_R ((M + 1) \text{ div } 2)) \otimes_S \varphi (\omega' [\frown]_R ((M + 1) \text{ div } 4)) \oplus_S$
 $\varphi (\omega' [\frown]_R ((M + 1) \text{ div } 4)) = \mathbf{0}_S$
 by (subst eq3) (auto simp: S.cring-simprules)
 also have $2 \wedge 2 \text{ dvd } (2 \wedge p :: \text{nat})$
 by (intro le-imp-power-dvd) (use p-gt-2 in auto)
 hence $4 \text{ dvd } (M + 1)$ by (auto simp: M-def)
 hence $(M + 1) \text{ div } 2 = (M + 1) \text{ div } 4 + (M + 1) \text{ div } 4$
 by presburger
 also have $\varphi (\omega [\frown]_R \dots) \otimes_S \varphi (\omega' [\frown]_R ((M + 1) \text{ div } 4)) =$
 $\varphi (\omega \otimes_R \omega') [\frown]_S ((M + 1) \text{ div } 4) \otimes_S \varphi (\omega [\frown]_R ((M + 1) \text{ div } 4))$
 by (simp add: S.cring-simprules S.nat-pow-distrib flip: S.nat-pow-mult)
 also have $\varphi (\omega \otimes_R \omega') = \varphi \mathbf{1}_R$ **unfolding** φ -def
 by (intro lucas-lehmer-hom-cong)
 (auto simp: ω -def ω' -def lucas-lehmer-ring-def lucas-lehmer-mult'-def)
 also have $(M + 1) \text{ div } 4 = 2 \wedge (p - 2)$
 using p-gt-2 by (auto simp: M-def power-diff)
 finally show eq4: $\varphi (\omega [\frown]_R (2 \wedge (p - 2) :: \text{nat}) \oplus_R \omega' [\frown]_R (2 \wedge (p - 2) ::$
 $\text{nat})) = \mathbf{0}_S$
 by simp
qed

 have φ ?lhs = $\mathbf{0}_S$
 by (rule eq4)
 also have lucas-lehmer-to-real ?lhs =
 $\text{lucas-lehmer-to-real } (\text{gen-lucas-lehmer-sequence } 4 (p - 2), 0)$
 by (simp add: ω -def ω' -def lucas-lehmer-to-real-simps gen-lucas-lehmer-sequence-4-closed-form1)
 hence ?lhs = (gen-lucas-lehmer-sequence 4 (p - 2), 0)
 by (rule injD[OF lucas-lehmer-to-real-inj])
 finally have gen-lucas-lehmer-sequence 4 (p - 2) mod M = 0 using M-gt-6
 by (auto simp: φ -def lucas-lehmer-hom-def S-def lucas-lehmer-ring-mod-def)
 thus $(2 \wedge p - 1) \text{ dvd gen-lucas-lehmer-sequence } 4 (p - 2)$
 by (simp add: M-def mod-eq-0-iff-dvd of-nat-diff)
qed

corollary lucas-lehmer-correct:
 $\text{prime } (2 \wedge p - 1 :: \text{nat}) \longleftrightarrow$
 $\text{prime } p \wedge (p = 2 \vee (2 \wedge p - 1) \text{ dvd gen-lucas-lehmer-sequence } 4 (p - 2))$
proof (intro iffI; (elim conjE)?)
 assume prime: prime $(2 \wedge p - 1 :: \text{nat})$
 from prime have $p \neq 0$ $p \neq 1$
 by (auto intro!: Nat.gr0I)
 hence $p = 2 \vee p > 2$ by auto
 thus prime $p \wedge (p = 2 \vee (2 \wedge p - 1) \text{ dvd gen-lucas-lehmer-sequence } 4 (p - 2))$
proof (elim disjE)
 assume $p > 2$
 with prime interpret mersenne-prime p $2 \wedge p - 1$
 by unfold-locales

```

    from lucas-lehmer-necessary p-prime show ?thesis by auto
  qed auto
next
  assume prime: prime p and *: p = 2 ∨ (2 ^ p - 1) dvd gen-lucas-lehmer-sequence
  4 (p - 2)
  from * consider p = 2 | p ≠ 2 (2 ^ p - 1) dvd gen-lucas-lehmer-sequence 4 (p
  - 2)
  by auto
  thus prime (2 ^ p - 1 :: nat)
  proof cases
    assume p ≠ 2 and dvd: (2 ^ p - 1) dvd gen-lucas-lehmer-sequence 4 (p - 2)
    from ⟨prime p⟩ and ⟨p ≠ 2⟩ have p > 2
    using prime-gt-1-nat[of p] by auto
    with prime have odd p by (auto simp: prime-odd-nat)
    with prime dvd show ?thesis
    by (intro lucas-lehmer-sufficient)
  qed auto
qed

```

```

corollary lucas-lehmer-correct':
  prime (2 ^ p - 1 :: nat) ↔ prime p ∧ (p = 2 ∨ lucas-lehmer-test p)
  using lucas-lehmer-correct[of p] prime-gt-1-nat[of p]
  by (auto simp: lucas-lehmer-test-def)

```

2.8 A first executable version Lucas–Lehmer test

The following is an implementation of the Lucas–Lehmer test using modular arithmetic on the integers. This is not the most efficient implementation – the modular arithmetic can be replaced by much cheaper bitwise operations, and we will do that in the next section.

```

primrec gen-lucas-lehmer-sequence' :: int ⇒ int ⇒ nat ⇒ int where
  gen-lucas-lehmer-sequence' m a 0 = a
| gen-lucas-lehmer-sequence' m a (Suc n) = gen-lucas-lehmer-sequence' m ((a ^ 2
- 2) mod m) n

```

```

lemma gen-lucas-lehmer-sequence'-Suc':
  gen-lucas-lehmer-sequence' m a (Suc n) = (gen-lucas-lehmer-sequence' m a n ^ 2
- 2) mod m
  by (induction n arbitrary: a) auto

```

```

lemma gen-lucas-lehmer-sequence'-correct:
  assumes a ∈ {0..<m}
  shows gen-lucas-lehmer-sequence' m a n = gen-lucas-lehmer-sequence a n mod
  m
  using assms
  proof (induction n)
    case (Suc n)
    have gen-lucas-lehmer-sequence' m a (Suc n) =

```

```

      ((gen-lucas-lehmer-sequence a n mod m)2 - 2) mod m
    using Suc unfolding gen-lucas-lehmer-sequence'-Suc' by simp
    also have ... = ((gen-lucas-lehmer-sequence a n)2 - 2) mod m
      by (intro congD cong-diff cong-pow cong-refl) (auto simp: cong-def)
    finally show ?case by simp
qed auto

```

```

lemma lucas-lehmer-test-code-arithmetic [code]:
  lucas-lehmer-test p = (p > 2 ∧
    gen-lucas-lehmer-sequence' (2 ^ p - 1) 4 (p - 2) = 0)
  unfolding lucas-lehmer-test-def
proof (intro conj-cong refl)
  assume p: p > 2
  from p have 2 ^ p ≥ (2 ^ 3 :: int) by (intro power-increasing) auto
  have (2 ^ p - 1 dvd gen-lucas-lehmer-sequence 4 (p - 2)) ↔
    gen-lucas-lehmer-sequence 4 (p - 2) mod (2 ^ p - 1) = 0
  by auto
  also have gen-lucas-lehmer-sequence 4 (p - 2) mod (2 ^ p - 1) =
    gen-lucas-lehmer-sequence' (2 ^ p - 1) 4 (p - 2)
  using ⟨2 ^ p ≥ 2 ^ 3⟩
  by (intro gen-lucas-lehmer-sequence'-correct [symmetric]) auto
  finally show (2 ^ p - 1 dvd gen-lucas-lehmer-sequence 4 (p - 2)) =
    (gen-lucas-lehmer-sequence' (2 ^ p - 1) 4 (p - 2) = 0) .
qed

```

```

lemma mersenne-prime-iff: mersenne-prime p ↔ p > 2 ∧ prime (2 ^ p - 1 ::
nat)
  by (simp add: mersenne-prime-def)

```

```

lemma mersenne-prime-code [code]:
  mersenne-prime p ↔ prime p ∧ lucas-lehmer-test p
  unfolding mersenne-prime-iff using lucas-lehmer-correct'[of p]
  by (auto simp: lucas-lehmer-test-def)

```

end

3 Efficient code for testing Mersenne primes

```

theory Lucas-Lehmer-Code
imports
  Lucas-Lehmer
  HOL-Library.Code-Target-Numeral
  HOL-Library.Code-Bit-Shifts-for-Arithmetic
begin

```

3.1 Efficient computation of remainders modulo a Mersenne number

We have $k = k \bmod 2^n + k \operatorname{div} 2^n \pmod{(2^n - 1)}$, and $k \bmod 2^n = k \& (2^n - 1)$ and $k \operatorname{div} 2^n = k \gg n$. Therefore, we can reduce k modulo $2^n - 1$ using only bitwise operations, addition, and bit shifts.

lemma *cong-mersenne-number-int*:

```

fixes  $k :: \text{int}$ 
shows  $[k \bmod 2^n + k \operatorname{div} 2^n = k] \pmod{(2^n - 1)}$ 
proof -
  have  $k = (2^n - 1 + 1) * (k \operatorname{div} 2^n) + (k \bmod 2^n)$ 
    by simp
  also have  $[\dots = (0 + 1) * (k \operatorname{div} 2^n) + (k \bmod 2^n)] \pmod{(2^n - 1)}$ 
    by (intro cong-add cong-mult cong-refl) (auto simp: cong-def)
  finally show ?thesis by (simp add: cong-sym add-ac)
qed

```

We encapsulate a single reduction step in the following operation. Note, however, that the result is not, in general, the same as $k \bmod (2^n - 1)$. Multiple reductions might be required in order to reduce it below 2^n , and a multiple of $2^n - 1$ can be reduced to $2^n - 1$, which is invariant to further reduction steps.

definition *mersenne-mod* $:: \text{int} \Rightarrow \text{nat} \Rightarrow \text{int}$ **where**

mersenne-mod $k\ n = k \bmod 2^n + k \operatorname{div} 2^n$

lemma *mersenne-mod-code* [*code*]:

```

mersenne-mod  $k\ n = \text{take-bit } n\ k + \text{drop-bit } n\ k$ 
by (simp add: mersenne-mod-def flip: take-bit-eq-mod drop-bit-eq-div)

```

lemma *cong-mersenne-mod*: $[\text{mersenne-mod } k\ n = k] \pmod{(2^n - 1)}$

unfolding *mersenne-mod-def* **by** (*rule cong-mersenne-number-int*)

lemma *mersenne-mod-nonneg* [*simp*]: $k \geq 0 \implies \text{mersenne-mod } k\ n \geq 0$

unfolding *mersenne-mod-def* **by** (*intro add-nonneg-nonneg*) (*simp-all add: pos-imp-zdiv-nonneg-iff*)

lemma *mersenne-mod-less*:

assumes $k \leq 2^m\ m \geq n$

shows $\text{mersenne-mod } k\ n < 2^n + 2^{m-n}$

proof -

have $\text{mersenne-mod } k\ n = k \bmod 2^n + k \operatorname{div} 2^n$

by (*simp add: mersenne-mod-def*)

also have $k \bmod 2^n < 2^n$

by *simp*

also {

have $k \operatorname{div} 2^n * 2^n + 0 \leq k \operatorname{div} 2^n * 2^n + k \bmod (2^n)$

by (*intro add-mono*) *auto*

also have $\dots = k$

by (*subst mult.commute*) *auto*

```

    also have ... ≤ 2 ^ m
      using assms by simp
    also have ... = 2 ^ (m - n) * 2 ^ n
      using assms by (simp flip: power-add)
    finally have k div 2 ^ n ≤ 2 ^ (m - n)
      by simp
  }
  finally show ?thesis by simp
qed

```

```

lemma mersenne-mod-less':
  assumes k ≤ 5 * 2 ^ n
  shows mersenne-mod k n < 2 ^ n + 5
proof -
  have mersenne-mod k n = k mod 2 ^ n + k div 2 ^ n
    by (simp add: mersenne-mod-def)
  also have k mod 2 ^ n < 2 ^ n
    by simp
  also {
    have k div 2 ^ n * 2 ^ n + 0 ≤ k div 2 ^ n * 2 ^ n + k mod (2 ^ n)
      by (intro add-mono) auto
    also have ... = k
      by (subst mult.commute) auto
    also have ... ≤ 5 * 2 ^ n
      using assms by simp
    finally have k div 2 ^ n ≤ 5
      by simp
  }
  finally show ?thesis by simp
qed

```

It turns out that for our use case, a single reduction is not enough to reduce the number in question enough (or at least I was unable to prove that it is). We therefore perform two reduction steps, which is enough to guarantee that our numbers are below $2^n + 4$ before and after every step in the Lucas–Lehmer sequence.

Whether one or two reductions are performed is not very important anyway, since the dominant step is the squaring anyway.

```

definition mersenne-mod2 :: int ⇒ nat ⇒ int where
  mersenne-mod2 k n = mersenne-mod (mersenne-mod k n) n

```

```

lemma cong-mersenne-mod2: [mersenne-mod2 k n = k] (mod (2 ^ n - 1))
  unfolding mersenne-mod2-def by (rule cong-trans) (rule cong-mersenne-mod)+

```

```

lemma mersenne-mod2-nonneg [simp]: k ≥ 0 ⇒ mersenne-mod2 k n ≥ 0
  unfolding mersenne-mod2-def by simp

```

```

lemma mersenne-mod2-less:

```

assumes $n > 2$ **and** $k \leq 2^{(2 * n + 2)}$
shows *mersenne-mod2* $k n < 2^n + 5$
proof –
from *assms* **have** $2^3 \leq (2^n :: int)$
by (*intro power-increasing*) *auto*
hence $2^n \geq (8 :: int)$ **by** *simp*
have *mersenne-mod* $k n < 2^n + 2^{(2 * n + 2 - n)}$
by (*rule mersenne-mod-less*) (*use assms in auto*)
also have $\dots \leq 5 * 2^n$
by (*simp add: power-add*)
finally have *mersenne-mod* (*mersenne-mod k n*) $n < 2^n + 5$
by (*intro mersenne-mod-less'*) *auto*
thus *?thesis* **by** (*simp add: mersenne-mod2-def*)
qed

Since we subtract 2 at one point, the intermediate results can become negative. This is not a problem since our reduction modulo $2^p - 1$ happens to make them positive again immediately.

lemma *mersenne-mod-nonneg-strong*:
 $\langle \text{mersenne-mod } a \ p \geq 0 \rangle$ **if** $\langle -(2^p) + 1 < a \rangle$
proof (*cases* $\langle a < 0 \rangle$)
case *False*
with that show *?thesis*
by *simp*
next
case *True*
have $\langle -a \text{ div } -(2^p) = -1 \rangle$
by (*rule div-pos-neg-trivial*) (*use* $\langle a < 0 \rangle$ *that in simp-all*)
then have $\langle a \text{ div } 2^p = -1 \rangle$
by *simp*
moreover have $\langle -a \text{ mod } -(2^p) = -a + -(2^p) \rangle$
by (*rule mod-pos-neg-trivial*) (*use* $\langle a < 0 \rangle$ *that in simp-all*)
then have $\langle a \text{ mod } 2^p = a + 2^p \rangle$
by *simp*
ultimately have $\langle \text{mersenne-mod } a \ p = a + 2^p - 1 \rangle$
by (*simp add: mersenne-mod-def*)
also have $\langle \dots > 0 \rangle$ **using that by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *mersenne-mod2-nonneg-strong*:
assumes $a > -(2^p) + 1$
shows *mersenne-mod2* $a \ p \geq 0$
unfolding *mersenne-mod2-def*
by (*rule mersenne-mod-nonneg*, *rule mersenne-mod-nonneg-strong*) (*use assms in auto*)

3.2 Efficient code for the Lucas–Lehmer sequence

primrec *gen-lucas-lehmer-sequence''* :: *nat* \Rightarrow *int* \Rightarrow *nat* \Rightarrow *int* **where**
gen-lucas-lehmer-sequence'' *p* *a* 0 = *a*
| *gen-lucas-lehmer-sequence''* *p* *a* (Suc *n*) =
gen-lucas-lehmer-sequence'' *p* (mersenne-mod2 ($a^2 - 2$) *p*) *n*

lemma *gen-lucas-lehmer-sequence''-correct*:

assumes [*a* = *a'*] (mod ($2^p - 1$))

shows [*gen-lucas-lehmer-sequence''* *p* *a* *n* = *gen-lucas-lehmer-sequence* *a'* *n*] (mod ($2^p - 1$))

using *assms*

proof (*induction n arbitrary: a a'*)

case (Suc *n*)

have [mersenne-mod2 ($a^2 - 2$) *p* = $a^2 - 2$] (mod ($2^p - 1$))

by (*rule cong-mersenne-mod2*)

also have [$a^2 - 2 = a'^2 - 2$] (mod ($2^p - 1$))

by (*intro cong-pow cong-diff Suc.premis cong-refl*)

finally have [*gen-lucas-lehmer-sequence''* *p* (mersenne-mod2 ($a^2 - 2$) *p*) *n* =
gen-lucas-lehmer-sequence ($a'^2 - 2$) *n*] (mod $2^p - 1$)

by (*rule Suc.IH*)

thus ?*case*

by (*auto simp del: gen-lucas-lehmer-sequence.simps simp: gen-lucas-lehmer-sequence-Suc'*)

qed *auto*

lemma *gen-lucas-lehmer-sequence''-bounds*:

assumes $a \geq 0$ $a < 2^p + 5$ $p > 2$

shows *gen-lucas-lehmer-sequence''* *p* *a* *n* $\in \{0..<2^p + 5\}$

using *assms*

proof (*induction n arbitrary: a*)

case (Suc *n*)

from *Suc.premis* **have** $a^2 < (2^p + 5)^2$

by (*intro power-strict-mono Suc.premis*) *auto*

also have $\dots \leq (2^{p+1})^2$

using *power-increasing*[of 3 *p* 2 :: *int*] $\langle p > 2 \rangle$ **by** (*intro power-mono*) *auto*

finally have $a^2 - 2 < 2^{2p+2}$

by (*simp flip: power-mult mult-ac*)

moreover {

from $\langle p > 2 \rangle$ **have** $(2^p) \geq (2^3)$:: *int*

by (*intro power-increasing*) *auto*

hence $-(2^p) + 1 < (-2)$:: *int*

by *simp*

also have $-2 \leq a^2 - 2$

by *simp*

finally have mersenne-mod2 ($a^2 - 2$) *p* ≥ 0

by (*rule mersenne-mod2-nonneg-strong*)

}

ultimately have *gen-lucas-lehmer-sequence''* *p* (mersenne-mod2 ($a^2 - 2$) *p*) *n*
 $\in \{0..<2^p + 5\}$

using $\langle p > 2 \rangle$ **by** (*intro Suc.IH mersenne-mod2-less*) *auto*

thus *?case* **by** *simp*
qed *auto*

3.3 Code for the Lucas–Lehmer test

lemma *lucas-lehmer-test-code* [*code*]:

```

lucas-lehmer-test p  $\longleftrightarrow$ 
  ( $2 < p \wedge (\text{let } x = \text{gen-lucas-lehmer-sequence'' } p \ 4 \ (p - 2) \text{ in } x = 0 \vee x =$ 
  (push-bit p 1) - 1))
unfolding lucas-lehmer-test-def
proof (rule conj-cong)
assume  $p > 2$ 
define x where  $x = \text{gen-lucas-lehmer-sequence'' } p \ 4 \ (p - 2)$ 
from  $\langle p > 2 \rangle$  have  $2^3 \leq (2^p :: \text{int})$  by (intro power-increasing) auto
hence  $2^p \geq (8 :: \text{int})$  by simp
hence bounds:  $x \in \{0..<2^p + 5\}$ 
unfolding x-def using  $\langle p > 2 \rangle$  by (intro gen-lucas-lehmer-sequence''-bounds)
auto
have  $2^p - 1 \text{ dvd } \text{gen-lucas-lehmer-sequence } 4 \ (p - 2) \longleftrightarrow 2^p - 1 \text{ dvd } x$ 
unfolding x-def by (intro cong-dvd-iff cong-sym[OF gen-lucas-lehmer-sequence''-correct])
auto
also have  $\dots \longleftrightarrow x \in \{0, 2^p - 1\}$ 
proof
assume  $2^p - 1 \text{ dvd } x$ 
then obtain k where  $x = (2^p - 1) * k$  by auto
have  $k \geq 0$  using bounds  $\langle 2^p \geq 8 \rangle$ 
by (auto simp: k zero-le-mult-iff)
moreover {
  have  $x < 2^p + 5$  using bounds by simp
  also have  $\dots \leq (2^p - 1) * 2$ 
  using  $\langle 2^p \geq 8 \rangle$  by simp
  finally have  $(2^p - 1) * k < (2^p - 1) * 2$ 
  unfolding k .
  hence  $k < 2$ 
  by (subst (asm) mult-less-cancel-left) auto
}
ultimately have  $k = 0 \vee k = 1$  by auto
thus  $x \in \{0, 2^p - 1\}$ 
using k by auto
qed auto
finally show  $(2^p - 1 \text{ dvd } \text{gen-lucas-lehmer-sequence } 4 \ (p - 2)) =$ 
  ( $(\text{let } x = x \text{ in } x = 0 \vee x = (\text{push-bit } p \ 1) - 1)$ )
by (simp add: Let-def push-bit-eq-mult)
qed auto

```

3.4 Examples

Note that for some reason, the clever bit-arithmetic version of the Lucas–Lehmer test is actually much slower than the one using integer arithmetic

when using PolyML, and even more so when using the built-in evaluator in Isabelle (which also uses PolyML with a slightly different setup).

I do not quite know why this is the case, but it is likely because of inefficient implementations of bit arithmetic operations in PolyML and/or the code generator setup for it.

When running with GHC, the bit-arithmetic version is *much* faster.

```
value filter mersenne-prime [0.. $<100$ ]
```

```
lemma prime ( $2^{521} - 1 :: \text{nat}$ )  
  by (subst lucas-lehmer-correct') eval
```

```
lemma prime ( $2^{4253} - 1 :: \text{nat}$ )  
  by (subst lucas-lehmer-correct') eval
```

```
end
```

References

- [1] J. W. Bruce. A really trivial proof of the Lucas-Lehmer test. *The American Mathematical Monthly*, 100(4):370–371, 1993.
- [2] Ö. J. Rödseth. A note on primality tests for $n = h \cdot 2^n - 1$. *BIT Numerical Mathematics*, 34(3):451–454, Sep 1994.
- [3] Wikipedia contributors. Lucas–Lehmer primality test — Wikipedia, the free encyclopedia, 2020. [Online; accessed 17 Jan 2020].
- [4] Wikipedia contributors. Mersenne prime — Wikipedia, the free encyclopedia, 2020. [Online; accessed 17 Jan 2020].