A Formalization of Pratt’s Primality Certificates

By Simon Wimmer and Lars Noschinski

February 23, 2021

Abstract

In 1975, Pratt introduced a proof system for certifying primes [1]. He showed that a number $p$ is prime iff a primality certificate for $p$ exists. By showing a logarithmic upper bound on the length of the certificates in size of the prime number, he concluded that the decision problem for prime numbers is in NP. This work formalizes soundness and completeness of Pratt’s proof system as well as an upper bound for the size of the certificate.

Contents

1 Pratt’s Primality Certificates 1
  1.1 Soundness 1
  1.2 Completeness 4
  1.3 Efficient modular exponentiation 12
  1.4 Executable certificate checker 14
  1.5 Proof method setup 18
  1.6 Code generation for Pratt certificates 19

1 Pratt’s Primality Certificates

theory Pratt-Certificate
imports
  Complex-Main
  Lehmer.Lehmer
begin

This work formalizes Pratt’s proof system as described in his article “Every Prime has a Succinct Certificate”[1].

The proof system makes use of two types of predicates:

- Prime($p$): $p$ is a prime number
- $(p, a, x): \forall q \in \text{prime-factors}(x). [a^{(p - 1) \text{ div } q} \neq 1] \pmod{p}$
We represent these predicates with the following datatype:

```
datatype pratt = Prime nat | Triple nat nat nat
```

Pratt describes an inference system consisting of the axiom \((p, a, 1)\) and the following inference rules:

- **R1:** If we know that \((p, a, x)\) and \([a^{(p - 1)} \div q \neq 1] (mod p)\) hold for some prime number \(q\) we can conclude \((p, a, qx)\) from that.

- **R2:** If we know that \((p, a, p - 1)\) and \([a^{(p - 1)} = 1] (mod p)\) hold, we can infer \(\text{Prime}(p)\).

Both rules follow from Lehmer’s theorem as we will show later on.

A list of predicates (i.e., values of type \(\text{pratt}\)) is a certificate, if it is built according to the inference system described above. I.e., a list \(x \# xs\) is a certificate if \(xs\) is a certificate and \(x\) is either an axiom or all preconditions of \(x\) occur in \(xs\).

We call a certificate \(xs\) a certificate for \(p\), if \(\text{Prime}(p)\) occurs in \(xs\).

The function \(\text{valid-cert}\) checks whether a list is a certificate.

```
fun valid-cert : pratt list ⇒ bool where
valid-cert [] = True
| R2: valid-cert (Prime p#xs) ←→ 1 < p ∧ valid-cert xs
  ∧ (∃ a . \([a^{(p - 1)} = 1] (mod p) ∧ Triple p a (p - 1) ∈ set xs) ∧ valid-cert xs ∧ \((p - 1) \div q \neq 1) (mod p))
| R1: valid-cert (Triple p a x # xs) ←→ p > 1 ∧ 0 < x ∧ valid-cert xs ∧ \((x = 1) \lor
  (∃ q y . x = q * y ∧ \text{Prime}(q) ∈ set xs ∧ Triple p a y ∈ set xs ∧ \([a^{(p - 1)} \div q \neq 1] (mod p)))
```

We define a function \(\text{size-cert}\) to measure the size of a certificate, assuming a binary encoding of numbers. We will use this to show that there is a certificate for a prime number \(p\) such that the size of the certificate is polynomially bounded in the size of the binary representation of \(p\).

```
fun size-pratt : pratt ⇒ real where
size-pratt (Prime p) = log 2 p
size-pratt (Triple p a x) = log 2 p + log 2 a + log 2 x

fun size-cert : pratt list ⇒ real where
size-cert [] = 0 |
size-cert (x # xs) = 1 + size-pratt x + size-cert xs
```

### 1.1 Soundness

In Section 1 we introduced the predicates \(\text{Prime}(p)\) and \((p, a, x)\). In this section we show that for a certificate every predicate occurring in this certificate holds. In particular, if \(\text{Prime}(p)\) occurs in a certificate, \(p\) is prime.

```
lemma prime-factors-one [simp]: shows prime-factors (Suc 0) = {}  
```
using prime-factorization-1 [where ?a = nat] by simp

lemma prime-factors-of-prime: fixes p :: nat assumes prime p shows prime-factors p = {p}
  using assms by (fact prime-prime-factors)

definition pratt-triple :: nat ⇒ nat ⇒ nat ⇒ bool where
  pratt-triple p a x ←→ x > 0 ∧ (∀q∈prime-factors x. [a ^((p - 1) div q) ≠ 1] (mod p))

lemma pratt-triple-1: p > 1 ⇒ x = 1 ⇒ pratt-triple p a x
  by (auto simp: pratt-triple-def)

lemma pratt-triple-extend:
  assumes prime q pratt-triple p a y
  p > 1 x > 0 x = q * y [a ^((p - 1) div q) ≠ 1] (mod p)
  shows pratt-triple p a x
  proof
    have prime-factors x = insert q (prime-factors y)
      using assms by (simp add: prime-factors-product prime-prime-factors)
    also have ∀r∈... [a ^((p - 1) div r) ≠ 1] (mod p)
      using assms by (auto simp: pratt-triple-def)
    finally show ?thesis using assms unfolding pratt-triple-def by blast
  qed

lemma pratt-triple-imp-prime:
  assumes pratt-triple p a x p
  p > 1 x = p - 1 [a ^((p - 1) - 1) = 1] (mod p)
  shows prime p
  using lehmers-theorem[of p a] assms by (auto simp: pratt-triple-def)

theorem pratt-sound:
  assumes 1: valid-cert c
  assumes 2: t ∈ set c
  shows (t = Prime p −→ prime p) ∧
    (t = Triple p a x −→ (∀q ∈ prime-factors x . [a ^((p - 1) div q) ≠ 1] (mod p)) ∧ 0<x))
  using assms
  proof (induction c arbitrary: p a t)
    case Nil then show ?case by force
    next
    case (Cons y ys)
      { assume y=Triple p a x=1
        then have (∀q ∈ prime-factors x . [a ^((p - 1) div q) ≠ 1] (mod p)) ∧ 0<x
          by simp
      }
      moreover
      { assume x-y: y=Triple p a x x'=1
        hence x>0 using Cons.prems by auto
      }
  qed
obtain $q, z$ where $x = q \cdot z$ \textbf{Prime} $q \in \text{set } ys$ \land \text{Triple} $p \ a \ z \in \text{set } ys$
and $\text{cong}:[a^\sim((p - 1) \ \text{div} \ q) \not\equiv 1] \ (\text{mod } p)$ using $\text{Cons.prems \ x-y by auto}$
then have \textbf{factors-IH}:(\forall \ r \in \text{prime-factors} \ z \ . \ [a^\sim((p - 1) \ \text{div} \ r) \not\equiv 1] \ (\text{mod } p)) \ \text{primen} \ q \ z > 0$
using $\text{Cons.IH \ Cons.prems \ (x>0: \ y= \text{Triple} \ p \ a \ x)$
by \textbf{force}+
then have \textbf{prime-factors} $x = \text{prime-factors} \ z \cup \{q\}$ \ using $x = q \cdot z \ \langle x>0\rangle$
by (simp add: \text{prime-factors-product \ prime-factors-of-prime})
then have $(\forall q \in \text{prime-factors} \ x . \ [a^\sim((p - 1) \ \text{div} \ q) \not\equiv 1] \ (\text{mod } p)) \ \land \ 0 < x$
using \textbf{factors-IH cong \ by} (simp add: $\langle x>0\rangle$)
}
ultimately have \textbf{y-Triple}: $y = \text{Triple} \ p \ a \ x \implies (\forall q \in \text{prime-factors} \ x . \ [a^\sim((p - 1) \ \text{div} \ q) \not\equiv 1] \ (\text{mod } p)) \ \land \ 0<x$
by \textbf{linarith}

moreover
{ assume $y= \text{Prime} \ p \ p>2$ \then have \textbf{prime} $p$ \using \textbf{simp} }
moreover
{ assume $y= \text{Prime} \ p$ \then have $p>1$ \using \textbf{Cons.prems \ by \ simp} }
ultimately have \textbf{y-Prime}: $y = \text{Prime} \ p \implies \text{prime} \ p$ \by \textbf{linarith}

\textbf{1.2 Completeness}

In this section we show completeness of Pratt’s proof system, i.e., we show that for every prime number $p$ there exists a certificate for $p$. We also give an upper bound for the size of a minimal certificate.

The prove we give is constructive. We assume that we have certificates for
all prime factors of $p - 1$ and use these to build a certificate for $p$ from that. It is important to note that certificates can be concatenated.

**Lemma valid-cert-appendI:**

- Assumes valid-cert $r$
- Assumes valid-cert $s$
- Shows valid-cert $(r \&@ s)$
- Using assms

**Proof (induction $r$)**

- Case $(\text{Cons } y \ ys)$ then show ?case by (cases $y$) auto

Qed simp

**Lemma valid-cert-concatI:** $(\forall x \in \text{set } xs. \text{valid-cert } x) \Rightarrow \text{valid-cert } (\text{concat } xs)$

by (induction $xs$) (auto simp add: valid-cert-appendI)

**Lemma size-pratt-le:**

- Fixes $d :: \text{real}$
- Assumes $\forall x \in \text{set } c. \text{size-pratt } x \leq d$
- Shows $\text{size-cert } c \leq \text{length } c * (1 + d)$ using assms

by (induction $c$) (simp-all add: algebra-simps)

**Fun build-fpc :: nat \Rightarrow nat \Rightarrow nat \Rightarrow \text{nat list \Rightarrow pratt list} where**

- $\text{build-fpc } p \ a \ r \ [] = [\text{Triple } p \ a \ r]$
- $\text{build-fpc } p \ a \ r \ (y \#\ ys) = \text{Triple } p \ a \ r \# \text{build-fpc } p \ a \ (r \text{ div } y) \ ys$

The function $\text{build-fpc}$ helps us to construct a certificate for $p$ from the certificates for the prime factors of $p - 1$. Called as $\text{build-fpc } p \ a \ (p - 1) \ qs$ where $qs = q_1 \ldots q_n$ is prime decomposition of $p - 1$ such that $q_1 \ldots q_n = p - 1$, it returns the following list of predicates:

$$(p, a, p - 1), (p, a, \frac{p - 1}{q_1}), (p, a, \frac{p - 1}{q_1 q_2}), \ldots, (p, a, \frac{p - 1}{q_1 \ldots q_n}) = (p, a, 1)$$

I.e., if there is an appropriate $a$ and and a certificate $rs$ for all prime factors of $p$, then we can construct a certificate for $p$ as

Prime $p \# \text{build-fpc } p \ a \ (p - 1) \ qs \&@ rs$

The following lemma shows that $\text{build-fpc}$ extends a certificate that satisfies the preconditions described before to a correct certificate.

**Lemma correct-fpc:**

- Assumes valid-cert $xs \ p > 1$
- Assumes prod-list $qs = r \ r \neq 0$
- Assumes $\forall q \in \text{set } qs. \text{Prime } q \in \text{set } xs$
- Assumes $\forall q \in \text{set } qs. \lfloor a^{-1}(p - 1) \text{ div } q \rfloor \neq 1\ (\text{mod } p)$
- Shows valid-cert $(\text{build-fpc } p \ a \ qs \&@ xs)$
- Using assms

**Proof (induction $qs$ arbitrary: $r$)**

- Case Nil thus ?case by auto
next
case (Cons y ys)
have prod-list ys = r div y using Cons.prems by auto
then have T-in: Triple p a (prod-list ys) ∈ set (build-fpc p a (r div y) ys @ xs)
  by (cases ys) auto

have valid-cert (build-fpc p a (r div y) ys @ xs)
  using Cons.prems by (intro Cons.IH) auto
then have valid-cert ( Triple p a r ≠ build-fpc p a (r div y) ys @ xs)
  using ⟨r ≠ 0⟩ T-in Cons.prems by auto
then show ?case by simp
qed

lemma length-fpc:
length (build-fpc p a r qs) = length qs + 1 by (induction qs arbitrary; r) auto

lemma div-gt-0:
fixes m n :: nat assumes m ≤ n 0 < m shows 0 < n div m
proof
  have 0 < m div m using ⟨0 < m⟩ div-self by auto
  also have m div m ≤ n div m using ⟨m ≤ n⟩ by (rule div-le-mono)
  finally show ?thesis.
qed

lemma size-pratt-fpc:
assumes a ≤ p r ≤ p 0 < a 0 < r 0 < p prod-list qs = r
shows ∀ x ∈ set (build-fpc p a r qs) . size-pratt x ≤ 3 * log 2 p using assms
proof (induction qs arbitrary: r)
case Nil
then have log 2 a ≤ log 2 p log 2 r ≤ log 2 p by auto
then show ?case by simp
next
case (Cons q qs)
then have log 2 a ≤ log 2 p log 2 r ≤ log 2 p by auto
then have log 2 a + log 2 r ≤ 2 * log 2 p by arith
moreover have r div q > 0 using Cons.prems by (fastforce intro: div-gt-0)
moreover hence prod-list qs = r div q using Cons.prems(6) by auto
moreover have r div q ≤ p using ⟨r ≤ p⟩ div-le-dividend[of r q] by linarith
ultimately show ?case using Cons by simp
qed

lemma concat-set:
assumes ∀ q ∈ qs . ∃ c ∈ set cs . Prime q ∈ set c
shows ∀ q ∈ qs . Prime q ∈ set (concat cs)
using assms by (induction cs) auto

lemma p-in-prime-factorsE:
fixes n :: nat
assumes p ∈ prime-factors n 0 < n
obtains $2 \leq p \leq n$ \( p \) dvd \( n \) \( \text{prime} \) \( p \)

proof
from \( \text{assms} \) show \( \text{prime} \) \( p \) by \( \text{auto} \)
then show \( 2 \leq p \) by \( \text{(auto dest: prime-gt-1-nat)} \)

from \( \text{assms} \) show \( p \) dvd \( n \) by \( \text{auto} \)
then show \( p \leq n \) using \( (\mathbb{0} < n) \) by \( \text{(rule dvd-imp-le)} \)
qed

lemma prime-factors-list-prime:
fixes \( n :: \text{nat} \)
assumes \( \text{prime} \) \( n \)
shows \( \exists \) \( qs \). \( \text{prime-factors} \) \( n \) = \( \text{set} \) \( qs \) \( \land \) \( \text{prod-list} \) \( qs \) = \( n \) \( \land \) \( \text{length} \) \( qs \) = \( 1 \)
using \( \text{assms} \) by \( \text{(auto simp add: prime-factorization-prime intro: exI [of - [n]])} \)

lemma prime-factors-list:
fixes \( n :: \text{nat} \)
assumes \( 3 < n \) \( \neg \text{prime} \) \( n \)
shows \( \exists \) \( qs \). \( \text{prime-factors} \) \( n \) = \( \text{set} \) \( qs \) \( \land \) \( \text{prod-list} \) \( qs \) = \( n \) \( \land \) \( \text{length} \) \( qs \) \( \geq \) \( 2 \)
using \( \text{assms} \) proof \( \text{(induction} \ n \ \text{rule: less-induct)} \)
\{ assume \( \text{n div} \) \( p \) \( > 3 \) \( \neg \text{prime} \) \( \text{(n div} \) \( p \) \)
then obtain \( qs \)
where \( \text{prime-factors} \) \( \text{(n div} \) \( p \) \) = \( \text{set} \) \( qs \) \( \text{prod-list} \) \( qs \) = \( \text{(n div} \) \( p \) \) \( \text{length} \) \( qs \) \( \geq \) \( 2 \)
using \( \text{p' by atomize-elim (auto intro: less simp: div-gt-0)} \)
moreover
have \( \text{prime-factors} \) \( \text{(p * (n div} \) \( p \) \)) = \( \text{insert} \) \( p \) \( \text{(prime-factors} \) \( \text{(n div} \) \( p \) \))
using \( \text{(3 < n) (2 \leq p) (p \leq n) (prime} \) \( p \) \))
by \( \text{(auto simp: prime-factors-product div-gt-0 prime-factors-of-prime)} \)
ultimately
have \( \text{prime-factors} \) \( n \) = \( \text{set} \) \( (p \# qs) \) \( \text{prod-list} \) \( (p \# qs) \) = \( n \) \( \text{length} \) \( (p\#qs) \) \( \geq \) \( 2 \)
using \( \text{(p dvd n) by simp-all} \)
hence \( \text{?case by blast} \)
\}
moreover
\{ assume \( \text{prime} \) \( \text{(n div} \) \( p \) \)
than obtain \( qs \)
where \( \text{prime-factors} \) \( \text{(n div} \) \( p \) \) = \( \text{set} \) \( qs \) \( \text{prod-list} \) \( qs \) = \( \text{(n div} \) \( p \) \) \( \text{length} \) \( qs \) = \( 1 \)
using \( \text{prime-factors-list-prime by blast} \)
moreover
have \( \text{prime-factors} \) \( \text{(p * (n div} \) \( p \) \)) = \( \text{insert} \) \( p \) \( \text{(prime-factors} \) \( \text{(n div} \) \( p \) \))
using \( \text{(3 < n) (2 \leq p) (p \leq n) (prime} \) \( p \) \))
by \( \text{(auto simp: prime-factors-product div-gt-0 prime-factors-of-prime)} \)
ultimately
have \( \text{prime-factors} \) \( n \) = \( \text{set} \) \( (p \# qs) \) \( \text{prod-list} \) \( (p \# qs) \) = \( n \) \( \text{length} \) \( (p\#qs) \) \( \geq \) \( 2 \)
using \langle p \text{ dvd } n \rangle \text{ by } simp-all
hence \( ? \text{ case by blast} \)
} note case-prime = this
moreover
\{ assume \( n \text{ div } p = 1 \)
hence \( n = p \) using \( \langle n \rangle \rangle \text{ by } One-leq-div[OF \langle p \text{ dvd } n \rangle] \) \( p' \langle 2 \rangle \) by force
hence \( ? \text{ case using case-prime by force} \)
}\}
moreover
\{ assume \( n \text{ div } p = 2 \)
hence \( ? \text{ case using case-prime by force} \)
}\}
moreover
\{ assume \( n \text{ div } p = 3 \)
hence \( ? \text{ case using case-prime by force} \)
}\}
ultimately show \( ? \text{ case using p' div-gt-0[of p n]} \) case-prime by fastforce
qed

lemma prod-list-ge:
fixes \( xs :: \text{nat list} \)
assumes \( \forall \ x \in \text{set } xs . \ x \geq 1 \)
shows \( \text{prod-list } xs \geq 1 \) using assms by (induction \( xs \)) auto

lemma sum-list-log:
fixes \( b :: \text{real} \)
fixes \( xs :: \text{nat list} \)
assumes \( b : b > 0 \) \( b \neq 1 \)
assumes \( xs : \forall x \in \text{set } xs . \ x \geq b \)
shows \( \sum x \leftarrow(xs . \ log b x) = \log b \) \( \text{prod-list } xs \)
using assms
proof (induction \( xs \))
  case Nil
  thus \( ? \text{ case by simp} \)
next
  case (Cons \( y \) \( ys \))
  have real \( \text{prod-list } ys \rangle > 0 \) using prod-list-ge Cons.premss by fastforce
  thus \( ? \text{ case using log-mult[OF Cons.premss(1-2)]} \) Cons by force
qed

lemma concat-length-le:
fixes \( g :: \text{nat} \Rightarrow \text{real} \)
assumes \( \forall x \in \text{set } xs . \ \text{real (length (} f x \rangle) \leq g x \)
shows \( \text{length (concat (map } f xs)) \leq \sum x \leftarrow(xs . \ g x) \) using assms
by (induction \( xs \)) force+

lemma prime-gt-3-impl-p-minus-one-not-prime:
fixes \( p :: \text{nat} \)
assumes prime p \(p > 3\)
shows \(\neg\) prime \((p - 1)\)

proof
  assume prime \((p - 1)\)
  have \(\neg\) even \(p\) using assms by (simp add: prime-odd-nat)
  hence \(2\) odd \((p - 1)\) by presburger
  then obtain \(q\) where \(p - 1 = 2 \ast q\) ..
  then have \(2 \in\) prime-factors \((p - 1)\) using \((p > 3)\)
    by (auto simp add: prime-factorization-times-prime)
  thus False using prime-factors-of-prime \((p > 3)\) \(\langle\) prime \((p - 1)\) \(\rangle\) by auto

We now prove that Pratt’s proof system is complete and derive upper bounds for the length and the size of the entries of a minimal certificate.

theorem pratt-complete':
  assumes prime \(p\)
  shows \(\exists c.\) \(\text{Prime} p \in\) set \(c\) \(\land\) valid-cert \(c\) \(\land\) length \(c\) \(\leq\) \(6 \ast\) log \(2 p - 4\) \(\land\) \((\forall x \in\) set \(c\). size-pratt \(x\) \(\leq\) \(3 \ast\) log \(2 p\)) using assms

proof (induction \(p\) rule: less-induct)
  case \((\less p)\)
  from \(\langle\) prime \(p\) \(\rangle\) have \(p > 1\) by (rule prime-gt-1-nat)
  then consider \(p = 2\) | \(p = 3\) | \(p > 3\) by force
  thus \(\langle\) case \(\rangle\)
    proof cases
      assume \([\text{simp}]\): \(p = 2\)
      have \(\text{Prime} p \in\) set \([\text{Prime} 2, \text{Triple} 2 1 1]\) by simp
      thus \(\langle\) case \(\rangle\) by fastforce
    next
      assume \([\text{simp}]\): \(p = 3\)
      let \(? \text{cert} = [\text{Prime} 3, \text{Triple} 3 2 2, \text{Triple} 3 2 1, \text{Prime} 2, \text{Triple} 2 1 1]\)
      have length \(? \text{cert} \leq 6 \ast\) log \(2 p - 4\) \(\iff\) \(3 \leq 2 \ast\) log \(2 3\) by simp
      also have \(2 \ast\) log \(2 3\) = log \(2 (3 ^ 2 ::\) real) by (subst log-nat-power) simp-all
      also have \(\ldots = \text{log} 2 9\) by simp
      also have \(3 \leq \text{log} 2 9\) \(\iff\) True by (subst le-log-iff) simp-all
      finally show \(\langle\) case \(\rangle\)
        by (intro exI[where \(x = \)?cert]) (simp add: cong-def)
  next
    assume \(p > 3\)
    have qlp: \(\forall q \in\) prime-factors \((p - 1)\). \(q < p\) using \((\text{prime} p)\)
      by (metis One-nat-def Suc-pred le-imp-less-Suc lessI less-trans p-in-prime-factorsE prime-gt-1-nat zero-less-diff)
    hence factor-certs: \(\forall q \in\) prime-factors \((p - 1)\). \((\exists c . (((\text{Prime} q \in\) set \(c\)) \(\land\) (valid-cert \(c\)) \(\land\) length \(c \leq 6 \ast\) log \(2 q - 4\)) \(\land\) \((\forall x \in\) set \(c\). size-pratt \(x\) \(\leq\) \(3 \ast\) log \(2 q\)))
      by (auto intro: less.IH)
    obtain \(a\) where \(a: [a ^{(p - 1) \mod p}] \langle\forall q. q \in\) prime-factors \((p - 1)\).
      \(\rightarrow [a ^{(p - 1) \div q} \neq 1 \mod p]\rangle\) \(\land\) a-size: \(a > 0 a < p\)
using converse-lehmer[OF ⟨prime p⟩] by blast

have ¬ prime (p − 1) using ⟨p > 3⟩ prime-3-impl-p-minus-one-not-prime ⟨prime p⟩ by auto
have p ≠ 4 using ⟨prime p⟩ by auto

then obtain qs where prod-qs-eq:prod-list qs = p − 1
and qs-eq:set qs = prime-factors (p − 1) and qs-length-eq: length qs ≥ 2
using prime-factors-list[OF (¬ prime (p − 1))] by auto
obtain f where f:∀ q ∈ prime-factors (p − 1) . ∃ c. f q = c
∧ ((Prime q ∈ set c) ∧ (valid-cert c) ∧ length c ≤ 6*log 2 q − 4)
∧ (∀ x ∈ set c. size-pratt x ≤ 3 * log 2 q)

using factor-certs by metis
let ?cs = map f qs
have cs: ∀ q ∈ prime-factors (p − 1) . (∃ c ∈ set ?cs . (Prime q ∈ set c) ∧ (valid-cert c))

using f qs-eq by auto

have cs-cert-size: ∀ c ∈ set ?cs . ∀ x ∈ set c. size-pratt x ≤ 3 * log 2 p
proof
fix c assume c ∈ set (map f qs)
then obtain q where c = f q and q ∈ set qs by auto
hence *:∀ x ∈ set c. size-pratt x ≤ 3 * log 2 q using f qs-eq by blast
have q < p q > 0 using qlp q ∈ set qs qs-eq prime-factors-gt-0-nat by auto
show ∀ x ∈ set c. size-pratt x ≤ 3 * log 2 p
proof
fix x assume x ∈ set c
hence size-pratt x ≤ 3 * log 2 q using * by fastforce
also have . . . ≤ 3 * log 2 p using ⟨q < p ⟩ ⟨q > 0 ⟩ ⟨p > 3⟩ by simp
finally show size-pratt x ≤ 3 * log 2 p .
qed
qed

have cs-valid-all: ∀ c ∈ set ?cs . valid-cert c
using f qs-eq by fastforce

have ∀ x ∈ set (build-fpc p a (p − 1) qs). size-pratt x ≤ 3 * log 2 p
using cs-cert-size a-size ⟨p > 3⟩ prod-qs-eq by (intro size-pratt-fpc) auto
hence ∀ x ∈ set (build-fpc p a (p − 1) qs @ concat ?cs) . size-pratt x ≤ 3 * log 2 p
using cs-cert-size by auto

moreover
have Triple p a (p − 1) ∈ set (build-fpc p a (p − 1) qs @ concat ?cs) by (cases qs) auto
moreover
have valid-cert ((build-fpc p a (p − 1) qs) @ concat ?cs)
proof (rule correct-fpc)
  show valid-cert (concat ?cs)
    using cs-valid-all by (auto simp: valid-cert-concatI)
  show prod-list qs = p − 1 by (rule prod-qs-eq)
  show p − 1 ̸= 0 using prime-gt-1-nat[OF :prime p] by arith
  show ∀ q ∈ set qs . Prime q ∈ set (concat ?cs)
    using concat-set[of prime-factors (p − 1)] cs qs-eq by blast
  show ∀ q ∈ set qs . [a ^ (p − 1) div q] ̸= 1 (mod p) using qs-eq a by auto
qed (insert (p > 3), simp-all)
moreover
  { let ?k = length qs
    have qs-ge-2:∀ q ∈ set qs . q ≥ 2 using qs-eq
      by (auto intro: prime-ge-2-nat)
    have ∀ x ∈ set qs . real (length (f x)) ≤ 6 * log 2 (real x) − 4 using f qs-eq by blast
      hence length (concat ?cs) ≤ (∑ q−qs . 6 * log 2 q − 4) using concat-length-le
        by fast
      hence length (Prime p # ((build-fpc p a (p − 1) qs) @ concat ?cs))
        ≤ ((∑ q−(map real qs) . 6 * log 2 q − 4) + ??k + 2)
        by (simp add: o-def length-fpc)
      also have . . . = (6 * (∑ qs−(map real qs) . log 2 q) + (-4 * real ??k) + ??k + 2)
        by (simp add: o-def sum-list-subtractf sum-list-triv sum-list-const-mult)
      also have . . . ≤ 6 * log 2 (p − 1) − 4 using (??k ≥ 2) prod-qs-eq sum-list-log[of
        2 qs] qs-ge-2
        by force
      also have . . . ≤ 6 * log 2 p − 4 using log-le-cancel-iff[of 2 p − 1 p] (p > 3) by force
    ultimately have length (Prime p # ((build-fpc p a (p − 1) qs) @ concat ?cs))
      ≤ 6 * log 2 p − 4 by linarith
  }
  ultimately obtain c where c:Triple p a (p − 1) ∈ set c valid-cert c
    length (Prime p # c) ≤ 6 * log 2 p − 4
    (∀ x ∈ set c . size-pratt x ≤ 3 * log 2 p) by blast
  hence Prime p ∈ set (Prime p # c) valid-cert (Prime p # c)
    (∀ x ∈ set (Prime p # c) . size-pratt x ≤ 3 * log 2 p)
    using a’prime p by (auto simp: Primes,prime-gt-Suc-0-nat)
    thus ?case using c by blast
  qed
  qed

We now recapitulate our results. A number p is prime if and only if there is a certificate for p. Moreover, for a prime p there always is a certificate whose size is polynomially bounded in the logarithm of p.

**corollary pratt:**

prime p ←→ (∃ c. Prime p ∈ set c ∧ valid-cert c)
using pratt-complete’ pratt-sound(1) by blast

**corollary pratt-size:**
assumes prime p
shows \( \exists c. \text{Prime } p \in \text{set } c \land \text{valid-cert } c \land \text{size-cert } c \leq (6 \ast \log 2 \ p - 4) \ast (1 + 3 \ast \log 2 \ p) \)
proof
obtain c where c: Prime p \in \text{set } c \land \text{valid-cert } c \land \text{size-cert } c \leq (6 \ast \log 2 \ p - 4) \ast (1 + 3 \ast \log 2 \ p)
using pratt-complete' assms by blast
hence size-cert c \leq \text{length } c \ast (1 + 3 \ast \log 2 \ p) by (simp add: size-cert-le)
also have \( \ldots \leq (6 \ast \log 2 \ p - 4) \ast (1 + 3 \ast \log 2 \ p) \) using len by simp
finally show \( ?\text{thesis} \) using c by blast
qed

1.3 Efficient modular exponentiation
locale efficient-power =
fixes f :: 'a \Rightarrow 'a \Rightarrow 'a
assumes f-assoc: \( \forall x z. \ f x \ (f x z) = f \ (f x x) z \)
begin
function efficient-power :: 'a \Rightarrow 'a \Rightarrow \text{nat} \Rightarrow 'a
where
\begin{align*}
\text{efficient-power } y \ x \ 0 &= y \\
\text{efficient-power } y \ x \ (\text{Suc } 0) &= f x y \\
\text{efficient-power } y \ x \ n &= \text{efficient-power } y \ (f x x) \ (n \div 2) & \text{if } n \neq 0 \text{ and even } n \\
\text{efficient-power } y \ x \ n &= \text{efficient-power } (f x y) \ (f x x) \ (n \div 2) & \text{if } n \neq 1 \text{ and odd } n
\end{align*}
by force+
termination by (relation measure \( \text{snd} \circ \text{snd} \)) (auto elim: oddE)

lemma efficient-power-code: efficient-power y x n =
\begin{align*}
\text{if } n = 0 \text{ then } y \\
\text{else if } n = 1 \text{ then } f x y \\
\text{else if even } n \text{ then efficient-power } y \ (f x x) \ (n \div 2) \\
\text{else efficient-power } (f x y) \ (f x x) \ (n \div 2)
\end{align*}
by (induction y x n rule: efficient-power.induct) auto

lemma efficient-power-correct: efficient-power y x n = (f x ^^ n) y
proof
have [simp]: f ^^ 2 = (\lambda x. f \ (f x)) for f :: 'a \Rightarrow 'a
by (simp add: eval-nat-numeral o-def)
show \( ?\text{thesis} \)
by (induction y x n rule: efficient-power.induct)
(auto elim!: evenE oddE simp: funpow-mult [symmetric] funpow-Suc-right f-assoc)
simp del: funpow.simps(2))
qed
end
interpretation \( \text{mod-exp-nat}: \text{efficient-power} \ \lambda x \ y :: \text{nat}. (x * y) \mod m \)
by standard (simp add: mod-mult-left-eq mod-mult-right-eq mult-ac)

definition \( \text{mod-exp-nat-aux} \) where \( \text{mod-exp-nat-aux} = \text{mod-exp-nat.efficient-power} \)

lemma \( \text{mod-exp-nat-aux-code} \) [code]:
\[
\text{mod-exp-nat-aux} \ m \ y \ x \ n =
\begin{cases}
  (\text{if} \ n = 0 \text{ then } y) \\
  (\text{else if} \ n = 1 \text{ then } (x * y) \mod m) \\
  (\text{else if} \ \text{even} \ n \text{ then } \text{mod-exp-nat-aux} \ m \ y ((x * x) \mod m) (n \div 2)) \\
  (\text{else mod-exp-nat-aux} \ m ((x * y) \mod m) ((x * x) \mod m) (n \div 2))
\end{cases}
\]

unfolding \( \text{mod-exp-nat-aux-def} \) by (rule mod-exp-nat.efficient-power-code)

lemma \( \text{mod-exp-nat-aux-correct} \):
\[
\text{mod-exp-nat-aux} \ m \ y \ x \ n \mod m = (x ^ n * y) \mod m
\]
proof −
have \( \text{mod-exp-nat-aux} \ m \ y \ x \ n = ((\lambda y. x * y \mod m) ^ n) y \)
by (simp add: mod-exp-nat-aux-def mod-exp-nat.efficient-power-correct)
also have \( ((\lambda y. x * y \mod m) ^ n) y \mod m = (x ^ n * y) \mod m \)
proof (induction \( n \))
case \( \text{Suc} \ n \)
hence \( x * ((\lambda y. x * y \mod m) ^ n) y \mod m = x * x ^ n * y \mod m \)
by (metis mod-mult-right-eq mult.assoc)
thus \( \text{case by auto} \)
qed auto
finally show \( \text{?thesis} \).
qed

definition \( \text{mod-exp-nat} :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \)
where [code-abbrev]: \( \text{mod-exp-nat} \ b \ e \ m = (b ^ e) \mod m \)

lemma \( \text{mod-exp-nat-code} \) [code]: \( \text{mod-exp-nat} \ b \ e \ m = \text{mod-exp-nat-aux} \ m \ 1 \ b \ e \mod m \)
by (simp add: mod-exp-nat-def mod-exp-nat-aux-correct)

lemmas [code-unfold] = cong-def

lemma \( \text{eval-mod-exp-nat-aux} \) [simp]:
\[
\begin{align*}
\text{mod-exp-nat-aux} \ m \ y \ 0 \ 0 & = y \\
\text{mod-exp-nat-aux} \ m \ y \ (\text{Suc} \ 0) & = (x * y) \mod m \\
\text{mod-exp-nat-aux} \ m \ y \ (\text{numeral} \ (\text{num.Bit0} \ n)) & = \\
\text{mod-exp-nat-aux} \ m \ y \ (x ^ 2 \mod m) & (\text{numeral} \ n) \\
\text{mod-exp-nat-aux} \ m \ y \ (\text{numeral} \ (\text{num.Bit1} \ n)) & = \\
\text{mod-exp-nat-aux} \ m ((x * y) \mod m) (x ^ 2 \mod m) & (\text{numeral} \ n)
\end{align*}
\]
proof −
define \( n' \) where \( n' = (\text{numeral} \ n :: \text{nat}) \)
have [simp]: \( n' \neq 0 \) by (auto simp: n'-def)
show \( \text{mod-exp-nat-aux } m \ y \ x \ 0 = y \) and \( \text{mod-exp-nat-aux } m \ y \ x \ (\text{Suc } 0) = (x \ast y) \mod m \)
by (simp-all add: mod-exp-nat-aux-def)

have \( \text{numeral } (\text{num.Bit0 } n) = (2 \ast n') \)
by (subst numeral.numeral-Bit0) (simp del: arith-simps add: n'-def)
also have \( \text{mod-exp-nat-aux } m \ y \ x \ \ldots \ = \text{mod-exp-nat-aux } m \ y \ (x^2 \mod m) \ n' \)
by (subst mod-exp-nat-aux-code) (simp-all add: power2-eq-square)
finally show \( \text{mod-exp-nat-aux } m \ y \ x \ (\text{numeral } (\text{num.Bit0 } n)) = \text{mod-exp-nat-aux } m \ y \ (x^2 \mod m) \ (\text{numeral } n) \)
by (simp add: n'-def)

have \( \text{numeral } (\text{num.Bit1 } n) = \text{Suc } (2 \ast n') \)
by (subst numeral.numeral-Bit1) (simp del: arith-simps add: n'-def)
also have \( \text{mod-exp-nat-aux } m \ y \ x \ \ldots \ = \text{mod-exp-nat-aux } m \ (\text{Suc } 0) \ (x^2 \mod m) \ n' \)
by (subst mod-exp-nat-aux-code) (simp-all add: power2-eq-square)
finally show \( \text{mod-exp-nat-aux } m \ y \ x \ (\text{numeral } (\text{num.Bit1 } n)) = \text{mod-exp-nat-aux } m \ ((x \ast y) \mod m) \ (x^2 \mod m) \ (\text{numeral } n) \)
by (simp add: n'-def)
qed

lemma eval-mod-exp [simp]:
\begin{align*}
\text{mod-exp-nat } b' \ 0 \ m' &= 1 \mod m' \\
\text{mod-exp-nat } b' \ 1 \ m' &= b' \ mod m' \\
\text{mod-exp-nat } b' \ (\text{Suc } 0) \ m' &= b' \mod m' \\
\text{mod-exp-nat } b' \ e' \ 0 &= b' \mod e' \\
\text{mod-exp-nat } b' \ e' \ 1 &= 0 \\
\text{mod-exp-nat } 0 \ 1 \ m' &= 0 \\
\text{mod-exp-nat } 0 \ (\text{Suc } 0) \ m' &= 0 \\
\text{mod-exp-nat } 0 \ (\text{numeral } e) \ m' &= 0 \\
\text{mod-exp-nat } 1 \ e' \ m' &= 1 \mod m' \\
\text{mod-exp-nat } (\text{Suc } 0) \ e' \ m' &= 1 \mod m' \\
\text{mod-exp-nat } (\text{numeral } b) \ (\text{numeral } e) \ (\text{numeral } m) &= \\
&= \text{mod-exp-nat-aux } (\text{numeral } m) \ 1 \ (\text{numeral } b) \ (\text{numeral } e) \ mod numeral m \\
by (simp-all add: mod-exp-nat-def mod-exp-nat-aux-correct)
\end{align*}

1.4 Executable certificate checker

lemmas [code] = valid-cert.simps(1)

context
begin

lemma valid-cert-Const1 [code]:
valid-cert (Prime p #\ x) \longleftrightarrow \\
p > 1 \land (\exists t \in set \ x. \ case t of Prime - \Rightarrow False | \\
\text{Triple } p' \ a \ x \Rightarrow p' = p \land x = p - 1 \land \text{mod-exp-nat } a \ (p - 1) \ p = 1 ) \land valid-cert

14
\(xs\)

(is \(?lhs = ?rhs\)

proof

assume \(?lhs\) thus \(?rhs\) by (auto simp: mod-exp-nat-def cong-def split: pratt.splits)

next

assume \(?rhs\)

hence \(p > 1\) valid-cert \(xs\) by blast

moreover from \(?rhs\) obtain \(t\) where \(t \in set xs\) case \(t\) of Prime - \(\Rightarrow False\) |

\(Triple p' a x \Rightarrow p' = p \land x = p - 1 \land [a^{(p-1)} = 1] (mod p)\)

by (auto simp: cong-def mod-exp-nat-def cong: pratt.case-cong)

ultimately show \(?lhs\) by (cases \(t\)) auto

qed

private lemma Suc-0-mod-eq-Suc-0-iff:

\(Suc 0 mod n = Suc 0 \iff n \neq Suc 0\)

proof

consider \(n = 0\) \(\mid n = Suc 0\) \(\mid n > 1\) by (cases \(n\)) auto

thus \(?thesis\) by cases auto

qed

private lemma Suc-0-eq-Suc-0-mod-iff:

\(Suc 0 = Suc 0 mod n \iff n \neq Suc 0\)

using Suc-0-mod-eq-Suc-0-iff by (simp add: eq-commute)

lemma valid-cert-Cons2 [code]:

\(valid-cert (Triple p a x \# xs) \iff x > 0 \land p > 1 \land (x = 1 \lor (\exists t \in set xs. case t of Prime - \Rightarrow False | \nTriple p' a' y \Rightarrow p' = p \land a' = a \land y dvd x \land (let q = x div y in Prime q \in set xs \land mod-exp-nat a ((p-1) div q) p \neq 1))))\)

\(\land valid-cert xs\)

(is \(?lhs = ?rhs\)

proof

assume \(?lhs\)

from \(?lhs\) have pos: \(x > 0\) and gt-1: \(p > 1\) and valid: valid-cert \(xs\) by simp-all

show \(?rhs\)

proof (cases \(x = 1\))

case True

with \(?lhs\) show \(?thesis\) by auto

next

case False

with \(?lhs\) have \((\exists q y. x = q * y \land Prime q \in set xs \land Triple p a y \in set xs \land [a^{((p-1) div q)} \neq 1] (mod p))\) by auto

then guess \(q y\) by (elim exE conjE) note \(qy = this\)

hence \((\exists t \in set xs. case t of Prime - \Rightarrow False | \nTriple p' a' y \Rightarrow p' = p \land a' = a \land y dvd x \land (let q = x div y in Prime q \in set xs \land mod-exp-nat a ((p-1) div q) p \neq 1))\)

using pos gt-1 by (intro bexI [of - Triple p a y])

(auto simp: Suc-0-mod-eq-Suc-0-iff Suc-0-eq-Suc-0-mod-iff cong-def mod-exp-nat-def)

with pos gt-1 valid show \(?thesis\) by blast
qed
next
assume ?rhs

hence \( \text{pos: } x > 0 \) and \( \text{gt-1: } p > 1 \) and \( \text{valid: } \text{valid-cert } xs \) by simp-all

show ?lhs

proof (cases \( x = 1 \))

  case True

  with \( \langle ?rhs \rangle \)

  show ?thesis

  by auto

next

  case False

  with \( \langle ?rhs \rangle \)

  obtain \( t \) where \( t \in \text{set } xs \) case \( t \) of

  Prime \( x \) \Rightarrow False

  | Triple \( p'\ a'\ y \Rightarrow p' = p \wedge a' = a \wedge y \text{ dvd } x \wedge (\text{let } q = x \text{ div } y \text{ in Prime } q \in \text{set } xs \wedge \text{mod-exp-nat } a \ ((p - 1) \text{ div } q) \neq 1) \) by auto

  then obtain \( y \) where \( y : t = \text{Triple } p a y \text{ dvd } x \text{ let } q = x \text{ div } y \text{ in Prime } q \in \text{set } xs \wedge \text{mod-exp-nat } a \ ((p - 1) \text{ div } q) \neq 1 \)

  by (auto simp cong-def Let-def mod-exp-nat-def Suc-0-mod-eq-Suc-0-iff Suc-0-eq-Suc-0-mod-iff)

  define \( q \) where \( q = x \text{ div } y \)

  have \( \exists q. x = q * y \wedge \text{Prime } q \in \text{set } xs \wedge \text{Tripl} e p a y \in \text{set } xs \wedge [a^-((p - 1) \text{ div } q) \neq 1] \) (mod p)

  by (auto simp: Let-def q-def)

  with \( \text{pos gt-1 valid } \) show ?thesis by simp

qed

declare valid-cert.simps(2,3) [simp del]

lemmas eval-valid-cert = valid-cert.simps(1) valid-cert-Cons1 valid-cert-Cons2

end

The following alternative tree representation of certificates is better suited for efficient checking.

datatype pratt-tree = Pratt-Node nat × nat × pratt-tree list

fun pratt-tree-number where

pratt-tree-number (Pratt-Node \( (n, -) \)) = n

The following function checks that a given list contains all the prime factors of the given number.

fun check-prime-factors-subset :: nat ⇒ nat list ⇒ bool where

check-prime-factors-subset \( n \) \[ \leftrightarrow n = 1 \]
| check-prime-factors-subset \( n \) \( (p \# ps) \) \[ \leftrightarrow (\text{if } n = 0 \text{ then False else} \)
  (if \( p > 1 \wedge p \text{ dvd } n \) then check-prime-factors-subset \( n \text{ div } p \) \( (p \# ps) \)
else check-prime-factors-subset \( n \text{ ps} \) )

16
lemma check-prime-factors-subset-0 [simp]: \( \neg \text{check-prime-factors-subset } 0 \text{ ps} \)
by (induction ps) auto

lemmas [simp del] = check-prime-factors-subset.simps(2)

lemma check-prime-factors-subset-Cons [simp]:
check-prime-factors-subset \((\text{Suc } 0) \text{ (p # ps)}\) \(\iff\) check-prime-factors-subset \((\text{Suc } 0) \text{ ps}\)
check-prime-factors-subset \(1 \text{ (p # ps)}\) \(\iff\) check-prime-factors-subset \(1 \text{ ps}\)
p \(> 1 \implies p \text{ dvd numeral } n \implies \text{check-prime-factors-subset } \text{(numeral } n \text{) (p # ps)}\)
\(\iff\)
check-prime-factors-subset \((\text{numeral } n \text{ div p}) \text{ (p # ps)}\)
p \(\leq 1 \lor \neg p \text{ dvd numeral } n \implies \text{check-prime-factors-subset } \text{(numeral } n \text{) (p # ps)}\)
\(\iff\)
check-prime-factors-subset \((\text{numeral } n) \text{ ps}\)
by (subst check-prime-factors-subset.simps; force+)

lemma check-prime-factors-subset-correct:
assumes check-prime-factors-subset \(n \text{ ps list-all prime ps}\)
shows prime-factors \(n \subseteq \text{set ps}\)
using assms
proof (induction \(n \text{ ps}\) rule: check-prime-factors-subset.induct)
case (2 \(n \text{ p ps}\))
note \(\ast\) = this
from 2.prems have prime \(p\) and \(p > 1\)
by (auto simp: prime-gt-Suc-0-nat)
consider \(n = 0 \mid n > 0 p \text{ dvd } n \mid n > 0 \neg(p \text{ dvd } n)\)
by blast
thus \(?case\)
proof cases
  case 2
  hence \(n \text{ div } p > 0\) by auto
  hence prime-factors \((n \text{ div } p) * p\) = insert \(p\) (prime-factors \((n \text{ div } p)\))
  using \((p > 1) \cdot \text{prime } p\) by (auto simp: prime-factors-product prime-prime-factors)
  also have \((n \text{ div } p) * p = n\)
  using 2 by auto
  finally show \(?thesis using\) \(2 \cdot (p > 1)\) * 
  by (auto simp: check-prime-factors-subset.simps(2)[of \(n\)])
next
  case 3
  with \(\ast\) and \((p > 1)\) show \(?thesis\)
  by (auto simp: check-prime-factors-subset.simps(2)[of \(n\)])
qed auto
qed auto

fun valid-pratt-tree where
valid-pratt-tree \( (\text{Pratt-Node} \ (n, \ a, \ ts)) \) \( \iff \)
\[
n \geq 2 \land
\text{check-prime-factors-subset} \ ((n - 1) \ (\text{map} \ \text{pratt-tree-number} \ ts)) \land
\left[ a \ 
\left( \frac{n - 1}{\text{pratt-tree-number} \ t} \right) \right] \ (\mod n) \land
(\forall t \in \text{set} \ ts. \ [a \ 
\left( \frac{n - 1}{\text{pratt-tree-number} \ t} \right) \neq 1] \ (\mod n)) \land
(\forall t \in \text{set} \ ts. \ \text{valid-pratt-tree} \ t)
\]

\text{lemma valid-pratt-tree-code [code]:}
\[
\text{valid-pratt-tree} \ (\text{Pratt-Node} \ (n, \ a, \ ts)) \iff
n \geq 2 \land
\text{check-prime-factors-subset} \ ((n - 1) \ (\text{map} \ \text{pratt-tree-number} \ ts)) \land
\text{mod-exp-nat} \ a \ ((n - 1)) \ n = 1 \land
(\forall t \in \text{set} \ ts. \ \text{mod-exp-nat} \ a \ ((n - 1) \ (\text{div} \ \text{pratt-tree-number} \ t)) \ n \neq 1) \land
(\forall t \in \text{set} \ ts. \ \text{valid-pratt-tree} \ t)
\]
\]
\text{by (simp add: mod-exp-nat-def cong-def)}

\text{lemma valid-pratt-tree-imp-prime:}
\text{assumes valid-pratt-tree} \ t
\text{shows prime} \ (\text{pratt-tree-number} \ t)
\text{using assms}
\text{proof (induction} \ t \ \text{rule: valid-pratt-tree.induct)}
\text{case } 1 \ n \ a \ ts
\text{from } 1 \ \text{have prime-factors} \ ((n - 1)) \subseteq \text{set} \ (\text{map} \ \text{pratt-tree-number} \ ts)
\text{by (intro check-prime-factors-subset-correct) (auto simp: list.pred-set)}
\text{with } 1 \ \text{show ?case}
\text{by (intro lehmers-theorem[where} \ a = \ a]) \ \text{auto}
\text{qed}

\text{lemma valid-pratt-tree-imp-prime':}
\text{assumes PROP} \ (\text{Trueprop} \ (\text{valid-pratt-tree} \ (\text{Pratt-Node} \ (n, \ a, \ ts)))) \equiv \text{PROP} \ (\text{Trueprop} \ \text{True})
\text{shows prime} \ n
\text{proof –}
\text{have valid-pratt-tree} \ (\text{Pratt-Node} \ (n, \ a, \ ts))
\text{by (subst assms) auto}
\text{from valid-pratt-tree-imp-prime}[OF this] \ \text{show ?thesis by simp}
\text{qed}

1.5 Proof method setup

\text{theorem lehmers-theorem':}
\text{fixes} \ p :: \ \text{nat}
\text{assumes list-all prime} \ ps \ a \equiv a \ n \equiv n
\text{assumes list-all} \ (\forall p. \ \text{mod-exp-nat} \ a \ ((n - 1) \ (\text{div} \ p)) \ n \neq 1) \ \text{ps} \ \text{mod-exp-nat} \ a \ (n - 1) \ n = 1
\text{assumes check-prime-factors-subset} \ ((n - 1)) \ \text{ps} \ 2 \leq n
\text{shows prime} \ n
\text{using assms check-prime-factors-subset-correct}[OF assms(6,1)]
\text{by (intro lehmers-theorem[where} \ a = \ a]) \ \text{(auto simp: cong-def mod-exp-nat-def)
lemma list-all-ConsI: $P \ x \implies list-all\ P\ xs \implies list-all\ P\ (x\ #\ xs)$
  by simp

ML-file ⟨pratt.ML⟩

method-setup pratt = ⟨
  Scan.lift (Pratt.tac-config-parser -- Scan.option Pratt.cert-cartouche) >>
  (fn (config, cert) => fn ctxt => SIMPLE-METHOD (HEADGOAL (Pratt.tac
          config cert ctxt))))⟩

\textit{Prove primality of natural numbers using Pratt certificates.}

The proof method replays a given Pratt certificate to prove the primality of a given number. If no certificate is given, the method attempts to compute one. The computed certificate is then also printed with a prompt to insert it into the proof document so that it does not have to be recomputed the next time.

The format of the certificates is compatible with those generated by Mathematica. Therefore, for larger numbers, certificates generated by Mathematica can be used with this method directly.

lemma prime (47 :: nat)
  by (pratt (silent))

lemma prime (2503 :: nat)
  by pratt

lemma prime (7919 :: nat)
  by pratt

lemma prime (131059 :: nat)
  by (pratt ⟨{131059, 2, [2, 3, 2, [2]}, [809, 3, [2, [101, 2, [2, [5, 2, [2]]]]]]}⟩)

theory Pratt-Certificate-Code
imports
  Pratt-Certificate
  HOL-Library.Code-Target-Numeral
begin

1.6 Code generation for Pratt certificates

The following one-time setup is required to set up code generation for the certificate checking. Other theories importing this theories do not have to do this again.

setup ⟨
We can now evaluate the efficiency of the procedure on some examples.

**Lemma** prime (131059 :: nat)

*by* (pratt (code))

**Lemma** prime (100000007 :: nat)

*by* (pratt (code))

**Lemma** prime (8504276003 :: nat)

*by* (pratt (code))

**Lemma** prime (52759926861157 :: nat)

*by* (pratt (code))

**Lemma** prime (39070009756439177203 :: nat)

*by* (pratt (code))

\[
\begin{align*}
\text{Context} & \text{ theory-map} \ (\text{Pratt.set-up-valid-cert-code-conv}) \\
& \left( \{ \text{computation-check} \right.
\left. \text{ terms: Trueprop valid-pratt-tree 0::nat 1::nat 2::nat 3::nat 4::nat} \}
\right) \text{ datatypes: pratt-tree list nat \times nat \times pratt-tree list nat\}
\end{align*}
\]

\[
\text{by}
\]

\[
\text{by}
\]

\[
\text{by}
\]

\[
\text{by}
\]
V. R. Pratt. Every prime has a succinct certificate.

References