The Correctness of the AKS Primality Test in Coq[*]

Flávio L. C. de Moura[1], **Ricardo Tadeu**[1]

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)
Caixa Postal 4466 – CEP 70910-900 – Brasília – DF – Brazil

flavio@cic.unb.br, rictad@gmail.com

Abstract. *In 2004, Agrawal, Kayal and Saxena published the paper "PRIMES is in P" where they present the first polinomial algorithm that can decide in deterministic polynomial time if a given number is prime or not. This algorithm, known as AKS, is of relevance for both mathematical and computer science comunities. In this work, we present our first results in order to get a complete formalization of AKS. The formal verification is done in the Coq proof assistant.*

1. Introduction

A primality test is a method for determining if a given number is prime or not. The study of prime numbers is of great importance since the ancient Greece, when the most popular method for computing primes, the Sieve of Eratosthenes, was created. Although it is still in use for educational purposes, the Sieve of Eratosthenes is not used to compute large prime numbers because it is inefficient.

In the 17th century, the work of Fermat, among others, provided a great improvement on the knowledge about prime numbers. In fact, Fermat's Little Theorem provides a better way to check if a given number is prime:

Theorem 1 (Fermat's Little Theorem) *For any numbers p and a, if p is prime then*

$$a^p = a \ (mod\ p) \tag{1}$$

The $(p-1)$-th power of a can be efficiently computed by repeatedly squaring. However, Fermat's Little Theorem is not a correct primality test because some composite numbers satisfy equation (1), for some a. Despite of this fact, it became the basis of many efficient primality tests.

Before the 1970s, these results were restricted to pure mathematics. This changed with the advent of cryptography, and since then computers have been largely used to compute large prime numbers [Ribenboim 1995, Ribenboim 2004]. In fact, primality testing has been investigated intensively [Pratt 1975, Miller 1976, Rabin 1980, Solovay and Strassen 1977, Adleman et al. 1983, Goldwasser and Kilian 1986, Adleman and Huang 1992], and the most efficient algorithms for primality testing are, nowadays, probabilistic.

In this work, we present our initial results in order to prove formally in Coq [Coq Development Team 2007], the correctness of the first (deterministic) algorithm that runs in polynomial time: the AKS algorithm [Agrawal et al. 2004]. This

[*]Research partially supported by Grant 8-004/2007 of FAP-DF

algorithm is of historical importance, and although probabilistic algorithms are much more efficient, we believe that our formalization allows a better understanding of the algorithm, and can give insights for improving its efficiency. In addition, there are several reasons for formalizing Mathematics [Barendregt and Wiedijk 2005, Friedman 2006, Kaliszyk and Wiedijk 2007]:

1. If one takes an existing informal textbook proof, and considers the gaps between the steps in that proof, a human mathematician will not even be consciously aware of the majority of those. In this sense, proof assistants should be important in taking care of all of these details.
2. It provides a natural way to approximate research areas that are in principle distinct, like Pure Mathematics and Formal Methods. In fact, software/hardware verification and formalization of Mathematics are closely related to each other, where the verification process must certificate all the underlying theories.
3. It is important for proof theory because sophisticated formal proofs can be very informative about the proof structure.

This is an ongoing work whose main goal is to produce a fully verified version of the AKS algorithm in Coq.

2. The Coq Proof Assistant

Coq is a formal proof management system based on a very expressive higher-order logic, called Calculus of Inductive Constructions. It has been successfully used both in industry and academia. It is based on the Curry-Howard isomorphism, where a formula A is provable iff it is inhabited, i.e., a proof of a formula A corresponds to a λ-term t of type A, written $t : A$ [Barendregt 1992, Barendregt and Geuvers 2001]. As a simple example, let A be the propositional formula (P → Q) → (Q → R) → P → R, where P, Q and R are propositional variables. In the Coq language, this can be done by first opening a section (just delimit the scope of local variables), followed by the declaration of the propositional variables P, Q and R:

```
Section direct_proof.
  Variables P Q R : Prop.
  Theorem trans : (P → Q) → (Q → R) → P → R.
  Proof.
    exact (fun (x:P→Q) (y:Q →R)(z:P) ⇒ y(x z)).
  Qed.
End direct_proof.
```

A proof is given by a λ-term with type (P → Q) → (Q → R) → P → R, i.e., by the λ-term $\lambda_{x:P\to Qy:Q\to Rz:P}.y(x\ z)$ that is written in Coq as (fun (x:P→Q) (y:Q →R)(z:P) ⇒ y(x z)). Nevertheless, theorems are not so simple in real life, and proof construction is usually a challenging task. Proof assistants based on type theory allow us to build proofs interactively. The user types in the so called *tactics*, guiding Coq to construct a proof. For instance, the word exact in the above example is a tactic that tells Coq to type check the given term (fun (x:P → Q)(y:Q →R)(z:P) ⇒ y(x z)), also known as *proof object* [Barendregt and Geuvers 2001], against the given type (P → Q) → (Q → R) → P → R. Once a proof is done, it can be used as a lemma in other proofs.

As a more interesting example (related to the subject of this article), we present a proof of Fermat's Little Theorem using induction.

Theorem 2 (Fermat's Little Theorem) *If n is a prime number then for any integer a, we have that $a^n = a \ (mod\ n)$.*

The whole proof is divided in two parts: we present the first one, called `fermat_little_def_pos`, that considers the case when $a \geq 0$. This is a classical proof by induction on a, where the induction base is trivial. The whole proof is available as a Coq file at `http://www.cic.unb.br/∼flavio/publications.html`. First of all, notice that $(a+1)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} = a^n + \sum_{k=1}^{n-1} \binom{n}{k} a^{n-k} + 1$. The key of the proof consists in proving that if n is prime then $\binom{n}{k} = 0 \ (\text{mod}\, n), \forall 1 \leq k \leq n-1$. Therefore, we have that $(a+1)^n = a^n + 1 \ (\text{mod}\ n)$, and we are done after an application of the induction hypothesis. This lemma is formalized in Coq as follows:

```
Lemma fermat_little_def_pos: forall n a: Z, a>=0 ->
                   prime n -> a ^ n mod n = a mod n.
```

After the initialization of the proof mode, we get the following:

```
1 subgoal
  ============================
forall n a : Z, a >= 0 -> prime n -> a ^ n mod n = a mod n
```

Therefore, we have one subgoal to prove that is displayed below the dashed line. The statements on the left hand side of implications are introduced as hypothesis above the dashed line. The induction base corresponds to the case $a = 0$ and is straightforward after a simplification step:

```
  n : Z
  a : Z
  H0 : a >= 0
  H1 : prime n
  H2 : 2 <= n
  H : (2 <= Zabs_nat n)%nat
  ============================
   0 ^ n mod n = 0 mod n
```

The induction step is as follows and requires the binomial theorem to expand the term $(\mathtt{n0}+1)^{\mathtt{n}}$:

```
  n : Z
  a : Z
  H0 : a >= 0
  H1 : prime n
  H2 : 2 <= n
  H : (2 <= Zabs_nat n)%nat
  n0 : nat
  IHn0 : Z_of_nat n0 ^ n mod n = Z_of_nat n0 mod n
  ============================
   (Z_of_nat n0 + 1) ** Zabs_nat n mod n =
   (Z_of_nat n0 + 1) mod n
```

In the above goal `**` represents exponentiation for natural powers, while `^` is used for integer powers. After an application of the lemma `bin_mod_n_prime`, we can apply the induction hypothesis `IHn0` and the proof is done. This is just the general overview of the proof, but the complete proof is about 80 lines of Coq code.

3. The AKS Algorithm

In 2004, Agrawal, Kayal and Saxena published the paper *"PRIMES is in P"* where they present a deterministic polynomial time algorithm for primality [Agrawal et al. 2004]. The main idea of this algorithm comes from a previous result of Agrawal and Biswas [Agrawal and Biswas 2003], where they present a deterministic primality test:

Theorem 3 *Let n and a be integers with $gcd(n, a) = 1$. Then, n is prime iff*

$$(x + a)^n = x^n + a \tag{2}$$

in $\mathbb{Z}_n[x]$.

This is a nice primality test, nevertheless it runs in exponential time! In fact, we need to evaluate n coefficients of the LHS of equation (2) in the worst case. A way to reduce the complexity of the computation of $(x + a)^n$ is to consider equation (2) in the quotient ring $\mathbb{Z}_n[x]/(x^r - 1)$, for some adequate integer r. This quotient ring is formed by the classes of polinomials Note that in this quotient ring, the degree of the polynomials are limited by r. The new identity is presented as follows:

$$(x + a)^n = x^n + a \textit{ in } \mathbb{Z}_n[x]/(x^r - 1) \tag{3}$$

Note that all primes that satisfy equation (2) also satisfy (3), but some composite numbers satisfy (3) and hence it is no longer a characterization of the prime numbers. In fact, for $n = 4$, $a = 3$ and $r = 2$, we have that $gcd(4, 3) = 1$ and $(x + 3)^4 = x^4 + 3$ in $\mathbb{Z}_4[x]/(x^2 - 1)$. Nevertheless, a characterization of the prime numbers can be partially recovered in the sense that, if for an appropriated r the equation (3) is satisfied for several a's, then n must be a prime power. The number of a's and the appropriate r are both bounded by a polynomial in $log\ n$. A modified version of the AKS algorithm due to H. W. Lenstra Jr. is presented in what follows. The correctness of this algorithm is given by the following theorem:

Theorem 4 *The algorithm AKS returns* prime *if, and only if, the input n is a prime number.*

A detailed proof of this theorem is given in [Agrawal et al. 2004], where it is divided in two steps:

Lemma 1 *If the input n is a prime number then algorithm AKS returns* prime.

Lemma 2 *If the algorithm AKS returns* prime *then the input n is a prime number.*

In the next section we present the details of the Coq formalization of Lemma 1. Several definitions and proofs presented in the next section are based on the notion of prime numbers, that is given in a Coq library called `Znumtheory`, as follows:

```
Inductive prime (p:Z) : Prop :=
  prime_intro : 1 < p ->
   (forall n:Z, 1 <= n < p -> rel_prime n p) -> prime p.
```

The unary predicate `prime` is defined inductively and its type is (the sort) `Prop`. The Coq system has two major sorts, called `Prop` and `Set`. The sort `Prop` is the place where the propositions live, while the sort `Set` is used for specifications or programs. The above definition of `prime` has one constructor, called `prime_intro`, whose type express the constructive notion of prime number in terms of another predicate called `rel_prime` that states the condition for which two integers are coprimes:

```
Definition rel_prime (a b:Z) : Prop := Zis_gcd a b 1.
```

Algorithm 1 AKS algorithm

```
Require: integer $n > 1$
Ensure: prime or composite
  STEP 1:
  if $n = a^b$ for $a \in \mathbb{N}$ and $b > 1$, then
    output composite.
  end if
  STEP 2:
  let $N = 2n(n-1)(n^2-1)(n^3-1)...(n^{4\lceil \log_2 n \rceil^2}-1)$. Find the smallest prime $r$, such that $r \nmid N$.
  STEP 3:
  if $q \mid n$, for some prime $q < r$, then
    if $q = n$ then
      output prime
    else
      output composite
    end if
  end if
  STEP 4:
  if $(X+a)^n = X^n + a$ in $\mathbb{Z}_n[X]/(X^r - 1), \forall a \in S = \{1, 2, ..., r\}$, then
    output prime
  else
    output composite
  end if
```

4. The Formalization of the proof in Coq

The proof of Lemma 1 is formalized according to the steps given by Algorithm 1. Note that in step 1, if the input is a prime number then the algorithm never outputs *prime*. This fact is formalized by the following lemma:

```
Lemma p_pow: forall p: Z, (exists n: Z, n>1 /\
      exists m:nat, gt m 1 /\ p = n ** m) -> ~prime p.
```

In words, Lemma p_pow states that if an integer p is such that $p = n^m$, for some integer $n > 1$, and some natural number $m > 1$ then p is not a prime number. This proof follows directly from the:

```
Lemma power_not_prime: forall (p n: Z) (m:nat), n > 1 ->
                 gt m 1 -> p = Zpower_nat n m -> ~prime p.
```

that, in turn, uses the fact that the only prime divisors of a prime number p are $\pm 1, \pm p$. In fact, all the proofs of this work are based on trivial algebraic facts and the hard work is just to joint these facts in the right way.

The step 3 looks for a potential divisor of the input n in a limited interval defined by the number r computed in step 2. Therefore, the important step in the proof of Lemma 1 is the step 4, which outputs *prime* whenever the equation (3) is satisfied for all integers $1 \leq a \leq r$. The formalization of the step 4 starts with the proof of one direction of Theorem 3:

```
Lemma agrawal_biswas: forall (a n x:Z), prime n ->
      Zis_mod ((x+a)^n) (x^n +a) n.
```

Note that the equation $(x+a)^n = x^n + a\ (mod\ n)$ is written in Coq as

```
Zis_mod ((x+a)^n) (x^n +a) n                    (4)
```

The proof of the above lemma requires the expansion of the binomial `(x+a)^n` using iteration. The definitions and some results about binomials were already formalized in Coq by a research group headed by Laurent Théry at INRIA Sophia Antipolis - France. The binomial expansion of the LHS of the equation $(x+a)^n = x^n + a\ (mod\ n)$ is given by:

$$\sum_{k=0}^{n} \binom{n}{k} x^{n-k} a^k$$

and the expanded version of equation (4) in Coq is given by:

```
sum_nm (Zabs_nat n) 0
  (fun k : nat =>
   binomial (Zabs_nat n) k *
    (x ** (Zabs_nat n - k) * a ** k)) mod n =
(x ** Zabs_nat n + a) mod n
```

where `sum_nm` is a function that iterates its argument in order to generate all the binomials $\binom{n}{1}, \binom{n}{2}, \ldots, \binom{n}{n-1}$. The next step is to prove that all these binomials are equal to zero modulo n:

```
sum_nm (pred (pred (Zabs_nat n))) 1
  (fun k : nat => binomial (Zabs_nat n) k *
  (x ** (Zabs_nat n - k) * a ** k)) mod n = 0
```

This proof is completed after an application of the lemma `bin_mod_n_prime` that states that if n is prime then $\binom{n}{k} = 0\ (mod\ n)$, for all $0 < k < n$. Lemma `agrawal_biswas` corresponds to the formalization of Theorem 3, and from this formalization is easy to see that, in fact, the primality test given by Theorem 3 runs in exponential time. The complete formalization of Lemma 1 is achieved with the proof of the following proposition:

Proposition 1 *If n is a prime number then for all r prime and a integer with $a \leq r$ we have that*

$$(x+a)^n = x^n + a \text{ in } \mathbb{Z}_n[x]/(x^r - 1).$$

From the fact that r is prime we infer that $x^r - 1 \neq 0$, and the congruence stated in Proposition 1 can be defined as follows:

```
Definition congr_AKS (n r:Z): Prop:= forall x a: Z,
  (x^r -1) <> 0 -> 1<=a<=r ->
Zis_mod (((x+a)^n) mod n) (((x^n)+a) mod n) (x^r-1).
```

The lemma in Coq that represents Proposition 1 is given by:

```
Lemma congr_AKS_1: forall n r: Z, prime n -> congr_AKS n r.
```

This proof is better understood after the introduction of the hypothesis when the goal to be proved is exactly the equation (3):

```
Zis_mod ((x + a)^n mod n) ((x^n + a) mod n) (x^r - 1)
```

Two polinomials $p_1, p_2 \in \mathbb{Z}_n[x]$ are equivalent in the quotient ring $\mathbb{Z}_n[x]/(x^r - 1)$ if they belong to the same coset, i.e., if the remainder of the division of these two polinomials by $x^r - 1$ are the same. Therefore, we need to show that the remainder of $(x + a)^n$ and of $x^r + a$ after division by $x^r - 1$ are the same, or equivalently that $x^r - 1$ divides $(x + a)^n - (x^r + a)$ which is the content of the following goal:

```
(x ^ r - 1 | (x + a) ^ n mod n - (x ^ n + a) mod n)
```

From the definition of polynomial division, this goal can be written as:

```
((x + a) ^ n mod n) mod - (x ^ r - 1) =
 ((x ^ n + a) mod n) mod - (x ^ r - 1)
```

The proof is completed after an application of Lemma `agrawal_biswas`. At this point, we have built all the necessary steps to prove that, if n is prime then Algorithm 1 returns *prime*. We join all these informations in the following definition:

```
Definition AKS (n: Z): Prop:=
    n>1 /\ ~p_pow n /\ (exists r:Z, min_r r
   (N n) /\ (q_div_eq r n \/
   (q_not_div r n /\ congr_AKS n r))).
```

That is, an integer `n` satisfies the predicate `AKS` if it is greater than 1, and is not a power, and satisfies the predicate `congr_AKS` for all integers $1 \leq a \leq r$, where r is the integer computed in step 2.

Our main result is then formalized in the next theorem whose proof consists essentially of calls to the other proofs detailed in this section:

```
Theorem AKS_correctness_1 : forall p:Z, prime p -> AKS p.
```

5. Conclusion and Future Work

In this work we presented our first results concerning a formalization of the AKS algorithm in the Coq proof assistant. Primality tests are important for generating large prime numbers used in cryptography. The AKS algorithm is the first deterministic primality test that runs in polynomial time.

In [Campos et al. 2004] is presented a partial formalization of the AKS algorithm in ACL2, which is based on first-order logic. The expressiveness of higher-order logic is more adequate to represent mathematical properties [Gordon et al. 2006], and hence Coq seems to be a more adequate environment for this kind of work.

As a future work, there are at least two goals to be achieved:

1. Formalization of the proof of Lemma 2: This goal is under development, and one of the difficulties concerns the development of a constructive procedure for computing the degree r of the ideal $(x^r - 1)$ of the quotient ring $\mathbb{Z}_n[x]/(x^r - 1)$.
2. Experimentation with the Coq code extraction tool[Bertot and Castéran 2004] including a detailed complexity analysis of the generated code. This can be useful for comparing different implementations of the AKS algorithm, as well as for comparinson with other primality tests.

References

Adleman, L. M. and Huang, M.-D. (1992). Primality testing and abelian varieties over finite fields. *Lecture Notes in Mathematics*, (1512). Springer-Verlag.

Adleman, L. M., Pomerance, C., and Rumely, R. S. (1983). On distinguishing prime numbers from composite numbers. *Annals of Mathematics*, (117):173–206.

Agrawal, M. and Biswas, S. (2003). Primality and Identity Testing via Chinese Remaindering. *Journal of the ACM*, 50(3):429–443.

Agrawal, M., Kayal, N., and Saxena, N. (2004). PRIMES is in P. *Annals of Mathematics*, 160(2):781–793.

Barendregt, H. and Geuvers, H. (2001). Proof-assistants using dependent type systems. In *Handbook of Automated Reasoning*, pages 1149–1238. Elsevier and MIT Press.

Barendregt, H. and Wiedijk, F. (2005). The challenge of computer mathematics. *Transactions A of the Royal Society*, 363(1835):2351–2375.

Barendregt, H. P. (1992). λ-calculi with types. *Handbook of Logic in Computer Science*, II.

Bertot, Y. and Castéran, P. (2004). *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag.

Campos, C., Modave, F., and Roach, S. (2004). The AKS Algorithm in ACL2. In *Fifth International Conference on Intelligent Technologies*.

Coq Development Team, T. (2007). *The Coq Proof Assistant Reference Manual Version 8.1*. INRIA-Rocquencourt. http://coq.inria.fr/doc-eng.html.

Friedman, H. M. (2006). Adventures in the verification of mathematics. In *Computer Science Colloquium*. Ohio State University.

Goldwasser, S. and Kilian, J. (1986). Almost all primes can be quickly certified. In *Annual ACM Symposium on the Theory of Computing*, pages 316–329.

Gordon, M. J. C., Reynolds, J., Hunt, W. A., and Kaufmann, M. (2006). An Integration of HOL and ACL2. In *FMCAD '06: Proceedings of the Formal Methods in Computer Aided Design*, pages 153–160, Washington, DC, USA. IEEE Computer Society.

Kaliszyk, C. and Wiedijk, F. (2007). Certified computer algebra on top of an interactive theorem prover. In *Calculemus/MKM*, pages 94–105.

Miller, G. L. (1976). Riemann's hypothesis and tests for primality. *J. Comput. Sys. Sci.*, (13):300–317.

Pratt, V. (1975). Every prime has a succinct certificate. *SIAM Journal on Computing*, (4):214–220.

Rabin, M. O. (1980). Probabilistic algorithm for testing primality. *J Number Theory*, (12):128–138.

Ribenboim, P. (1995). *The New Book of Prime Number Records*. Springer, New York, NY.

Ribenboim, P. (2004). *The Little Book of Bigger Primes*. Springer, New York, NY, 3 edition.

Solovay, R. and Strassen, V. (1977). A fast Monte-Carlo test for primality. *SIAM Journal on Computing*, (6):84–86.