Mechanisation of the AKS Algorithm (Theory)

Hing Lun Chan

College of Engineering and Computer Science
Australian National University

PhD Thesis, November 2019

Outline

Achievements

Theorem (The AKS algorithm is a primality test.)

$$\vdash \text{prime } n \iff \text{aks } n$$

Theorem (The AKS algorithm is in polynomial-time class.)

$$\vdash \text{valueOf } (\text{aksM } n) \iff \text{aks } n$$
$$\vdash \text{stepsOf} \circ \text{aksM} \in \mathcal{O}(\lceil \log n \rceil^{21})$$

Achievements

Theorem (The AKS algorithm is a primality test.)

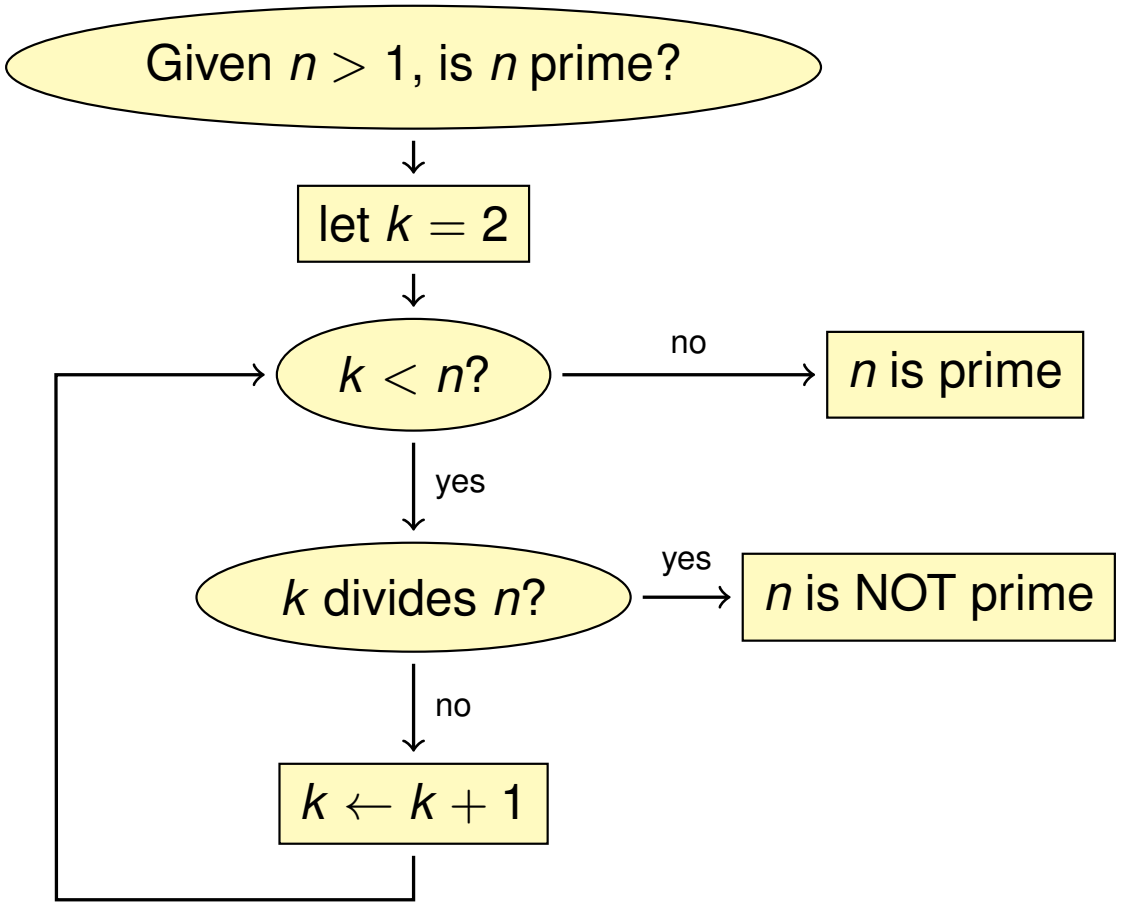
$$\vdash \text{prime } n \iff \text{aks } n$$

Theorem (The AKS algorithm is in polynomial-time class.)

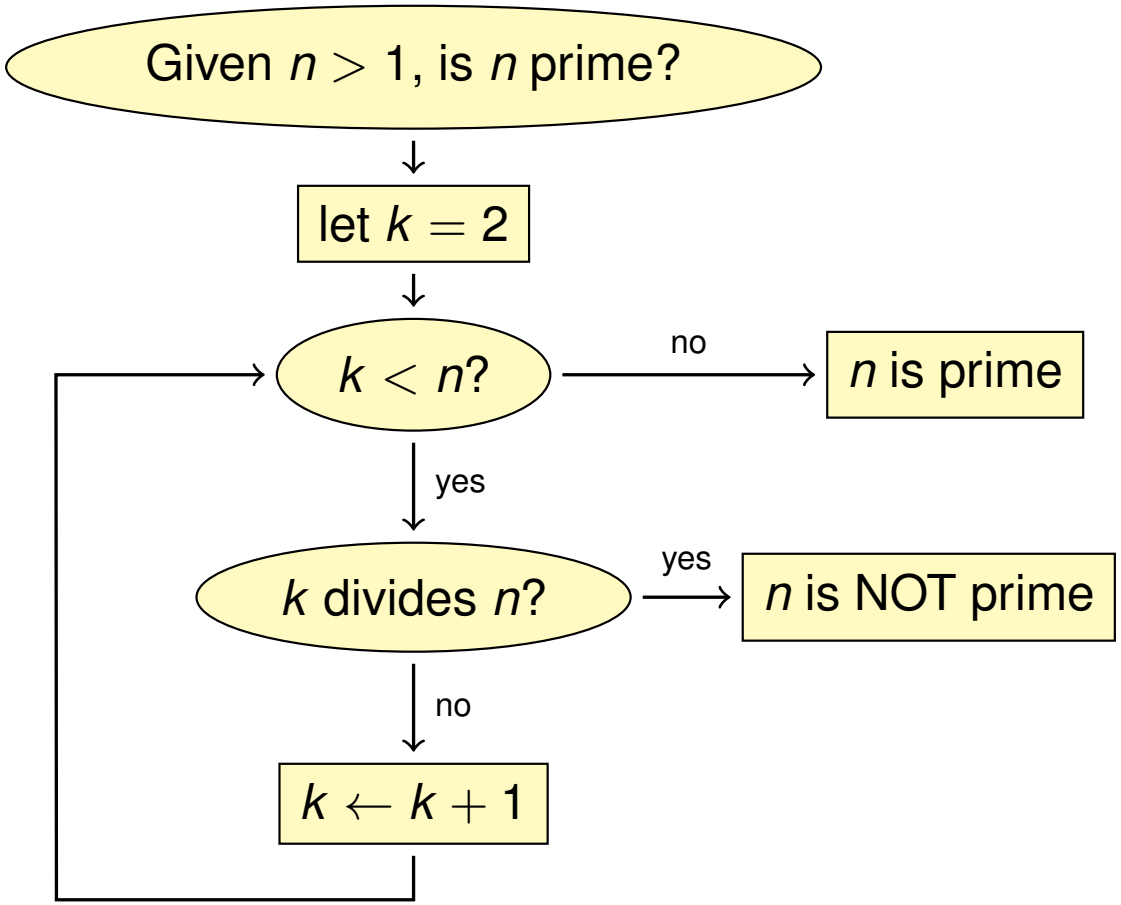
$$\vdash \text{valueOf } (\text{aksM } n) \iff \text{aks } n$$
$$\vdash \text{stepsOf} \circ \text{aksM} \in \mathcal{O}(\lceil \log n \rceil^{21})$$

- aks n — a formulation of the AKS algorithm.
- aksM n — an implementation of (aks n) in a machine.
- valueOf (aksM n) — final result of running (aksM n).
- stepsOf (aksM n) — number of steps when running (aksM n).
- All formal proofs are done in the HOL4 theorem-prover.

Primality Testing

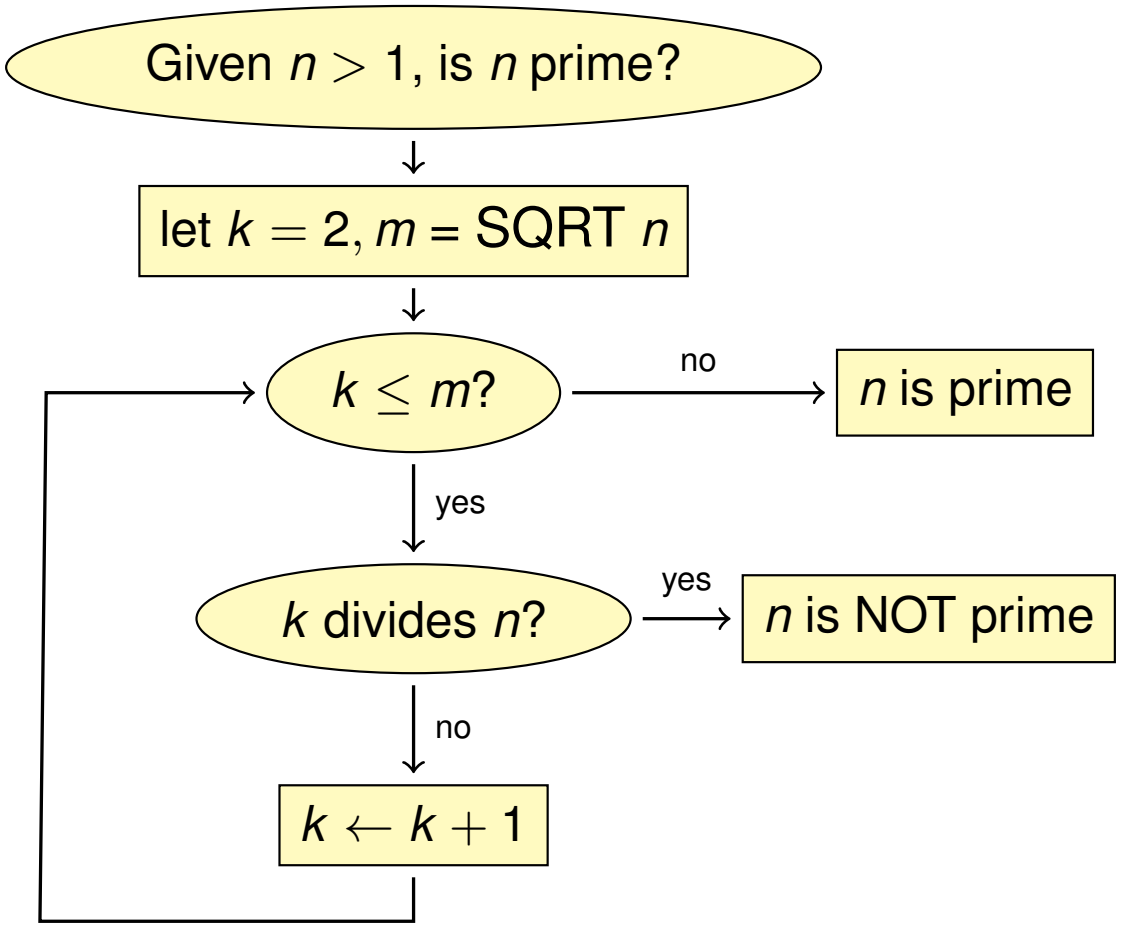


Primality Testing

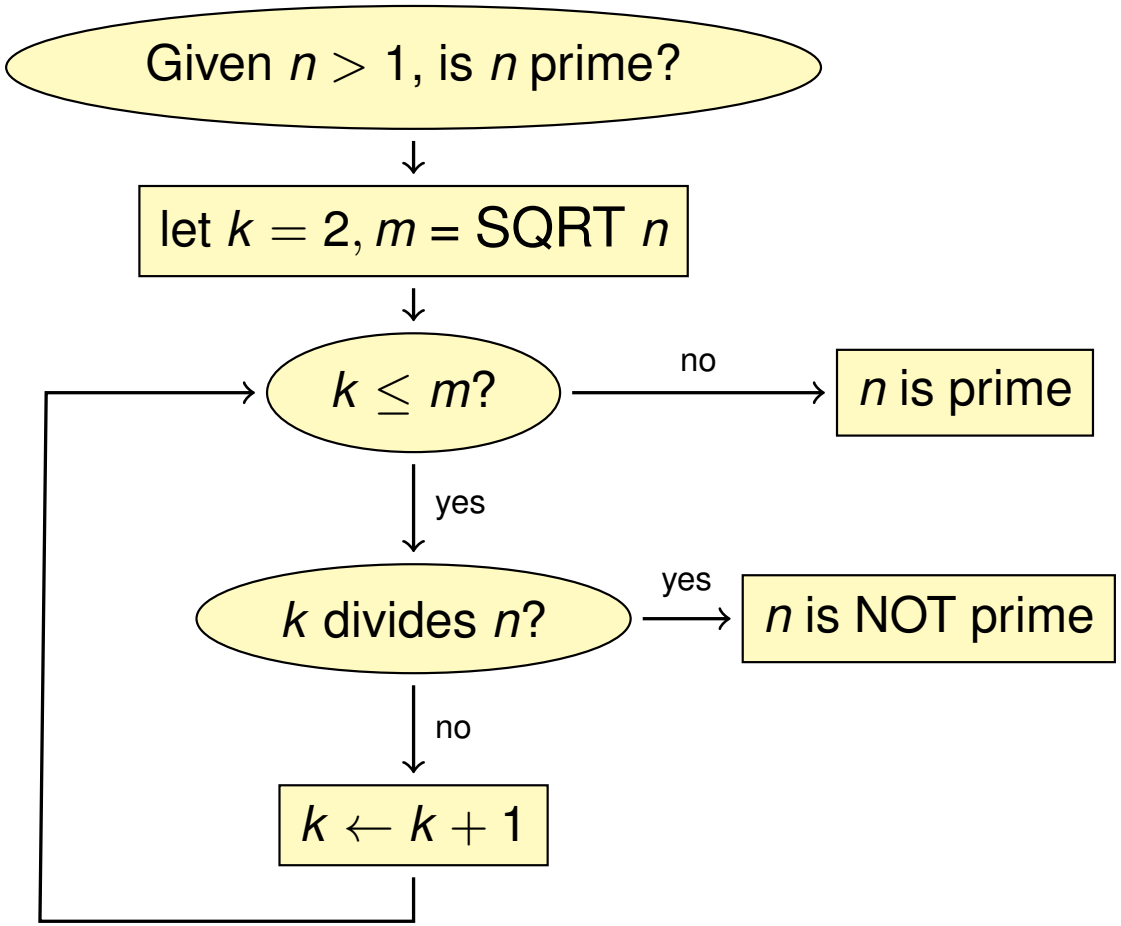


- When k is not a factor of n, *throw it away*.
- This algorithm has runtime $\mathcal{O}(n)$.

PRIMES in P?

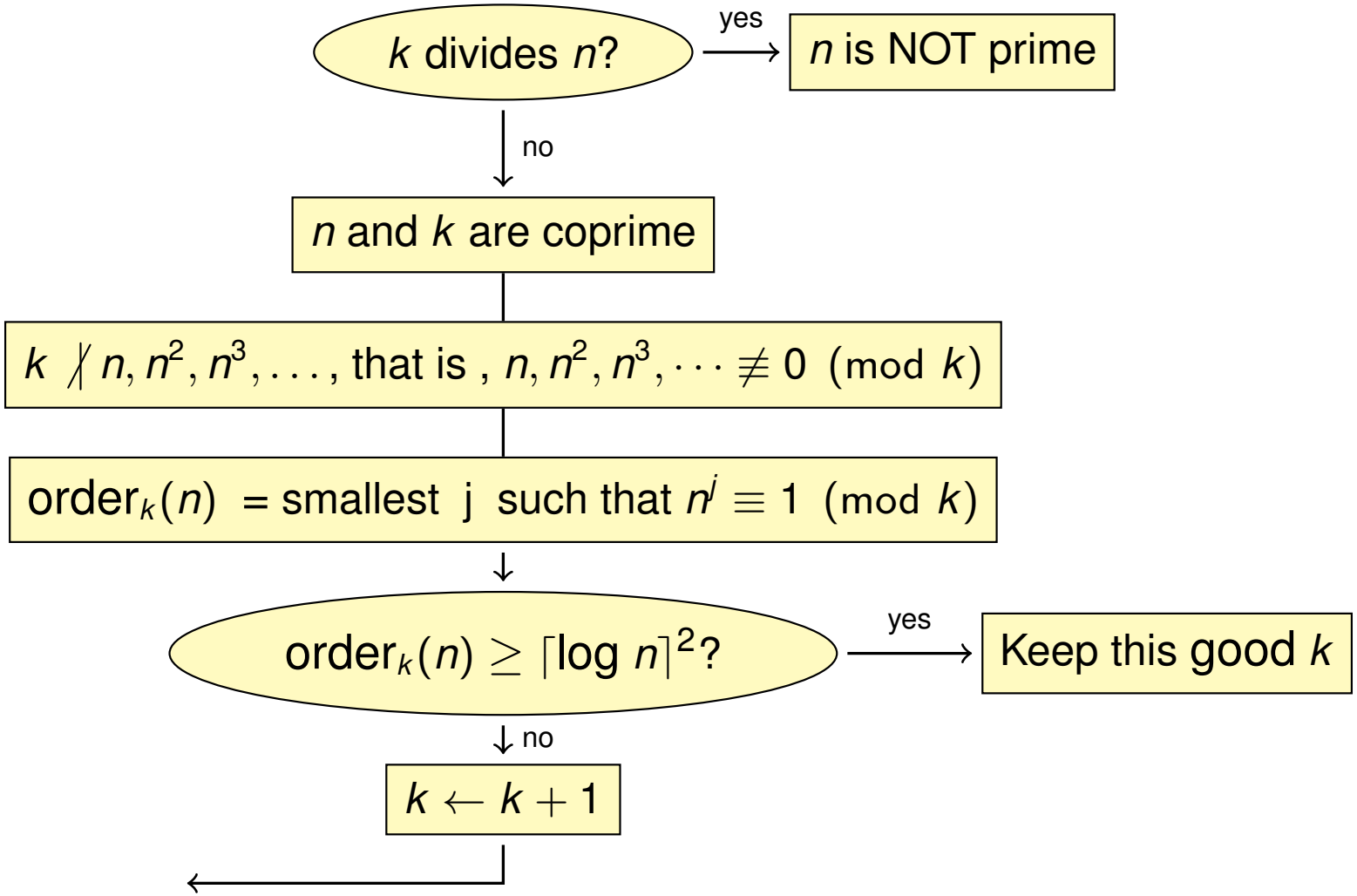

Given $n > 1$, is n prime?
let $k = 2, m$ = SQRT n
$k \leq m$?
no
n is prime
yes
k divides n?
yes
n is NOT prime
no
$k \leftarrow k + 1$

PRIMES in P?

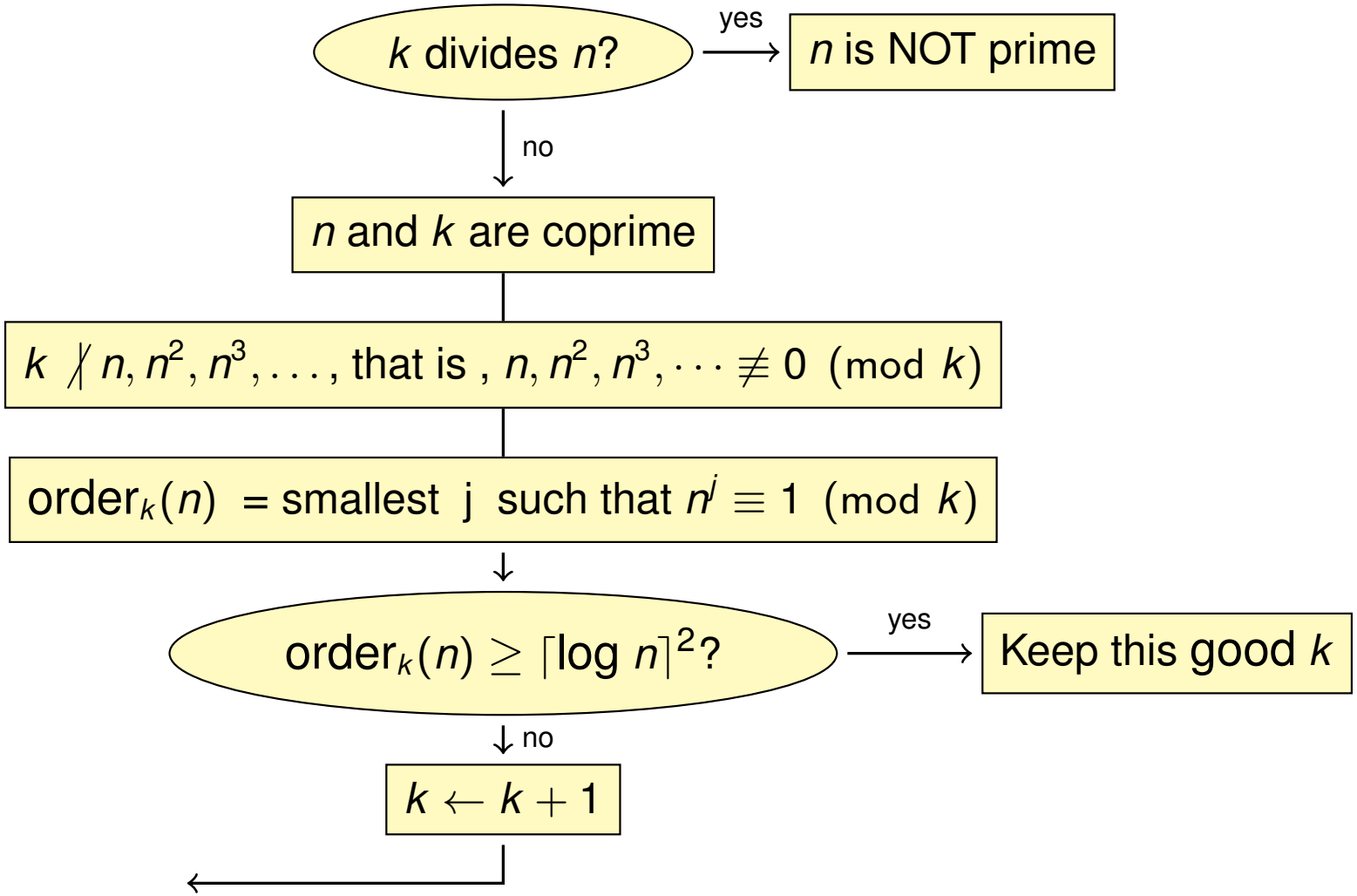


- This improved algorithm has runtime $\mathcal{O}(n^{\frac{1}{2}})$.
- Is there a primality test with runtime $\mathcal{O}(\lceil \log n \rceil^t)$?

AKS parameter



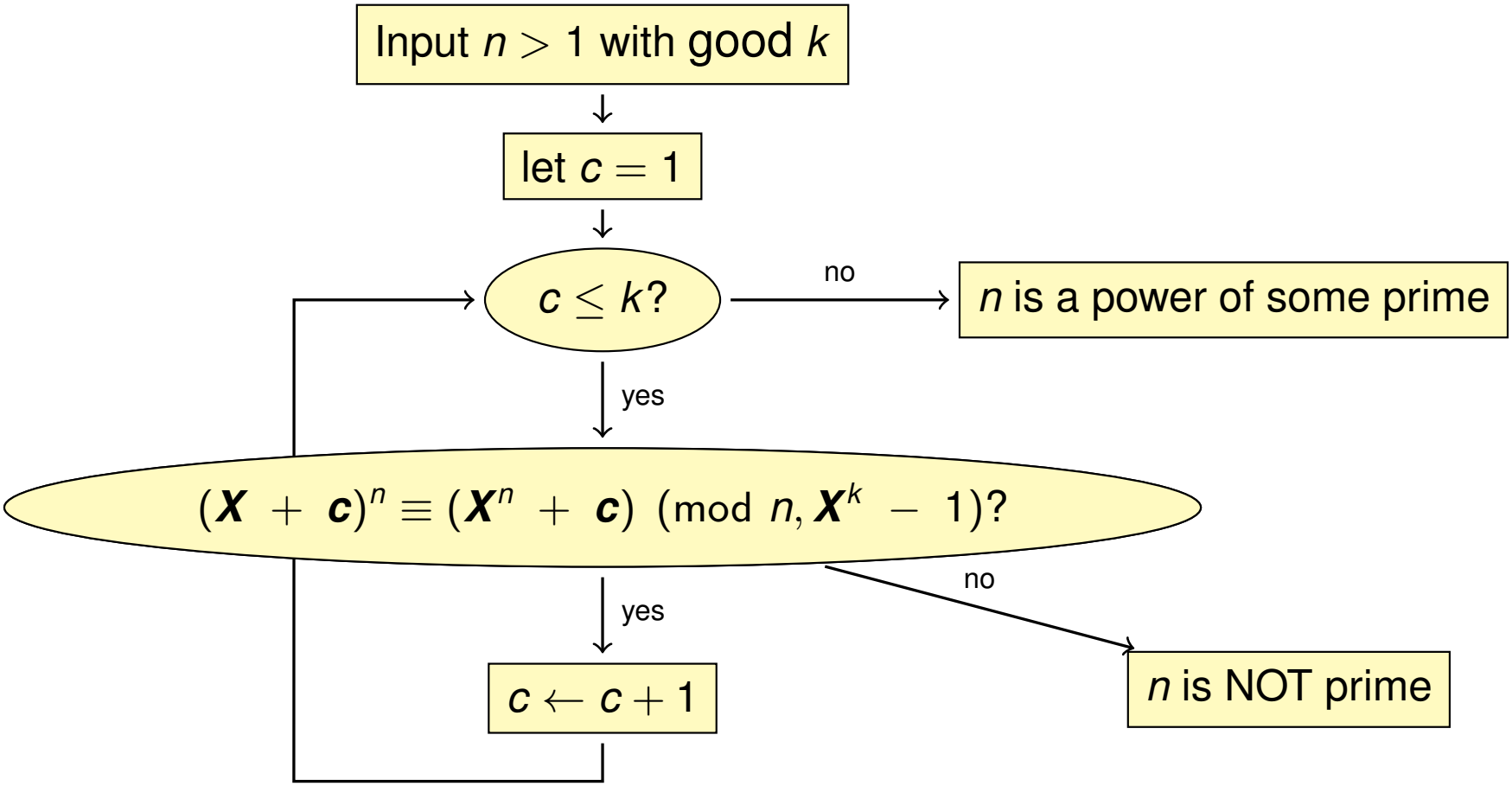
AKS parameter



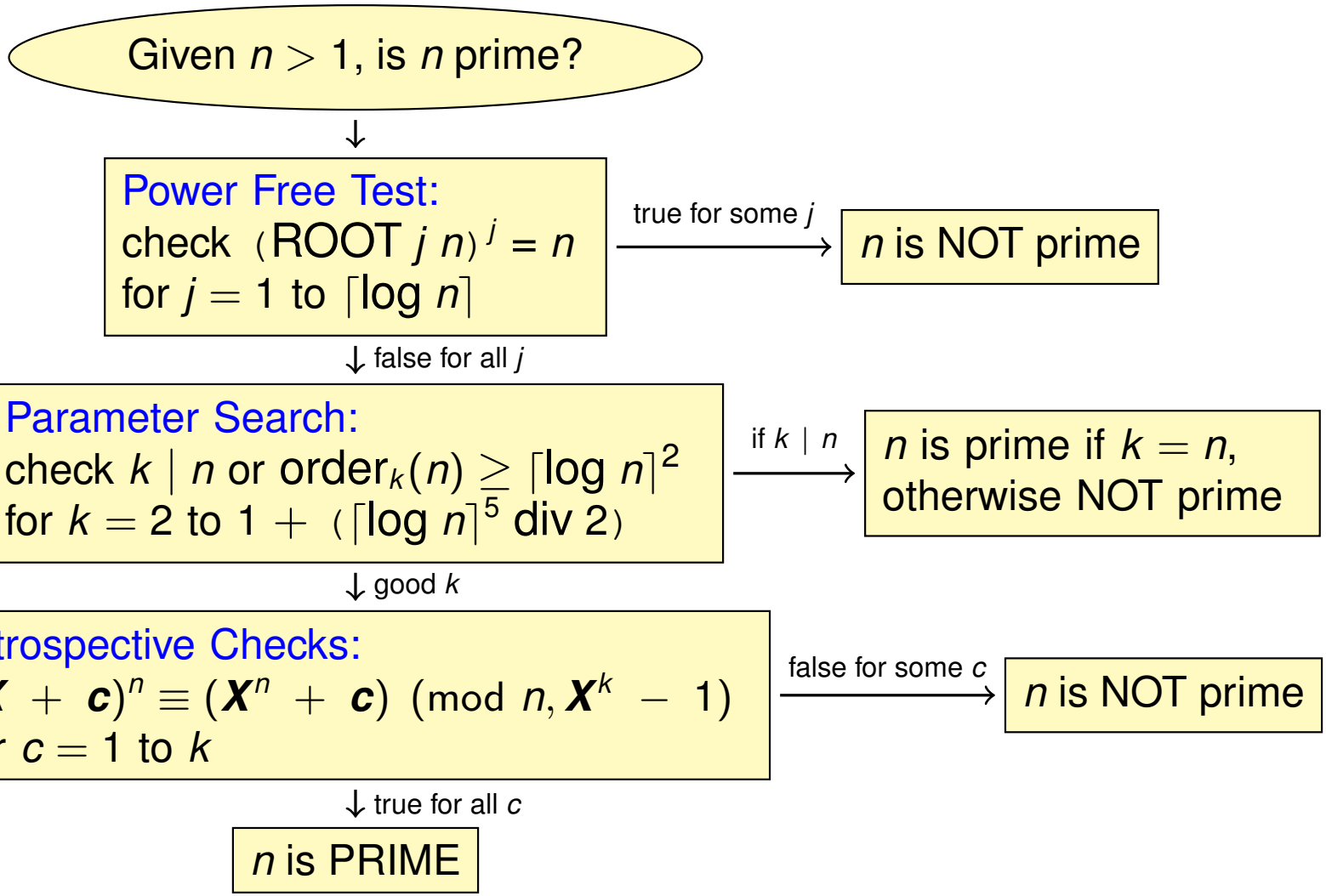
Don't throw away without thinking hard!

Introspective Checks

The good k is used in polynomial identity checks:



The AKS Algorithm



Machine Model

Definition (Some Machine Primitives)

$$\text{addM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick max}(\text{size } x, \text{size } y);\ \textbf{return}\ (x + y)\ \textbf{od}$$

$$\text{subM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick max}(\text{size } x, \text{size } y);\ \textbf{return}\ (x - y)\ \textbf{od}$$

$$\text{mulM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick}\ (\text{size } x \times \text{size } y);\ \textbf{return}\ (x \times y)\ \textbf{od}$$

$$\text{eqM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick max}(\text{size } x, \text{size } y);\ \textbf{return}\ (x = y)\ \textbf{od}$$

Machine Model

Definition (Some Machine Primitives)

$$\text{addM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick max}(\text{size } x, \text{size } y);\ \textbf{return}\ (x + y)\ \textbf{od}$$
$$\text{subM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick max}(\text{size } x, \text{size } y);\ \textbf{return}\ (x - y)\ \textbf{od}$$
$$\text{mulM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick}\ (\text{size } x \times \text{size } y);\ \textbf{return}\ (x \times y)\ \textbf{od}$$
$$\text{eqM } x\ y \stackrel{\text{def}}{=} \textbf{do}\ \text{tick max}(\text{size } x, \text{size } y);\ \textbf{return}\ (x = y)\ \textbf{od}$$

- The binary length of number n is $\text{size } n \approx \lceil \log n \rceil$.
- The clock is measured in **ticks**.
- Each primitive updates the clock by adding ticks, and returns the expected value.
- Primitives are used to build subroutines.
- Subroutines are used to implement algorithms.

Subroutine: Exponentiation

The exponential form b^n can be computed by "repeat squaring":

$$\vdash \; b^n = \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } (\textbf{if } \mathrm{EVEN}\ n \textbf{ then } 1 \textbf{ else } b) \times (b^2)^{n \operatorname{div} 2}$$

Subroutine: Exponentiation

The exponential form b^n can be computed by "repeat squaring":

$$\vdash\ b^n = \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } (\textbf{if } \mathrm{EVEN}\ n \textbf{ then } 1 \textbf{ else } b) \times (b^2)^{n\ \mathrm{div}\ 2}$$

Definition (Exponentiation in monadic style)

```
expM b n ≝
 do
  z ← zeroM n;
  if z then return 1
  else
   do
    b' ← sqM b;
    n' ← halfM n;
    r ← expM b' n';
    ifM (evenM n) (idM r) (mulM b r)
   od
 od
```

Subroutine: Exponentiation

The exponential form b^n can be computed by "repeat squaring":

$$\vdash\ b^n = \textbf{if } n = 0 \textbf{ then } 1 \textbf{ else } (\textbf{if } \mathrm{EVEN}\ n \textbf{ then } 1 \textbf{ else } b) \times (b^2)^{n\ \mathrm{div}\ 2}$$

Definition (Exponentiation in monadic style)

```
expM b n ≝
 do
  z ← zeroM n;
  if z then return 1
  else
   do
    b' ← sqM b;
    n' ← halfM n;
    r ← expM b' n';
    ifM (evenM n) (idM r) (mulM b r)
   od
 od
```

$\vdash$ valueOf (expM b n) = b^n

$\vdash$ stepsOf (expM b n) $\leq 1 + \mathrm{size}\ n + 5(\mathrm{size}\ n)^2 + 8(\mathrm{size}\ n)n^3(\mathrm{size}\ b)^2$

AKS Phases

Phase 1: Power Free Test

$$\vdash \text{valueOf } (\text{power\_freeM } n) \iff \text{power\_free } n$$
$$\vdash \text{stepsOf } (\text{power\_freeM } n) \leq 207(\text{size } n)^9$$

AKS Phases

Phase 1: Power Free Test

$$\vdash \text{valueOf}\ (\text{power\_freeM}\ n) \iff \text{power\_free}\ n$$
$$\vdash \text{stepsOf}\ (\text{power\_freeM}\ n) \leq 207(\text{size}\ n)^9$$

Phase 2: Parameter Search

$$\vdash \text{valueOf}\ (\text{paramM}\ n) = \text{param}\ n$$
$$\vdash \text{stepsOf}\ (\text{paramM}\ n) \leq 1418157969(\text{size}\ n)^{20}$$

AKS Phases

Phase 1: Power Free Test

$$\vdash \text{valueOf}\ (\text{power\_freeM}\ n) \iff \text{power\_free}\ n$$
$$\vdash \text{stepsOf}\ (\text{power\_freeM}\ n) \leq 207(\text{size}\ n)^9$$

Phase 2: Parameter Search

$$\vdash \text{valueOf}\ (\text{paramM}\ n) = \text{param}\ n$$
$$\vdash \text{stepsOf}\ (\text{paramM}\ n) \leq 1418157969(\text{size}\ n)^{20}$$

Phase 3: Introspective Checks

$$\vdash 1 < n \wedge 1 < k \Rightarrow$$
$$(\text{valueOf}\ (\text{intro\_rangeM}\ n\ k\ s) \iff \text{intro\_range}\ \mathbb{Z}_n\ k\ n\ s)$$
$$\vdash 0 < n \Rightarrow$$
$$\text{stepsOf}\ (\text{intro\_rangeM}\ n\ k\ s) \leq$$
$$163\max(1, s)\max(1, k)^2(\text{size}\ k)(\text{size}\ s)(\text{size}\ n)^4$$

AKS Implementation

Definition (AKS in monadic style)

```
aksM n ≝
 do
  b ← power_freeM n;
  if b then
   do
    c ← paramM n;
    case c of
     | nice k ⇒ eqM k n
     | good k ⇒ intro_rangeM n k k
     | bad ⇒ return F
   od
  else return F
 od
```

AKS Implementation

Definition (AKS in monadic style)

```
aksM n ≝
 do
  b ← power_freeM n;
  if b then
   do
    c ← paramM n;
    case c of
     | nice k ⇒ eqM k n
     | good k ⇒ intro_rangeM n k k
     | bad ⇒ return F
   od
  else return F
 od
```

- On our simple machine, (aksM n) has runtime $\mathcal{O}(\lceil \log n \rceil^{21})$.
- With best-known arithmetic subroutines, the AKS paper gives a runtime $\mathcal{O}(\lceil \log n \rceil^{\frac{15}{2}})$.