Mechanisation of the AKS Algorithm

Hing-Lun Chan

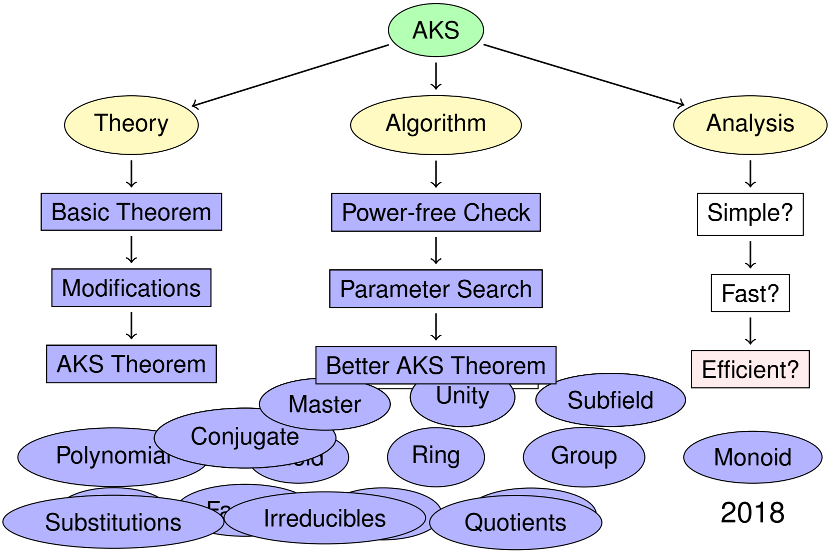
College of Engineering and Computer Science
Australian National University

PhD Monitoring, April 2018

Outline

1. Introduction
   - Road Map
   - Complexity
   - The Machine
2. Algorithms
   - Long multiplication
   - Fast exponentiation
3. Look Ahead
   - Review
   - Plans
   - Publications

Mechanisation of AKS Algorithm – Road Map



Complexity

Aim: to show that AKS primality testing is a polynomial-time algorithm

- Given a number n, the algorithm terminates with an answer to the question: is n a prime?
- The size of input n is measured by its number of bits, *i.e.*, $\lceil \log n \rceil$.
- The number of steps for the algorithm is bounded by a polynomial function of $\lceil \log n \rceil$.

The Machine

Design a CPU with machine codes, and these parts:

- a map of names to lists of machine codes,
- a list of Registers,
- a list of Memory Cells,

The Machine

Design a CPU with machine codes, and these parts:

- a map of names to lists of machine codes,
- a list of Registers,
- a list of Memory Cells,
- a program counter along a code list,
- a base pointer to allocate/deallocate cells,
- a stack for call and return, and
- a boolean flag for test result.

The Machine

Design a CPU with machine codes, and these parts:

- a map of names to lists of machine codes,
- a list of Registers,
- a list of Memory Cells,
- a program counter along a code list,
- a base pointer to allocate/deallocate cells,
- a stack for call and return, and
- a boolean flag for test result.

Some book-keeping parts:

- the clock tick to record execution time,
- the maximum number of registers,
- the maximum allocated cells, and
- the maximum level of the stack.

9×5

decimal 5 = binary 101

9			9				
$\times 5$	1	0	1				
45			9	$(9 \times 1) \times 1$	$=$	9
		0		$(9 \times 2) \times 0$	$=$	0
	9			$(9 \times 4) \times 1$	$=$	36
				sum	$=$	45

9×5

decimal 5 = binary 101

9			9					
$\times 5$	1	0	1					
45			9	$(9 \times 1) \times 1$	$=$	9		
		0		$(9 \times 2) \times 0$	$=$	0		
	9			$(9 \times 4) \times 1$	$=$	36		
				sum	$=$	45		

"Egyptian" multiplication: $m \times n$ giving r.

double m	half n	result r	add: if ODD n then m else 0	next r
9	5	0	$+9$	9
18	2	9	$+0$	9
36	1	9	$+36$	45
72	0	45		STOP

Multiplication

Pseudo code:

Input: integers m, n.

1. $r \leftarrow 0$
2. while $n \neq 0$:
   - If (ODD n), $r \leftarrow r + m$.
   - $m \leftarrow 2 \times m$
   - $n \leftarrow \frac{n}{2}$
3. return $r = m \times n$.

Figure : Long Multiplication

Multiplication

Machine code for “mult”:

Given m, n		registers: $m :: n :: tail$
$r \leftarrow 0$	PUSH 0 LOCAL 2 PUT 0 PUT 1	registers: $0 :: m :: n :: tail$ allocate two cells: r and m $r \leftarrow 0$, registers: $m :: n :: tail$ $m \leftarrow m$, registers: $n :: tail$
		loop:
while ($n \neq 0$) begin	TZERO JY 12	Is $n = 0$? yes, exit
if (ODD m),	TEVEN JY 5	Is n EVEN? yes, skip
then $r \leftarrow r + m$	GET 1 GET 0 ADD PUT 0	registers: $m :: n :: tail$ registers: $r :: m :: n :: tail$ registers: $(r + m) :: n :: tail$ $r \leftarrow r + m$, registers: $n :: tail$
		skip:
$m \leftarrow 2 \times m$	GET 1 LSHIFT PUT 1	registers: $m :: n :: tail$ registers: $2m :: n :: tail$ $m \leftarrow 2m$, registers: $n :: tail$
$n \leftarrow \frac{n}{2}$	RSHIFT	registers: $\frac{n}{2} :: tail$
end while	JP -12	back to loop
		exit:
return r	POP GET 0 FREE RETURN	registers: $tail$ registers: $r :: tail$ discard allocated cells back to caller

Long multiplication

Theorem

The machine code for long multiplication is implemented correctly.

$$
\begin{array}{l}
\vdash\ m.\mathrm{state} = \mathrm{RUNNING} \wedge m.\mathrm{registers} = n{::}k{::}t \wedge \\
\quad m.\mathrm{codeMap} = \mathrm{program\_codes} \wedge m \text{ sees } [\mathrm{CALL}\ \text{“mult”}] \Rightarrow \\
\quad\quad \mathrm{FUNPOW}\ \mathrm{cpu\_step}\ (11 + \mathrm{mult\_loop\_step}\ k)\ m = \\
\quad\quad\quad m\ \mathtt{with} \\
\quad\quad\quad <| \mathrm{registers} := n \times k{::}t; \\
\quad\quad\quad\quad \mathrm{rcount} := \max(m.\mathrm{rcount}, 3 + \mathrm{LENGTH}\ t); \\
\quad\quad\quad\quad \mathrm{ccount} := \max(m.\mathrm{ccount}, 2 + \mathrm{LENGTH}\ m.\mathrm{cells}); \\
\quad\quad\quad\quad \mathrm{level} := \max(m.\mathrm{level}, 1 + \mathrm{LENGTH}\ m.\mathrm{stack});\ \mathrm{flag} := \mathrm{T}; \\
\quad\quad\quad\quad \mathrm{pc} := m.\mathrm{pc} + 1; \\
\quad\quad\quad\quad \mathrm{clock} := m.\mathrm{clock} + (13 + \mathrm{mult\_loop\_tick}\ k\ n\ 0) |>
\end{array}
$$

Exponentiation

Multiplication:

- Slow: multiplication = iterated addition
  $9 \times 5 = 9 + 9 + 9 + 9 + 9$
- Fast: $m \times n$ = double m, half n, update result by add if (ODD n).

Exponentiation

Multiplication:

- Slow: multiplication = iterated addition
  $9 \times 5 = 9 + 9 + 9 + 9 + 9$
- Fast: $m \times n$ = double m, half n, update result by add if (ODD n).

Exponentiation:

- Slow: exponentiation = iterated multiplication
  $9^5 = 9 \times 9 \times 9 \times 9 \times 9$
- Fast: m^n = square m, half n, update result by multiply if (ODD n).

9^5

decimal 5 = binary 101

9			9			
to power 5	1	0	1			
59049			9	$(9^1)^1$	$=$	9
		0		$(9^2)^0$	$=$	1
	9			$(9^4)^1$	$=$	6561
				product	$=$	59049

9^5

decimal 5 = binary 101

9			9			
to power 5	1	0	1			
59049			9	$(9^1)^1$	$=$	9
		0		$(9^2)^0$	$=$	1
	9			$(9^4)^1$	$=$	6561
				product	$=$	59049

Fast exponentiation: m^n giving r.

square m	half n	result r	times: if ODD n then m else 1	next r
9	5	1	$\times 9$	9
81	2	9	$\times 1$	9
6561	1	9	$\times 6561$	59049
43046721	0	59049		STOP

Exponentiation

Pseudo code:

Input: integers m, n.

1. $r \leftarrow 1$
2. while $n \neq 0$:
   - If (ODD n), $r \leftarrow r \times m$.
   - $m \leftarrow m^2$
   - $n \leftarrow \frac{n}{2}$
3. return $r = m^n$.

Figure : Fast Exponentiation

Exponentiation

Machine code for “exp”:

Given m, n		registers: $m :: n :: tail$
$r \leftarrow 1$	PUSH 1 LOCAL 2 PUT 0 PUT 1	registers: $1 :: m :: n :: tail$ allocate two cells: r and m $r \leftarrow 1$, registers: $m :: n :: tail$ $m \leftarrow m$, registers: $n :: tail$
		loop:
while ($n \neq 0$) begin	TZERO JY 12	Is $n = 0$? yes, exit
if (ODD m),	TEVEN JY 5	Is n EVEN? yes, skip
then $r \leftarrow r \times m$	GET 1 GET 0 CALL “mult” PUT 0	registers: $m :: n :: tail$ registers: $r :: m :: n :: tail$ registers: $(r \times m) :: n :: tail$ $r \leftarrow r \times m$, registers: $n :: tail$
		skip:
$m \leftarrow m^2$	GET 1 CALL “square” PUT 1	registers: $m :: n :: tail$ registers: $m^2 :: n :: tail$ $m \leftarrow m^2$, registers: $n :: tail$
$n \leftarrow \frac{n}{2}$	RSHIFT	registers: $\frac{n}{2} :: tail$
end while	JP -12	back to loop
		exit:
return r	POP GET 0 FREE RETURN	registers: $tail$ registers: $r :: tail$ discard allocated cells back to caller

Fast exponentiation

Theorem

The machine code for fast exponentiation is implemented correctly.

$\vdash$ m.state = RUNNING $\wedge$ m.registers = n::k::t $\wedge$
m.codeMap = program_codes $\wedge$ m sees [CALL "exp"] $\Rightarrow$
FUNPOW cpu_step (11 + exp_loop_step k n) m =
m with
<| registers := n^k::t;
rcount := max(m.rcount, (if k = 0 then 3 else 4) + LEN
ccount := max(m.ccount, (if k = 0 then 2 else 4) + LEN
level := max(m.level, (if k = 0 then 1 else 3) + LENGTH
flag := T; pc := m.pc + 1;
clock := m.clock + (13 + exp_loop_tick k n 1) |>

AKS Pseudo Code

Input: integer $n > 1$.

1. **Power Free Test**
   For each $j = 2$ to $\lceil \log n \rceil$:
   - If (integer j-th root of n)$^j = n$, COMPOSITE.
2. **Parameter Search**
   For each $k = 2$ to $2 + \frac{\lceil \log n \rceil^5}{2}$:
   - If $k \mid n$, then if $k = n$ PRIME else COMPOSITE.
   - If $k \geq \lceil \log n \rceil^2 \wedge \text{order}_k(n) \geq \lceil \log n \rceil^2$, go to Step 3.
3. **Identity Checks**
   For each $c = 1$ to $\sqrt{\varphi(k)} \times \lceil \log n \rceil$:
   - if $(\boldsymbol{X} + \boldsymbol{c})^n \not\equiv (\boldsymbol{X}^n + \boldsymbol{c}) \pmod{n, \boldsymbol{X}^k - 1}$, COMPOSITE.
4. return PRIME.

Progress

Achieved:

- Designed a simple model for a Machine.
- Established properties of macros to compose machine codes.
- Proved the correctness of machine codes for simple arithmetic.
- Verified the machine code for root extraction (by unrolling recursion).
- Implemented the Power Free Test (AKS step 1) and proved its correctness.

Progress

Achieved:

- Designed a simple model for a Machine.
- Established properties of macros to compose machine codes.
- Proved the correctness of machine codes for simple arithmetic.
- Verified the machine code for root extraction (by unrolling recursion).
- Implemented the Power Free Test (AKS step 1) and proved its correctness.

To Do:

- Use the Big-O notation to simplify machine statistics.
- Establish machine codes for modular computations (required for AKS step 2).
- Improve the machine model to handle polynomials (necessary for AKS step 3).

Possible Timeline

Thesis plan:

April, 2015:	AKS Main Theorem (with suitable prime k)
June, 2016:	AKS Main Theorem (with suitable k)
November, 2017:	Machine model to support AKS Computation
September, 2018:	AKS Computational Model for polynomial time
December, 2018:	Thesis written (hopefully!)

Publications

Publications:

- CPP2012: A String of Pearls: Proofs of Fermat's Little Theorem
- JFR2013: Extended version for Journal of Formalized Reasoning
- ITP2015: Mechanisation of AKS Algorithm Part 1: Main Theorem
- ITP2016: Bounding LCMs with Triangles (a simple lower bound)
- JAR2017: Bounding LCMs with Triangles (both lower and upper bounds)
- JAR2018? (to be revised) Classification of Finite Fields with Applications