Addition Machines, Automatic Functions and Open Problems of Floyd and Knuth

Sanjay Jain, Xiaodong Jia,

Ammar Fathin Sabili, Frank Stephan

National University of Singapore

JCSS 136:135-156, 2023

History

Turing machines have quite small primitive steps and difficult to read programs.

Alternative: Operations on numbers, registers store arbitrary large integers and do certain primitive operations in $\mathbf{O(1)}$.

Counter machines: Primitive operations PLUS 1 (increment) MINUS 1 (decrement) and comparison with $\mathbf{0}$. Turing equivalent to Turing machines, but addition of $\mathbf{n}$-digit numbers takes $\mathbf{\Theta(2^n)}$ steps.

Hartmanis and Simon [1974] showed machines with addition, multiplication, subtraction, comparison and bitwise operations can solve all NP problems in polynomial time; without multiplication, the notion is the same as P.

Model of Floyd and Knuth

Primitive Operations: $+$, $-$, $<, =, >$, Read, Write.
Examples: $\mathbf{x = y + 3}$; $\mathbf{z = v + w}$; If $\mathbf{x < 55}$ then Goto line 12.

Floyd and Knuth themselves did not allow operations and comparisons with integer constants; however, besides allowing constants, we did not change their model. In particular on right hand side of assignments there are only one operation and two operands. Comparisons for goto instructions are between two registers or register and constant.

Floyd and Knuth showed (for model without constant operands) that greatest common divisor can be computed in linear time ($\mathbf{n}$ is the number of digits of the largest input) with three registers, but not with two. Furthermore, most arithmetic operations are in linear time.

Open Problems of Floyd and Knuth

Problems 1 and 6 not relevant, $\log(\text{input}) = \text{size}(\text{input})$.

2. Can an integer addition machine with only 5 registers compute x^2 in $O(\log x)$ operations? Can it compute the quotient $\lfloor y/z \rfloor$ in $O(\log y/z)$ operations?

3. Can an integer addition machine compute x^y mod z in $o((\log y)(\log z))$ operations, given $0 \leq x, y < z$?

4. Can an integer addition machine sort an arbitrary sequence of positive integers $\langle q_1, q_2, \ldots, q_m \rangle$ in $o((m + \log(q_1 \cdot q_2 \cdot \ldots \cdot q_m)) \log m)$ steps?

5. Can the powers of 2 in the binary representation of x be computed and output by an integer addition machine in $o((\log x)^2)$ steps? For example, if $x = 13$, the program should output the numbers $8, 4, 1$ in some order. (This means, it does not need to be the top-down default order.)

Answers

Theorem Linear time multiplication can be done with four registers (five if no constants allowed).

Theorem The listings of the powers of two in a binary representation of the input can be computed in linear time.

Remark There are various versions on how to define the big Oh and little Oh of several variables; for the Wikipedia version, it is shown that Open Problems (3) and (4) have a negative answer; however, for other versions, no result is obtained.

Methods to obtain the proof and programs follow in the next slides.

Fibonacci Method of Floyd and Knuth

Fibonacci Numbers: $\mathbf{F_1 = F_2 = 1}$; $\mathbf{F_{n+2} = F_n + F_{n+1}}$.

1. Read $\mathbf{x, y}$ With $\mathbf{1 \leq y < x}$;
2. $\mathbf{z = y}$;
3. $\mathbf{z = z + y}$; $\mathbf{y = z - y}$; If $\mathbf{z < x}$ Then Goto 3;
4. If $\mathbf{x \geq z}$ Then Begin $\mathbf{x = x - z}$ End;
5. If $\mathbf{y = z}$ Then Goto 7;
6. $\mathbf{y = z - y}$; $\mathbf{z = z - y}$; Goto 4;
7. Write $\mathbf{x}$.

The values of $\mathbf{y, z}$ are always the product of Fibonacci numbers with the input value of $\mathbf{y}$, the shifting up in line 3 and the shifting down in line 6 allow to compute the remainder in linear time. With this technique and six registers, Floyd and Knuth multiply and divide in linear time.

Optimisation of Register Numbers

(a) Constant-Range Variables can be coded in program blocks – each possible value is another program block.

(b) A variable holding a power of 2 can be used to read out a number at the top.

(c) Coding bits at the end can be used to signal when a number is completely read out.

Example with Input 11010;
Coding bit appended: 110101;
Power of 2 Created larger than input: 1000000;
Doubling up and comparing with power of 2 allow to read out bit per bit:
1101010: Greater than 1000000, bit = 1, subtract 1000000;
1010100: Greater than 1000000, bit = 1, subtract 1000000;
0101000: Smaller than 1000000, bit = 0, no subtraction; . . .
1000000: Equal to 1000000, coding bit reached, the end.

Example: Digit Sum Mod 2

1. Read $\mathbf{x}$; $\mathbf{x} = \mathbf{x} + \mathbf{x}$; $\mathbf{x} = \mathbf{x} + \mathbf{1}$; $\mathbf{y} = \mathbf{1}$;
2. If $\mathbf{y} > \mathbf{x}$ Goto 3; $\mathbf{y} = \mathbf{y} + \mathbf{y}$; Goto 2;
3. If $\mathbf{x} = \mathbf{y}$ Then Goto 5; If $\mathbf{x} < \mathbf{y}$ Then Goto 4; $\mathbf{x} = \mathbf{x} - \mathbf{y}$; Goto 7;
4. $\mathbf{x} = \mathbf{x} + \mathbf{x}$; Goto 3;
5. Write $\mathbf{0}$; End;
6. If $\mathbf{x} = \mathbf{y}$ Then Goto 8; If $\mathbf{x} < \mathbf{y}$ Then Goto 7; $\mathbf{x} = \mathbf{x} - \mathbf{y}$; Goto 4;
7. $\mathbf{x} = \mathbf{x} + \mathbf{x}$; Goto 6;
8. Write $\mathbf{1}$; End.

Ideas: (a) Register $\mathbf{y}$ used to read out top bit $\mathbf{a}$ of $\mathbf{x}$.
(b) Program position codes $\mathbf{a}$: $\mathbf{a} = \mathbf{0}$: Lines 3-5; $\mathbf{a} = \mathbf{1}$: Lines 6-8.

1. Read $\mathbf{x}$; $\mathbf{x} = \mathbf{x} + \mathbf{x} + \mathbf{1}$; $\mathbf{a} = \mathbf{0}$; $\mathbf{y} = \mathbf{1}$;
2. If $\mathbf{y} > \mathbf{x}$ Goto 3; $\mathbf{y} = \mathbf{y} + \mathbf{y}$; Goto 2;
3. If $\mathbf{x} \neq \mathbf{y}$ then Goto 4; Write $\mathbf{a}$; End;
4. If $\mathbf{x} < \mathbf{y}$ then Goto 5; $\mathbf{x} = \mathbf{x} - \mathbf{y}$; $\mathbf{a} = \mathbf{1} - \mathbf{a}$;
5. $\mathbf{x} = \mathbf{x} + \mathbf{x}$; Goto 3.

Example of Multiplication

```
Multiply 1234 with 111, Running Sum
                        with Frontshifts
  111                    0 + 111 =    111
   222                1110 + 222 =   1332
    333              13320 + 333 =  13653
     444            136530 + 444 = 136974
  ------
  136974
```

Algorithm: Read digits of one number (here 1234) out at the top and add product of digit with other number to running sum times 10.

Use binary numbers to avoid multiplication by digits 2,3,4,...,9 and decide only on adding the second number (bit 1) or skipping the addition of the second number (bit 0).

Example with Coding Digit

Example of Multiplication:

```
12340 (x) times 111 (y) Digit Running sum
1000000 (Comparator v)   -               w=0
1234010                  1               111
2340100                  2              1332
3401000                  3             13653
4010000                  4            136974
0100000                  0           1369740
1000000   Coding digit reached, the End
```

Algorithm x times y:

1. Append Coding digit 1 to x;
2. Comparator v is power of 10 greater x;
3. Shift x up, while ($x > v$) do x=x-v;
4. Digit is number of subtractions needed;
5. Shift w up, Add digit times y to w;
6. If x = v then stop else goto 3.

Algorithm for Multiplication

1. Begin Read $\mathbf{x}$; Read $\mathbf{y}$; Let $\mathbf{a} = \mathbf{1}$;
2. If $\mathbf{x} < \mathbf{0}$ Then Begin Let $\mathbf{x} = -\mathbf{x}$; Let $\mathbf{a} = -\mathbf{a}$ End;
3. If $\mathbf{y} < \mathbf{0}$ Then Begin Let $\mathbf{y} = -\mathbf{y}$; Let $\mathbf{a} = -\mathbf{a}$ End;
4. If $\mathbf{y} < \mathbf{x}$ Then Begin Let $\mathbf{v} = \mathbf{x}$; Let $\mathbf{x} = \mathbf{y}$; Let $\mathbf{y} = \mathbf{v}$ End;
5. Let $\mathbf{v} = \mathbf{1}$; Let $\mathbf{w} = \mathbf{0}$; Let $\mathbf{x} = \mathbf{x} + \mathbf{x}$; Let $\mathbf{x} = \mathbf{x} + \mathbf{1}$;
6. If $\mathbf{v} > \mathbf{x}$ Then Goto $\mathbf{7}$;
   Let $\mathbf{v} = \mathbf{v} + \mathbf{v}$; Goto $\mathbf{6}$;
7. Let $\mathbf{x} = \mathbf{x} + \mathbf{x}$; If $\mathbf{v} = \mathbf{x}$ Then Goto $\mathbf{8}$ Else Let $\mathbf{w} = \mathbf{w} + \mathbf{w}$;
   If $\mathbf{x} > \mathbf{v}$ Then Begin Let $\mathbf{w} = \mathbf{w} + \mathbf{y}$; Let $\mathbf{x} = \mathbf{x} - \mathbf{v}$ End;
   Goto $\mathbf{7}$;
8. If $\mathbf{a} = -\mathbf{1}$ Then Begin Let $\mathbf{w} = -\mathbf{w}$ End;
   Write $\mathbf{w}$; End.

Example for Division

```
Division of 136974 by 1234
           136974 - 123400 = 13574, digit 1
           135740 - 123400 = 12340, digit 1
           123400 - 123400 =     0, digit 1
           Result 111
```

One shifts the second (smaller) number to the front until it reaches the first digit of the first number and then subtracts from the first number in each round the largest multiple of the second number which is below the second number, receiving the corresponding digit. The first number is shifted to the front for getting the next digit of the division.

Again binary digits distinguish only times 1 and times 0, thus the largest multiple which goes below the first number is either 0 or the second number itself.

Algorithm for Division

1. Begin Read $\mathbf{y}$; Read $\mathbf{z}$; Let $\mathbf{x = 0}$;
2. If $\mathbf{z < 0}$ Then Begin $\mathbf{z = -z}$; $\mathbf{y = -y}$ End;
   If $\mathbf{z = 0}$ Then Goto 6;
   If $\mathbf{y < 0}$ Then Begin $\mathbf{a = -1}$; $\mathbf{y = -y}$ End
   Else Begin $\mathbf{a = 1}$ End;
   Let $\mathbf{u = z}$;
3. If $\mathbf{u \geq y}$ Then Goto 4; $\mathbf{u = u + u}$; Goto 3;
4. If $\mathbf{y \geq u}$ Then Begin $\mathbf{y = y - u}$; $\mathbf{x = x + 1}$ End;
   If $\mathbf{u = z}$ Then Goto 5;
   Let $\mathbf{x = x + x}$; Let $\mathbf{y = y + y}$; Let $\mathbf{z = z + z}$; Goto 4;
5. If $\mathbf{a = -1}$ Then Begin Let $\mathbf{x = -x}$;
   If $\mathbf{y > 0}$ Then Let $\mathbf{x = x - 1}$ End;
6. Write $\mathbf{x}$ End.

Reading Out Powers of Two

The fifth problem of Floyd and Knuth asked to output, for some input x, the shortest sequence of powers of two which sum up to x; the order of the powers of two in this output does not matter.

The idea is to first invert the order of bits in the number - for this one reads the top bit out and adds a register which then gets doubled up for the next loop.

Once the number is replaced by its binary mirror image, one can read it out again and use an extra variable to track the powers of two from $1, 2, 4, \ldots$ onwards. As the bits read out had in the original number these binary place values, one outputs each time this extra variable whenever the corresponding bit is 1.

Formal Algorithm

1. Begin Read $\mathbf{x}$; If $\mathbf{x} < \mathbf{0}$ Then Begin Let $\mathbf{x} = -\mathbf{x}$ End; Let $\mathbf{y} = \mathbf{1}$; Let $\mathbf{z} = \mathbf{1}$; Let $\mathbf{x} = \mathbf{x} + \mathbf{x}$; Let $\mathbf{x} = \mathbf{x} + \mathbf{1}$; Let $\mathbf{u} = \mathbf{0}$;
2. If $\mathbf{y} > \mathbf{x}$ Then Goto 3; Let $\mathbf{y} = \mathbf{y} + \mathbf{y}$; Goto 2;
3. If $\mathbf{y} = \mathbf{x}$ Then Goto 4;
   If $\mathbf{x} > \mathbf{y}$ Then Begin Let $\mathbf{u} = \mathbf{u} + \mathbf{z}$; Let $\mathbf{x} = \mathbf{x} - \mathbf{y}$ End;
   Let $\mathbf{x} = \mathbf{x} + \mathbf{x}$; Let $\mathbf{z} = \mathbf{z} + \mathbf{z}$; Goto 3;
4. Let $\mathbf{z} = \mathbf{1}$; Let $\mathbf{x} = \mathbf{u} + \mathbf{u}$;
5. If $\mathbf{x} \geq \mathbf{y}$ Then Begin Write $\mathbf{z}$; Let $\mathbf{x} = \mathbf{x} - \mathbf{y}$ End;
   Let $\mathbf{z} = \mathbf{z} + \mathbf{z}$; Let $\mathbf{x} = \mathbf{x} + \mathbf{x}$; If $\mathbf{x} > \mathbf{0}$ Then Goto 5;
   End.

Regular Sets

Regular sets are sets of words which are recognised by a finite automaton. For numbers, membership in such a set depends on the representation of the number; similarly the number of registers depends on the representation. Binary, Octal and Hexadecimal numbers need 2 registers, decimal and ternary numbers need 3 registers. In general 2 registers are sufficient iff the basis of the representation is a power of 2.

The algorithm for witnessing this is a direct simulation of the automaton which w.l.o.g. reads the digits of the number from high order digits to low order digits.

Automatic Functions

A function f is automatic iff one of the following equivalent conditions holds:

1. The graph of f is recognised by a finite automaton, that is, the automaton reads x and y in parallel with special symbols supplied after the end of x and y, respectively, is reached and accepts iff $f(x)$ is defined and equal to y.
2. There is a deterministic one-tape linear time Turing machine which replaced input x by $f(x)$ on the tape with input and output starting at the same position.
3. There is a nondeterministic one-tape linear time Turing machine which replaced input x by $f(x)$ on the tape with input and output starting at the same position.

Regular sets are sets where the characteristic function is automatic.

Results for Automatic functions

Theorem.
(a) Every addition machine with one register can only compute automatic functions and not even all of them.
(b) Every automatic function can be computed by a register machine with three registers (for binary representation of integers) and with four registers in general. If there are several inputs, one register more is needed.

Theorem.
A function computing the membership in a regular set has a register machine with two registers in the case of the binary representation or equivalent representations (like octal or hexadecimal where the base is a power of two). For other bases, three registers are used.

Membership in Regular Set

This algorithm tests whether a number – viewed in $\mathbf{k}$-ary representation – is member of a fixed regular set – with δ being the dfa-transition-function. The commands for multiplication with constant $\mathbf{k}$ depend on $\mathbf{k}$.

1. Begin Read $\mathbf{x}$; Let $\mathbf{y = 1}$; Let $\mathbf{a = start}$; Let $\mathbf{x = k \cdot x + 1}$ (using register $\mathbf{z}$ when $\mathbf{k}$ is not a power of $\mathbf{2}$);
2. If $\mathbf{y > x}$ Then Goto $\mathbf{3}$
   Else Begin Let $\mathbf{y = k \cdot y}$ Using $\mathbf{z}$; Goto $\mathbf{2}$ End;
3. Let $\mathbf{x = k \cdot x}$ (using register $\mathbf{z}$ if needed);
   If $\mathbf{x \neq y}$ Then Begin Let
   $\mathbf{(b, x) = (Floor(x/y), Remainder(x/y))}$;
   Let $\mathbf{a} = \delta(\mathbf{a, b})$; Goto $\mathbf{3}$ End;
4. If $\mathbf{a \in accept}$ Then Write $\mathbf{1}$ Else Write $\mathbf{0}$ End.

Big Oh of Several Variables

(a) The definition on Wikipedia, based on the algorithms textbook of Cormen, Leiserson, Rivest and Stein from 2009, a function $\mathbf{f(m,n)}$ is in $\mathbf{O(g(m,n))}$ iff there exist a constant $\mathbf{c > 0}$ and numbers $\mathbf{m_0, n_0}$ such that whenever $\mathbf{m \geq m_0}$ or $\mathbf{n \geq n_0}$ then $\mathbf{f(m,n) \leq c \cdot g(m,n)}$.

(b) The alternative (also popular) definition is to use an "and" instead of an "or": A function $\mathbf{f(m,n)}$ is in $\mathbf{O(g(m,n)}$ iff there is a constant $\mathbf{c > 0}$ and numbers $\mathbf{m_0, n_0}$ such that, for all $\mathbf{m, n}$, if $\mathbf{m \geq m_0}$ and $\mathbf{n \geq n_0}$ then $\mathbf{f(m,n) \leq c \cdot g(m,n)}$.

Little Oh of Several Variables

Correspondingly, version (a) says $\mathbf{f(m,n) \in o(g(m,n))}$ if there is for every $\mathbf{q > 0}$ a $\mathbf{k}$ such that whenever $\max\{\mathbf{m,n}\} \geq \mathbf{k}$ then $\mathbf{f(m,n) \leq q \cdot o(g(m,n))}$.

Version (b) says $\mathbf{f(m,n) \in o(g(m,n))}$ if there is for every $\mathbf{q > 0}$ a $\mathbf{k}$ such that whenever $\min\{\mathbf{m,n}\} \geq \mathbf{k}$ then $\mathbf{f(m,n) \leq q \cdot o(g(m,n))}$.

Note for version (a), $\mathbf{m + n \notin o(m \cdot n)}$. Fix $\mathbf{m = 2}$. Now $\mathbf{2 + n \notin o(2 \cdot n)}$ as one can consider $\mathbf{q = 1/3}$ and sees that the relation $\mathbf{2 + n < 2n/3}$ is not satisfied for almost all $\mathbf{n}$. However, for version (b), $\mathbf{m + n \in o(m \cdot n)}$. Given $\mathbf{q}$, one chooses $\mathbf{k > 2/q}$ and has for all $\mathbf{m, n}$ with $\mathbf{k} \leq \min\{\mathbf{m,n}\}$ that $\mathbf{m \cdot n \geq q \cdot (m + n)}$; here assume $\mathbf{n \geq m}$ and then $\mathbf{m + n \leq 2n}$ while $\mathbf{m \cdot n > 2/q \cdot n}$ and thus $\mathbf{q \cdot m \cdot n > 2n}$.

Problem (3) of Floyd and Knuth

Floyd and Knuth asked whether one can compute $\mathbf{x^y}$ modulo $\mathbf{z}$ in time $\mathbf{o(n \cdot m)}$ where $\mathbf{n}$ is the number of digits of $\mathbf{y}$ and $\mathbf{m}$ is the number of digits of $\mathbf{z}$.

In the following Question (3) is answered for variant (a) (Wikipedia version) of the Little Oh calculus and this answer does not work for variant (b).

Theorem. Let $\mathbf{y = 3}$ (or some bigger constant) and $\mathbf{n}$ be the number of digits of $\mathbf{y}$. Then, for sufficiently large $\mathbf{z}$, one cannot compute the function $\mathbf{x \mapsto x^y}$ modulo $\mathbf{z}$ in time $\mathbf{o(m \cdot n)}$.

Proof on next slide. Idea for (4) is similar.

Proof of Solution to Question (3)

One represents the numbers modulo $\mathbf{z}$ as $-\mathbf{z}/2, -\mathbf{z}/2+1, \ldots, \mathbf{z}/2-1, \mathbf{z}/2$ and let the size of a number $\mathbf{u}$ be the number of bits needed to represent the absolute value of this number as a number modulo $\mathbf{z}$. Assume that the constant of the little Oh is below $0.1/(\mathbf{y} \cdot \mathbf{n})$. Now let $\mathbf{x} = 2^{\mathbf{m}/(\mathbf{y}+1)}$ and, modulo $\mathbf{z}$, and one needs to output $\mathbf{x}^{\mathbf{y}} = 2^{\mathbf{m} \cdot \mathbf{y}/(\mathbf{y}+1)}$ which is smaller than $\mathbf{z}/2$. Thus one would need that, modulo $\mathbf{z}$, the first nonzero digit of the largest registers goes from $\mathbf{m}/(\mathbf{y}+1)$ to $\mathbf{m} \cdot \mathbf{y}/(\mathbf{y}+1)$ which requires at least $\mathbf{m} \cdot (\mathbf{y}-1)/(\mathbf{y}+1)$ additions. This amount of additions is larger than $\mathbf{c} \cdot \mathbf{m} \cdot \mathbf{n}$, as $\mathbf{c} \leq 0.1/(\mathbf{n} \cdot \mathbf{y})$ and $(\mathbf{y}-1)/(\mathbf{y}+1) \geq 0.5$, so the algorithm cannot make enough additions and subtractions for producing a result which, modulo $\mathbf{z}$, equals $\mathbf{x}^{\mathbf{y}}$.

Decidability Properties (Overview)

For one-tape linear time Turing machines, one can decide whether two programs are equal by using that the first-order theory of automatic structures is decidable.

For addition machines with one register, the same holds.

For linear time addition machine functions, equality of programs is undecidable, as one can code Hilbert's Tenth Problem. The output is 0 if the inputs for the Matiyasevich polynomial are mapped to 0 and it is 1 otherwise; this function is compared with the constant 1 function.

For multitape linear time Turing machines, equality is also undecidable, as one can code Post's Correspondence Problem. The input is a possible solution coded in a natural number and the output is 1 if the translation of the input to the upper word and lower word gives equal words, otherwise the output is 0.

Conclusion

Addition machines have, as primitive operations, addition, subtraction, comparison (greater, smaller, equal) and read / write; these primitive operations are in $O(1)$ time. Thus their model differs from that of Turing machines, which have much more restricted primitive operations.

Floyd and Knuth showed that multiplication and division can be done in linear time with six registers and that the listing of the powers of two (in any order) in an input number can be done in quadratic time. They asked whether these results can be improved.

The answer is affirmative. Multiplication and division can be done with four registers and the listing of powers of two in an input can be done in linear time.

Further results relate addition machines with automatic functions and ask when the equality is decidable.