The Frobenius Problem and Its Generalizations

Jeffrey Shallit
School of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
Canada
shallit@cs.uwaterloo.ca
http://www.cs.uwaterloo.ca/~shallit

The Frobenius Problem

The **Frobenius problem** is the following: given positive integers $x_1, x_2, \ldots, x_n$ with $\gcd(x_1, x_2, \ldots, x_n) = 1$, compute the largest integer **not** representable as a non-negative integer linear combination of the x_i.

The Frobenius Problem

The **Frobenius problem** is the following: given positive integers $x_1, x_2, \ldots, x_n$ with $\gcd(x_1, x_2, \ldots, x_n) = 1$, compute the largest integer **not** representable as a non-negative integer linear combination of the x_i.

This largest integer is sometimes denoted $g(x_1, \ldots, x_n)$.

The **Frobenius problem** is the following: given positive integers $x_1, x_2, \ldots, x_n$ with $\gcd(x_1, x_2, \ldots, x_n) = 1$, compute the largest integer **not** representable as a non-negative integer linear combination of the x_i.

This largest integer is sometimes denoted $g(x_1, \ldots, x_n)$.

The restriction $\gcd(x_1, x_2, \ldots, x_n) = 1$ is necessary for the definition to be meaningful, for otherwise every non-negative integer linear combination is divisible by this gcd.

The Chicken McNuggets Problem

A famous problem in elementary arithmetic books:

The Chicken McNuggets Problem

A famous problem in elementary arithmetic books:

At McDonald's, Chicken McNuggets are available in packs of either 6, 9, or 20 nuggets. What is the largest number of McNuggets that one cannot purchase?

The Chicken McNuggets Problem

Answer: 43.

Answer: 43.

To see that 43 is not representable, observe that we can choose either 0, 1, or 2 packs of 20. If we choose 0 or 1 packs, then we have to represent 43 or 23 as a linear combination of 6 and 9, which is impossible. So we have to choose two packs of 20. But then we cannot get 43.

The Chicken McNuggets Example

To see that every larger number is representable, note that

$$44 = 1 \cdot 20 + 0 \cdot 9 + 4 \cdot 6$$

The Chicken McNuggets Example

To see that every larger number is representable, note that

$$\begin{aligned} 44 &= 1 \cdot 20 + 0 \cdot 9 + 4 \cdot 6 \\ 45 &= 0 \cdot 20 + 3 \cdot 9 + 3 \cdot 6 \end{aligned}$$

The Chicken McNuggets Example

To see that every larger number is representable, note that

$$
\begin{aligned}
44 &= 1 \cdot 20 + 0 \cdot 9 + 4 \cdot 6 \\
45 &= 0 \cdot 20 + 3 \cdot 9 + 3 \cdot 6 \\
46 &= 2 \cdot 20 + 0 \cdot 9 + 1 \cdot 6
\end{aligned}
$$

The Chicken McNuggets Example

To see that every larger number is representable, note that

$$
\begin{aligned}
44 &= 1 \cdot 20 + 0 \cdot 9 + 4 \cdot 6 \\
45 &= 0 \cdot 20 + 3 \cdot 9 + 3 \cdot 6 \\
46 &= 2 \cdot 20 + 0 \cdot 9 + 1 \cdot 6 \\
47 &= 1 \cdot 20 + 3 \cdot 9 + 0 \cdot 6
\end{aligned}
$$

The Chicken McNuggets Example

To see that every larger number is representable, note that

$$\begin{aligned} 44 &= 1 \cdot 20 + 0 \cdot 9 + 4 \cdot 6 \\ 45 &= 0 \cdot 20 + 3 \cdot 9 + 3 \cdot 6 \\ 46 &= 2 \cdot 20 + 0 \cdot 9 + 1 \cdot 6 \\ 47 &= 1 \cdot 20 + 3 \cdot 9 + 0 \cdot 6 \\ 48 &= 0 \cdot 20 + 0 \cdot 9 + 8 \cdot 6 \end{aligned}$$

The Chicken McNuggets Example

To see that every larger number is representable, note that

$$
\begin{aligned}
44 &= 1 \cdot 20 + 0 \cdot 9 + 4 \cdot 6 \\
45 &= 0 \cdot 20 + 3 \cdot 9 + 3 \cdot 6 \\
46 &= 2 \cdot 20 + 0 \cdot 9 + 1 \cdot 6 \\
47 &= 1 \cdot 20 + 3 \cdot 9 + 0 \cdot 6 \\
48 &= 0 \cdot 20 + 0 \cdot 9 + 8 \cdot 6 \\
49 &= 2 \cdot 20 + 1 \cdot 9 + 0 \cdot 6
\end{aligned}
$$

and every larger number can be written as a multiple of 6 plus one of these numbers.

History of the Frobenius problem

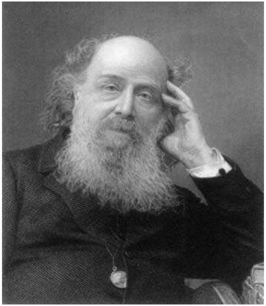
- Problem discussed by Frobenius (1849–1917) in his lectures in the late 1800's — but Frobenius never published anything

History of the Frobenius problem

- A related problem was discussed by Sylvester in 1882: he gave a formula for $h(x_1, x_2, \ldots, x_n)$, the total number of non-negative integers not representable as a linear combination of the x_i, in the case $n = 2$

History of the Frobenius problem

- A related problem was discussed by Sylvester in 1882: he gave a formula for $h(x_1, x_2, \ldots, x_n)$, the total number of non-negative integers not representable as a linear combination of the x_i, in the case $n = 2$



- Applications of the Frobenius problem occur in number theory, automata theory, sorting algorithms, etc.

Research on the Frobenius problem

- Formulas for g where dimension is bounded

Research on the Frobenius problem

- Formulas for g where dimension is bounded
- Upper and lower bounds for g

Research on the Frobenius problem

- Formulas for g where dimension is bounded
- Upper and lower bounds for g
- Formulas for g in special cases

Research on the Frobenius problem

- Formulas for g where dimension is bounded
- Upper and lower bounds for g
- Formulas for g in special cases
- Complexity of computing g

Research on the Frobenius problem

- Formulas for g where dimension is bounded
- Upper and lower bounds for g
- Formulas for g in special cases
- Complexity of computing g
- Average-case behavior of g

Formulas for g

In the case where $n = 2$, we have $g(x, y) = xy - x - y$.

Formulas for g

In the case where $n = 2$, we have $g(x, y) = xy - x - y$.

Proof. Suppose $xy - x - y$ is representable as $ax + by$.

Formulas for g

In the case where $n = 2$, we have $g(x, y) = xy - x - y$.

Proof. Suppose $xy - x - y$ is representable as $ax + by$.

Then, taking the result modulo x, we have $-y \equiv by \pmod{x}$, so $b \equiv -1 \pmod{x}$.

Formulas for g

In the case where $n = 2$, we have $g(x, y) = xy - x - y$.

Proof. Suppose $xy - x - y$ is representable as $ax + by$.

Then, taking the result modulo x, we have $-y \equiv by \pmod{x}$, so $b \equiv -1 \pmod{x}$.

Similarly, modulo y, we get $-x \equiv ax$, so $a \equiv -1 \pmod{y}$.

Formulas for g

In the case where $n = 2$, we have $g(x, y) = xy - x - y$.

Proof. Suppose $xy - x - y$ is representable as $ax + by$.

Then, taking the result modulo x, we have $-y \equiv by \pmod{x}$, so $b \equiv -1 \pmod{x}$.

Similarly, modulo y, we get $-x \equiv ax$, so $a \equiv -1 \pmod{y}$.

But then $ax + by \geq (y - 1)x + (x - 1)y = 2xy - x - y$, a contradiction.

Formulas for g

In the case where $n = 2$, we have $g(x, y) = xy - x - y$.

Proof. Suppose $xy - x - y$ is representable as $ax + by$.

Then, taking the result modulo x, we have $-y \equiv by \pmod{x}$, so $b \equiv -1 \pmod{x}$.

Similarly, modulo y, we get $-x \equiv ax$, so $a \equiv -1 \pmod{y}$.

But then $ax + by \geq (y-1)x + (x-1)y = 2xy - x - y$, a contradiction.

So $xy - x - y$ is not representable.

Formulas for g

To prove every integer larger than $xy - x - y$ is representable, let $c = x^{-1} \bmod y$ and $d = y^{-1} \bmod x$.

Formulas for g

To prove every integer larger than $xy - x - y$ is representable, let $c = x^{-1} \bmod y$ and $d = y^{-1} \bmod x$. Then a simple calculation shows that $(c-1)x + (d-1)y = xy - x - y + 1$, so this gives a representation for $g(x,y) + 1$.

Formulas for g

To prove every integer larger than $xy - x - y$ is representable, let $c = x^{-1} \bmod y$ and $d = y^{-1} \bmod x$. Then a simple calculation shows that $(c-1)x + (d-1)y = xy - x - y + 1$, so this gives a representation for $g(x, y) + 1$.

To get a representation for larger numbers, we use the extended Euclidean algorithm to find integers e, f such that $ex - fy = 1$.

Formulas for g

To prove every integer larger than $xy - x - y$ is representable, let $c = x^{-1} \bmod y$ and $d = y^{-1} \bmod x$. Then a simple calculation shows that $(c-1)x + (d-1)y = xy - x - y + 1$, so this gives a representation for $g(x, y) + 1$.

To get a representation for larger numbers, we use the extended Euclidean algorithm to find integers e, f such that $ex - fy = 1$. We just add the appropriate multiple of this equation, reducing, if necessary, by $(-y)x + xy$ or $yx + (-x)y$ if a coefficient becomes negative.

Formulas for g

To prove every integer larger than $xy - x - y$ is representable, let $c = x^{-1} \bmod y$ and $d = y^{-1} \bmod x$. Then a simple calculation shows that $(c-1)x + (d-1)y = xy - x - y + 1$, so this gives a representation for $g(x, y) + 1$.

To get a representation for larger numbers, we use the extended Euclidean algorithm to find integers e, f such that $ex - fy = 1$. We just add the appropriate multiple of this equation, reducing, if necessary, by $(-y)x + xy$ or $yx + (-x)y$ if a coefficient becomes negative.

For example, for $[x, y] = [13, 19]$, we find $[2, 10] \cdot [x, y] = 216$. Also $[3, -2] \cdot [x, y] = 1$. To get a representation for 217, we just add these two vectors to get $[5, 8]$.

Formulas for g

For 3 numbers, more complicated (but still polynomial-time) algorithms have been given by Greenberg and Davison (independently).

Formulas for g

For 3 numbers, more complicated (but still polynomial-time) algorithms have been given by Greenberg and Davison (independently).

Kannan has given a polynomial-time algorithm for any fixed dimension, but the time depends at least exponentially on the dimension and the algorithm is very complicated.

Computational Complexity of g

Ramírez-Alfonsín has proven that computing g is NP-hard under Turing-reductions, by reducing from the integer knapsack problem.

Ramírez-Alfonsín has proven that computing g is NP-hard under Turing-reductions, by reducing from the integer knapsack problem.

The **integer knapsack problem** is, given $x_1, x_2, \ldots, x_n$, and a target t, do there exist non-negative integers a_i such that $\sum_{1 \le i \le n} a_i x_i = t$?

Computational Complexity of g

Ramírez-Alfonsín has proven that computing g is NP-hard under Turing-reductions, by reducing from the integer knapsack problem.

The **integer knapsack problem** is, given $x_1, x_2, \ldots, x_n$, and a target t, do there exist non-negative integers a_i such that $\sum_{1 \leq i \leq n} a_i x_i = t$?

His reduction requires 3 calls to a subroutine for the Frobenius number g.

Upper bound for the Frobenius number

A simple upper bound can be obtained by dynamic programming.

Upper bound for the Frobenius number

A simple upper bound can be obtained by dynamic programming.

Suppose $x_1 < x_2 < \cdots < x_n$. Consider testing each number $0, 1, 2, \ldots$ in turn to see if it is representable as a non-negative integer linear combination of the x_i.

Upper bound for the Frobenius number

A simple upper bound can be obtained by dynamic programming.

Suppose $x_1 < x_2 < \cdots < x_n$. Consider testing each number $0, 1, 2, \ldots$ in turn to see if it is representable as a non-negative integer linear combination of the x_i. Then r is representable if and only if at least one of $r - x_1, r - x_2, \ldots, r - x_n$ is representable.

Upper bound for the Frobenius number

A simple upper bound can be obtained by dynamic programming.

Suppose $x_1 < x_2 < \cdots < x_n$. Consider testing each number $0, 1, 2, \ldots$ in turn to see if it is representable as a non-negative integer linear combination of the x_i. Then r is representable if and only if at least one of $r - x_1, r - x_2, \ldots, r - x_n$ is representable. Now group the numbers in blocks of size x_n, and write a 1 if the number is representable, 0 otherwise.

Upper bound for the Frobenius number

A simple upper bound can be obtained by dynamic programming.

Suppose $x_1 < x_2 < \cdots < x_n$. Consider testing each number $0, 1, 2, \ldots$ in turn to see if it is representable as a non-negative integer linear combination of the x_i. Then r is representable if and only if at least one of $r - x_1, r - x_2, \ldots, r - x_n$ is representable. Now group the numbers in blocks of size x_n, and write a 1 if the number is representable, 0 otherwise. Clearly if j is representable, so is $j + x_n$, so each consecutive block has 1's in the same positions as the previous, plus maybe some new 1's.

Upper bound for the Frobenius number

A simple upper bound can be obtained by dynamic programming.

Suppose $x_1 < x_2 < \cdots < x_n$. Consider testing each number $0, 1, 2, \ldots$ in turn to see if it is representable as a non-negative integer linear combination of the x_i. Then r is representable if and only if at least one of $r - x_1, r - x_2, \ldots, r - x_n$ is representable. Now group the numbers in blocks of size x_n, and write a 1 if the number is representable, 0 otherwise. Clearly if j is representable, so is $j + x_n$, so each consecutive block has 1's in the same positions as the previous, plus maybe some new 1's. In fact, new 1's must appear in each consecutive block, until it is full of 1's, for otherwise the Frobenius number would be infinite.

Upper bound for the Frobenius number

A simple upper bound can be obtained by dynamic programming.

Suppose $x_1 < x_2 < \cdots < x_n$. Consider testing each number $0, 1, 2, \ldots$ in turn to see if it is representable as a non-negative integer linear combination of the x_i. Then r is representable if and only if at least one of $r - x_1, r - x_2, \ldots, r - x_n$ is representable. Now group the numbers in blocks of size x_n, and write a 1 if the number is representable, 0 otherwise. Clearly if j is representable, so is $j + x_n$, so each consecutive block has 1's in the same positions as the previous, plus maybe some new 1's. In fact, new 1's must appear in each consecutive block, until it is full of 1's, for otherwise the Frobenius number would be infinite. So we need to examine at most x_n blocks.

A simple upper bound can be obtained by dynamic programming.

Suppose $x_1 < x_2 < \cdots < x_n$. Consider testing each number $0, 1, 2, \ldots$ in turn to see if it is representable as a non-negative integer linear combination of the x_i. Then r is representable if and only if at least one of $r - x_1, r - x_2, \ldots, r - x_n$ is representable. Now group the numbers in blocks of size x_n, and write a 1 if the number is representable, 0 otherwise. Clearly if j is representable, so is $j + x_n$, so each consecutive block has 1's in the same positions as the previous, plus maybe some new 1's. In fact, new 1's must appear in each consecutive block, until it is full of 1's, for otherwise the Frobenius number would be infinite. So we need to examine at most x_n blocks. Once a block is full, every subsequent number is representable.

Upper bound for the Frobenius number

A simple upper bound can be obtained by dynamic programming.

Suppose $x_1 < x_2 < \cdots < x_n$. Consider testing each number $0, 1, 2, \ldots$ in turn to see if it is representable as a non-negative integer linear combination of the x_i. Then r is representable if and only if at least one of $r - x_1, r - x_2, \ldots, r - x_n$ is representable. Now group the numbers in blocks of size x_n, and write a 1 if the number is representable, 0 otherwise. Clearly if j is representable, so is $j + x_n$, so each consecutive block has 1's in the same positions as the previous, plus maybe some new 1's. In fact, new 1's must appear in each consecutive block, until it is full of 1's, for otherwise the Frobenius number would be infinite. So we need to examine at most x_n blocks. Once a block is full, every subsequent number is representable. Thus we have shown $g(x_1, x_2, \ldots, x_n) < x_n^2$.

More bounds for the Frobenius number

Erdős and Graham:

$$g(x_1, x_2, \ldots, x_n) \leq 2x_n \left\lfloor \frac{x_1}{n} \right\rfloor - x_1.$$

More bounds for the Frobenius number

Erdős and Graham:

$$g(x_1, x_2, \ldots, x_n) \leq 2x_n \left\lfloor \frac{x_1}{n} \right\rfloor - x_1.$$

Davison:

$$g(x_1, x_2, x_3) \geq \sqrt{3x_1x_2x_3} - (x_1 + x_2 + x_3)$$

Applications of the Frobenius Number

- Shell sort - a sorting algorithm devised by D. Shell in 1959.

- Shell sort - a sorting algorithm devised by D. Shell in 1959.
- Basic idea: arrange list in j columns; sort columns; decrease j; repeat

Shellsort Example

Start with 10 5 12 13 4 6 9 11 8 1 7

Shellsort Example

Start with 10 5 12 13 4 6 9 11 8 1 7
Arrange in 5 columns:

10	5	12	13	4
6	9	11	8	1
7				

Shellsort Example

Start with 10 5 12 13 4 6 9 11 8 1 7
Arrange in 5 columns:

10	5	12	13	4
6	9	11	8	1
7				

Sort each column:

6	5	11	8	1
7	9	12	13	4
10				

Shellsort Example

Now arrange in 3 columns:

6	5	11
8	1	7
9	12	13
4	10	

Shellsort Example

Now arrange in 3 columns:

6	5	11
8	1	7
9	12	13
4	10	

Sort each column:

4	1	7
6	5	11
8	10	13
9	12	

Shellsort Example

We now have
4 1 7 6 5 11 8 10 13 9 12.

Shellsort Example

We now have
4 1 7 6 5 11 8 10 13 9 12.

Finally, sort the remaining elements:
1 4 5 6 7 8 9 10 11 12 13

Choosing the Increments in Shellsort

- Running time depends on increments

Choosing the Increments in Shellsort

- Running time depends on increments
- Original version used increments a power of 2, but this gives quadratic running time.

Choosing the Increments in Shellsort

- Running time depends on increments
- Original version used increments a power of 2, but this gives quadratic running time.
- It is $O(n^{3/2})$ if increments $1, 3, 7, 15, 31, \ldots$ are used.

Choosing the Increments in Shellsort

- Running time depends on increments
- Original version used increments a power of 2, but this gives quadratic running time.
- It is $O(n^{3/2})$ if increments $1, 3, 7, 15, 31, \ldots$ are used. (Powers of 2, minus 1.)

Choosing the Increments in Shellsort

- Running time depends on increments
- Original version used increments a power of 2, but this gives quadratic running time.
- It is $O(n^{3/2})$ if increments $1, 3, 7, 15, 31, \ldots$ are used. (Powers of 2, minus 1.)
- It is $O(n^{4/3})$ if increments $1, 8, 23, 77, \ldots$ are used

Choosing the Increments in Shellsort

- Running time depends on increments
- Original version used increments a power of 2, but this gives quadratic running time.
- It is $O(n^{3/2})$ if increments $1, 3, 7, 15, 31, \ldots$ are used. (Powers of 2, minus 1.)
- It is $O(n^{4/3})$ if increments $1, 8, 23, 77, \ldots$ are used (Numbers of the form $4^{j+1} + 3 \cdot 2^j + 1$).

Choosing the Increments in Shellsort

- Running time depends on increments
- Original version used increments a power of 2, but this gives quadratic running time.
- It is $O(n^{3/2})$ if increments $1, 3, 7, 15, 31, \ldots$ are used. (Powers of 2, minus 1.)
- It is $O(n^{4/3})$ if increments $1, 8, 23, 77, \ldots$ are used (Numbers of the form $4^{j+1} + 3 \cdot 2^j + 1$).
- It is $O(n(\log n)^2)$ if increments $1, 2, 3, 4, 6, 9, 8, 12, 18, 27, 16, 24, \ldots$ are used

Choosing the Increments in Shellsort

- Running time depends on increments
- Original version used increments a power of 2, but this gives quadratic running time.
- It is $O(n^{3/2})$ if increments $1, 3, 7, 15, 31, \ldots$ are used. (Powers of 2, minus 1.)
- It is $O(n^{4/3})$ if increments $1, 8, 23, 77, \ldots$ are used (Numbers of the form $4^{j+1} + 3 \cdot 2^j + 1$).
- It is $O(n(\log n)^2)$ if increments $1, 2, 3, 4, 6, 9, 8, 12, 18, 27, 16, 24, \ldots$ are used (Numbers of the form $2^i 3^j$).

Theorem. The number of steps required to r-sort a file $a[1..N]$ that is already $r_1, r_2, \ldots, r_t$-sorted is $\leq \frac{N}{r} g(r_1, r_2, \ldots, r_t)$.

Theorem. The number of steps required to r-sort a file $a[1..N]$ that is already $r_1, r_2, \ldots, r_t$-sorted is $\leq \frac{N}{r} g(r_1, r_2, \ldots, r_t)$.

Proof. The number of steps to insert $a[i]$ is the number of elements in $a[i-r], a[i-2r], \ldots$ that are greater than $a[i]$.

Theorem. The number of steps required to r-sort a file $a[1..N]$ that is already $r_1, r_2, \ldots, r_t$-sorted is $\leq \frac{N}{r} g(r_1, r_2, \ldots, r_t)$.

Proof. The number of steps to insert $a[i]$ is the number of elements in $a[i-r], a[i-2r], \ldots$ that are greater than $a[i]$. But if x is a linear combination of $r_1, r_2, \ldots, r_t$, then $a[i-x] < a[i]$, since the file is $r_1, r_2, \ldots, r_t$-sorted.

Shellsort and the Frobenius Problem

Theorem. The number of steps required to r-sort a file $a[1..N]$ that is already $r_1, r_2, \ldots, r_t$-sorted is $\leq \frac{N}{r} g(r_1, r_2, \ldots, r_t)$.

Proof. The number of steps to insert $a[i]$ is the number of elements in $a[i-r], a[i-2r], \ldots$ that are greater than $a[i]$. But if x is a linear combination of $r_1, r_2, \ldots, r_t$, then $a[i-x] < a[i]$, since the file is $r_1, r_2, \ldots, r_t$-sorted. Thus the number of steps to insert $a[i]$ is $\leq$ the number of multiples of r that are not linear combinations of $r_1, r_2, \ldots, r_t$.

Theorem. The number of steps required to r-sort a file $a[1..N]$ that is already $r_1, r_2, \ldots, r_t$-sorted is $\leq \frac{N}{r} g(r_1, r_2, \ldots, r_t)$.

Proof. The number of steps to insert $a[i]$ is the number of elements in $a[i-r], a[i-2r], \ldots$ that are greater than $a[i]$. But if x is a linear combination of $r_1, r_2, \ldots, r_t$, then $a[i-x] < a[i]$, since the file is $r_1, r_2, \ldots, r_t$-sorted. Thus the number of steps to insert $a[i]$ is $\leq$ the number of multiples of r that are not linear combinations of $r_1, r_2, \ldots, r_t$. This number is $\leq g(r_1, r_2, \ldots, r_t)/r$.

- A deterministic finite automaton (DFA) is a simple model of a computer

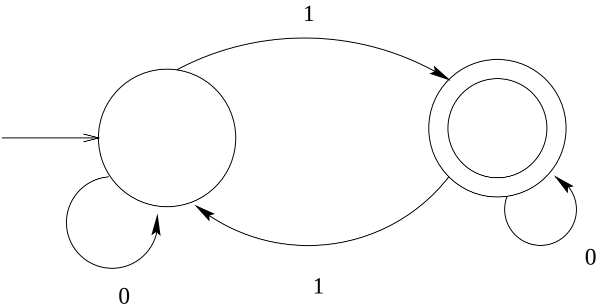
- A deterministic finite automaton (DFA) is a simple model of a computer
- It consists of a finite set of states and transitions between the states

- A deterministic finite automaton (DFA) is a simple model of a computer
- It consists of a finite set of states and transitions between the states
- At each step, the machine enters a new state based on its current state and the symbol being scanned

The Frobenius Problem and NFA to DFA Conversion

- A deterministic finite automaton (DFA) is a simple model of a computer
- It consists of a finite set of states and transitions between the states
- At each step, the machine enters a new state based on its current state and the symbol being scanned
- If an input string causes the machine to enter a "final state", it is accepted; otherwise it is rejected

Example of a DFA



Nondeterminism - NFA

- A generalization of the DFA is the NFA - nondeterministic finite automaton

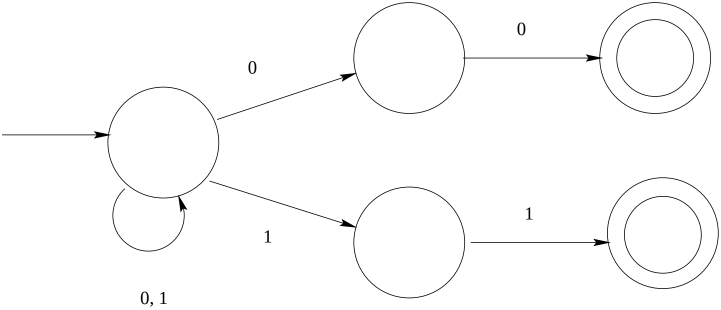
Nondeterminism - NFA

- A generalization of the DFA is the NFA - nondeterministic finite automaton
- Here transitions on a symbol go to a set of states, not just a single state

Nondeterminism - NFA

- A generalization of the DFA is the NFA - nondeterministic finite automaton
- Here transitions on a symbol go to a set of states, not just a single state
- A string x is accepted if some path labeled x leads to a final state.

Example of an NFA



The Frobenius Problem and NFA to DFA Conversion

When converting an NFA of n states to an equivalent DFA via the subset construction, 2^n states are sufficient by the "subset construction".

The Frobenius Problem and NFA to DFA Conversion

When converting an NFA of n states to an equivalent DFA via the subset construction, 2^n states are sufficient by the "subset construction".

What may be less well-known is that this construction is optimal in the case of a binary or larger input alphabet, in that there exist languages L that can be accepted by an NFA with n states, but no DFA with $< 2^n$ states accepts L.

The Frobenius Problem and NFA to DFA Conversion

When converting an NFA of n states to an equivalent DFA via the subset construction, 2^n states are sufficient by the "subset construction".

What may be less well-known is that this construction is optimal in the case of a binary or larger input alphabet, in that there exist languages L that can be accepted by an NFA with n states, but no DFA with $< 2^n$ states accepts L.

However, for unary (1-letter) languages, the 2^n bound is not attainable.

Unary NFA to DFA Conversion

It can be proved that approximately $e^{\sqrt{n \log n}}$ states are necessary and sufficient in the worst case to go from a unary n-state NFA to a DFA.

Unary NFA to DFA Conversion

It can be proved that approximately $e^{\sqrt{n \log n}}$ states are necessary and sufficient in the worst case to go from a unary n-state NFA to a DFA.

Chrobak showed that any unary n-state NFA can be put into a certain normal form, where there is a “tail” of $< n^2$ states, followed by a single nondeterministic state which has branches into different cycles, where the total number of states in all the cycles is $\leq n$.

Unary NFA to DFA Conversion

It can be proved that approximately $e^{\sqrt{n \log n}}$ states are necessary and sufficient in the worst case to go from a unary n-state NFA to a DFA.

Chrobak showed that any unary n-state NFA can be put into a certain normal form, where there is a “tail” of $< n^2$ states, followed by a single nondeterministic state which has branches into different cycles, where the total number of states in all the cycles is $\leq n$.

The bound of n^2 for the number of states in the tail comes from the bound we have already seen on the Frobenius problem.

An Exercise

Use the Frobenius problem on two variables to show that the language

$$L_n = \{\mathtt{a}^i \ : \ i \neq n\}$$

can be accepted by an NFA with $O(\sqrt{n})$ states.

Related Problems

As we already have seen, Sylvester published a paper in 1882 where he defined $h(x_1, x_2, \ldots, x_n)$ to be the total number of integers not representable as an integer linear combination of the x_i.

Related Problems

As we already have seen, Sylvester published a paper in 1882 where he defined $h(x_1, x_2, \ldots, x_n)$ to be the total number of integers not representable as an integer linear combination of the x_i.

He also gave the formula $h(x_1, x_2) = \frac{1}{2}(x_1 - 1)(x_2 - 1)$.

Related Problems

As we already have seen, Sylvester published a paper in 1882 where he defined $h(x_1, x_2, \ldots, x_n)$ to be the total number of integers not representable as an integer linear combination of the x_i.

He also gave the formula $h(x_1, x_2) = \frac{1}{2}(x_1 - 1)(x_2 - 1)$.

There is a very simple proof of this formula. Consider all the numbers between 0 and $(x_1 - 1)(x_2 - 1)$. Then it is not hard to see that every representable number in this range is paired with a non-representable number via the map $c \to c'$, where $c' = (x_1 - 1)(x_2 - 1) - c - 1$, and vice-versa.

Computing h is NP-hard

Computing h is NP-hard:

Computing h is NP-hard

Computing h is NP-hard:

Theorem. $h(x_1, x_2, \ldots, x_k) = h(x_1, x_2, \ldots, x_k, d)$ if and only iff d can be expressed as a non-negative integer linear combination of the x_i.

Computing h is NP-hard

Computing h is NP-hard:

Theorem. $h(x_1, x_2, \ldots, x_k) = h(x_1, x_2, \ldots, x_k, d)$ if and only iff d can be expressed as a non-negative integer linear combination of the x_i.

It follows that the integer knapsack problem (known to be NP-complete) can be reduced to the problem of computing h, and so computing h is also NP-hard (under Turing reductions).

In this problem, we are given a set of denominations $1 = x_1, x_2, \ldots, x_k$ of stamps, and an envelope that can contain at most t stamps. We want to determine the *smallest* amount of postage we *cannot* provide. Call it $N_t(x_1, x_2, \ldots, x_k)$.

The Local Postage Stamp Problem

In this problem, we are given a set of denominations $1 = x_1, x_2, \ldots, x_k$ of stamps, and an envelope that can contain at most t stamps. We want to determine the *smallest* amount of postage we *cannot* provide. Call it $N_t(x_1, x_2, \ldots, x_k)$.

For example, $N_3(1, 4, 7, 8) = 25$.

The Local Postage Stamp Problem

Many papers have been written about this problem, especially in Germany and Norway. Algorithms have been given for many special cases.

The Local Postage Stamp Problem

Many papers have been written about this problem, especially in Germany and Norway. Algorithms have been given for many special cases.

Alter and Barnett asked (1980) if $N_t(x_1, x_2, \ldots, x_k)$ can be "expressed by a simple formula".

The Local Postage Stamp Problem

Many papers have been written about this problem, especially in Germany and Norway. Algorithms have been given for many special cases.

Alter and Barnett asked (1980) if $N_t(x_1, x_2, \ldots, x_k)$ can be "expressed by a simple formula".

The answer is, probably not. I proved computing $N_t(x_1, x_2, \ldots, x_k)$ is NP-hard in 2001.

The Global Postage-Stamp Problem

The global postage-stamp problem is yet another variant: now we are given a limit t on the number of stamps to be used, and an integer k, and the goal is to find a set of k denominations $x_1, x_2, \ldots, x_k$ that maximizes $N_t(x_1, x_2, \ldots, x_k)$.

The Global Postage-Stamp Problem

The global postage-stamp problem is yet another variant: now we are given a limit t on the number of stamps to be used, and an integer k, and the goal is to find a set of k denominations $x_1, x_2, \ldots, x_k$ that maximizes $N_t(x_1, x_2, \ldots, x_k)$.

The complexity of this problem is unknown.

The Optimal Coin Change Problem

Yet another variant is the optimal change problem: here we are given a bound on the number of distinct coin denominations we can use (but allowing arbitrarily many of each denomination), and we want to find a set that minimizes the average number of coins needed to make each amount in some range.

The Optimal Coin Change Problem

Yet another variant is the optimal change problem: here we are given a bound on the number of distinct coin denominations we can use (but allowing arbitrarily many of each denomination), and we want to find a set that minimizes the average number of coins needed to make each amount in some range.

For example, in the US we currently use 4 denominations for change under 1 dollar: 1¢, 5¢, 10¢, and 25¢. These can make change for every amount between 0¢ and 99¢, with an average cost of 4.7 coins per amount.

The Optimal Coin Change Problem

Yet another variant is the optimal change problem: here we are given a bound on the number of distinct coin denominations we can use (but allowing arbitrarily many of each denomination), and we want to find a set that minimizes the average number of coins needed to make each amount in some range.

For example, in the US we currently use 4 denominations for change under 1 dollar: 1¢, 5¢, 10¢, and 25¢. These can make change for every amount between 0¢ and 99¢, with an average cost of 4.7 coins per amount.

It turns out that the system of denominations $(1, 5, 18, 25)$ is optimal, with an average cost of only 3.89 coins per amount.

Improving the Current Coin System

You could also ask, what single denomination could we add to the current system to improve its efficiency in making change?

Improving the Current Coin System

You could also ask, what single denomination could we add to the current system to improve its efficiency in making change?

The answer is, add a 32-cent piece.

Improving the Current Coin System

You could also ask, what single denomination could we add to the current system to improve its efficiency in making change?

The answer is, add a 32-cent piece.

In Canada, where there are 1-dollar and 2-dollar coins, the best coin to add is an 83-cent piece.

Improving the Current Coin System

You could also ask, what single denomination could we add to the current system to improve its efficiency in making change?

The answer is, add a 32-cent piece.

In Canada, where there are 1-dollar and 2-dollar coins, the best coin to add is an 83-cent piece.

Improving the Euro coin system

Europe uses a system of coins based on $1, 2, 5$:

$$1, 2, 5, \quad 10, 20, 50, \quad 100, 200, \ldots$$

Improving the Euro coin system

Europe uses a system of coins based on $1, 2, 5$:

$$1, 2, 5, \quad 10, 20, 50, \quad 100, 200, \ldots$$

This may seem natural, but a small change to

$$1, 3, 4, \quad 10, 30, 40, \quad 100, 300, 400, \ldots$$

would significantly decrease the average number of coins per transaction.

Improving the Euro coin system

This new system has the following advantages:

Improving the Euro coin system

This new system has the following advantages:

- Change can still be made on a digit-by-digit basis. For example, to make change for 348, first do the hundreds digit (getting 300), then the tens (getting 40), and then the ones (getting 4+4).

Improving the Euro coin system

This new system has the following advantages:

- Change can still be made on a digit-by-digit basis. For example, to make change for 348, first do the hundreds digit (getting 300), then the tens (getting 40), and then the ones (getting $4+4$).
- The greedy algorithm can be used in all cases but one. The exception is that $6 = 3+3$ and not $4+1+1$. (Similarly, $60 = 30+30$, etc.)

Improving the Euro coin system

This new system has the following advantages:

- Change can still be made on a digit-by-digit basis. For example, to make change for 348, first do the hundreds digit (getting 300), then the tens (getting 40), and then the ones (getting 4+4).
- The greedy algorithm can be used in all cases but one. The exception is that $6 = 3+3$ and not $4+1+1$. (Similarly, $60 = 30+30$, etc.)
- Assuming the uniform distribution of change denominations, on all scales (10, 100, 1000, etc.) the new system is about 6% better.

Improving the Euro coin system

This new system has the following advantages:

- Change can still be made on a digit-by-digit basis. For example, to make change for 348, first do the hundreds digit (getting 300), then the tens (getting 40), and then the ones (getting 4+4).
- The greedy algorithm can be used in all cases but one. The exception is that $6 = 3+3$ and not $4+1+1$. (Similarly, $60 = 30+30$, etc.)
- Assuming the uniform distribution of change denominations, on all scales (10, 100, 1000, etc.) the new system is about 6% better.
- If one assumes change denominations are distributed by Benford's law, the new system is about 7% better up to 10, about 6% better up to 100, and about 6% better up to 1000.

Before, we had defined $g(x_1, x_2, \ldots, x_k)$ to be the largest integer not representable as a non-negative integer linear combination of the x_i.

Generalizing the Frobenius Problem to Words

Before, we had defined $g(x_1, x_2, \ldots, x_k)$ to be the largest integer not representable as a non-negative integer linear combination of the x_i.

We can now replace the integers x_i with words (strings of symbols over a finite alphabet Σ), and ask, what is the right generalization of the Frobenius problem?

Generalizing the Frobenius Problem to Words

There are several possible answers.

There are several possible answers.

One is as follows:

Instead of non-negative integer linear combinations of the x_i, we could consider the regular expressions

$$x_1^* x_2^* \cdots x_k^*$$

There are several possible answers.

One is as follows:

Instead of non-negative integer linear combinations of the x_i, we could consider the regular expressions

$$x_1^* x_2^* \cdots x_k^*$$

or

$$\{x_1, x_2, \ldots, x_k\}^*.$$

Generalizing the Frobenius Problem to Words

Instead of the condition that $\gcd(x_1, x_2, \ldots, x_k) = 1$, which was used to ensure that the number of unrepresentable integers is finite, we could demand that

$$\Sigma^* - x_1^* x_2^* \cdots x_k^*$$

Generalizing the Frobenius Problem to Words

Instead of the condition that $\gcd(x_1, x_2, \ldots, x_k) = 1$, which was used to ensure that the number of unrepresentable integers is finite, we could demand that

$$\Sigma^* - x_1^* x_2^* \cdots x_k^*$$

or

$$\Sigma^* - \{x_1, x_2, \ldots, x_k\}^*$$

be finite, or in other words, that

$$x_1^* x_2^* \cdots x_k^*$$

Instead of the condition that $\gcd(x_1, x_2, \ldots, x_k) = 1$, which was used to ensure that the number of unrepresentable integers is finite, we could demand that

$$\Sigma^* - x_1^* x_2^* \cdots x_k^*$$

or

$$\Sigma^* - \{x_1, x_2, \ldots, x_k\}^*$$

be finite, or in other words, that

$$x_1^* x_2^* \cdots x_k^*$$

or

$$\{x_1, x_2, \ldots, x_k\}^*$$

be *co-finite*.

And instead of looking for the largest non-representable integer, we could ask for the **length of the longest word** not in

$$x_1^* x_2^* \cdots x_k^*$$

Generalizing the Frobenius Problem to Words

And instead of looking for the largest non-representable integer, we could ask for the **length of the longest word** not in

$$x_1^* x_2^* \cdots x_k^*$$

or

$$\{x_1, x_2, \ldots, x_k\}^*.$$

Theorem. Let $x_1, x_2, \ldots, x_k \in \Sigma^+$. Then $x_1^* x_2^* \cdots x_k^*$ is co-finite if and only if $|\Sigma| = 1$ and $\gcd(|x_1|, \ldots, |x_k|) = 1$.

Theorem. Let $x_1, x_2, \ldots, x_k \in \Sigma^+$. Then $x_1^* x_2^* \cdots x_k^*$ is co-finite if and only if $|\Sigma| = 1$ and $\gcd(|x_1|, \ldots, |x_k|) = 1$.

Proof. Let $Q = x_1^* x_2^* \cdots x_k^*$.

Theorem. Let $x_1, x_2, \ldots, x_k \in \Sigma^+$. Then $x_1^* x_2^* \cdots x_k^*$ is co-finite if and only if $|\Sigma| = 1$ and $\gcd(|x_1|, \ldots, |x_k|) = 1$.

Proof. Let $Q = x_1^* x_2^* \cdots x_k^*$.

If $|\Sigma| = 1$ and $\gcd(|x_1|, \ldots, |x_k|) = 1$, then every sufficiently long unary word can be obtained by concatenations of the x_i, so Q is co-finite.

Theorem. Let $x_1, x_2, \ldots, x_k \in \Sigma^+$. Then $x_1^* x_2^* \cdots x_k^*$ is co-finite if and only if $|\Sigma| = 1$ and $\gcd(|x_1|, \ldots, |x_k|) = 1$.

Proof. Let $Q = x_1^* x_2^* \cdots x_k^*$.

If $|\Sigma| = 1$ and $\gcd(|x_1|, \ldots, |x_k|) = 1$, then every sufficiently long unary word can be obtained by concatenations of the x_i, so Q is co-finite.

For the other direction, suppose Q is co-finite. If $|\Sigma| = 1$, let $\gcd(|x_1|, \ldots, |x_k|) = d$. If $d > 1$, Q contains only words of length divisible by d, and so is not co-finite. So $d = 1$.

Hence assume $|\Sigma| \geq 2$, and let a, b be distinct letters in Σ.

Hence assume $|\Sigma| \geq 2$, and let a, b be distinct letters in Σ.

Let $\ell = \max_{1 \leq i \leq k} |x_i|$, the length of the longest word among the x_i.

Hence assume $|\Sigma| \geq 2$, and let a, b be distinct letters in Σ.

Let $\ell = \max_{1 \leq i \leq k} |x_i|$, the length of the longest word among the x_i.

Let $Q' = ((a^{2\ell} b^{2\ell})^k)^+$. Then we claim that $Q' \cap Q = \emptyset$.

Hence assume $|\Sigma| \geq 2$, and let a, b be distinct letters in Σ.

Let $\ell = \max_{1 \leq i \leq k} |x_i|$, the length of the longest word among the x_i.

Let $Q' = ((a^{2\ell} b^{2\ell})^k)^+$. Then we claim that $Q' \cap Q = \emptyset$.

For if none of the x_i consists of powers of a single letter, then the longest block of consecutive identical letters in any word in Q is $< 2\ell$, so no word in Q' can be in Q.

Otherwise, say some of the x_i consist of powers of a single letter.

Otherwise, say some of the x_i consist of powers of a single letter.

Take any word w in Q, and count the number $n(w)$ of maximal blocks of 2ℓ or more consecutive identical letters in w. (Here "maximal" means such a block is delimited on both sides by either the beginning or end of the word, or a different letter.)

Otherwise, say some of the x_i consist of powers of a single letter.

Take any word w in Q, and count the number $n(w)$ of maximal blocks of 2ℓ or more consecutive identical letters in w. (Here "maximal" means such a block is delimited on both sides by either the beginning or end of the word, or a different letter.)

Clearly $n(w) \leq k$.

Otherwise, say some of the x_i consist of powers of a single letter.

Take any word w in Q, and count the number $n(w)$ of maximal blocks of 2ℓ or more consecutive identical letters in w. (Here "maximal" means such a block is delimited on both sides by either the beginning or end of the word, or a different letter.)

Clearly $n(w) \leq k$.

But $n(w') \geq 2k$ for any word w' in Q'. Thus Q is not co-finite, as it omits all the words in Q'. □

$\{x_1, x_2, \ldots, x_k\}^*$

Suppose $\max_{1 \leq i \leq k} |x_i| = n$.

$\{x_1, x_2, \ldots, x_k\}^*$

Suppose $\max_{1 \le i \le k} |x_i| = n$.

We can obtain an exponential upper bound on length of the longest omitted word, as follows:

$\{x_1, x_2, \ldots, x_k\}^*$

Suppose $\max_{1 \leq i \leq k} |x_i| = n$.

We can obtain an exponential upper bound on length of the longest omitted word, as follows:

Given $x_1, x_2, \ldots, x_k$, create a DFA accepting $\Sigma^* - \{x_1, x_2, \ldots, x_k\}^*$. This DFA keeps track of the last $n - 1$ symbols seen, together with markers indicating all positions within those $n - 1$ symbols where a partial factorization of the input into the x_i could end.

$\{x_1, x_2, \ldots, x_k\}^*$

Suppose $\max_{1 \le i \le k} |x_i| = n$.

We can obtain an exponential upper bound on length of the longest omitted word, as follows:

Given $x_1, x_2, \ldots, x_k$, create a DFA accepting $\Sigma^* - \{x_1, x_2, \ldots, x_k\}^*$. This DFA keeps track of the last $n-1$ symbols seen, together with markers indicating all positions within those $n-1$ symbols where a partial factorization of the input into the x_i could end.

Since this DFA accepts a finite language, the longest word it accepts is bounded by the number of states.

$\{x_1, x_2, \ldots, x_k\}^*$

But is this exponential upper bound attainable?

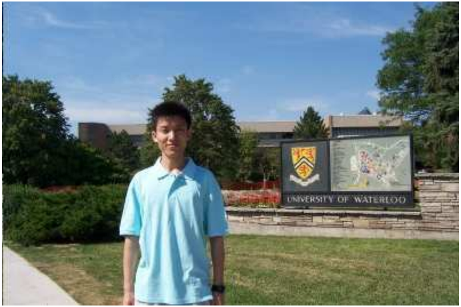
$\{x_1, x_2, \ldots, x_k\}^*$

But is this exponential upper bound attainable?

Yes.

But is this exponential upper bound attainable?

Yes.



My student Zhi Xu has recently produced a class of examples $\{x_1, x_2, \ldots, x_k\}$ in which the length of the longest word is n, but the longest word in $\Sigma^* - \{x_1, x_2, \ldots, x_k\}^*$ is exponential in n.

$\{x_1, x_2, \ldots, x_k\}^*$: Zhi Xu's Examples

Let $r(n, k, l)$ denote the word of length l representing n in base k, possibly with leading zeros. For example, $r(3, 2, 3) = 011$.

$\{x_1, x_2, \ldots, x_k\}^*$: Zhi Xu's Examples

Let $r(n, k, l)$ denote the word of length l representing n in base k, possibly with leading zeros. For example, $r(3, 2, 3) = 011$.

Let $T(m, n) = \{r(i, |\Sigma|, n-m)0^{2m-n}r(i+1, |\Sigma|, n-m) \ : \ 0 \leq i \leq |\Sigma|^{n-m} - 2\}$.

$\{x_1, x_2, \ldots, x_k\}^*$: Zhi Xu's Examples

Let $r(n, k, l)$ denote the word of length l representing n in base k, possibly with leading zeros. For example, $r(3, 2, 3) = 011$.

Let $T(m, n) = \{r(i, |\Sigma|, n - m)0^{2m-n}r(i + 1, |\Sigma|, n - m) \ : \ 0 \leq i \leq |\Sigma|^{n-m} - 2\}$.

Theorem. Let m, n be integers with $0 < m < n < 2m$ and $\gcd(m, n) = 1$, and let $S = \Sigma^m + \Sigma^n - T(m, n)$. Then S^* is co-finite and the longest words not in S^* are of length $g(m, l)$, where $l = m|\Sigma|^{n-m} + n - m$.

$\{x_1, x_2, \ldots, x_k\}^*$: Zhi Xu's Examples

Let $r(n, k, l)$ denote the word of length l representing n in base k, possibly with leading zeros. For example, $r(3, 2, 3) = 011$.

Let $T(m, n) = \{r(i, |\Sigma|, n - m)0^{2m-n}r(i + 1, |\Sigma|, n - m) \ : \ 0 \leq i \leq |\Sigma|^{n-m} - 2\}$.

Theorem. Let m, n be integers with $0 < m < n < 2m$ and $\gcd(m, n) = 1$, and let $S = \Sigma^m + \Sigma^n - T(m, n)$. Then S^* is co-finite and the longest words not in S^* are of length $g(m, l)$, where $l = m|\Sigma|^{n-m} + n - m$.

Example. Let $m = 3, n = 5, \Sigma = \{0, 1\}$. In this case, $l = 3 \cdot 2^2 + 2 = 14$, $S = \Sigma^3 + \Sigma^5 - \{00001, 01010, 10011\}$. Then a longest word not in S^* is

$$00001010011\ 000\ 00001010011$$

of length $25 = g(3, 14)$.

Counting the Omitted Words

Zhi Xu has also generated some examples where the number of omitted words is doubly exponential in n, the length of the longest word.

Counting the Omitted Words

Zhi Xu has also generated some examples where the number of omitted words is doubly exponential in n, the length of the longest word.

Let $T'(m,n) = \{r(i, |\Sigma|, n-m)0^{2m-n}r(j, |\Sigma|, n-m) \ : \ 0 \leq i < j \leq |\Sigma|^{n-m} - 1\}$.

Counting the Omitted Words

Zhi Xu has also generated some examples where the number of omitted words is doubly exponential in n, the length of the longest word.

Let $T'(m,n) = \{r(i, |\Sigma|, n-m)0^{2m-n}r(j, |\Sigma|, n-m) \ : \ 0 \le i < j \le |\Sigma|^{n-m} - 1\}$.

Theorem. Let m, n be integers with $0 < m < n < 2m$ and $\gcd(m,n) = 1$, and let $S = \Sigma^m + \Sigma^n - T'(m,n)$. Then S^* is co-finite and S^* omits at least $2^{|\Sigma|^{n-m}} - |\Sigma|^{n-m} - 1$ words.

Counting the Omitted Words

Zhi Xu has also generated some examples where the number of omitted words is doubly exponential in n, the length of the longest word.

Let $T'(m,n) = \{r(i,|\Sigma|,n-m)0^{2m-n}r(j,|\Sigma|,n-m) \ : \ 0 \leq i < j \leq |\Sigma|^{n-m} - 1\}$.

Theorem. Let m, n be integers with $0 < m < n < 2m$ and $\gcd(m,n) = 1$, and let $S = \Sigma^m + \Sigma^n - T'(m,n)$. Then S^* is co-finite and S^* omits at least $2^{|\Sigma|^{n-m}} - |\Sigma|^{n-m} - 1$ words.

Example. Let $m = 3, n = 5, \Sigma = \{0,1\}$. Then $S = \Sigma^3 + \Sigma^5 - \{00001, 00010, 00011, 01010, 01011, 10011\}$. Then S^* omits $1712 > 11 = 2^{2^2} - 2^2 - 1$ words.

Other Possible Generalizations

Instead of considering the longest word omitted by $x_1^* x_2^* \cdots x_k^*$ or $\{x_1, x_2, \ldots, x_k\}^*$, we might consider their state complexity.

Other Possible Generalizations

Instead of considering the longest word omitted by $x_1^* x_2^* \cdots x_k^*$ or $\{x_1, x_2, \ldots, x_k\}^*$, we might consider their state complexity.

The *state complexity* of a regular language L is the smallest number of states in any DFA that accepts L. It is written $\mathrm{sc}(L)$.

Other Possible Generalizations

Instead of considering the longest word omitted by $x_1^* x_2^* \cdots x_k^*$ or $\{x_1, x_2, \ldots, x_k\}^*$, we might consider their state complexity.

The *state complexity* of a regular language L is the smallest number of states in any DFA that accepts L. It is written $\mathrm{sc}(L)$.

It turns out that the state complexity of $\{x_1, x_2, \ldots, x_k\}^*$ can be exponential in both the length of the longest word and the number of words.

Theorem. Let t be an integer ≥ 2, and define words as follows:

$$y := 01^{t-1}0$$

and

$$x_i := 1^{t-i-1}01^{i+1}$$

for $0 \leq i \leq t-2$. Let $S_t := \{0, x_0, x_1, \ldots, x_{t-2}, y\}$. Then S_t^* has state complexity $3t2^{t-2} + 2^{t-1}$.

State Complexity

Theorem. Let t be an integer ≥ 2, and define words as follows:

$$y := 01^{t-1}0$$

and

$$x_i := 1^{t-i-1}01^{i+1}$$

for $0 \leq i \leq t-2$. Let $S_t := \{0, x_0, x_1, \ldots, x_{t-2}, y\}$. Then S_t^* has state complexity $3t2^{t-2} + 2^{t-1}$.

Example. For $t = 6$ the words in S_t are 0 and

$$\begin{aligned} y &= 0111110 \\ x_0 &= 1111101 \\ x_1 &= 1111011 \\ x_2 &= 1110111 \\ x_3 &= 1101111 \\ x_4 &= 1011111 \end{aligned}$$

State Complexity

Using similar ideas, we can also create an example achieving subexponential state complexity for $x_1^* x_2^* \cdots x_k^*$.

State Complexity

Using similar ideas, we can also create an example achieving subexponential state complexity for $x_1^* x_2^* \cdots x_k^*$.

Theorem. Let y and x_i be as defined above. Let $L = (0^* x_1^* x_2^* \cdots x_{n-1}^* y^*)^e$ where $e = (t+1)(t-2)/2 + 2t$. Then $\mathrm{sc}(L) \geq 2^{t-2}$.

This example is due to Jui-Yi Kao.

Complexity

Theorem. If S, a finite list of words, is represented by either an NFA or a regular expression, then determining if S^* is co-finite is NP-hard and is in PSPACE.

Complexity

Theorem. If S, a finite list of words, is represented by either an NFA or a regular expression, then determining if S^* is co-finite is NP-hard and is in PSPACE.

Theorem. If S is a unary language (possibly infinite) represented by an NFA, then we can decide in polynomial time if S^* is co-finite.

Open Problem

We still do not know the complexity of the following problem:

Open Problem

We still do not know the complexity of the following problem:

Given a finite list of words $S = \{x_1, x_2, \ldots, x_k\}$, determine if S^* is co-finite.

Another generalization

Define $g_j(a_1, \ldots, a_n)$ to be the largest integer having exactly j representations as a non-negative integer linear combination of the integers a_i.

Another generalization

Define $g_j(a_1, \ldots, a_n)$ to be the largest integer having exactly j representations as a non-negative integer linear combination of the integers a_i.

It seems reasonable that $g_0(\cdots) < g_1(\cdots)$, but this is not always true.

Another generalization

Define $g_j(a_1, \ldots, a_n)$ to be the largest integer having exactly j representations as a non-negative integer linear combination of the integers a_i.

It seems reasonable that $g_0(\cdots) < g_1(\cdots)$, but this is not always true.

We constructed a class of 5-tuples for which $g_0(\cdots) = n^2 - O(n)$, but $g_k(X_n) = (6k+3)n - 1$ for all sufficiently large n.

Another generalization

Define $g_j(a_1, \ldots, a_n)$ to be the largest integer having exactly j representations as a non-negative integer linear combination of the integers a_i.

It seems reasonable that $g_0(\cdots) < g_1(\cdots)$, but this is not always true.

We constructed a class of 5-tuples for which $g_0(\cdots) = n^2 - O(n)$, but $g_k(X_n) = (6k+3)n - 1$ for all sufficiently large n.

Also $g_0(24, 26, 36, 39) = 181$ but $g_1(24, 26, 36, 39) = 175$.

Another generalization

Define $g_j(a_1, \ldots, a_n)$ to be the largest integer having exactly j representations as a non-negative integer linear combination of the integers a_i.

It seems reasonable that $g_0(\cdots) < g_1(\cdots)$, but this is not always true.

We constructed a class of 5-tuples for which $g_0(\cdots) = n^2 - O(n)$, but $g_k(X_n) = (6k+3)n - 1$ for all sufficiently large n.

Also $g_0(24, 26, 36, 39) = 181$ but $g_1(24, 26, 36, 39) = 175$.

Conjecture: for all triples a_1, a_2, a_3 of distinct integers we have $g_0 < g_1 < \cdots < g_{14}$.

Another generalization

Define $g_j(a_1, \ldots, a_n)$ to be the largest integer having exactly j representations as a non-negative integer linear combination of the integers a_i.

It seems reasonable that $g_0(\cdots) < g_1(\cdots)$, but this is not always true.

We constructed a class of 5-tuples for which $g_0(\cdots) = n^2 - O(n)$, but $g_k(X_n) = (6k+3)n - 1$ for all sufficiently large n.

Also $g_0(24, 26, 36, 39) = 181$ but $g_1(24, 26, 36, 39) = 175$.

Conjecture: for all triples a_1, a_2, a_3 of distinct integers we have $g_0 < g_1 < \cdots < g_{14}$.

This would be best possible, since $g_{14}(8, 9, 15) = 172$, but $g_{15}(8, 9, 15) = 169$.

For Further Reading

- J. L. Ramírez Alfonsín, *The Diophantine Frobenius Problem*, Oxford University Press, 2005.
- J. Shallit, The computational complexity of the local postage stamp problem, *SIGACT News* **33** (1) (March 2002), 90–94.
- J. Shallit, What this country needs is an 18-cent piece, *Math. Intelligencer* **25** (2) (2003), 20–23.
- Jui-Yi Kao, J. Shallit, and Zhi Xu, "The Frobenius problem in a free monoid", in S. Albers and P. Weil, eds., *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science*, 2008, pp. 421–432.
- J. Shallit and J. Stankewicz, Unbounded discrepancy in Frobenius numbers, *INTEGERS* **11** (2011), paper #A2.