A Space-efficient Gödel Numbering with Chinese Remainder Theorem

Shi-Chun Tsai* Jen-Chun Chang† Rong-Jaye Chen‡

Abstract

Representing a sequence of nonnegative integers in terms of two integers is important in various considerations dealing with logic and mathematics. It is commonly used in the so-called Gödel numbering in mathematical logic. We developed a space-efficient algorithm to represent a sequence of nonnegative integers with two integers based on the Chinese Remainder Theorem [2]. This improved a previous result [2, 3] by significantly reducing the size of the resulting integers.

1 Introduction

Gödel numbering [1, 6, 3] is originally used to encode symbols in logic, where it enables one to regard interpreted languages to be the natural numbers and thus one can represent expressions in a first order language as natural numbers. The arithmetization of first order logic, illustrated by the Gödel numbering, has been used in first order logic to prove the well-known completeness and incompleteness [1, 6]. With Gödel encoding method, a sequence of non-negative integers $a_1, a_2, \cdots, a_t$ can be represented by its length t and another two integers [3, 2]. In this paper we show how to encode Gödel numbers efficiently in space by using the Chinese Remainder Theorem (CRT) [2]. Our result uses $O(t \log a)$ space to store the encoded numbers, which improves the previous result with space complexity $O(t \log a + t^2 \log t)$ [2, 3], where a is the largest one among the t integers. Throughout this paper the base of logarithm is 2 unless stated otherwise. The space complexity of an integer is simply the number of bits in its binary representation. From the perspective of computational complexity [8, 7, 4], space complexity is an important measurement for classifying the difficulty of computational problems. Many interesting game related problems are categorized with space complexity [8].

CRT has a long list of applcaitions in mathematics, algorithms, coding theory, cryptography, etc [2]. First let's review the theorem.

Theorem 1.1 *Let $a_0, \cdots, a_t$ be any t integers and $m_0, \cdots, m_t$ be relatively prime in pairs. Then there is a number x with the property $x \equiv a_i \pmod{m_i}$, $i = 0, \cdots, t$. Moreover, x is unique in the following senses. Let M be $m_0 \cdots m_t$ and let y satisfy the above system of congruences, then $y \equiv x \pmod{M}$.*

As pointed out by Kunen [5], Gödel numbering, in practice, would be extremely awkward to deal with. This paper shows an improvement on the efficiency of Gödelinzation. A previous method [2, 3] provided a representation of t nonnegative integers

*Computer Science and Info Engineering Dept, National Chiao-Tung University, Hsinchu 30050, Taiwan, Email: sctsai@csie.nctu.edu.tw. The work was supported in part by the National Science Council of Taiwan under contract NSC 89-2213-E-260-009.

†Department of Information Management, Ming-Hsin Institute of Technology, Hsinfong 304, Hsinchu, Taiwan, Email: jcchang@csie.nctu.edu.tw

‡Computer Science and Info. Engineering Dept, National Chiao-Tung University, Hsinchu 30050, Taiwan, Email: rjchen@csie.nctu.edu.tw *2000 Mathematics Subject Classification:* 03F40, 03F20, 68W40. *Key words and phrases:* Computational complexity, Godel numbering, Chinese Remainder Theorem

a_i, $i = 1, \cdots, t$, with two integers u and v. Let $a = \max_{i=1}^{t} a_i$, $v = 2a(t-1)!$ and $m_i = 1 + v \cdot i$, $1 \leq i \leq t$. Then, by the Chinese Remainder Theorem (CRT), u is obtained from the system of congruences $u \equiv a_i \pmod{m_i}$, $1 \leq i \leq t$. We summarize the old method in Figure 1.

OLDENCODER(p, t)
Input: p is a vector of $t (\geq 3)$
nonnegative integers $a_1, \cdots, a_t$.
Output: Two integers u and v.

1. $a = \max_{i=1}^{t} a_i$;
2. $v = 2a \cdot (t-1)!$;
3. **for**$i = 1$ to t **do**
4. $\quad m_i = 1 + v \cdot i$;
5. With CRT, find u from the system of congruences $u \equiv a_i \pmod{m_i}$, $i = 1..t$;
6. **return** u and v;

OLDDECODER(u, v, t)
Input: Two large integers u and v;
t is the number of integers to be decoded.
Output: t nonnegative integers $a_1, \cdots, a_t$..

1. **for** $i = 1$ to t **do**
2. $\quad a_i = u \bmod (1 + i \cdot v)$;

Figure 1. The old method of representing integers.

By a simple analysis on the size of u and v in Figure 1, we find that $size(v) = O(\log a + t \log t)$ and u can be as large as $O(\Pi_{i=1}^{t} m_i) = O(\Pi_{i=1}^{t}(1 + i \cdot v)) = O(v^t t!)$, thus $size(u) = O(t \log a + t^2 \log t)$. In the following, we show our method based on the CRT to represent a sequence of non-negative integers with two integers whose sizes are significantly smaller than the above. But before that, let's observe the following simple fact, which shows how to represent two non-negative integers with one.

Fact 1.2 *Given two nonnegative integers n_1 and n_2, let c be an integer greater than or equal to n_1 and n_2. Then we can find an integer u such that $n_1 = u \bmod (c+1)$ and $n_2 = u \bmod (c+2)$.*

Since $c+1$ and $c+2$ are relatively prime for any nonnegative integer c, this fact is clear by CRT. There is always an u, $0 \leq u < (c+1)(c+2)$, such that $n_1 = u \bmod (c+1)$ and $n_2 = u \bmod (c+2)$. Note that the upper bound of u is $O(c^2)$.

We outline our algorithm in Figure 1 and 1. The encoding procedure is presented in Figure 1. In this procedure, if $t = 3$, we use the OLDENCODER routine to encode the input integers. Otherwise, in each round we choose a number c and use it to merge every two integers into one by CRT. Then the number c and the resulting $\lceil \frac{t}{2} \rceil$ integers become the input of the next round. The process is repeated until there are only three integers left. Then the OLDENCODER routine is applied to obtain u and v. The decoding procedure in Figure 1 works reversely. Note that in both the encoding and decoding procedures the value of c carries two pieces of information. One is to indicate that the modulo bases are $c+1$ and $c+2$, and the other is to indicate whether the last integer in the list in each round is encoded from one or two integers in the previous round. We choose c such that if c is odd, the last integer is encoded from an integer (i.e, the last integer itself); otherwise it is encoded from two integers. Thus we can use the parity of c as a tag to decode the last integer as in lines 13–17 of procedure DECODENUMBERS.

Fact 1.3 *In line 4 of procedure* ENCODENUMBERS*, c is odd iff t is an odd number.*

Proof. Let a be an arbitrary integer. If t is odd, then $c = a + ((a+t) \bmod 2) = a + ((a+1) \bmod 2)$, which is always odd. If t is even, then $c = a + ((a+t) \bmod 2) = a + (a \bmod 2)$, which is even. □

EncodeNumbers(p, t)

Input: p is a list of $t (\geq 3)$ nonnegative integers $a_1, \cdots, a_t$.

Output: Two integers u and v.

1. **if** $(t == 3)$ **then return** OldEncoder(p, t);
2. Let p' be an empty new list;
3. Let a be the maximum among the remaining integers in p ;
4. Let $c = a + ((a + t) \bmod 2)$;
   /* c is odd if and only if there are odd number of integers in the list p. */
5. Append c to the end of p';
6. **while** (list p has at least two integers) **do**
7.   Remove the first two integers, n_1 and n_2 from p ;
8.   With CRT, find x satisfying $x \equiv n_1 \pmod{c+1}$ and $x \equiv n_2 \pmod{c+2}$;
       /* $x \equiv n_2 + (c+2)(n_1 - n_2) \pmod{(c+1)(c+2)}$ */
9.   Append x to the end of p' ;
10. **if** (p still has one integer left) **then** remove and append it to the end of list p';
11. EncodeNumbers($p', \lceil \frac{t}{2} \rceil + 1$);

Figure 2. The encoding procedure.

DecodeNumbers(u, v, t)

Input: u and v are two integers; t is the number of integers to be decoded.

Output: t nonnegative integers $a_1, \cdots, a_t$.

1. $i = 3$;
2. OldDecoder($u, v, 3$);
3. Organize the decoded integers into a list p ;
4. **while** $(i < t)$ **do**
5.   Let c be the first integer in p and remove it from p ;
6.   Let p' be an empty list ;
7.   **while** (list p has more than one integer) **do**
8.   Let u be the first integer in p and remove it from p ;
9.   Append $(u \bmod (c+1))$ to p' ;
10.   Append $(u \bmod (c+2))$ to p' ;
11. $i = i + 2$;
12.   Let u be the last integer in p and remove it from p ;
13.   **if** (c is odd) **then**
14.   Append u to p' ;
15. $i = i + 1$;
16.   **else do** Append $(u \bmod (c+1))$ to p' ;
17.   Append $(u \bmod (c+2))$ to p' ;
18. $i = i + 2$;
19. $p = p'$;
20. **return** the integers in list p;

Figure 3. The decoding procedure.

2 Analysis

First let's analyze the time complexity of the algorithm. The base case in our encoding (or decoding) procedure is when $t = 3$, where we use the method of Ding et al. for encoding and decoding. Observe that for any $t > 3$ by applying the formula $\lceil \frac{t}{2} \rceil + 1$ recursively, the sequence starting with t will converge to 3. This fact can be shown by induction. It is trivial for $t = 4$. Suppose it is true for all $t < n$. For $t = n$, it is clear that $\lceil \frac{n}{2} \rceil + 1 < n$ and then by induction hypothesis we know t always converges to 3 and thus our algorithm does terminate. We check that the algorithm EncodeNumbers recurs $O(\log t)$ times, since the number of intgers decreases by half in each recursive call. For each call, it takes $O(t)$ steps. Thus the total time complexity is $O(t \log t)$. Reversely, it takes $O(t \log t)$ steps to decode the encoded t non-negative integers.

Next, we analyze the space complexity. Let a be the largest integer in the input. According to the CRT and our encoding algorithm, we know that after each recurrsive call each integer can be as large as the suqare of the largest integer of the prevevious round. Since the recurrsive depth is $O(\log t)$, we have the integers u and v as large as $O(a^t)$. Thus $size(u)$ and $size(v)$ both have size $O(t \log a)$, which improves the previous result $O(t \log a + t^2 \log t)$. It is clear that we use significantly less space when t gets larger. We summarize the result in the following theorem.

Theorem 2.1 (Main result) *Let $a_0, \cdots, a_t$ be a finite sequence of non-negative integers. Then by applying Chinese Remainder Theorem two integers u and v are sufficient to represent the above integers, where $size(u)$ and $size(v)$ are both $O(t \log a)$, $a = \max_i a_i$.*

3 Concluding Remarks

We have shown a new space-efficient method of representing a sequence of non-negative integers via the CRT. We also have thought about to compress the integers with fewer bits with the CRT, but our initial try was not successful. It seemed every time we applied the CRT, the size of integers would double in the worst case. It will be ineteresting to know if we can use the CRT for compression.

References

[1] G. Boolos and R. Jeffrey Computability and Logic, 3rd Ed, Cambridge University Press, 1989.

[2] C. Ding and D. Pei and A. Salomaa Chinese Remainder Theorem– Applications in Computing, Coding and Cryptography, World Scientific Publishing Co., 1996.

[3] Y. V. Matiyasevich Hilbert's Tenth Problem, The MIT Press, Cambridge, London, England, 1993.

[4] D.-Z. Du and Ker-I Ko Theory of Computational Compexity, John Wiley & Sons Inc., 2000.

[5] K. Kunen A Ramsey Theorem in Boyer-Moore Logic, Journal of Automated Reasoning (15), pp. 217-235, 1995.

[6] E. Mendelson Introduction to Mathematical logic, Van Nostrand, Princeton, 1964.

[7] C. Papadimitriou Computational Complexity, Addison-Wesley Publishing Co,, 1995.

[8] M. Sipser Introduction to the Theory Computation, PWS Publishing Company, 1997.