On arithmetic operations with M-out-of-N-codes

On arithmetic operations with M-out-of-N-codes

by

Wilhelmina M.C.J. van Overveld

85/02

February 1985

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
All rights reserved
editor: M.L. Potters

On arithmetic operations with M-out-of-N- codes

Wilhelmina M. C. J. van Overveld
Department of Mathematics and Computing Science,
Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

Abstract

We consider an M-out-of-N- code with M=n, N=2n. This means that all rows of 2n bits with exactly n ones are used as codewords to represent the numbers 0,1,... $\binom{2n}{n}$-1. We are interested in arithmetic operations on these codewords; of course algorithms for addition and multiplication depend on the code chosen. We give several examples of this.
To obtain nice, efficient algorithms, it appears to be advantageous to use only part of all codewords.

Contents

0. The problem

Suppose we have N channels with output '0' or '1' for each channel. The input of the channels does not matter here. With these N channel bits we want to represent integers. An obvious way to do this is to use binary representation: each bit can be '0' or '1', so we can represent 2^N numbers $(0,1,\ldots 2^N-1)$.

In practice this representation has the following disadvantage. It is possible that the N channels do not deliver their outputs at the same time; some may be slower than others. Suppose the initial state of all bits is '0'. Some of them may change into '1' after a while, but we don't know how long it will take before all bits have reached their final state. Only when we are sure that no '0' will turn into '1' any more, we have reached the final state and we can read the represented number.

A solution to this problem uses so called 'M-out-of-N- codes'. This means that we allow only those final states where exactly M of the N bits are '1' and N-M bits are '0'.

Indeed, as soon as M bits have turned into '1', we know that there will be no more changes: we are in the final state and the represented number is known.

However, this solution also has its disadvantages, since we can't represent 2^N numbers anymore. There are only $\binom{N}{M}$ possible rows of N bits with exactly M ones, so there are $\binom{N}{M}$ final states allowed. A final state of this form will also be called a 'codeword' or just 'word'. Still, we would like to represent as many numbers as possible in this way. It can be easily seen that the best choices for M and N are: $N=2n$, $M=n$ $(n \in \mathbb{N})$.

In the sequel we shall only consider 'n-out-of-2n- codes': the codewords are rows of 2n bits with n ones. With these words we represent the integers $0,1,\ldots\binom{2n}{n}-1$.

There is one more problem concerning this way of coding: in the binary system there is an easy way to add or multiply two numbers as bit sequences, but it is not clear whether this is also possible with n-out-of-2n- codes!

Of course there is one trivial way to add codewords: convert the words to the corresponding numbers, add these numbers and convert the sum back to a codeword. We wonder whether there is also a way to add two words directly, without converting.

This paper is the result of some research done into the question of 'how difficult' it is to use arithmetic operations with n-out-of-2n-codes. Obviously we can construct many different codes of this type, and the addition algorithm for a code will depend on the construction of the code. That is why we shall consider quite a few examples of codes, and for each code we shall give algorithms for converting words to numbers and vice versa, and an algorithm for adding two words. In some cases we shall even be able to multiply words in a fairly easy way.

For the sake of simplicity, we shall never consider overflow.

The structure of this paper is as follows.

- In chapter 1 we shall consider a number of ways to code all numbers $0,1,\dots\binom{2n}{n}-1$. In most cases, it will appear that the addition algorithms come down to a somewhat disguised form of 'convert to binary numbers, add the numbers and convert back to a word'.
- In chapter 2 we shall see that we do find ways to perform a 'direct' addition of codewords, if only we weaken our constraints. We shall not code all of the $\binom{2n}{n}$ numbers, but we shall consider a limited number of the $\binom{2n}{n}$ possible codewords.

For the codes we find in this manner we shall be able to give some nice addition algorithms, but we shall also prove that the number of used codewords (i.e. the number of represented numbers) is a very small fraction of the total number of words which is $\binom{2n}{n}$.
In fact, in all cases the fraction

$$\frac{\#\text{ codewords}}{\binom{2n}{n}}$$

vanishes for $n \to \infty$. If we look at the quotient of the logarithms, i.e.

$$\frac{\log_2 (\#\text{ codewords})}{\log_2 \binom{2n}{n}} ,$$

we have a measure for the number of bits used for the codewords compared to the number of bits it would take if we would code all words. The behaviour of this quotient for $n \to \infty$ will also be investigated.

- Chapter 3, finally, consists of some ideas for coding that gave rise to problems during research. However, it is imaginable that the principles of the ideas are nevertheless useful, and they may yield practical addition algorithms.

1. Codes that use all words

1.0 The lexicographical ordering

We consider all words of length 2n with n zeros and n ones, and we order them lexicographically. Let '$\prec$' denote 'lexicographically less than', then $0 \prec 1$. E.g. for n=4 we have:
$00001111 \prec 00010111 \prec 00011011 \prec \ldots$
With each word we associate a 'lexicographical number': the number that word has in the lexicographical ordering. So, in our example:
$00001111 \leftrightarrow 0$, $00010111 \leftrightarrow 1$, $00011011 \leftrightarrow 2$, etcetera.
These numbers lie in the range $0 \ldots \binom{2n}{n}-1$.

We now define the coding; let $g \in \left\{0,1,\ldots,\binom{2n}{n}-1\right\}$.
The number g will be coded as the g^{th} word in the lexicographical ordering, that is, the word with lexicographical number g. This word will be written as $\underline{x}_g$: a rowvector over $\{0,1\}$ of length 2n.

There are algorithms known for the conversion of g to $\underline{x}_g$, and $\underline{x}_g$ to g (cf. [1]). The most straightforward algorithms are the following.

<u>A0</u> Convert $\underline{x}_g$ to g. Given an integer array $w(i:1 \leq i \leq 2n)$ satisfying $\forall i: 1 \leq i \leq 2n: w(i) \in \{0,1\}$. This array contains the word $\underline{x}_g$. We calculate the lexicographical number, g.

```
i:= 2n; g:= 0; a:= 0;
do i ≠ 0 → if w(i)= 0 → skip
           [] w(i)= 1 → g:= g + $\binom{2n-i}{a+1}$; a:= a + 1
           fi;
           i:= i - 1
od
```

Invariant relation: a= the number of ones in $w(j: i < j \leq 2n)$

$$g = \sum_{\text{all ones in } w(j:i<j\leq 2n)} \binom{\#\text{positions in } w(j:i<j\leq 2n) \text{ to the right of this '1'}}{\#\text{ones in } w(j:i<j\leq 2n) \text{ to the right of this '1'} + 1}$$

From this we see that the '1' in the figure below causes the term $\binom{2n-i}{a+1}$ in the sum of g.

$$\underbrace{\lfloor (n-a-1 \text{ ones}) \rfloor}_{i-1 \text{ positions}} 1 \underbrace{\lfloor (a \text{ ones}) \rfloor}_{2n-i \text{ positions}}$$

Apparently, the lexicographical number of the word in $w(i{:}1 \leq i \leq 2n)$ is

$$g = \sum_{\text{all ones in } w(i:1\leq i\leq 2n)} \binom{\#\text{ positions to the right of this '1'}}{\#\text{ ones to the right of this '1'} \;+1}$$

We can see this using the following argument. The number corresponding to a word $\underline{x}$ equals the number of words that are lexicographically less than $\underline{x}$. Suppose the first bit of $\underline{x}$ (on the left) is a '1', then all words starting with '0' are lexicographically less than $\underline{x}$. There are $\binom{2n-1}{n}$ of them. So this first '1' causes a term $\binom{2n-1}{n}$ in g. Note that this is indeed one of the terms in the given expression for g. We can argue in the same way if $\underline{x}$ starts with 0^k1 (k zeros followed by a 1). Analogously the remaining n-1 ones in $\underline{x}$ each yield a term in the summation of g. The interested reader is referred to [1].

<u>BO</u> Convert g to $\underline{x}_g$. Given a number g, $0 \leq g \leq \binom{2n}{n} - 1$, we want to construct $\underline{x}_g$. The word $\underline{x}_g$ will be stored in the array $w(i{:}1 \leq i \leq 2n)$.

```
i:= 1; a:= 0; s:= 0;
do a ≠ n → if g ≥ s + $\binom{2n-i}{n-a}$ → w(i):= 1; s:= s + $\binom{2n-i}{n-a}$; a:= a + 1
           [] g < s + $\binom{2n-i}{n-a}$ → w(i):= 0
           fi;
           i:= i + 1
od;
do i ≠ 2n+1 → w(i):= 0; i:= i + 1 od
```

Invariant relation: $a =$ the number of ones in $w(j{:}\ 1 \leq j < i)$

$$s = \sum_{\text{all ones in } w(j:\ 1\leq j< i)} \binom{\#\text{ positions to the right of the '1'}}{\#\text{ ones to the right of the '1'} \;+1}$$

Explanation: we check position i to see if we can put a '1' in that position. This '1' would contribute $\binom{2n-i}{n-a}$ to the lexicographical number. If the total contribution of all ones in $w(j\colon 1 \leq j \leq i)$ does not exceed g, we may indeed set $w(i)$ to '1', otherwise it will have to be '0'. After termination of the first 'do'- loop, all ones are in the right positions. If necessary, we complete the word with zeros in the final positions.

We can easily generalize the algorithms A0 and B0 for words of length ℓ with exactly k ones and ℓ-k zeros. The lexicographical number of such a word can be found in exactly the same way: the '1' in the word below contributes $\binom{\ell-i}{a+1}$ to the number.

$$\underbrace{\text{(k-a-1 ones)}}_{\text{i-1 pos.}}\ 1\ \underbrace{\text{(a ones)}}_{\ell\text{-i pos.}}$$

We give both algorithms (called A1 and B1) in a version that is at the same time more efficient than the one above. Indeed, the programs A0 and B0 need a computing time of $O(n^2)$: a loop is executed n times and each time we calculate $\binom{2n-i}{n-a}$. By introducing an extra variable we can compute the binomial coefficients more efficiently, thus reducing the computing time to $O(n)$ - or $O(\ell)$ in the generalized case.
We can do this by using the following relations:

$$\binom{\ell-i+1}{a+2} = \frac{\ell-i+1}{a+2}\binom{\ell-i}{a+1} \quad , \quad \binom{\ell-i+1}{a+1} = \frac{\ell-i+1}{\ell-i-a}\binom{\ell-i}{a+1} \quad \text{etc.}$$

```
A1   i:= $\ell$ ; g:= 0; bin:= 0;
  do i $\neq$ 0 $\rightarrow$ if w(i) = 0 $\rightarrow$ if bin = 0 $\rightarrow$ bin:= 1
                                  [] bin > 0 $\rightarrow$ bin:= bin $* \frac{(\ell-i+1)}{(\ell-i-a)}$
                                  fi;
                   [] w(i) = 1 $\rightarrow$ g:= g + bin; bin:= bin $* \frac{(\ell-i+1)}{(a+2)}$;  a:= a + 1
                   fi;
                   i:= i-- 1
  od
```

Invariant relation: a= # ones in $w(j\colon i < j \leq \ell)$, bin= $\binom{\ell-i}{a+1}$ and

$$g = \sum_{\text{ones in } w(j:\ i<j\leq \ell)} \binom{\#\text{ positions to the right of the '1'}}{\#\text{ ones to the right of the '1' }+1}$$

```
B1   i:= 1; a:= 0; bin:= $\binom{\ell-1}{k}$; s:= 0;
  do a $\neq$ k $\rightarrow$ if g $\geq$ s + bin $\rightarrow$ w(i):= 1; s:= s + bin;
                               if i < $\ell$ $\rightarrow$ bin:= bin $*$ $\frac{(k-a)}{(\ell-i)}$
                               [] i = $\ell$ $\rightarrow$ bin:= 0
                               fi; a:= a + 1
                 [] g < s + bin $\rightarrow$ w(i):= 0;
                               if i < $\ell$ $\rightarrow$ bin:= bin $*$ $\frac{(\ell-i-k+a)}{(\ell-i)}$
                               [] i = $\ell$ $\rightarrow$ bin:= 0
                               fi
                 fi; i:= i + 1
  od;
  do i $\neq$ $\ell$+1 $\rightarrow$ w(i) := 0; i:= i + 1 od
```

Invariant relation: a= # ones in w(j: $1 \leq j < i$), bin= $\binom{\ell-i}{k-a}$,

$$s = \sum_{\text{ones in } w(j:\ 1 \leq j < i)} \binom{\#\text{ positions to the right of the '1'}}{\#\text{ ones to the right of the '1'} + 1}$$

In the next paragraphs we shall use these generalized algorithms frequently, for arbitrary k and ℓ; for this reason we have given them explicitly.

At this point, we can already draw an important conclusion from the fact that the conversion algorithms have a computing time that is linear in n. Suppose we want to add $\underline{x}_i$ and $\underline{x}_j$, then we can do that by converting to i and j and afterwards converting i+j to $\underline{x}_{i+j}$. From the foregoing we conclude that this addition also has a computing time of O(n). Further, we realize that any addition algorithm must be at least O(n): we shall have to take a look at all bits of $\underline{x}_i$ and $\underline{x}_j$ at least once. Hence our conclusion is that we can't possibly design an addition algorithm that is more efficient than the one above. This only holds for worst-case behaviour; we may be able to construct an algorithm with a better average-case behaviour.

Nevertheless we are going to make an attempt to add two words $\underline{x}_i$ and $\underline{x}_j$ directly, without the use of i and j. For example, there is an easy way to derive the lexicographical successor of a word $\underline{x}_i$; this corresponds to increasing i by 1.Description of the algorithm:

i) determine the rightmost pair '01' in $\underline{x}_i$. This pair always exists, except when $i = \binom{2n}{n}-1$: then $\underline{x}_i = 1^n0^n$.

ii) Replace this pair by '10'.

iii) Make the part of the word to the right of this pair lexicographically as small as possible, that is, put all ones in this part in the rightmost positions. Thus the word will look like $\ldots 10\ 0^a1^b$, $a,b \geqslant 0$.

E.g. if $\underline{x}_i = 010\underline{01}110$ then $\underline{x}_{i+1} = 010\underline{10}011$ ($i = 18$).
It is just as easy to determine the lexicographical predecessor of a word (in fact, this algorithm is obtained from the one above by merely interchanging '1' and '0'.). Both algorithms are $O(n)$.

Thus we arrive at the trivial addition algorithm below.

```
if $\underline{x}_i \preceq \underline{x}_j \rightarrow \underline{x} := \underline{x}_j;\ \underline{y} := \underline{x}_i$
[] $\underline{x}_i \succeq \underline{x}_j \rightarrow \underline{x} := \underline{x}_i;\ \underline{y} := \underline{x}_j$
fi; $\{\underline{x} \succeq \underline{y}\}$
do $\underline{y} \neq 0^n1^n \rightarrow$ $\underline{x}$:= lex. sucessor of $\underline{x}$;
                    $\underline{y}$:= lex. predecessor of $\underline{y}$
od
```

Of course this is not a practical algorithm, since it is $O(\binom{2n}{n})$.

However, apart from increasing i by 1, there are more 'elementary' additions possible, even of order $O(1)$. Suppose $\underline{x}_i$ is of the following type.

$$\underbrace{(n-a \text{ ones})}_{2n-k-2}\ \big|\ 0\ \big|\ 1\ \big|\ \underbrace{(a-1 \text{ ones})}_{k}$$

The indicated '1' contributes $\binom{k}{a}$ to the lexicographical number. If we change the pair '01' into '10', the '1' contributes $\binom{k+1}{a}$; the other terms in the summation do not change. Hence, by this replacement we add $\binom{k+1}{a} - \binom{k}{a} = \binom{k}{a-1}$ to the initial word $\underline{x}_i$.

The consequence is , that for every word $\underline{x}_i$ there are certain numbers j such that $\underline{x}_{i+j}$ can be easily derived from $\underline{x}_i$. For such a pair i,j we call this an elementary addition. The general problem is still open; an addition will have to be split up into elementary additions and subtractions.
It is not clear how this can be implemented.

1.1 Variants of the lexicographical ordering

1.1.0

For a word $\underline{x}$ we define the numbers i_0 and j_0:

- i_0 is defined as the number of ones preceding the leftmost '0' in the word $\underline{x}$;
- j_0 is the lexicographical number of the word $\underline{x}'$, consisting of the rightmost $2n-i_0-1$ bits of $\underline{x}$. This word $\underline{x}'$ has $n-i_0$ ones; see figure below.

$$\underline{x} = \underbrace{11\ldots1}_{i_0 \text{ pos.}}\,0\,\underbrace{(n-i_0 \text{ ones})}_{=\underline{x}'}$$

The word $\underline{x}$ will be denoted as $\underline{x}_{i_0,j_0}$. The ordering in the words $\underline{x}_{i_0,j_0}$ is defined according to the lexicographical ordering of the pairs (i_0,j_0).
Hence: $\underline{x}_{i_0,j_0} \prec \underline{x}_{i_1,j_1} \Leftrightarrow (i_0,j_0) \prec (i_1,j_1) \Leftrightarrow (i_0 < i_1) \vee (i_0 = i_1 \wedge j_0 < j_1)$.

Note that this ordering in the words is exactly the same as the lexicographical ordering of § 1.0.
The number of words with i ones preceding the first '0' is $\binom{2n-i-1}{n-i}$.
Hence the number corresponding to $\underline{x}_{i_0,j_0}$ is given by

$$\sum_{i=0}^{i_0-1} \binom{2n-i-1}{n-i} + j_0$$

(We count all words $\underline{x}_{i,j}$ with $i < i_0$; there are j_0 words $\underline{x}_{i_0,j}$ with $j < j_0$.)

Now we shall give an algorithm to add $\underline{x}_{i_0,j_0}$ and $\underline{x}_{i_1,j_1}$; we call the sum $\underline{x}_{i_2,j_2}$. We use two tables, stored in integer arrays:

$$\text{sigma}(i_0\text{: } 0 \leq i_0 \leq n) \text{ with} \begin{cases} \text{sigma}(0) = 0, \\ \text{sigma}(i_0) = \sum_{i=0}^{i_0-1} \binom{2n-i-1}{n-i}, & i_0 > 0 \end{cases}$$

and term(i: $0 \leq i \leq n$) with
$\text{term}(i) = \binom{2n-i-1}{n-i}$.

Algorithm:

i) Convert $\underline{x}_{i_0,j_0}$ and $\underline{x}_{i_1,j_1}$ to (i_0,j_0) and (i_1,j_1). For j_0 we use the algorithm A1 (§ 1.0) with $\ell = 2n-i_0-1$, $k = n-i_0$, and the word $\underline{x}'$ mentioned in the definition of $\underline{x}_{i_0,j_0}$. Analogously we find j_1.

ii) Now the following holds: the number of $\underline{x}_{i_0,j_0} = \text{sigma}(i_0) + j_0$
the number of $\underline{x}_{i_1,j_1} = \text{sigma}(i_1) + j_1$.

We add these numbers in two steps:

- calculate sigma(i_0) + j_0 + y for y = sigma (i_1); write the sum as sigma (i_2') + j_2'.
- calculate sigma(i_2') + j_2' + y for y = j_1; the sum is sigma (i_2) + j_2.

We only give the first step.

```
s:= sigma($i_0$); $i_2'$:= $i_0$; y:= y+$j_0$;
do y $\geq$ term($i_2'$) $\rightarrow$ s:= s + term ($i_2'$); y:= y - term($i_2'$); $i_2'$:= $i_2'$ + 1
od ; $j_2'$:= y
```

Invariant relation: sigma(i_0) + j_0 + sigma(i_1) = s + y, s= sigma(i_2').
After termination: answer= sigma(i_0) + j_0 + sigma (i_1) = sigma(i_2') + j_2'.

iii) Convert (i_2,j_2) to $\underline{x}_{i_2,j_2}$ using algorithm B1.

About the complexity of this algorithm, we can say that step ii) is certainly O(n), and in the average case it is much better. Steps i) and iii) are $O(2n-i_0-1)$ because of the algorithms A1 and B1. So we see that for small values of i_0, there is hardly any difference with the addition algorithm of § 1.0 that uses conversion.

For large values of i_0 the algorithms A1 and B1 need much less computing time, and step ii) does not really take more time than it does for small i_0. We conclude that this algorithm is somewhat more efficient in the average case than the one in § 1.0, but it makes use of O(n) tables.

All of the following algorithms in § 1.1 are also based on the handling of indices like i_0 and j_0.

1.1.1

For a word $\underline{x}$ we define i_0, j_0 and k_0:

- i_0:= the number of ones in the first n positions of $\underline{x}$ $(0 \leq i_0 \leq n)$.
- j_0:= the lexicographical number of the word $\underline{x}'$, formed by the first n positions of $\underline{x}$. $\underline{x}'$ has length n and i_0 ones.
- k_0:= the lexicographical number of the word $\underline{x}''$, formed by the last n positions of $\underline{x}$. It has $n-i_0$ ones.

Note that $0 \leq j_0 < \binom{n}{i_0}$ and $0 \leq k_0 < \binom{n}{n-i_0} = \binom{n}{i_0}$.

Again we denote $\underline{x}$ as $\underline{x}_{i_0,j_0,k_0}$.

The ordering in these words corresponds to the lexicographical ordering in (i_0,j_0,k_0); this is not the same as the lexicographical ordering in the words! To illustrate this, we give the first 11 words for n=4: on the left, ordered lexicographically; on the right, ordered according to (i_0,j_0,k_0). Note the differences in the last two words.

			i_0,j_0,k_0
0	00001111	00001111	0, 0, 0
1	00010111	00010111	1, 0, 0
2	00011011	00011011	1, 0, 1
3	00011101	00011101	1, 0, 2
4	00011110	00011110	1, 0, 3
5	00100111	00100111	1, 1, 0
6	00101011	00101011	1, 1, 1
7	00101101	00101101	1, 1, 2
8	00101110	00101110	1, 1, 3
9	00110011	01000111	1, 2, 0
10	00110101	01001011	1, 2, 1

There are $\binom{n}{i}\binom{n}{n-i} = \binom{n}{i}^2$ words with i ones in the first n positions. Hence the number of words $\underline{x}_{i,j,k}$ with $i < i_0$ is

$$\sum_{i=0}^{i_0-1} \binom{n}{i}^2 .$$

The number of words $\underline{x}_{i_0,j,k}$ with $j < j_0$ is $j_0\binom{n}{n-i_0} = j_0\binom{n}{i_0}$ (for every j we have $\binom{n}{n-i_0}$ possibilities for k).
For the number represented by $\underline{x}_{i_0,j_0,k_0}$ we find

$$\sum_{i=0}^{i_0-1} \binom{n}{i}^2 + j_0\binom{n}{i_0} + k_0.$$

For the addition of $\underline{x}_{i_0,j_0,k_0}$ and $\underline{x}_{i_1,j_1,k_1}$ -with result $\underline{x}_{i_2,j_2,k_2}$ - we use three integer arrays:
nchoosei(i_0):= $\binom{n}{i_0}$, nchoosei2 (i_0):= $\binom{n}{i_0}^2$ and

sigma(i_0):= $\sum_{i=0}^{i_0-1} \binom{n}{i}^2$ for $0 \leq i_0 \leq n$.

Algorithm:

i) Determine (i_0,j_0,k_0) and (i_1,j_1,k_1) with algorithm A1.

ii) Calculate sigma(i_0) + j_0 nchoosei(i_0) + k_0 + y for y= sigma(i_1). Write the result as sigma(i_2') + j_2' nchoosei(i_2') + k_2' and add to this y= j_1 nchoosei(i_1) and y= k_1, respectively.
Again, only the first part is written in full.

```
s:= sigma($i_0$); $i_2'$:= $i_0$; y:= y + $j_0$*nchoosei($i_0$) + $k_0$;
do y ≥ nchoosei2($i_2'$) → s:= s + nchoosei2($i_2'$);
                          y:= y - nchoosei2($i_2'$); $i_2'$:= $i_2'$ + 1
od;
$j_2'$:= y div nchoosei($i_2'$);  $k_2'$:= y mod nchoosei($i_2'$)
```

Invariant relation: answer= sigma(i_0) + j_0 nchoosei(i_0) + k_0 + sigma (i_1) = s + y and s = sigma(i_2').

iii) Convert (i_2,j_2,k_2) to $\underline{x}_{i_2,j_2,k_2}$ with algorithm B1.

This, too, is an algorithm of O(n).

1.1.2

We abbreviate a:= 00, b:= 01, c:= 10, d:= 11.
A word in $\{0,1\}^{2n}$ corresponds to a word in $\{a,b,c,d\}^n$: we take two successive bits together to form a word in $\{a,b,c,d\}^n$.
E.g. 01|00|11|01 = badb.

Further, if $\underline{x} \in \{a,b,c,d\}^n$, we denote #a:= the number of a's in $\underline{x}$. Similarly we write #b, #c, #d. For the words in our n-out-of-2n- code we know the number of '1's is n. For a corresponding word in $\{a,b,c,d\}^n$ this means #b + #c + 2#d = n and #a + #b + #c + #d = n, hence #a = #d.

We define the numbers i_0,j_0,k_0 and ℓ_0 for a word $\underline{x} \in \{a,b,c,d\}^n$:

- i_0:= #a, so $0 \leq i_0 \leq \lfloor n/2 \rfloor$
- j_0:= the lexicographical number of the word we get from $\underline{x}$ if we distinguish between a's and non-a's only. Let a correspond to '1' and non-a to '0'. This word has length n and i_0 ones, so $0 \leq j_0 < \binom{n}{i_0}$
- k_0:= the lexicographical number of the word of length $n-i_0$ that remains if we delete all a's from $\underline{x}$, where we distinguish between d's and non-d's. d corresponds to '1', non-d to '0'. We find $0 \leq k_0 < \binom{n-i_0}{i_0}$.

- ℓ_0:= the number represented by the b's and c's in the word $\underline{x}$ if we regard the b's and c's as 0's and 1's ,respectively, in a binary representation. There are $n-2i_0$ positions that are b or c, thus $0 \le \ell_0 < 2^{n-2i_0}$.

Example: consider the word abbacdbd (n=8). Then $i_0 = \#a = 2$.
j_0 can be found by substituting '0' for b,c and d and '1' for a: 10010000. j_0= lex. number of 10010000 = $\binom{4}{1} + \binom{7}{2} = 25$.
We find k_0 by deleting the a's: bbcdbd. Write d= 1, b= c= 0 to get k_0 = lex. number of 000101 = 1. To find ℓ_0 we delete the d's: bbcb, and think of this as the binary number 0010. Hence $\ell_0 = 2$.

The word corresponding to i_0,j_0,k_0,ℓ_0 is denoted as $\underline{x}_{i_0,j_0,k_0,\ell_0}$; the ordering in these words is defined as before. Reasoning as in § 1.1.0 and § 1.1.1 we deduce the number of $\underline{x}_{i_0,j_0,k_0,\ell_0}$; it is:

$$\sum_{i=0}^{i_0-1} \binom{n}{i}\binom{n-i}{i} 2^{n-2i} + j_0 \binom{n-i_0}{i_0} 2^{n-2i_0} + k_0\, 2^{n-2i_0} + \ell_0 \;.$$

From the example we see that the conversion of $\underline{x}_{i_0,j_0,k_0,\ell_0}$ to (i_0,j_0,k_0,ℓ_0) and vice versa is somewhat more complicated than in the previous examples, but the computation remains O(n).

For the addition we use some tables, containing

$$\sum_{i=0}^{i_0-1} \binom{n}{i}\binom{n-i}{i} 2^{n-2i} \;,\quad \binom{n}{i_0}\binom{n-i_0}{i_0} 2^{n-2i_0} \quad \text{and} \quad 2^{n-2i_0}\binom{n-i_0}{i_0}.$$

For transparency, we shall not use arraynames for these tables in the program below, but write $\binom{n}{i_0}\binom{n-i_0}{i_0}2^{n-i_0}$, etc.
Keep in mind that these are not O(n) computations, but direct accesses to a table. Like the preceding algorithms, we only give the addition

$$\sum_{i=0}^{i_0-1} \binom{n}{i}\binom{n-i}{i} 2^{n-2i} + j_0 \binom{n-i_0}{i_0} 2^{n-2i_0} + k_0\, 2^{n-2i_0} + \ell_0 + y.$$

Here and in the next sections we shall omit the steps i) and iii), since they are the same for every algorithm.

```
$s := \sum_{i=0}^{i_0-1} \binom{n}{i}\binom{n-i}{i} 2^{n-2i}$; $i_2' := i_0$; $y := y + j_0 \binom{n-i_0}{i_0} 2^{n-2i_0} + k_0\, 2^{n-2i_0} + \ell_0$;
do $y \geq \binom{n}{i_2'}\binom{n-i_2'}{i_2'} 2^{n-2i_2'}$ → $s := s + \binom{n}{i_2'}\binom{n-i_2'}{i_2'} 2^{n-2i_2'}$;
                                   $y := y - \binom{n}{i_2'}\binom{n-i_2'}{i_2'} 2^{n-2i_2'}$; $i_2' := i_2' + 1$
od;
$\ell_2' := y$ mod $2^{n-2i_2'}$; $k_2' := (y-\ell_2')2^{-n+2i_2'}$ mod $\binom{n-i_2'}{i_2'}$;
$j_2' := (y-\ell_2')\, 2^{-n+2i_2'}$ div $\binom{n-i_2'}{i_2'}$
Invariant relation: answer = $y+s$; $s = \sum_{i=0}^{i_2'-1} \binom{n}{i}\binom{n-i}{i} 2^{n-2i}$ .
```

This algorithm may be preferred to the one in § 1.1.1 in the sense that the tables are of length $\lceil n/2 \rceil$ instead of n, since from the definition of i_0 it follows that $i_0 \leq \lfloor n/2 \rfloor$.
Apart from this, the algorithm is O(n).

A final remark of this section deals with a generalized form of the algorithm. If we have words with even length ℓ and exactly k ones, we can also use a's, b's, c's and d's. The requirements change into:
$\#a + \#b + \#c + \#d = \ell/2$ and $\#b + \#c + 2\#d = k$.
Hence $\#a - \#d = \ell/2 - k$. Once we choose #a for a word, #d is fixed: $\#d = \#a - \ell/2 + k$. The b's and c's can be chosen arbitrarily on the $\ell/2 - \#a - \#d$ ($= \ell - k - 2\#a$) remaining positions.

Again, let $i_0 :=$ #a, $j_0 :=$ lexicographical number only minding the a's, $k_0 :=$ lexicographical number in the d's, and $\ell_0 :=$ the binary number represented by the b's and c's. The number of $\underline{x}_{i_0, j_0, k_0, \ell_0}$ we find is:

$$\sum_{i=0}^{i_0-1} \binom{\ell/2}{i}\binom{\ell/2-i}{i-\ell/2+k} 2^{\ell-k-2i} + j_0 \binom{\ell/2-i_0}{i_0-\ell/2+k} 2^{\ell-k-2i_0} + k_0\, 2^{\ell-k-2i_0} + \ell_0 .$$

1.1.3

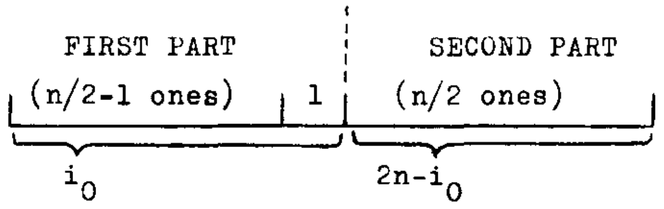
A method related to the idea of § 1.1.1: now we don't devide the word $\underline{x}$ into the first n positions and the last n positions, but we look at the distribution of the ones. Let n be even. We split $\underline{x}$ into two parts, where the first part consists of the first position upto and including the position of the $(n/2)^{th}$ one. The second part consists of the remaining rightmost positions. Therefore we define:

i_0:= the length of the first part of $\underline{x}$ (defined as above)

j_0:= the lexicographical number of the word formed by the first i_0-1 positions of $\underline{x}$. It contains n/2-1 ones: $0 \leq j_0 < \binom{i_0-1}{n/2-1}$

k_0:= the lexicographical number of the word formed by the last $2n-i_0$ positions, with n/2 ones (= the second part of $\underline{x}$).

As an illustration, see the sketch below.



Writing $\underline{x}$ as $\underline{x}_{i_0,j_0,k_0}$ and ordering the words as usual we find for the number of $\underline{x}_{i_0,j_0,k_0}$:

$$\sum_{i=0}^{i_0-1} \binom{i-1}{n/2-1}\binom{2n-i}{n/2} + j_0\binom{2n-i_0}{n/2} + k_0.$$

For the addition algorithm we use tables containing

$$\sum_{i=0}^{i_0-1} \binom{i-1}{n/2-1}\binom{2n-i}{n/2} , \binom{i_0-1}{n/2-1}\binom{2n-i_0}{n/2} \text{ and } \binom{2n-i_0}{n/2}.$$

Again, we shall not refer to these tables with arraynames. We calculate

$$\sum_{i=0}^{i_0-1} \binom{i-1}{n/2-1}\binom{2n-i}{n/2} + j_0\binom{2n-i_0}{n/2} + k_0 + y .$$

```
s:= $\sum_{i=0}^{i_0-1} \binom{i-1}{n/2-1}\binom{2n-i}{n/2}$; $i_2'$:= $i_0$; y:= y + $j_0 * \binom{2n-i_0}{n/2} + k_0$;

do y $\geq \binom{i_2'-1}{n/2-1}\binom{2n-i_2'}{n/2}$ $\rightarrow$ s:= s + $\binom{i_2'-1}{n/2-1}\binom{2n-i_2'}{n/2}$ ;
                                        y:= y - $\binom{i_2'-1}{n/2-1}\binom{2n-i_2'}{n/2}$ ; $i_2'$:= $i_2'$ + 1
od;
$j_2'$:= y div $\binom{2n-i_2'}{n/2}$ ; $k_2'$:= y mod $\binom{2n-i_2'}{n/2}$
```

There is not really any reason to prefer this algorithm to the one in § 1.1.1; they are equally efficient -O(n)-, and in fact they resemble each other very much.

1.1.4

Finally we mention a somewhat less practical variant. Define for a word $\underline{x}$ i_0, j_0 and k_0 as follows:

i_0:= the number of positions starting at the first '1', upto and including the last '1' in the word. So $n \leq i_0 \leq 2n$.

j_0:= the lexicographical number of the word of length i_0-2 occupying the positions between the first and n^{th} one: $0 \leq j_0 < \binom{i_0-2}{n-2}$.

k_0:= $2n-i_0-$ (the number of leading zeros in $\underline{x}$) : $0 \leq k_0 \leq 2n-i_0$.

See figure below.

$$\underbrace{00\ldots0}_{2n-i_0-k_0}\,1\,\underbrace{(n-2 \text{ ones})}_{i_0-2}\,1\,\underbrace{0\ldots0}_{k_0}$$

Order the words $\underline{x}_{i_0,j_0,k_0}$ according to (i_0,j_0,k_0). Note that all words $\underline{x}_{i_0,j_0,k}$ (for fixed i_0 and j_0) are cyclic shifts of each other. If we start from the word $\underline{x}_{i_0,j_0,0}$, the word $\underline{x}_{i_0,j_0,k}$ is derived by shifting k positions to the left. Obviously there are $2n-i_0+1$ words of the type $\underline{x}_{i_0,j_0,k}$. Using this we find for the number of $\underline{x}_{i_0,j_0,k_0}$:

$$\sum_{i=0}^{i_0-1} \binom{i-2}{n-2}(2n-i+1) + j_0(2n-i_0+1) + k_0.$$

The conversion $\underline{x}_{i_0,j_0,k_0} \leftrightarrow (i_0, j_0, k_0)$ can be easily performed; the computing time is O(n). Again we give the algorithm for increasing the number represented by $\underline{x}_{i_0,j_0,k_0}$ by y.

We use two arrays containing

$$\sum_{i=0}^{i_0-1} \binom{i-2}{n-2}(2n-i+1) \quad \text{and} \quad \binom{i_0-2}{n-2}(2n-i_0+1), \text{ respectively.}$$

```
s:= $\sum_{i=0}^{i_0-1} \binom{i-2}{n-2}(2n-i+1)$;  $i_2'$:= $i_0$; y:= $y + j_0(2n-i_0+1) + k_0$ ;
do y $\geq \binom{i_2'-2}{n-2}(2n-i_2'+1)$ → s:= $s + \binom{i_2'-2}{n-2}(2n-i_2'+1)$;
                                        y:= $y - \binom{i_2'-2}{n-2}(2n-i_2'+1)$; $i_2'$:= $i_2' + 1$
od;
$j_2'$:= y div $(2n-i_2'+1)$;
$k_2'$:= y mod $(2n-i_2'+1)$
```

For large values of i_0 this is not a practical algorithm: the conversion $\underline{x}_{i_0,j_0,k_0} \leftrightarrow (i_0, j_0, k_0)$ consumes the lion's share of the computing time. Thus we do essentially the same work as we did in the algorithm of § 1.0 (converting words and adding the lexicographic numbers). In the average case however, the conversions will consume less time, whilst the middle part of the algorithm consumes even less time than the conversion. So in fact we can make the same remark we made in § 1.1.0.

1.2 Further remarks and conclusions

1.2.0

Firstly, we make a general remark. In all cases we have seen so far, we assigned a word $\underline{x}_i$ to every number i $(0 \leq i < \binom{2n}{n})$. However, it suffices to assign codewords to half of the numbers, e.g. the even numbers only. We must do this in such a way that the codeword for the odd number 2k+1 can be easily derived from the word assigned to 2k. We give two examples of this.

1) Consider only those words in $\{0,1\}^{2n}$ with n ones that start with '0'. There are $\binom{2n-1}{n} = \frac{1}{2}\binom{2n}{n}$ words like this. Order these words in some way as $\underline{y}_k$, for $0 \leq k < \frac{1}{2}\binom{2n}{n}$.
Denote $(\underline{y}_k)^* :=$ the complement of the word $\underline{y}_k$, i.e. '0' and '1' swapped in $\underline{y}_k$. Now we define codeword $\underline{x}_i$ for $0 \leq i < \binom{2n}{n}$:

$$\left.\begin{array}{l} \underline{x}_{2k} := \underline{y}_k \\ \underline{x}_{2k+1} := (\underline{y}_k)^* \end{array}\right\} \; 0 \leq k < \tfrac{1}{2}\binom{2n}{n}$$

Assume we have some addition algorithm for the words $\underline{y}_k$, with the addition operator denoted as $\oplus$. Then we can define the addition for the words $\underline{x}_i$:

$$\begin{array}{llll}
\underline{x}_{2k} \oplus \underline{x}_{2\ell} := \underline{y}_k \oplus \underline{y}_\ell & \text{corresponding to} & 2k + 2\ell = 2(k+\ell) \\
\underline{x}_{2k} \oplus \underline{x}_{2\ell+1} := (\underline{y}_k \oplus \underline{y}_\ell)^* & \text{"} \quad \text{"} & 2k + (2\ell+1) = 2(k+\ell) + 1 \\
\underline{x}_{2k+1} \oplus \underline{x}_{2\ell+1} := \underline{y}_k \oplus \underline{y}_\ell \oplus \underline{y}_1 & \text{"} \quad \text{"} & (2k+1)+(2\ell+1) = 2(k+\ell+1)
\end{array}$$

We shall encounter this method once more in chapter 3.

2) Consider the code of §1.1.1, restricted to the words $\underline{x}_{i_0,j_0,k_0}$ with $i_0 \leq \lceil n/2 \rceil$. These words will play the role of the words $\underline{y}_k$ in

example no. 1). We assume n even; in that case our 'restricted code' contains exactly $\frac{1}{2}\binom{2n}{n}$ words. The ordering in the words $\underline{y}_k$ is of course the same as in § 1.1.1, and we use the same addition algorithm for these words. Again we define $\underline{x}_{2k} := \underline{y}_k$ and $\underline{x}_{2k+1} := (\underline{y}_k)^*$, with the addition defined as in the above example.

Note that the words $(\underline{y}_k)^*$ are exactly the words with more than n/2 ones in the left half: the words $\underline{x}_{i_0,j_0,k_0}$ with $i_0 > n/2$.
The tables of §1.1.1 can thus be reduced in length to n/2; the algorithm will also decrease in computing time. These savings can also be applied to some of the other algorithms given.

<u>1.2.1</u>

We summarize the results of this chapter.

Apart from the lexicographical ordering, all algorithms are based on the same principle. To every word $\underline{x}$ we assign indices like i_0, j_0, ... and we denote the word as $\underline{x}_{i_0,j_0\ldots}$. From these indices, the number represented by $\underline{x}_{i_0,j_0\ldots}$ can be calculated directly (by means of a more or less complicated expression). If we want to add the words $\underline{x}_{i_0,j_0\ldots}$ and $\underline{x}_{i_1,j_1\ldots}$ with the result called $\underline{x}_{i_2,j_2\ldots}$, we start by determining the indices $(i_0, j_0,\ldots)$ and $(i_1, j_1,\ldots)$. Then we calculate $(i_2, j_2,\ldots)$ from these; this algorithm works in such a way that we add the numbers corresponding to $(i_0, j_0,\ldots)$ and $(i_1, j_1,\ldots)$ in several steps, and we deduce the numbers $(i_2, j_2,\ldots)$ from the sum.
Afterwards we convert $(i_2, j_2,\ldots)$ to $\underline{x}_{i_2,j_2,\ldots}$.

The only difference in the algorithms is the efficiency of the middle part: calculating $(i_2, j_2,\ldots)$ from the known indices. Using 'div' and 'mod' operations, this can be done very easily.

Nevertheless we have seen that all algorithms are O(n) and that no addition can possibly need less computing time.

2. Codes that do not use all words

2.0 Motivation

From chapter 1 we have learned that all algorithms we could think of make use of the lexicographical ordering, and that addition is never done directly: we make use of an intermediate step (the indices). Indeed we have never been able to really add two bit sequences, as can be done in the binary system -or in any positional system, for that matter.

We can construct direct algorithms if we drop the requirement that all integers 0, 1 through $\binom{2n}{n}$ -1 must be coded. In that case we can design a number of algorithms that rather deviate from the ones in chapter 1, and are much easier in general. It will even be possible to give programs for multiplication; something that is hardly feasible for the codes in chapter 1 ! Section 2.3 contains a very interesting example of this.

Therefore, the main reason for giving the codes in this chapter is the fact that we can show a variety of nice addition algorithms. The practical use of these codes is rather doubtful, since we can only code a very small part of the $\binom{2n}{n}$ numbers. As mentioned before, the fraction of codable numbers goes to 0 for $n \to \infty$. We shall prove this in all cases. However, the quotient

$$\frac{\log_2 (\#\ \text{codewords})}{\log_2 \binom{2n}{n}}$$

has also been examined for all codes, as announced in the introduction. In most cases it will appear that this quotient does not vanish for large n. For some codes we even find that the limit of this quotient for $n \to \infty$ is 1. So in those cases, the result is not too bad: we can still code a considerable amount of numbers.

2.1 A code that uses permutations

2.1.0

Arbitrarily, we pick a word $\underline{x}_0$ as codeword for the number 0. Let $\pi \in S_{2n}$. Consider the effect π has on the positions of $\underline{x}_0$. By this permutation we get a new word that we call $\underline{x}_1$. We denote this as $\pi\,\underline{x}_0 = \underline{x}_1$.

Note that $\underline{x}_1$ can be equal to $\underline{x}_0$ even if π is not the identical permutation. We give an example of this for n=4.
$\underline{x}_0$:= 00001111. Number the positions from left to right as 1,2,..8.
Let $\pi := (1\ 2\ 3)(6\ 8)$, then $\pi\underline{x}_0 = \underline{x}_0$.
Let $\pi' := (1\ 2\ 5)$, then $\pi'\underline{x}_0 = 01000111 \neq \underline{x}_0$.

Repeated actions of π on $\underline{x}_0$ yield $\underline{x}_1$, $\underline{x}_2$, $\underline{x}_3$,...; in general, $\underline{x}_i = \pi^i\ \underline{x}_0$.
Let $\underline{x}_i$ be the codeword for i, then we can easily add $\underline{x}_i$ and $\underline{x}_j$:
$\underline{x}_i \oplus \underline{x}_j = \pi^{i+j}\ \underline{x}_0 = \pi^i\ \underline{x}_j = \pi^j\ \underline{x}_i$.
In case the permutations π^i are not too difficult to compute, this is a simple algorithm. Multiplication can be achieved in the same way:
$\underline{x}_i \otimes \underline{x}_j = \pi^{ij}\ \underline{x}_0 = (\pi^i)^j\ \underline{x}_0$ etc. , where $\otimes$ denotes multiplication.

However, we must recall that for $i \neq j$ it is possible that $\pi^i\ \underline{x}_0 = \pi^j\ \underline{x}_0$, or $\underline{x}_i = \underline{x}_j$ holds. In the above example this holds for i=1, j=0. Apparently, if this occurs, we cannot represent both i and j in the code, since coding should be one-to-one.
For this reason we shall investigate how many words the set $\{\pi^i\ \underline{x}_0 \mid i \in \mathbb{N}\}$ can contain, for any $\pi \in S_{2n}$. Maximizing this over S_{2n}, we find the largest possible number of codewords such a code can have.

In fact, we shall not be able to calculate this number, but we can give an upper bound for it which guarantees that it is an infinitely small fraction of $\binom{2n}{n}$ for $n \to \infty$. As for $\log_2$ (#codewords), this number is also unknown, but the quotient mentioned in § 2.0 will be bounded. We shall come back to this later.

2.1.1

A first upper bound for the number of codewords can be derived as follows. Let $\pi \in S_{2n}$. Let k be the largest length of any cycle in the disjunct cycle notation of π.
W.l.o.g. we write $\pi = (1\ 2 \ldots k)(\ldots)(\ldots)\ldots$ with $1 \leq k \leq 2n$.
(Here, the identical permutation is denoted as (1), so k=1.)
The first k positions of $\pi^i\ \underline{x}_0$, for any i, can only be a cyclic permutation of the first k positions of $\underline{x}_0$. Hence there are only k possibilities for this part of the word.
If $\underline{x}_0$ has m ones in the final 2n-k positions, any word $\underline{x}_i$ also has m

ones in those positions: there are at most $\binom{2n-k}{m}$ possibilities for this part of $\underline{x}_i$. Since $\binom{2n-k}{m} \leq \binom{2n-k}{\lfloor\frac{1}{2}(2n-k)\rfloor}$ for all m, we can have at most $k\binom{2n-k}{\lfloor\frac{1}{2}(2n-k)\rfloor}$ words of the type $\underline{x}_i = \pi^i \underline{x}_0$.

We have thus derived the upper bound:

$$\underline{\left|\left\{\pi^i \underline{x}_0 \mid i \in \mathbb{N}\right\}\right| \leq k\binom{2n-k}{\lfloor\frac{1}{2}(2n-k)\rfloor}} \qquad (1)$$

Here $|A|$ denotes the number of elements of the set A.

This bound is very weak for small k, but strong for large k. Fortunately our next bound behaves just the other way around: it is weak for large k and strong for small k. This bound is derived by considering the order of π; obviously, the order of π is an upper bound for the number of words $\pi^i \underline{x}_0$. Again we write π with disjunct cycles, $\pi = (1\ 2 \ldots k)(\ldots)\ldots$. Observe that the order of π is equal to the least common multiple (lcm) of all cyclelengths occurring in π. Since all cycles have length at most k, we have
order of $\pi \leq \operatorname{lcm}\{1,2,\ldots,k-1,k\}$. Hence:

$$\underline{\left|\left\{\pi^i \underline{x}_0 \mid i \in \mathbb{N}\right\}\right| \leq \operatorname{lcm}\{1,2,\ldots,k-1,k\}} \qquad (2)$$

Combining (1) and (2) we find

$$\left|\left\{\pi^i \underline{x}_0 \mid i \in \mathbb{N}\right\}\right| \leq \min\left\{k\binom{2n-k}{\lfloor\frac{1}{2}(2n-k)\rfloor}, \operatorname{lcm}\{1,2,\ldots,k\}\right\} .$$

<u>Lemma 2.1.1.0</u> : $\operatorname{lcm}\{1,2,\ldots,k\}$ is a not-decreasing function of k for k= 1,2,..2n.
If $n \geq 3$, $k \cdot \binom{2n-k}{\lfloor\frac{1}{2}(2n-k)\rfloor}$ is a not-increasing function of k for k= 2,3..2n-1.

<u>Proof</u>: for $\operatorname{lcm}\{1,2\ldots,k\}$: trivial.

Let k be even.

$$(k+1)\binom{2n-k-1}{\lfloor\frac{1}{2}(2n-k-1)\rfloor} = (k+1)\binom{2n-k-1}{\frac{1}{2}(2n-k)-1} = \frac{(k+1)}{2}\binom{2n-k}{\frac{1}{2}(2n-k)} \leq$$

$$\leq k\binom{2n-k}{\frac{1}{2}(2n-k)} \quad \text{for } k \geq 1.$$

Let k be odd.

$$(k+1)\binom{2n-k-1}{\frac{1}{2}(2n-k-1)} = (k+1)\cdot\frac{\frac{1}{2}(2n-k+1)}{2n-k}\binom{2n-k}{\frac{1}{2}(2n-k-1)} =$$

$$= \frac{(k+1)}{2k} \frac{(2n-k+1)}{(2n-k)} \cdot k \binom{2n-k}{\frac{1}{2}(2n-k-1)}$$

We have to prove: $\frac{(k+1)(2n-k+1)}{2k(2n-k)} \leqslant 1$ for $2 \leqslant k \leqslant 2n-2$.

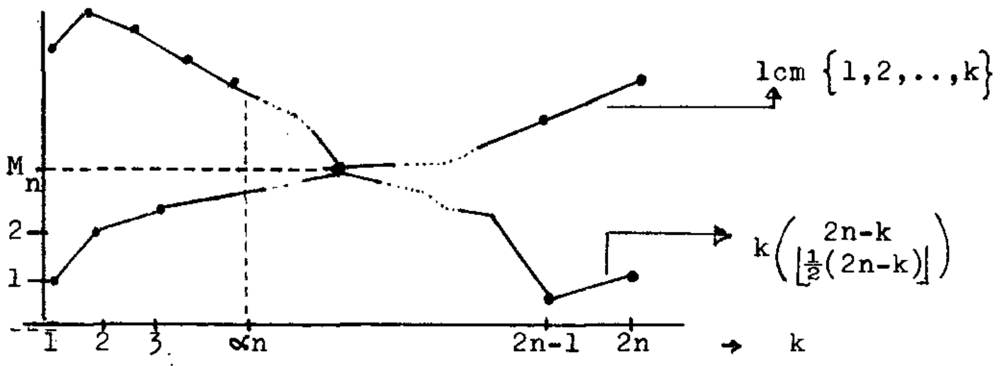
We determine the roots of the equation (in k) :

$(k+1)(2n-k+1) = 2k(2n-k)$ or $k^2-2nk+2n+1 = 0$

The roots are $k_1 := n + \sqrt{n^2-2n-1}$ and $k_2 := n - \sqrt{n^2-2n-1}$.
Since $n \geqslant 3$, $n^2-2n-1 > (n-2)^2$. Hence $k_1 < 2$ and $k_2 > 2n-2$.
So, for $k_1 \leqslant k \leqslant k_2$ or $2 \leqslant k \leqslant 2n-2$:

$(k+1)(2n-k+1) \leqslant 2k(2n-k)$ □

From this lemma and some special checking of the cases k=1 and k=2n-1 we learn that the situation is as in the picture below.



Of course the functions are not continuous; the picture is only meant to give a rough idea of the situation.
We are interested in the number

$$\max_k \min \left\{ k\binom{2n-k}{\lfloor\frac{1}{2}(2n-k)\rfloor} , \text{lcm} \{1,2,..,k\} \right\}$$

since this is an upper bound for the number of codewords $\pi^i \underline{x}_0$ that can be obtained using any $\pi \in S_{2n}$. We call this number M_n; it is also indicated on the vertical axis of the figure.
We are now ready to state our theorem.
Theorem 2.1.1.1 : the fraction of usable codewords in any code of the type described in § 2.1.0 is upper bounded by $M_n / \binom{2n}{n}$, and

$$\lim_{n \to \infty} \frac{M_n}{\binom{2n}{n}} = 0.$$

Proof: we already know that the expression is an upper bound for the fraction of words. To prove the rest, we shall show that there exists an $\alpha \in (0,1)$ such that:

(i) for $0 < k \leq \alpha n$, $\lim_{n \to \infty} \dfrac{\mathrm{lcm}\,\{1,2,..,k\}}{\binom{2n}{n}} = 0$

(ii) for $\alpha n \leq k \leq 2n$, $\lim_{n \to \infty} \dfrac{k\binom{2n-k}{\lfloor \frac{1}{2}(2n-k) \rfloor}}{\binom{2n}{n}} = 0$

That it suffices to show this, is a consequence of lemma 2.1.1.0. Also, see the picture; note that the point $k = \alpha n$ does not necessarily coincide with the value of k where M_n is reached.

Since (ii) is the easiest part to prove, we shall start with this.

(ii) Let $\alpha \in (0,1)$ (an appropriate value of α will be fixed later), and $k \geq \alpha n$.

$$\frac{k\binom{2n-k}{\lfloor \frac{1}{2}(2n-k) \rfloor}}{\binom{2n}{n}} \leq \frac{k}{\binom{k}{\lfloor \frac{1}{2}k \rfloor}} \leq \frac{k}{2^{\lfloor \frac{1}{2}k \rfloor}} = O\left(\frac{k}{(\sqrt{2})^k}\right), \ (k \to \infty).$$

The first inequality is based on $\binom{2n}{n} \geq \binom{2n-k}{\lfloor \frac{1}{2}(2n-k) \rfloor}\binom{k}{\lfloor \frac{1}{2}k \rfloor}$; this is a special case of a wellknown combinatorial fact: choosing n out of 2n can be done by choosing $\lfloor \frac{1}{2}k \rfloor$ out of the first k, and $\lceil \frac{1}{2}(2n-k) \rceil$ out of the remaining 2n-k.

The second inequality is in fact the same as $\binom{2n}{n} \geq 2^n$, which is trivial if we write

$$\binom{2n}{n} = \begin{pmatrix}2n\\ n\end{pmatrix}\begin{pmatrix}2n-1\\ n-1\end{pmatrix} \frac{\cdots}{\cdots} \begin{pmatrix}n+1\\ 1\end{pmatrix}$$

Since $k \geq \alpha n$, the above inequality implies

$$\lim_{n \to \infty} \frac{k\binom{2n-k}{\lfloor \frac{1}{2}(2n-k) \rfloor}}{\binom{2n}{n}} = \lim_{k \to \infty} \frac{k}{(\sqrt{2})^k} = 0.$$

(i) We need some number theory (cf. [2]); we define for $x \in \mathbb{R}$:

$\pi(x)$:= the number of prime numbers less than or equal to x.

A wellknown theorem states that there are constants C_1 and C_2 such that, for $x \geq 2$,

$$C_1 \leq \frac{\pi(x)}{x} \log_2(x) \leq C_2 \qquad (*)$$

In fact, we only need the second inequality; $C_2 \approx 12 \log_2(e) \approx 17.3$.

lcm $\{1,2,..,k\}$ (for any k) is the product of prime powers: if p is a prime, we get $p^i \mid \text{lcm}\{1,2,..,k\}$ and $p^{i+1} \nmid \text{lcm}\{1,2,..,k\}$ iff p^i is the largest power of p occurring in the prime decompositions of 1,2 through k. Hence

if $p^i \leqslant k$ and $p^{i+1} > k$ we find $i \leqslant \log_p k$, $i+1 > \log_p k$ or $i = \lfloor \log_p k \rfloor$.

This yields

$$\text{lcm}\{1,2,..,k\} = \prod_{\substack{p \leqslant k \\ p \text{ prime}}} p^{\lfloor \log_p k \rfloor} \leqslant \prod_{\substack{p \leqslant k \\ p \text{ prime}}} k = k^{\pi(k)}.$$

Let $\alpha \in (0,1)$ and $k \leqslant \alpha n$. Then

$\text{lcm}\{1,2,..,k\} \leqslant \text{lcm}\{1,2,..\alpha n\} \leqslant \alpha n^{\pi(\alpha n)}$.

Define $\ell := \lceil \log_2(\alpha n) \rceil$; $\alpha n \leqslant 2^\ell$, $\alpha n > 2^{\ell-1}$. Then

$$\frac{\text{lcm}\{1,2,..,k\}}{\binom{2n}{n}} \leqslant \frac{\alpha n^{\pi(\alpha n)}}{\binom{2n}{n}} \leqslant \frac{2^{\ell\pi(\alpha n)}}{2^n} = 2^{\ell\pi(\alpha n)-n}$$

Let $n \geqslant 2$,then with (*):

$$\ell\pi(\alpha n)-n \leqslant C_2 \ell \cdot \frac{\alpha n}{\log_2(\alpha n)} - n = C_2 \lceil \log_2(\alpha n) \rceil \frac{\alpha n}{\log_2(\alpha n)} - n \leqslant$$

$$\leqslant \left\{ \frac{C_2(\log_2(\alpha n) + 1)\alpha}{\log_2(\alpha n)} \right\} n - n = \left\{ C_2\alpha + \frac{C_2\alpha}{\log_2(\alpha n)} - 1 \right\} n$$

Next we choose an appropriate value for α : $\alpha_0 := \frac{1}{2C_2}$ then $\alpha_0 \in (0,1)$.

Let $n > \max(2, 4C_2)$, then $\log_2(\alpha_0 n) \geqslant \log_2 2 + \delta = 1 + \delta$ for some $\delta > 0$. This yields

$$\ell\pi(\alpha_0 n) - n \leqslant \left\{ \tfrac{1}{2} + \frac{\tfrac{1}{2}}{\log_2(\alpha_0 n)} - 1 \right\} n \leqslant \left\{ \frac{\tfrac{1}{2}}{1+\delta} - \tfrac{1}{2} \right\} n \leqslant$$

$$\leqslant -\varepsilon n, \quad \text{for some } \varepsilon > 0.$$

Hence $\dfrac{\text{lcm}\{1,2,..,k\}}{\binom{2n}{n}} \leqslant 2^{\ell\pi(\alpha_0 n)-n} \leqslant 2^{-\varepsilon n}$

So $\lim\limits_{n \to \infty} \dfrac{\text{lcm}\{1,2,..,k\}}{\binom{2n}{n}} = 0$. $\square$

In the appendix we list:

$\min\left\{\operatorname{lcm}\{1,2,..,k\}\ ,\ k\binom{2n-k}{\lfloor\frac{1}{2}(2n-k)\rfloor}\right\}$ for $1\leq k\leq 2n$, M_n and $\dfrac{M_n}{\binom{2n}{n}}$ for some values of n: n= 4, 5, 10, 20.

From these tables we can already see that $M_n \,/\, \binom{2n}{n} \to 0$.

In the appendix we also list the fractions

$\dfrac{\log_2 M_n}{\log_2\binom{2n}{n}}$; we see that $\dfrac{\log_2 M_n}{\log_2\binom{2n}{n}} \approx 0.64$ for large n.

After the next section we shall be able to show that there exists a code that satisfies

$\dfrac{\log_2 (\#\text{codewords})}{\log_2 \binom{2n}{n}} \geq 0.30$ for reasonable values of n $(2n < 100)$.

As mentioned in § 2.1.0, the exact value of $\lim\limits_{n\to\infty}\dfrac{\log_2 (\text{max. \# codewords})}{\log_2 \binom{2n}{n}}$ could not be determined.

2.1.2

Suppose that we do want to use a code as described in § 2.1.0, notwithstanding the negative results of the preceding paragraph. In that case we use a permutation $\pi\in S_{2n}$ that yields as many words as possible. This can be done by taking the cycle lengths of π relatively prime (remember that the order of π is the least common multiple of all cycle lengths).

W.l.o.g., let one of the cycles of π be $(1\ 2\ \dots k{-}1\ k)$. We have $\underline{x}_i = \pi^i\,\underline{x}_0$ for all i, where $\underline{x}_0$ is chosen s.t. the first k positions are 00..01..1: $\lfloor\frac{1}{2}k\rfloor$ zeros and $\lceil\frac{1}{2}k\rceil$ ones. The other positions of $\underline{x}_0$ are also chosen according to the cycles of π: for any cycle of length ℓ, we put zeros in the first $\lfloor\ell/2\rfloor$ positions of $\underline{x}$ (corresponding to the numbers in the cycle), and ones in the other positions. We must take care that there are n zeros and n ones, so we must sometimes take $\lceil\ell/2\rceil$ zeros instead of $\lfloor\ell/2\rfloor$.

Now suppose we want to add $\underline{x}_i$ and $\underline{x}_j$. Suppose the first k positions of $\underline{x}_i$ resp. $\underline{x}_j$ contain the m_ith resp. m_jth permutation of symbols, compared with 00..01..1. (The m^{th} permutation means: a cyclic shift to the left over m positions.)

Then we know that $i\equiv m_i \pmod k$, $j\equiv m_j \pmod k$. So the sum i+j satisfies $i+j\equiv m_i + m_j \pmod k$.

The only thing we have to do is to determine m_j , which is easy, and shift the first k positions of $\underline{x}_i$ over m_j positions to the left.
After that, these k positions contain the $(m_i + m_j)$th permutation of 00..01..1. This means that these k positions now contain the right configuration for $\underline{x}_{i+j}$. Do the same for all other cycles.
This algorithm is O(n). We can also do this for multiplication: in the same notation, if we know m_i and m_j and $i \equiv m_i \pmod k$, $j \equiv m_j \pmod k$ then $ij \equiv m_i m_j \pmod k$. So we must shift 00..01..1 over $m_i m_j$ positions to the left to find the first k symbols of $\underline{x}_{ij}$.

Note that conversion of i to $\underline{x}_i$ is not difficult (determine $m_i =$ $= i \,\underline{\text{mod}}\, k$, shift over m_i positions), but $\underline{x}_i \to i$ is more complicated.

<u>2.1.3</u>

Using the code of § 2.1.2, we want to calculate

$$\frac{\log_2(\#\text{codewords})}{\log_2 \binom{2n}{n}} \quad .$$

For this code we choose the cycle lengths relatively prime; we shall take prime numbers for these lengths. Let p_i denote the i^{th} prime number ($p_1 = 2$), and let $\pi^{(k)}$ be a permutation with k disjunct cycles of lengths $p_1, p_2, ..p_k$ respectively, such that $\pi^{(k)}$ is a permutation of $\sum_{i=1}^{k} p_i$ positions.

So, w.l.o.g. $\pi^{(4)} = \underbrace{(1\ 2)}_{2}\underbrace{(3\ 4\ 5)}_{3}\underbrace{(6\ 7\ 8\ 9\ 10)}_{5}\underbrace{(11\ 12\ 13\ 14\ 15\ 16\ 17)}_{7}$

If $\sum_{i=1}^{k} p_i$ is even, $\pi^{(k)}$ corresponds to a code as in §2.1.2 with $n(k) := \frac{1}{2} \left(\sum_{i=1}^{k} p_i \right)$.

If the sum is odd, we can add a cycle of length 1 to obtain a code; in that case:

$$n(k) := \tfrac{1}{2}\left(\sum_{i=1}^{k} p_i + 1\right).$$

In either case we have a code of length $2 \cdot n(k)$ with n(k) ones and zeros.
The code contains $\prod_{i=1}^{k} p_i$ words (= the order of $\pi^{(k)}$).

Now $\log_2(\#\text{codewords}) / \log_2 \binom{2n(k)}{n(k)} = \log_2 \prod_{i=1}^{k} p_i \,/\, \log_2 \binom{2n(k)}{n(k)}$

Since $\binom{2n(k)}{n(k)} \leqslant 2^{2n(k)}$, we have

$$\frac{\log_2 \prod_{i=1}^{k} p_i}{\log_2 \binom{2n(k)}{n(k)}} \geqslant \frac{\log_2 \prod_{i=1}^{k} p_i}{2n(k)} \quad .$$

In the appendix we list the value of $\dfrac{\log_2 \prod_{i=1}^{k} p_i}{2n(k)}$ for $1 \leqslant k \leqslant 12$.

We find that $\dfrac{\log_2 (\#\text{codewords})}{\log_2 \binom{2n(k)}{n(k)}} \geqslant 0.30$ for $2n(k) < 100$.

2.2 Examples of positional systems

2.2.0

In the proof of theorem 2.1.1.1 we have already encountered the inequality $\binom{2n}{n} > 2^n$. We give an easy way to code 2^n numbers in an n-out-of-2n- code. Let $0 \leqslant i < 2^n$.
To code i, we write i in the binary system. This representation of i has at most n ones, and a length at most n. We always make the length equal to n, by adding leading zeros. If this 'word' of length n does not have n ones, we must put the remaining ones to the right of this 'word', followed by remaining zeros, if any.
Example: n=4, i=5: 0101|1100.

If we want to add words in this code, we only consider the first n positions. These first halves of the words can be added using binary addition. Afterwards we complete the word with remaining ones and zeros on the right. This is a very trivial algorithm.
However, from the next lemma we see that $\lim_{n \to \infty} \dfrac{2^n}{\binom{2n}{n}} = 0$.

Lemma 2.2.0.0 : if $n \geqslant 4$, $\binom{2n}{n} > 2^{\lfloor 3n/2 \rfloor}$.

Proof: let n be even, $n = 2m$, $m \geqslant 2$.

$$\binom{2n}{n} = \frac{(2n)!}{n!\; n!} = \frac{(4m)(4m-1)\dots(2m+1)}{(2m)(2m-1)\dots 2\cdot 1} =$$

$$= \frac{(4m)(4m-2)\dots(2m+2)}{(2m)(2m-1)\dots(m+1)} * \frac{(4m-1)(4m-3)\dots(2m+1)}{m(m-1)\dots 2\cdot 1} =$$

$$= 2^m \prod_{i=0}^{m-1} \frac{(4m-2i-1)}{(m-i)} = 2^m \cdot \frac{(4m-1)}{m} \cdot (2m+1) \prod_{i=1}^{m-2} \frac{(4m-2i-1)}{(m-i)}$$

For $m \geqslant 2$, we find $8m^2 - 14m - 1 > 0$ (zeros are $m= \frac{14 \pm \sqrt{228}}{16}$)
Hence

$$\frac{(4m-1)}{m} * (2m+1) = \frac{8m^2 + 2m - 1}{m} > 16.$$

For $i \geqslant 1$: $4m - 2i - 1 > 4(m - i)$ so $\frac{4m - 2i - 1}{m -- i} > 4$

$$\therefore \binom{2n}{n} = 2^m \frac{(4m-1)}{m} (2m+1) \prod_{i=1}^{m-2} \frac{(4m-2i-1)}{m-i}$$

$$2^m.16.4^{(m-2)} = 2^{3m} = 2^{3n/2} .$$

Let n be odd, $n= 2m+1$, then analogously

$$\binom{2n}{n} = \frac{(4m+2)(4m)\ldots(2m+2)}{(2m+1)(2m)\ldots(m+1)} \cdot \frac{(4m+1)(4m-1)\ldots(2m+3)}{m\ (m-1)\ \ldots\ 2.1} =$$

$$= 2^{m+1} \prod_{i=0}^{m-1} \frac{(4m-2i+1)}{m-i} \geqslant 2^{m+1}.4^m = 2^{3m+1} = 2^{(3n-1)/2}$$

where the inequality follows from $4m - 2i + 1 > 4(m - i)$, for all $i \geqslant 0$. □

As for $\lim_{n \to \infty} \frac{\log_2 2^n}{\log_2 \binom{2n}{n}}$, since $\binom{2n}{n} \leqslant 2^{2n}$, we know

$$\frac{\log_2 2^n}{\log_2 \binom{2n}{n}} \geqslant \frac{n}{2n} = \frac{1}{2} \text{ for all n.}$$

As a matter of fact the limit of this quotient for $n \to \infty$ is $\frac{1}{2}$, which will follow from theorem 2.2.1.0.

2.2.1

With the preceding lemma, we can generate $2^{\lfloor 3n/2 \rfloor}$ words, and in fact even more. Let n be even. Partition the words into k blocks of length $2n/k$, from left to right. Now we require that every block has n/k zeros and n/k ones. Here we suppose $2k \mid n$.

The number of words that can be generated in this way is $\binom{2n/k}{n/k}^k$ ($\binom{2n/k}{n/k}$ possibilities per block).

If $n/k \geqslant 4$ we can apply lemma 2.2.0.0 :

$$\binom{2n/k}{n/k}^k \geq 2^{\lfloor (3n)/2k \rfloor k} = 2^{3n/2} .$$

Indeed, we have more than $2^{3n/2}$ words with a very nice property: they are written in a positional system with base $\binom{2n/k}{n/k}$. In the positional system, each position corresponds to a block of length $2n/k$.

The arithmetic in such a system is just as in the binary (or decimal) system: addition and multiplication can be performed in a straightforward manner. We shall not give the algorithms for this; we only remark that they are $O(n)$.

It is clear that arithmetic in a system with base b is easiest when b is as small as possible. Since we have base $\binom{2n/k}{n/k}$ with the restriction $n/k \geq 4$, $2k | n$, we must take $n/k = 4$, so $n = 4k$.

Thus we find a system with base $\binom{8}{4} = 70$. With this system we can represent 70^k numbers, which is more than the promised $2^{\lfloor 3n/2 \rfloor} = 2^{6k} = 64^k$.

Regrettably, this is still not enough: $\lim_{k \to \infty} \frac{70^k}{\binom{8k}{4k}} = 0$, as can be concluded from the following theorem. From this theorem we can also conclude

$$\lim_{k \to \infty} \frac{\log_2 (70^k)}{\log_2 \binom{8k}{4k}} = \frac{\log_2 (70)}{8} \approx 0.766$$, but we come back to this later.

<u>Theorem 2.2.1.0</u> :

$$\forall a \in (0,4) \quad \frac{1}{\binom{2n}{n}} = O(a^{-n}) \qquad ,(n \to \infty).$$

<u>Proof</u>: let $0 < a < 4$ and $N \in \mathbb{N}$ s.t. $\frac{2n+1}{n+1} \geq \frac{1}{2}a$ for all $n \geq N$.

Then for all $n \geq N$

$$\binom{2n+2}{n+1} = \frac{(2n+2)(2n+1)}{(n+1)(n+1)} \binom{2n}{n} \geq a \binom{2n}{n} \geq \dots \geq a^{n-N+1} \binom{2N}{N} .$$

Let $C := a^{-N}\binom{2N}{N}$, then for $n \geq N$

$$\frac{1}{\binom{2n}{n}} \leq \frac{1}{Ca^n}$$

<u>Corollary</u>: $\frac{70^k}{\binom{8k}{4k}} = O\left(\left(\frac{70}{a^4}\right)^k \right) \quad ,(k \to \infty)$ for $0 < a^4 < 256$.

If we write theorem 2.2.1.0 in a different way:

$$\forall a \in (0,4) \ \exists C \in \mathbb{R} \ \exists N \in \mathbb{N} \ \forall n > N \quad \binom{2n}{n} \geq C.a^n$$

Taking logarithms and using $\log_2 \binom{2n}{n} \leq 2n$:

$$\forall a<2 \ \exists C\in\mathbb{R} \ \exists N\in\mathbb{N} \ \forall n>N \quad an + C \leq \log_2 \binom{2n}{n} \leq 2n$$

Therefore

$$\forall a<8 \ \exists C\in\mathbb{R} \ \exists N\in\mathbb{N} \ \forall k>N \quad \frac{\log_2 70}{8} \leq \frac{\log_2 70^k}{\log_2 \binom{8k}{4k}} \leq \frac{k.\log_2 70}{C + ak}$$

Applying the domination principle, we find for all $a<8$

$$\frac{\log_2 70}{8} \leq \lim_{k\to\infty} \frac{\log_2 70^k}{\log_2 \binom{8k}{4k}} \leq \frac{\log_2 70}{a}$$

thus $\displaystyle\lim_{k\to\infty} \frac{\log_2 70^k}{\log_2 \binom{8k}{4k}} = \frac{\log_2 70}{8}$ (≈ 0.766) .

Note that the base of this number system does not depend on n. In the next section we shall construct a code that also uses a positional system, but the base will depend on n. The idea behind this is, that we can represent more words if the base of the system is larger: if the blocks with equally many zeros and ones are larger, the constraints on the words are weaker so we allow more words. However, addition and multiplication will be harder in a system with larger base.

2.2.2

In this section 2n will be a square: $n = 2.s^2$, $2n = (2s)^2$ for some $s\in\mathbb{N}$. We think of a word as a 2s by 2s grid with $(2s)^2$ little squares. $2s^2$ of these squares contain a '0' and $2s^2$ contain a '1'. Thus, the word can be found reading the successive rows from left to right.

We make the restriction that the words (or grids) must have s zeros and s ones per row.

We give an example of this for s=3 (so n=18): we only write the '1's; the blank squares are to be filled with '0's. This grid corresponds to the word

$\underline{101100}\,\underline{110100}\,\underline{010110}\,\underline{100101}\,\underline{110010}\,\underline{001011}$.

1		1	1		
1	1		1		
	1		1	1	
1			1		1
1	1			1	
		1		1	1

Per row we have $\binom{2s}{s}$ possibilities, in total $\binom{2s}{s}^{2s}$. So we have constructed $\binom{2s}{s}^{s}$ words in a number system with base $\binom{2s}{s}$.
Since the lexicographical number of a row can be determined in O(s), addition and multiplication algorithms are also O(s). O(s) = O(n).
Again we shall investigate the asymptotic behaviour of

$$\frac{\binom{2s}{s}^{2s}}{\binom{4s^2}{2s^2}} \quad \text{and} \quad \frac{\log_2 \binom{2s}{s}^{2s}}{\log_2 \binom{4s^2}{2s^2}} .$$

Theorem 2.2.2.0 :

Let $f(s) := \binom{2s}{s}^{2s} \Big/ \binom{4s^2}{2s^2}$. Then $f(s) = O(4^{-s})$, $(s \to \infty)$.

Proof: we want to compare f(s+1) to f(s); in fact, we want to show that $f(s+1) \leq \frac{1}{4} f(s)$ since this yields $f(s) = O(\frac{1}{4}^s)$, $s \to \infty$.

For the numerator of f(s):

$$\binom{2s+2}{s+1}^{2s+2} = \left\{\frac{(2s+2)(2s+1)}{(s+1)(s+1)}\right\}^{2s+2} \binom{2s}{s}^{2s+2}$$
$$= \left\{2\cdot\frac{(2s+1)}{(s+1)}\right\}^{2s+2} \binom{2s}{s}^{2} \binom{2s}{s}^{2s}$$

For the denominator:

$$\binom{4s^2+8s+4}{2s^2+4s+2} = \frac{(4s^2+8s+4)\ldots(4s^2+1)}{(2s^2+4s+2)^2\ldots(2s^2+1)^2}\binom{4s^2}{2s^2}$$
$$= \left(\prod_{i=1}^{4s+2}\frac{(4s^2+2i-1)(4s^2+2i)}{(2s^2+i)(2s^2+i)}\right)\binom{4s^2}{2s^2}$$

Hence

$$f(s+1) = 2^{2s+2}\left\{\frac{2s+1}{s+1}\right\}^{2s+2}\binom{2s}{s}^{2}\prod_{i=1}^{4s+2}\frac{(2s^2+i)}{2(4s^2+2i-1)} * f(s)$$

$$= 2^{-2s}\left\{\frac{2s+1}{s+1}\right\}^{2s+2}\binom{2s}{s}^{2}\prod_{i=1}^{4s+2}\frac{(2s^2+i)}{(4s^2+2i-1)} * f(s).$$

For all i we have $\dfrac{2s^2+i}{4s^2+2i-1} \leq \dfrac{2s^2}{4s^2-1}$ $\quad (i \geq 1)$

Hence

$$f(s+1) \leq 2^{-2s}\left(\frac{2s+1}{s+1}\right)^{2s+2}\binom{2s}{s}^2 2^{4s+2}\left(\frac{s^2}{4s^2-1}\right)^{4s+2} * f(s)$$

$$= 2^{2s+2}\binom{2s}{s}^2 \frac{s^4}{(s+1)^2(2s-1)^2}\left\{\frac{s^4}{(2s+1)(2s-1)^2(s+1)}\right\}^{2s} * f(s)$$

It can be readily checked that $\frac{s^4}{(s+1)^2(2s-1)^2} \leq \frac{1}{4}$ for $s \geq 1$,

and $\frac{s^4}{(2s+1)(2s-1)^2(s+1)} \leq \frac{1}{8}$ for $s \geq 2$.

Therefore, if $s \geq 2$: $f(s+1) \leq 2^{-4s}\binom{2s}{s}^2 f(s)$.

Finally we show that $\binom{2s}{s}2^{-2s} \leq \frac{1}{2}$ (so $2^{-4s}\binom{2s}{s}^2 \leq \frac{1}{4}$):

$$2^{2s} = \sum_{i=0}^{2s}\binom{2s}{i} \geq \binom{2s}{s-1} + \binom{2s}{s} + \binom{2s}{s+1} = 2 \cdot \binom{2s}{s-1} + \binom{2s}{s} =$$

$$= \frac{2s}{s+1}\binom{2s}{s} + \binom{2s}{s} = \frac{(3s+1)}{s+1}\binom{2s}{s}$$

So $\binom{2s}{s} 2^{-2s} \leq \frac{s+1}{3s+1} \leq \frac{1}{2}$ for $s \geq 1$. □

<u>Theorem 2.2.2.1</u> : $\lim_{s\to\infty} \frac{\log_2 \binom{2s}{s}^{2s}}{\log_2\binom{4s^2}{2s^2}} = 1.$

<u>Proof</u>: this can be found in the same way as in § 2.2.1. We have:

$\forall a < 2 \; \exists C,N \; \forall s > N \quad as + C \leq \log_2\binom{2s}{s} \leq 2s$ and similarly

$\forall a' < 2 \; \exists C',N' \; \forall s > N' \quad a'2s^2 + C' \leq \log_2\binom{4s^2}{2s^2} \leq 4s^2$. So:

$$\forall a,a' < 2 \; \exists C,C',N \; \forall s > N \quad \frac{2s(as+C)}{4s^2} \leq \frac{2s\log_2\binom{2s}{s}}{\log_2\binom{4s^2}{2s^2}} \leq \frac{(2s)(2s)}{a'2s^2 + C'}$$

Again this yields $\forall a,a' < 2 \quad \frac{2a}{4} \leq \lim_{s\to\infty}\frac{\log_2\binom{2s}{s}^{2s}}{\log_2\binom{4s^2}{2s^2}} \leq \frac{4}{2a'}$

So $$\lim_{s\to\infty} \frac{\log_2 \binom{2s}{s}^{2s}}{\log_2 \binom{4s^2}{2s^2}} = 1. \qquad \square$$

We could consider to extend this code: we could also allow words that have exactly s zeros and s ones per column. Notice that some of these words are already in the code, so the number of words would not even double if we added these words. For the asymptotic behaviour this would not make any difference. Therefore it is not a wise thing to do, since we would loose the nice properties of a positional system.

To conclude this section, we come back on the remark made in the previous section about the number of words increasing with the base of the number system (or the block length).
Of course, if we would take block length 2n, we would have base $\binom{2n}{n}$.
In other words, we would allow all sequences with n zeros and n ones. We already know from chapter 1 that addition is very complex in this case. The next largest block length is n; this means a base $\binom{n}{n/2}$ (let n=2k) and we would have only two positions in a word. Addition is still complex, and it would not be very practical to do this. However, to illustrate the remark in § 2.2.1 we shall investigate the fractions

$$\binom{2k}{k}^2 \Big/ \binom{4k}{2k} \quad \text{and} \quad \log_2 \binom{2k}{k}^2 \Big/ \log_2 \binom{4k}{2k} \quad \text{for } k \to \infty .$$

<u>Theorem 2.2.2.2</u> : $$\frac{\binom{2k}{k}^2}{\binom{4k}{2k}} \geq \frac{1}{1+2k} \text{ , or } \frac{\binom{4k}{2k}}{\binom{2k}{k}^2} = O(k), \ (k \to \infty)$$

<u>Proof</u>: from § 1.1.1 we can see that

$$\binom{4k}{2k} = \sum_{i=0}^{2k} \binom{2k}{i}^2 = \binom{2k}{k}^2 + 2 \cdot \sum_{i=0}^{k-1} \binom{2k}{i}^2$$

$$(1 + 2k) \binom{2k}{k}^2 \qquad \square$$

<u>Theorem 2.2.2.3</u> : $$\lim_{k\to\infty} \frac{\log_2 \binom{2k}{k}^2}{\log_2 \binom{4k}{2k}} = 1$$

<u>Proof</u>: analogous to theorem 2.2.2.1 . $\square$

2.3 An iterative approach

2.3.0

In this section we shall introduce a code with particularly nice algorithms for conversion, addition and multiplication. The codewords will be constructed in an iterative way.

Let $n \geqslant 1$. Let V_n be a set of words of length 2n with n ones and n zeros. V_n is partitioned into the sets A_n and B_n, where A_n contains all words ending on '0', and B_n contains the words ending on '1'.

We need one more notation for concatenation of sequences: if $\underline{x}$ is a sequence of bits, the $\underline{x}$ + '0' denotes the sequence we get if we attach a '0' to the right end of $\underline{x}$. We define $\underline{x}$ + '0100' etc. in the same way.

We construct V_{n+1} from V_n as follows:

1) for all $\underline{x}$ + '0' in A_n, $\underline{x}$ + '010', $\underline{x}$ + '100' and $\underline{x}$ + '001' are in V_{n+1}
2) for all $\underline{x}$ + '1' in B_n, $\underline{x}$ + '110', $\underline{x}$ + '101' and $\underline{x}$ + '011' are in V_{n+1}
3) V_{n+1} contains no other words than the ones constructed with the rules 1) and 2).

It is easy to check that all these words in V_{n+1} are different, provided that all words in V_n are different.

Starting with $V_1 := \{01, 10\}$, we can prove by induction that $|A_n| = |B_n| = 3^{n-1}$, $|V_n| = 2.3^{n-1}$ for $n \geqslant 1$.

We already know that $\dfrac{2.3^{n-1}}{\binom{2n}{n}} = O\left(\left(\frac{3}{a}\right)^n\right)$, $(n \to \infty)$ for $0 < a < 4$,

and $$\lim_{n \to \infty} \frac{\log_2 2.3^{n-1}}{\log_2 \binom{2n}{n}} = \frac{\log_2(3)}{2} \approx 0.792$$ (compare th. 2.2.2.1)

We order the words iteratively. Suppose we have an ordering in V_n such that the words in A_n represent $0,1,..,3^{n-1}-1$ and B_n represents $3^{n-1},...,2.3^{n-1}-1$.

Let $N(\underline{x})$ denote the number represented by $\underline{x}$, then we define the ordering in V_{n+1} according to the list below:

for $\underline{x}$ + '0' $\in A_n$: $N(\underline{x}$ + '100'$) := N(\underline{x}$ + '0'$)$

for $\underline{x}$ + '0' $\in A_n$: $N(\underline{x}$ + '010'$) := N(\underline{x}) + 3^{n-1}$

for $\underline{x}$ + '1' $\in B_n$: $N(\underline{x}$ + '110'$) := N(\underline{x}) + 3^{n-1}$

for $\underline{x}$ + '0' $\in A_n$: $N(\underline{x}$ + '001'$) := N(\underline{x}) + 3^n$

for $\underline{x}$ + '1' $\in B_n$: $N(\underline{x}$ + '101'$) := N(\underline{x}) + 3^n$

for $\underline{x}$ + '1' $\in B_n$: $N(\underline{x}$ + '011'$) := N(\underline{x}$ + '1'$) + 3^n + 3^{n-1}$

Example for n=2:

A_3	A_2	1101 00	0	= 000
		1011 00	1	= 001
		0111 00	2	= 002
	A_2	1100 10	3	= 010
		1010 10	4	= 011
		0110 10	5	= 012
	B_2	1001 10	6	= 020
		0101 10	7	= 021
		0011 10	8	= 022

B_3	A_2	1100 01	9	= 100
		1010 01	10	= 101
		0110 01	11	= 102
	B_2	1001 01	12	= 110
		0101 01	13	= 111
		0011 01	14	= 112
	B_2	1000 11	15	= 120
		0100 11	16	= 121
		0010 11	17	= 122

In this example, the first column contains the codeword; the second column contains the number $N(\underline{x})$ corresponding to that word, $\underline{x}$; in the third column this number is written in the ternary system.
The next two algorithms take care of the conversion from a word to its ternary number and vice versa. It turns out that the ternary number can be obtained almost directly from the structure of the word. We do not need the 'real' number; for addition and multiplication, it suffices to have the number represented in the ternary system. We shall not give the algorithms for these well known arithmetic operations.

<u>A</u>) Convert a word, stored in array $w(i\colon 0 \leq i < 2n)$, to its ternary number; the positions of this number will be stored in an integer array $t(i\colon 0 \leq i < n)$ in such a way that the number is $\sum_{i=0}^{n-1} t(i).3^{n-i-1}$.

```
t(0):= w(2n-1); m:= 2n-1; j:=1;
do m > 1 → if w(m-2) = 1 ∧ w(m-1) = 0 ∧ w(m) = 0 → t(j):=0; w(m-2):=0
             ▯    "    0    "    1    "    0 → t(j):= 1
             ▯    "    1    "    1    "    0 → t(j):= 2
             ▯    "    0    "    0    "    1 → t(j):= 0
             ▯    "    1    "    0    "    1 → t(j):= 1
             ▯    "    0    "    1    "    1 → t(j):=2; w(m-2):=1
           fi;  j:= j + 1; m:= m - 2
od
```

Explanation: the last digit of the word, $w(2n-1)$, determines whether the word is in A_n or B_n; if $w(2n-1) = 0$, the word is in A_n so its number is smaller than 3^{n-1}, so $t(0)$ should be 0. If $w(2n-1)$ we find $t(0) = 1$. Furthermore, the last three digits of the word determine the position of the word in one of the six parts of the list on page 34/35. From the same list -and the example below it- we can derive $t(1)$.
Then we make use of the iterative structure of the ordering: we omit the last two symbols of the word, thus finding the word in V_{n-1} it was constructed from. We must be careful if the word in V_n belongs to the 1st or 6th part of the list: to find the word in V_{n-1} we must also change the third bit on the right.
We can go on omitting symbols until there are only two symbols left, $w(0)$ and $w(1)$: they are '01' or '10'. From an example with n=1 or n=2 it follows that these have already been taken care of in $t(n-1)$.

<u>B</u>) A ternary number is stored in $t(i\colon 0 \leq i < n)$; we want to find the corresponding word and store it in $w(i\colon 0 \leq i < 2n)$. This algorithm works in exactly the same way as the one under <u>A</u>; we just need two more variables, satisfying $k = \#$ ones in $w(i\colon m \leq i < 2n)$; w corresponds to the last bit of the word in $V_{(m+1)/2}$. This is not always the same as $w(m)$, since it is changed if this word is in the 1st or 6th part of the list (see comment under <u>A</u>).

```
w(2n-1):= t(0); k:= t(0); m:= 2n-1; w:= w(m); j:= 1;
do m > 1 → if t(j) = 0 ∧ w = 0 → w(m-1):= 0; w(m-2):= 1; w:= 0; k:= k+1
           ▯   "   1   "  0 →    "       1    "       0    " 0; k:= k+1
           ▯   "   2   "  0 →    "       1    "       1    " 1; k:= k+2
           ▯   "   0   "  1 →    "       0    "       0    " 0
           ▯   "   1   "  1 →    "       0    "       1    " 1; k:= k+1
           ▯   "   2   "  1 →    "       1    "       0    " 1; k:= k+1
           fi; m:= m - 2; j:= j + 1
od; w(0):= n - k
```

After termination of the do-loop $W(i\colon 1 \leq i < 2n)$ is filled; $w(0)$ follows from the fact that $w(i\colon 0 \leq i < 2n)$ has n zeros and n ones.

Both algorithms are $O(n)$. Adding two ternary numbers can also be done in $O(n)$ computing time, so the total algorithm for addition is $O(n)$.

Some final remarks on this way of coding:

- note that the ordering of the words in the list is quite arbitrary. If we had put the six parts of the list in another order, we would also have found conversion algorithms of the same type as A and B. The ordering we have chosen has the property that the words are ordered lexicographically, as can be seen from the list.

- the first position of the ternary number of a word -t(0) in the program- can never be 2. This may serve as an easy checking method for overflow.

2.3.1

We generalize the method of the preceding section. The code of the preceding section does not contain words ending on '000' or '111'. We do get these words if we generalize as follows.

Again, let V_n be a set of words of length 2n with n ones. V_n is partitioned into $A_n \cup B_n \cup C_n \cup D_n$:
A_n contains the words ending on '00'; B_n, C_n and D_n contain words ending on '01', '10', and '11' respectively.
Construction of V_{n+2}:

1) for all $\underline{x}$ + '00' in A_n, $\underline{x}$ + '001100', $\underline{x}$ + '001010', $\underline{x}$ + '001001', $\underline{x}$ + '000011', $\underline{x}$ + '000101', $\underline{x}$ + '000110' $\underline{x}$ + '110000', $\underline{x}$ + '$01\left\{\begin{matrix}0001\\0010\\0100\\1000\end{matrix}\right.$', and $\underline{x}$ + '$10\left\{\begin{matrix}0001\\0010\\0100\\1000\end{matrix}\right.$' are in V_{n+2}.

2) for all $\underline{x}$ + '11' in D_n, $\underline{x}$ + '$11\left\{\begin{matrix}1100\\1010\\1001\\0011\\0101\\0110\end{matrix}\right.$', $\underline{x}$ + '001111', $\underline{x}$ + '$01\left\{\begin{matrix}1110\\1101\\1011\\0111\end{matrix}\right.$', and $\underline{x}$ + '$10\left\{\begin{matrix}1110\\1101\\1011\\0111\end{matrix}\right.$' are in V_{n+2}.

3) for all $\underline{x}$ + '01' in B_n, $\underline{x}$ + '$01\left\{\begin{matrix}1100\\1010\\1001\\0011\\0101\\0110\end{matrix}\right.$', $\underline{x}$ + '$00\left\{\begin{matrix}1110\\1101\\1011\\0111\end{matrix}\right.$' are in V_{n+2}.

4) for all $\underline{x}$ + '10' in C_n, $\underline{x}$ + '$11\left\{\begin{matrix}1100\\1010\\1001\\0011\\0101\\0110\end{matrix}\right.$', $\underline{x}$ + '$11\left\{\begin{matrix}0001\\0010\\0100\\1000\end{matrix}\right.$' are in V_{n+2}.

5) There are no other words in V_{n+2}.

We can easily check that these words are all different. Of course the ordering in V_n is lexicographical again; it is essentially the same as in § 2.3.0 and the conversion algorithms are constructed in the same way. We shall not discuss this in detail. We only want to show that this generalized code contains many more words than the original code. To calculate $|V_n|$, we use the recurrence relations (counting words):

$$(1)\quad \begin{cases} |A_{n+2}| = 6\,|A_n| + |B_n| + 3\,|C_n| + |D_n| \\ |B_{n+2}| = 4\,|A_n| + 3|B_n| + 3\,|C_n| + 4\,|D_n| \\ |C_{n+2}| = 4\,|A_n| + 3|B_n| + 3\,|C_n| + 4\,|D_n| \\ |D_{n+2}| = |A_n| + 3|B_n| + |C_n| + 6\,|D_n| \end{cases}$$

Since $V_1 = \{01, 10\}$, $|A_1| = |D_1| = 0$, $|B_1| = |C_1| = 1$, we get $|A_n| = |D_n|$ and $|B_n| = |C_n|$ for all n(induction).
(1) Reduces to

$$(2)\quad \begin{cases} |A_{n+2}| = 7\,|A_n| + 4\,|B_n| \\ |B_{n+2}| = 8\,|A_n| + 6\,|B_n| \end{cases}$$

with $|V_n| = 2|A_n| + 2|B_n|$ for $n \geq 1$.

We can write this as

$$\begin{pmatrix} |A_{n+2}| \\ |B_{n+2}| \end{pmatrix} = M \begin{pmatrix} |A_n| \\ |B_n| \end{pmatrix} \quad \text{with} \quad M = \begin{pmatrix} 7 & 4 \\ 8 & 6 \end{pmatrix}.$$

Let λ_1 and λ_2 be the eigenvalues of M: $\lambda_1 = \frac{13 - \sqrt{129}}{2}$, $\lambda_2 = \frac{13 + \sqrt{129}}{2}$

so $\lambda_1 \approx 0.8211$, $\lambda_2 \approx 12.18$.
Then we know that $|V_{2t+1}| = \alpha_1 \lambda_1^{\,t} + \alpha_2 \lambda_2^{\,t}$ for $t \geq 1$, $n = 2t+1$ and some unimportant constants α_1, α_2.
So $|V_{2t+1}|$ is asymptotically equal to $\alpha_2 \lambda_2^{\,t}$ and $|V_n| \sim \alpha_2' \sqrt{\lambda_2}^{\,n}$, $n \to \infty$ for some α_2'.
Since $\sqrt{\lambda_2} > 3$, the number of codewords is very large compared to the number of codewords in § 2.3.0 which was 2.3^{n-1}. Also,

$$\lim_{n \to \infty} \frac{\log \alpha_2 \sqrt{\lambda_2}^{\,n}}{\log \binom{2n}{n}} = \frac{\log \sqrt{\lambda_2}}{2} \approx 0{,}902 \quad (\text{compare } 0.792,\ \S\ 2.3.0)$$

The problem is how to generalize this iterative procedure in such a way that we can construct all words. The construction would have to like the following example.

Suppose we have V_n containing all $\binom{2n}{n}$ possible words. If $\underline{x} \in V_n$, we add $\underline{x}$ + '01', $\underline{x}$ + '10' to V_{n+1}; change a '0' in $\underline{x}$ into a '1' (n possibilities to do this); call this $\underline{x}'$ and add $\underline{x}'$ + '00' to V_{n+1}. Analogously we change a '1' into '0' and attach '11' to the end of the word. In this way we would have had every word ending on '00' n+1 times, and every word ending on '11' would also be there n+1 times. Delete all words occurring more than once. In this way we would have all words exactly once, but we don't have a way to order them iteratively. Therefore we will not be able to give conversion algorithms like $\underline{A}$ and $\underline{B}$ (§ 2.3.0).

2.4 Summary

We have seen many examples of codes. In section 2.1 we used permutations; addition and multiplication are easy but the code has very few words. In section 2.2 we have seen some codes based on positional systems; arithmetic is easy in such a code, but we have seen that the number of words is quite small too, depending on the base of the system. Section 2.3 deals with a totally different kind of code, using an iterative construction method. We especially recommend this code because of the elegant conversion algorithms.

3. Unsolved problems.

3.0 Partitioning all words into equivalence classes.

We define an equivalence relation $\sim$ on all words $\underline{x} \in \{0,1\}^{2n}$ that have n ones. Let $\underline{x}$ and $\underline{y}$ be words, then

$\underline{x} \sim \underline{y} \Leftrightarrow$ $\underline{y}$ can be obtained from $\underline{x}$ by a cyclic shift on the positions of $\underline{x}$.

It is not true that all equivalence classes contain the same number of words; this may be clear observing the following example.
For n=2 there are 2 equivalence classes: $\{0101, 1010\}$ and $\{0011, 0110, 1100, 1001\}$.

For reasons that will be clear in a moment, we should like all equivalence classes to have equally many words. This would be the case if the word length ℓ were prime: we could shift a word to the left and get ℓ different words, so every class would have ℓ words in it. Obviously 2n cannot be prime unless n=1; therefore we don't look at all positions of the word but only at the last 2n-1 positions. We assume that 2n-1 is prime. Now we only consider words (of length 2n) that start with a '0'. In § 1.2.0 we have seen that it suffices to use these words; we shall call them $\underline{y}_k$ as in the example.
If we disregard the leading '0', which is the same for all words, we now have words of length 2n-1. We introduce the equivalence relation for these words and now all classes contain the same number of words, namely 2n-1.

We choose fixed representatives of all classes, for instance the lexicographically least word of the class. Let the classes be ordered in some way: we call them $C_0, C_1, C_2 .. C_{m-1}$ where m is the number of classes. So

$$m = \binom{2n-1}{n} / (2n-1).$$

The representative of C_i is denoted as $\underline{y}_i^{(0)}$ $(0 \leq i < m)$.
As in chapter 1, we define the indices (i_0, j_0) for a word $\underline{y}$:
i_0 is the number of the equivalence class $\underline{y}$ belongs to, so $\underline{y} \in C_{i_0}$.
j_0 is the number of shifts to the left needed to obtain $\underline{y}$ from the class representative, $\underline{y}_{i_0}^{(0)}$.

Thus, if π denotes a cyclic shift to the left over one position, we have

$\underline{y} = \pi^{j_0}(\underline{y}_{i_0}^{(0)})$ (still disregarding the leading zero!).

As usual the word $\underline{y}$ is called $\underline{y}_{i_0,j_0}$ and the words are ordered according to the lexicographical ordering in (i_0, j_0).
The number corresponding to $\underline{y}_{i_0,j_0}$ is $i_0(2n - 1) + j_0$ (trivial), so the algorithms for addition and multiplication are very simple in this case, using 'div' and 'mod'.

The remaining problem is the ordering in the equivalence classes C_i; it must be such that i_0 can be determined for a word $\underline{y}_{i_0,j_0}$ without too much computing time. If we know i_0 and $\underline{y}_{i_0}^{(0)}$, j_0 can be found easily.

3.1 A connection with Catalan numbers

For a word $\underline{x}$ we define i_0 as the smallest number such that the first $2i_0$ positions of $\underline{x}$ have i_0 zeros and i_0 ones. So $1 \leq i_0 \leq n$.
Let M_i be defined as the number of possibilities for these $2i_0$ positions. E.g. we have $M_2 = 2$: the first 4 positions can be 0011 and 1100; a word starting with 01.. or 10.. has $i_0 = 1$. We also see $M_1 = 2$.
We arrive at a recurrence relation:

$$M_{i_0} = \binom{2i_0}{i_0} - \sum_{i=1}^{i_0-1} M_i \cdot \binom{2i_0-2i}{i_0-i} \quad ,$$

since $M_i \cdot \binom{2i_0-2i}{i_0-i}$ is the number of words of length $2i_0$ that have i zeros and i ones in the first 2i positions already.
We can also give the numbers M_{i_0} explicitly, because they are closely related to the Catalan numbers in combinatorics (cf. [3]):

$$M_{i_0} = \frac{2(2i_0-2)!}{(i_0-1)!\, i_0!}$$

Knowing this, we can also define j_0 and k_0 for a word:
j_0 is the lexicographical number for the first $2i_0$ positions in the M_i possible words.
k_0 is the lexicographical number of the remaining $2n-2i_0$ positions.
Disadvantages of this code are: j_0 is probably very hard to find for a given word. Besides, for small values of i_0 the determining of k_0 consumes most of the computing time, and we are back to our original algorithm for the lexicographical ordering.
Possibly this code is of combinatorial interest only because of the link with Catalan numbers.

3.2 Another kind of lexicographical ordering.

Since many of the given algorithms rely heavily upon the lexicographical ordering, it is perhaps not a bad idea to take a closer look at this lexicographical ordering. We could even consider to change it into a more appropriate algorithm for ordering words.

This final section is devoted to an example of such a different ordering. We certainly do not claim that this ordering is in any way better than the lexicographical ordering; probably it is much worse, but it only serves as an illustration to show that there are other possibilities.

Let 0 be coded as $\underline{x}_0 = 0^n1^n$. Every other number i will be coded as $\underline{x}_i$; the word $\underline{x}_i$ will be associated with a permutation π_i satisfying

$\pi_i \, \underline{x}_0 = \underline{x}_i$.

Of course there are many permutations π that map $\underline{x}_0$ onto $\underline{x}_i$, but we shall make a special choice. This will be explained using an example.

Let the word $\underline{x}_i$ be $\frac{0\ 1\ 1\ 0\ 1\ 1\ 0\ 0}{1\ 2\ 3\ 4\ 5\ 6\ 7\ 8}$: the positions are numbered 1,2..8.

Compare $\underline{x}_i$ with $\underline{x}_0$. The first '1' in $\underline{x}_i$ is on position 2; since $\underline{x}_0$ does not have a '1' in that position, the permutation π_i must put a '1' in position 2. We assume that π_i consists of disjunct transpositions only, so the cycle containing '2' can only be (2 5), (2 6), (2 7) or (2 8) (it swaps a '0' and a '1' in $\underline{x}_0$).

Looking at $\underline{x}_i$ we see that the zeros in positions 1 and 4 have not changed; the first zero that has changed (compared with $\underline{x}_0$) is in position 7. Therefore we assume that this zero has been swapped with the '1' in position 2: the cycle is (2 7).

There is one more '1' that has changed its position compared with $\underline{x}_0$: it is in position 3. Reasoning in the same way we assume that it changed places with the '0' in position 8. Hence we find π_i = (2 7)(3 8).

In this way we can find permutations π_i for every word $\underline{x}_i$. Now we order the <u>permutations</u> lexicographically, looking at their disjunct cycle notation. E.g. (1 5)(2 7)(4 8) $\prec$ (2 6)(3 8) etc.

Thus we find π_0 = (1), π_1 = (1 5), π_2 = (1 5)(2 6) etc., for n=4.

Since $\underline{x}_i = \pi_i \, \underline{x}_0$, the words $\underline{x}_i$ are ordered in the same way.

For n=3 we find the list on page 43.

i	π_i	$\underline{x}_i$	i	π_i	$\underline{x}_i$
0	(1)	000111	10	(1 6)	100110
1	(1 4)	100011	11	(2 4)	010011
2	(1 4)(2 5)	110001	12	(2 4)(3 5)	011001
3	(1 4)(2 5)(3 6)	111000	13	(2 4)(3 6)	011010
4	(1 4)(2 6)	110010	14	(2 5)	010101
5	(1 4)(3 5)	101001	15	(2 5)(3 6)	011100
6	(1 4)(3 6)	101010	16	(2 6)	010110
7	(1 5)	100101	17	(3 4)	001011
8	(1 5)(2 6)	110100	18	(3 5)	001101
9	(1 5)(3 6)	101100	19	(3 6)	001110

We see that the conversion $\underline{x}_i \to \pi_i$ is quite easy, but $\underline{x}_i \to i$ or $\pi_i \to i$ is much more complex. In fact it is possible to calculate i from π_i, e.g. the number of permutations starting with '1' is $1 + \binom{2n-1}{n}$ (1 for π_0 and $\binom{2n-1}{n}$ for the <u>words</u> starting with '1'). These calculations resemble the algorithms for lexicographical ordering in § 1.0; actually, they are even more difficult. Therefore we still prefer the lexicographical ordering at the moment. However, other orderings may be found that are better suitable for these purposes.

<u>Acknowledgement</u>

The author wishes to thank professor M. Rem for his stimulating support during this work.

<u>References</u>

1 J.P.M. Schalkwijk, 'An algorithm for source coding', IEEE Trans. Inform. Theory, vol IT-18, pp 395-399, May 1972

2 Hua Loo Keng, 'Introduction to Number Theory', Springer Verlag, Berlin Heidelberg New York, 1982

3 L. Comtet, 'Advanced Combinatorics', D. Reidel Publishing Company Dordrecht-Holland Boston- U.S.A.

Appendix.

Examples for §2.1.1: we list $f_n(k) := \min\left\{\operatorname{lcm}\{1,2,..,k\},\ k\binom{2n-k}{\lfloor\frac{1}{2}(2n-k)\rfloor}\right\}$ for $1 \le k \le 2n$,

$M_n = \max\left\{f_n(k) \mid 1 \le k \le 2n\right.$, $\dfrac{M_n}{\binom{2n}{n}}$ and $\dfrac{\log_2 M_n}{\log_2\binom{2n}{n}}$ for n= 4, 5, 10, 20.

n = 4

k	1	2	3	4	5	6	7	8
$f_n(k)$	1	2	6	12	15	12	7	8

$M_n = 15$, $\dfrac{M_n}{\binom{2n}{n}} = 0.21$, $\dfrac{\log M_n}{\log\binom{2n}{n}} = 0.637$

n = 5

k	1	2	3	4	5	6	7	8	9	10
$f_n(k)$	1	2	6	12	50	36	21	16	9	10

$M_n = 50$, $\dfrac{M_n}{\binom{2n}{n}} = 0.20$,

$\dfrac{\log M_n}{\log\binom{2n}{n}} = 0.707$

n = 10

k	1	2	3	4	5	6	7	8	9	10	11	12
$f_n(k)$	1	2	6	12	60	60	420	840	2520	2520	1386	840

k	13	14	15	16	17	18	19	20
$f_n(k)$	455	280	150	96	51	36	19	20

$M_n = 2520$, $\dfrac{M_n}{\binom{2n}{n}} = 0.014$, $\dfrac{\log M_n}{\log\binom{2n}{n}} = 0.646$

n = 20

k	1	2	3	4	5	6	7	8	9	10	11	12	13
$f_n(k)$	1	2	6	12	60	60	420	840	2520	2520	27720	27720	360360

k	14	15	16	17	18	19	20
$f_n(k)$	360360	360360	720720	12252240	12252240	6701604	3695120

k	21	22	23	24	25	26	27	28
$f_n(k)$	1939938	1069640	559130	308880	160875	89232	46332	25872

k	29	30	31	32	34	35	36	37	38	39	40
$f_n(k)$	13398	7560	3906	2240	680	350	216	111	76	39	40

$M_n = 12{,}252{,}240$; $M_n/\binom{2n}{n} = 8.89 \cdot 10^{-5}$; $\log M_n \ / \log\binom{2n}{n} = 0.636.$

Example of § 2.1.3:

k	p_k	$2.n(k)$	$\prod_{i=1}^{k} p_i$	$\frac{\log_2(\prod_{i=1}^{k} p_i)}{2.n(k)}$
1	2	2	2	0.5000
2	3	6	6	0.4308
3	5	10	30	0.4963
4	7	18	210	0.4286
5	11	28	2310	0.3991
6	13	42	30030	0.3541
7	17	58	510510	0.3269
8	19	78	9699690	0.2976
9	23	100	$2.23093 \cdot 10^{8}$	0.2773
10	27	128	$6.02351 \cdot 10^{9}$	0.2538
11	29	156	$1.74682 \cdot 10^{11}$	0.2394
12	31	188	$5.41513 \cdot 10^{12}$	0.2250