Eindhoven University of Technology

MASTER

Continuous strategies for the binary multiplying delay channel

Diederiks, Patrick J.E.

Award date:
1994

Link to publication

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 27. Apr. 2025

719 3

Faculty of Electrical Engineering
Section of Information and Communication Theory

MASTER'S THESIS

Continuous Strategies for the Binary Multiplying Delay Channel

by Patrick J.E. Diederiks

Coach and Supervisor : prof.dr.ir. J.P.M. Schalkwijk

Eindhoven, june 1994

The Faculty of Electrical Engineering of the Eindhoven University accepts no responsibility for the contents of graduation reports.

Summary

In this thesis we make a first attempt in constructing continuous coding strategies for the Binary Multiplying Delay Channel. This channel is derived from the Binary Multiplying Channel, by adding some delay between the two users, such that it takes some time before an input symbol reaches the other receiver. This delay, which arises from the use of high bit rates and/or large distances, makes that two important properties of the BMC are no longer valid. In the first place is it no longer true that the output of user 1 is equal to the output of user 2, which means that the BMDC is not a so called T-channel. In the second place does user 1 not know what the output is of user 2 and vice versa. Because of these differences it is not possible to utilize the existing coding strategies, which are designed for the BMC, with the BMDC.
In an attempt to exceed Shannon's inner bound by using feedback, we therefore have designed two new coding strategies, which are both implemented by using the unit square. The first coding strategy, called the simultaneous coding strategy, is realized up to depth 5 and gives a simultaneous description of the outputs of both users.
From the results it follows that these strategies do not make use of the present feedback, which indicates that the depth 5 coding strategy is not deep enough. However, the implementation of coding strategies of greater depth appeared to be too complex, so we had to come up with a new coding strategy that could be described with less parameters. To accomplish this, we used the fact that in the time domain the BMDC can be described by two independent paths, which made it possible to consider only sequential outputs of both users. The involving decrease of complexity made it possible to construct sequential coding strategies up to depth 7. The depth 6 coding strategy could not benefit from the feedback, but the depth 7 coding strategy however, turned out to be the first coding strategy that could actually surpass Shannon's inner bound.

Contents

1 Introduction

For many years now Blackwells Binary Multiplying Channel (BMC) has been studied in order to determine it's capacity region. Despite the fact that this goal has not been reached up to now, many progress has been made on finding good coding strategies which outrun the inner bound region for this channel. To demonstrate the usefulness of these strategies, two personal computers were used to communicate through a wired AND channel, both using the advantages of the coding strategies [1]. This demonstration model could only be used with low transmission speed, but when higher bit-rates will be used, one has to deal with a time-delay-factor, i.e. one can't neglect the time it takes before an input signal reaches the other receiver. In this way this physical property adds some sort of memory to the BMC and therefore creates a new channel called the Binary Multiplying Delay Channel (BMDC). An example of the BMDC is depicted in figure 1.

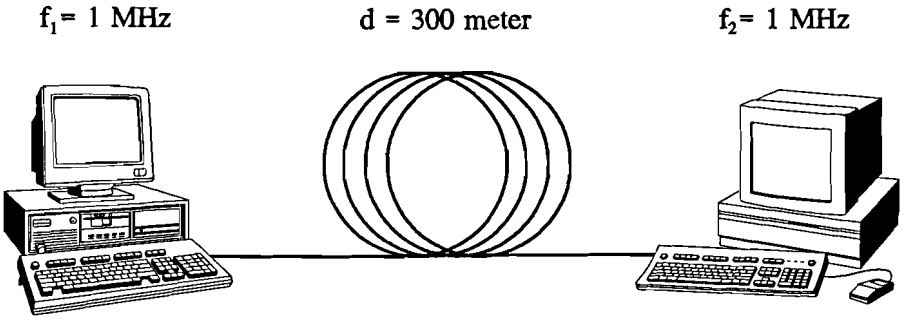

Figure 1: An example of the Binary Multiplying Delay Channel

In this figure the BMC is represented as a fibreglass, whereby a light pulse corresponds with a zero. The time-delay-factor subsequently is created by letting two terminals communicate over a fibreglass of length 300 m. Assuming that light moves with $3.0 \cdot 10^8$ m s^{-1} through fibreglass, it can be easily verified that, when both terminals communicate at a frequency of 1 MHz, the time-delay-factor is exactly 1 bit, in this way creating the BMDC.

In his 1961 paper on two-way channels [3], Shannon already briefly discussed two-way channels with certain types of memory. He concluded that the determination of the capacity region G is more complex than the case in which the channel is memoryless. Nevertheless will this report describe a first attempt on trying to construct strategies for the Binary Multiplying Delay Channel in an effort to exceed Shannon's inner bound.

2 Basic definitions

Shannon [3] described the general discrete two-way channel with memory as a channel with input sequences:

$$\begin{aligned} X_{1,n} &= (x_{1,1}, x_{1,2}, \ldots, x_{1,n}) \\ X_{2,n} &= (x_{2,1}, x_{2,2}, \ldots, x_{2,n}) \end{aligned} \tag{1}$$

and output sequences:

$$\begin{aligned} Y_{1,n} &= (y_{1,1}, y_{1,2}, \ldots, y_{1,n}) \\ Y_{2,n} &= (y_{2,1}, y_{2,2}, \ldots, y_{2,n}) \end{aligned} \tag{2}$$

which is defined by a set of conditional probabilities:

$$P\left[y_{1,n}, y_{2,n} \mid x_{1,1}, \ldots, x_{1,n}; x_{2,1}, \ldots, x_{2,n}; y_{1,1}, \ldots, y_{1,n-1}; y_{2,1}, \ldots, y_{2,n-1}\right] \tag{3}$$

This is the probability of the nth output pair $y_{1,n}$, $y_{2,n}$ conditional on the preceding history from time $t = 0$, that is, the input and output sequences from the starting time in using the channel. Figure 2 shows a two-way channel configuration with two terminals, each having an encoder to encode messages to the input sequences and a decoder to decode the output sequences back to messages.

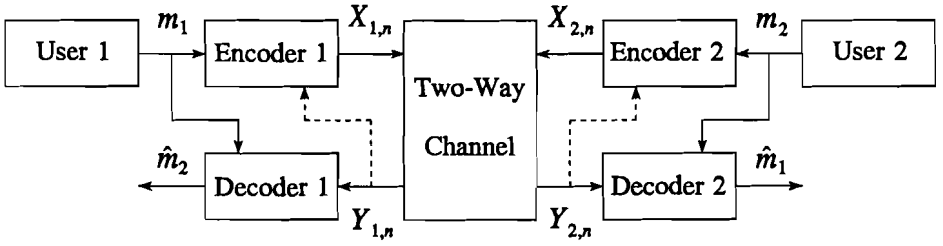

Figure 2: The general discrete two-way channel with memory

The dotted lines in figure 2, representing the two different situations which will come up in this thesis, indicate whether or not the channel output is known to the encoder. If true, the encoder can use this information to adapt its next channel input to the previous channel outputs, i.e. the encoder uses a *strategy* to construct the output symbols. Henceforth this situation will be described as the *feedback* situation. When the encoder does not have this feedback, it will just generate *codes*.

Definition (3) gives the most general description of a two-way channel with memory, but from now on we will concentrate on one specific channel, namely the Binary Multiplying Delay Channel (BMDC), which is schematically represented in figure 3.

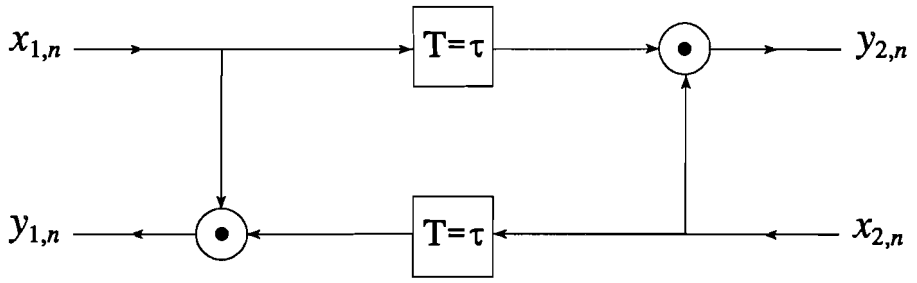

Figure 3: The Binary Multiplying Delay Channel

It follows from figure 3 that the nth output pair $y_{1,n}$, $y_{2,n}$ is given by:

$$\begin{aligned} y_{1,n} &= x_{1,n} \cdot x_{2,n-\tau} \\ y_{2,n} &= x_{2,n} \cdot x_{1,n-\tau} \end{aligned} \tag{4}$$

where the input and output symbols $x_{1,n}$, $x_{2,n}$, $y_{1,n}$, $y_{2,n} \in \{0,1\}$ and the multiplication is a logical AND.

This means that, unlike the BMC, the BMDC is not a so called T-channel, since the outputs $y_{1,n}$ and $y_{2,n}$ of user 1 and 2, respectively, are not the same. Therefore it is not possible to use the coding strategies which are designed for the BMC. So we implemented two new coding strategies for the BMDC, which will be discussed in the next two chapters.

Without loss of generality we have designed these coding strategies for the BMDC with time-delay-factor 1, since one can interleave τ of these coding strategies, when the time-delay-factor of the BMDC equals τ. This means that both users use τ independent coding strategies at the same time, of which the τ input sequences of length n of user 1 are defined by:

$$\begin{aligned} X_{1,n}^{1} &= \left(x_{1,1}, x_{1,\tau+1}, \ldots, x_{1,(n-1)\tau+1}\right) \\ X_{1,n}^{2} &= \left(x_{1,2}, x_{1,\tau+2}, \ldots, x_{1,(n-1)\tau+2}\right) \\ &\vdots \\ X_{1,n}^{i} &= \left(x_{1,i}, x_{1,\tau+i}, \ldots, x_{1,(n-1)\tau+i}\right) \end{aligned} \tag{5}$$

and the τ input sequences of user 2:

$$\begin{aligned} X_{1,n}^{1} &= \left(x_{1,1}, x_{1,\tau+1}, \ldots, x_{1,(n-1)\tau+1}\right) \\ X_{1,n}^{2} &= \left(x_{1,2}, x_{1,\tau+2}, \ldots, x_{1,(n-1)\tau+2}\right) \\ &\vdots \\ X_{1,n}^{i} &= \left(x_{1,i}, x_{1,\tau+i}, \ldots, x_{1,(n-1)\tau+i}\right) \end{aligned} \tag{6}$$

where $1 \leq i \leq \tau$ is the index of the interleaved coding strategy.

The output sequences of user 1 and 2 are defined by:

$$\begin{aligned} Y_{1,n}^{i} &= \left(y_{1,i}, y_{1,\tau+i}, \ldots, y_{1,(n-1)\tau+i}\right) \\ Y_{2,n}^{i} &= \left(y_{2,i}, y_{2,\tau+i}, \ldots, y_{2,(n-1)\tau+i}\right) \end{aligned} \tag{7}$$

Therefore will we from now on use:

$$y_{1,n} = x_{1,n} \cdot x_{2,n-1}$$
$$y_{2,n} = x_{2,n} \cdot x_{1,n-1} \tag{8}$$

to define the nth output pair $y_{1,n}$, $y_{2,n}$.

3 The simultaneous coding strategy

3.1 Strategy description

The main difference with the BMC is that the two output symbols of the BMDC are different and time dependent. Nevertheless will we try to apply the same approach in constructing a coding strategy as the ones described in [2] and [4]. We therefore reintroduce the unit square, to visualize the coding strategy. Again each sender tries to send information that can be taken as a subinterval Θ of a [0,1] interval, where the length of the subinterval specifies the amount of information which will be sent. Furthermore define θ_1 and θ_2 as the middle points of the subintervals of senders 1 and 2, respectively. Assuming that both users know nothing in advance, each subinterval is uniformly distributed over the [0,1] interval.
According to (1), $x_{1,0}$ and $x_{2,0}$ are not defined, which denote that the first output symbols $y_{1,1}$ and $y_{2,1}$ are not specified. For that reason (2) is redefined by:

$$\begin{aligned} Y_{1,n} &= (y_{1,2}, y_{1,3}, \cdots, y_{1,n}, y_{1,n+1}) \\ Y_{2,n} &= (y_{2,2}, y_{2,3}, \cdots, y_{2,n}, y_{2,n+1}) \end{aligned} \tag{9}$$

This means that the first input symbols $x_{1,1}$ and $x_{2,1}$ are only used to initialize the channel. But the introduction of $y_{1,n+1}$ and $y_{2,n+1}$ requires the existence of $x_{1,n+1}$ and $x_{2,n+1}$. Since these are actually used to 'clean up' the channel, they are always set to 1. In this way (1) becomes:

$$\begin{aligned} X_{1,n} &= (x_{1,1}, x_{1,2}, \cdots, x_{1,n}, 1) \\ X_{2,n} &= (x_{2,1}, x_{2,2}, \cdots, x_{2,n}, 1) \end{aligned} \tag{10}$$

which implies that when a strategy of depth n is used, both senders in fact have to send $n + 1$ input symbols.

At first we are interested in sending equal rates in both directions, so the transmission situation relative to the senders is completely symmetrical. Figure 4 shows the situation after both senders have sent their first two input symbols. Because one has to make a distinction between the output symbols $y_{1,n}$ and $y_{2,n}$, they are simultaneously displayed in the unit square. In this figure α_i represents a relative parameter, thus $\alpha_i \in [0, 1]$, where $i = 1,2,3$. To save space we also define $\beta_i = 1 - \alpha_i$. Now the first input symbols $x_{1,1}$ and $x_{2,1}$ can be determined as follows: if $\theta_j \in [0, \alpha_1]$ then $x_{j,1} = 1$, otherwise if $\theta_j \in \langle \alpha_1, 1]$ then $x_{j,1} = 0$, where $j = 1,2$. Since the first output symbols $y_{1,1}$ and $y_{2,1}$ are not specified, they do not provide the sender any information. This will happen when the second input symbol is sent. As can be seen in figure 4, encoder j produces an input $x_{j,2} = 1$ if $\theta_j \in [0, \alpha_1\alpha_2] \cup [\alpha_1, \alpha_1 + \beta_1\alpha_3]$, and $x_{j,2} = 0$ if $\theta_j \in \langle \alpha_1\alpha_2, \alpha_1) \cup \langle \alpha_1 + \beta_1\alpha_3, 1]$, where $j = 1,2$.

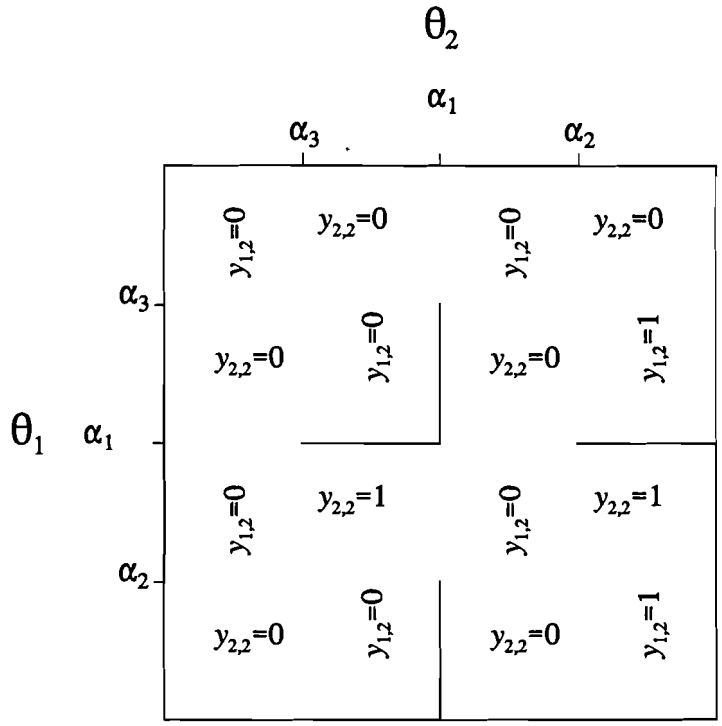

Figure 4: Subdivision of the unit square after 2 transmissions

The average mutual information in the $1 \rightarrow 2$ direction is given by:

$$\begin{aligned} I(\theta_1; Y_{2,2} \mid \theta_2) = {} & P[x_{2,2}=1 \wedge X_{2,1}]\, I(\theta_1; Y_{2,2} \mid x_{2,2}=1 \wedge X_{2,1}) + \\ & P[x_{2,2}=0 \wedge X_{2,1}]\, I(\theta_1; Y_{2,2} \mid x_{2,2}=0 \wedge X_{2,1}) \end{aligned} \tag{11}$$

which can also been written as:

$$\begin{aligned} I(\theta_1; Y_{2,2} \mid \theta_2) = {} & P[\theta_2 \in [0, \alpha_1\alpha_2]]\, I(\theta_1; Y_{2,2} \mid \theta_2 \in [0, \alpha_1\alpha_2]) + \\ & P[\theta_2 \in \langle \alpha_1\alpha_2, \alpha_1 \rangle]\, I(\theta_1; Y_{2,2} \mid \theta_2 \in \langle \alpha_1\alpha_2, \alpha_1 \rangle) + \\ & P[\theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3]]\, I(\theta_1; Y_{2,2} \mid \theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3]) + \\ & P[\theta_2 \in \langle \alpha_1 + \beta_1\alpha_3, 1]]\, I(\theta_1; Y_{2,2} \mid \theta_2 \in \langle \alpha_1 + \beta_1\alpha_3, 1]) \end{aligned} \tag{12}$$

Now $I(\theta_1; Y_{2,2} \mid \theta_2 \in \langle \alpha_1\alpha_2, \alpha_1 \rangle) = I(\theta_1; Y_{2,2} \mid \theta_2 \in \langle \alpha_1 + \beta_1\alpha_3, 1]) = 0$ since for $\theta_2 \in \langle \alpha_1\alpha_2, \alpha_1 \rangle \cup \langle \alpha_1 + \beta_1\alpha_3, 1]$ the output at terminal 2 is $y_{2,2} = 0$ regardless of the value of θ_1. For $\theta_2 \in [0, \alpha_1\alpha_2] \cup [\alpha_1, \alpha_1 + \beta_1\alpha_3]$ the output at terminal 2 is $y_{2,2} = 1$ or 0 depending on whether $\theta_1 \in [0, \alpha_1]$ or $\theta_1 \in \langle \alpha_1, 1]$, respectively. With $P[\theta_1 \in [0, \alpha_1]] = \alpha_1$ and $P[\theta_1 \in (\alpha_1, 1]] = \beta_1$ it follows that the

conditional average mutual information is given by:

$$I(\theta_1; Y_{2,2} \mid \theta_2 \in [0, \alpha_1\alpha_2]) = h(\alpha_1)$$
$$I(\theta_1; Y_{2,2} \mid \theta_2 \in [\alpha_1, \alpha_1+\beta_1\alpha_3]) = h(\alpha_1) \tag{13}$$

where $h(\alpha)$ is the binary entropy function $-\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha)$. Substituting (13) and the probabilities $P[\theta_2 \in [0, \alpha_1\alpha_2]] = \alpha_1\alpha_2$ and $P[\theta_2 \in [\alpha_1, \alpha_1+\beta_1\alpha_3]] = \beta_1\alpha_3$ in (12) leads to the following equation for the average mutual information in the $1 \rightarrow 2$ direction, and because of symmetry also in the $2 \rightarrow 1$ direction, after two transmissions:

$$I(\theta_1; Y_{2,2} \mid \theta_2) = I(\theta_2; Y_{1,2} \mid \theta_1) = (\alpha_1\alpha_2 + \beta_1\alpha_3)h(\alpha_1) \tag{14}$$

Up to now it is not possible to implement feedback with the strategy, but after $y_{j,2}$ has been received and $x_{j,2}$ is a 1, it is possible for encoder j to discriminate between $y_{j,2} = 0$ and $y_{j,2} = 1$, so he can use this knowledge to determine $x_{j,3}$, where $j = 1,2$. This can be seen in figure 5, where, because of clarity, the output symbols $y_{j,3}$ have been left out. Additionally have the previous input symbols $x_{j,2} = 1$ been indicated by dotted lines.

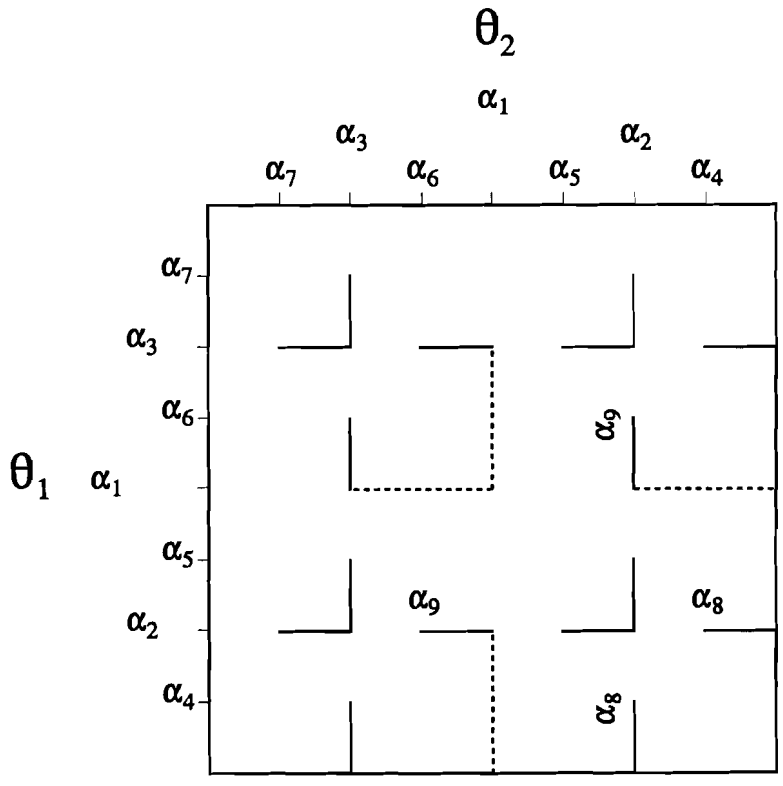

Figure 5: Subdivision of the unit square after three transmissions

One can see in this figure that the feedback is realized when $y_{j,2} = 1$ by replacing the two parameters α_4 and α_6 by α_8 and α_9, respectively. So $x_{j,3} = 1$ if $y_{j,2} = 0$ and $\theta_j \in [0, \alpha_1\alpha_2\alpha_4] \cup$

$\langle\alpha_1\alpha_2, \alpha_1\alpha_2 + \alpha_1\beta_2\alpha_5] \cup [\alpha_1, \alpha_1 + \beta_1\alpha_3\alpha_6] \cup \langle\alpha_1 + \beta_1\alpha_3, \alpha_1 + \beta_1\alpha_3 + \beta_1\beta_3\alpha_7]$ or $y_{j,2} = 1$ and $\theta_j \in [0, \alpha_1\alpha_2\alpha_8] \cup [\alpha_1, \alpha_1 + \beta_1\alpha_3\alpha_9]$ and $x_{j,3} = 0$ otherwise, where $j = 1,2$. For the average mutual information in the $1 \rightarrow 2$ direction after the third transmission follows:

$$\begin{aligned} I(\theta_1; Y_{2,3} \mid \theta_2, Y_{2,2}) = {} & P[x_{2,3} = 1 \wedge X_{2,2} \wedge Y_{2,2}]\, I(\theta_1; Y_{2,3} \mid x_{2,3} = 1, X_{2,2}, Y_{2,2}) + \\ & P[x_{2,3} = 0 \wedge X_{2,2} \wedge Y_{2,2}]\, I(\theta_1; Y_{2,3} \mid x_{2,3} = 0, X_{2,2}, Y_{2,2}) \end{aligned} \tag{15}$$

Using the fact that $I(\theta_1; Y_{2,3} \mid x_{2,3} = 0, X_{2,2}, Y_{2,2}) = 0$ we can write (15) as:

$$\begin{aligned} I(\theta_1; Y_{2,3} \mid \theta_2, Y_{2,2}) = {} & P[\theta_2 \in [0, \alpha_1\alpha_2\alpha_4]]\, P[y_{2,2} = 0 \mid \theta_2 \in [0, \alpha_1\alpha_2]] \cdot \\ & I(\theta_1; Y_{2,3} \mid \theta_2 \in [0, \alpha_1\alpha_2\alpha_4] \wedge y_{2,2} = 0) + \\ & P[\theta_2 \in [0, \alpha_1\alpha_2\alpha_8]]\, P[y_{2,2} = 1 \mid \theta_2 \in [0, \alpha_1\alpha_2]] \cdot \\ & I(\theta_1; Y_{2,3} \mid \theta_2 \in [0, \alpha_1\alpha_2\alpha_8] \wedge y_{2,2} = 1) + \\ & P[\theta_2 \in \langle\alpha_1\alpha_2, \alpha_1\alpha_2 + \alpha_1\beta_2\alpha_5]]\, P[y_{2,2} = 0 \mid \theta_2 \in \langle\alpha_1\alpha_2, \alpha_1\rangle] \cdot \\ & I(\theta_1; Y_{2,3} \mid \theta_2 \in \langle\alpha_1\alpha_2, \alpha_1\alpha_2 + \alpha_1\beta_2\alpha_5] \wedge y_{2,2} = 0) + \\ & P[\theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3\alpha_6]]\, P[y_{2,2} = 0 \mid \theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3]] \cdot \\ & I(\theta_1; Y_{2,3} \mid \theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3\alpha_6] \wedge y_{2,2} = 0) + \\ & P[\theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3\alpha_9]]\, P[y_{2,2} = 1 \mid \theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3]] \cdot \\ & I(\theta_1; Y_{2,3} \mid \theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3\alpha_9] \wedge y_{2,2} = 1) + \\ & P[\theta_2 \in \langle\alpha_1 + \beta_1\alpha_3, \alpha_1 + \beta_1\alpha_3 + \beta_1\beta_3\alpha_7]]\, P[y_{2,2} = 0 \mid \theta_2 \in \langle\alpha_1 + \beta_1\alpha_3, 1]] \cdot \\ & I(\theta_1; Y_{2,3} \mid \theta_2 \in \langle\alpha_1 + \beta_1\alpha_3, \alpha_1 + \beta_1\alpha_3 + \beta_1\beta_3\alpha_7] \wedge y_{2,2} = 0) \end{aligned} \tag{16}$$

Pursuing the same analogy as described before we can calculate (16) with only one difference, namely that the previous output $y_{2,2}$ has to be taken into account. First of all, when $x_{2,2} = 1$ and $y_{2,2} = 0$, then $P[y_{2,2} = 0 \mid \theta_2 \in [0, \alpha_1\alpha_2]] = P[y_{2,2} = 0 \mid \theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3]] = \beta_1$ and accordingly $I(\theta_1; Y_{2,3} \mid \theta_2 \in [0, \alpha_1\alpha_2\alpha_4] \wedge y_{2,2} = 0) = I(\theta_1; Y_{2,3} \mid \theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3\alpha_6] \wedge y_{2,2} = 0) = h(\alpha_3)$. Secondly, when $x_{2,2} = 1$ and $y_{2,2} = 1$, then $P[y_{2,2} = 1 \mid \theta_2 \in [0, \alpha_1\alpha_2]] = P[y_{2,2} = 1 \mid \theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3]] = \alpha_1$ and $I(\theta_1; Y_{2,3} \mid \theta_2 \in [0, \alpha_1\alpha_2\alpha_8] \wedge y_{2,2} = 1) = I(\theta_1; Y_{2,3} \mid \theta_2 \in [\alpha_1, \alpha_1 + \beta_1\alpha_3\alpha_9] \wedge y_{2,2} = 1) = h(\alpha_2)$. When $x_{2,2} = 0$ then $y_{2,2}$ is always 0, so $P[y_{2,2} = 0 \mid \theta_2 \in \langle\alpha_1\alpha_2, \alpha_1\rangle] = P[y_{2,2} = 0 \mid \theta_2 \in \langle\alpha_1 + \beta_1\alpha_3, 1]] = 1$ and $I(\theta_1; Y_{2,3} \mid \theta_2 \in \langle\alpha_1\alpha_2, \alpha_1\alpha_2 + \alpha_1\beta_2\alpha_5] \wedge y_{2,2} = 0) =$
$I(\theta_1; Y_{2,3} \mid \theta_2 \in \langle\alpha_1 + \beta_1\alpha_3, \alpha_1 + \beta_1\alpha_3 + \beta_1\beta_3\alpha_7] \wedge y_{2,2} = 0) = h(\alpha_1\alpha_2 + \beta_1\alpha_3)$. Substituting this in (16) finally results in the average mutual information in the $1 \rightarrow 2$ as well as the $2 \rightarrow 1$ direction after the third transmission:

$$\begin{aligned} I(\theta_1; Y_{2,3} \mid \theta_2, Y_{2,2}) = I(\theta_2; Y_{1,3} \mid \theta_1, Y_{1,2}) = {} & \alpha_1\alpha_2\alpha_4 \cdot \beta_1 \cdot h(\alpha_3) + \\ & \alpha_1\alpha_2\alpha_8 \cdot \alpha_1 \cdot h(\alpha_2) + \\ & \alpha_1\beta_2\alpha_5 \cdot 1 \cdot h(\alpha_1\alpha_2 + \beta_1\alpha_3) + \\ & \beta_1\alpha_3\alpha_6 \cdot \beta_1 \cdot h(\alpha_3) + \\ & \beta_1\alpha_3\alpha_9 \cdot \alpha_1 \cdot h(\alpha_2) + \\ & \beta_1\beta_3\alpha_7 \cdot 1 \cdot h(\alpha_1\alpha_2 + \beta_1\alpha_3) \end{aligned} \tag{17}$$

Depending on the depth n of the coding strategy, one must repeat the last step n - 3 times, before both users send their last input symbols $x_{1,n+1}$ and $x_{2,n+1}$, which are, according to (10), equal to 1.

Every new input symbol can lead to three different states: when $x_{i,j}=1$ and $y_{i,j}=0$, when $x_{i,j}=1$ and $y_{i,j}=0$, and finally when $x_{i,j}=0$, where i = 1,2, j < n + 1. Defining p_n as the number of parameters for a depth n coding strategy, it follows that:

$$p_n = 3^{n-1} \tag{18}$$

For the depth 3 coding strategy we have depicted the unit square in figure 6 for user 2, after he has sent $x_{2,4}$.

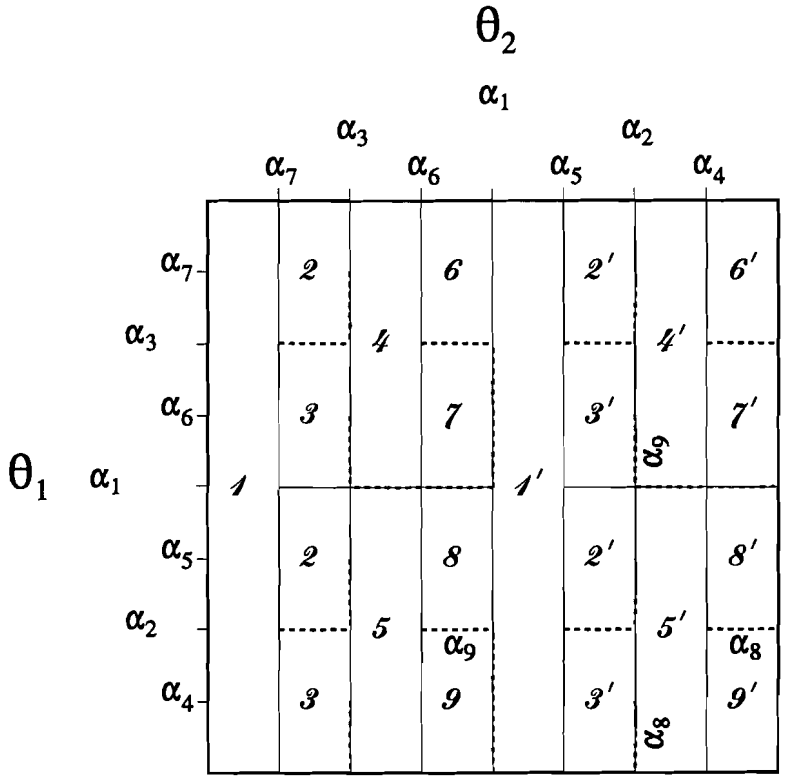

Figure 6: The unit square for the depth 3 coding strategy after 4 input symbols

The additional italic numbers in this figure denote the areas that have the same conditional average mutual information. Table I gives for each area the corresponding conditional average mutual information and the accompanying conditional probabilities.

Table I: The average mutual information for each area in figure 6

area	$I(\theta_1; Y_{2,4} \mid X_{2,3}, Y_{2,3})$	$P[X_{2,3} \wedge Y_{2,3}]$
1	$h(\alpha_1(\alpha_2\alpha_4 + \beta_2\alpha_5) + \beta_1(\alpha_3\alpha_6 + \beta_3\alpha_7))$	$\beta_1\beta_3\beta_7$
2	$h((\alpha_1\beta_1\alpha_5 + \beta_1\beta_3\alpha_7) / (\alpha_1\beta_2 + \beta_1\beta_3))$	$\beta_1\beta_3\alpha_7\,(\alpha_1\beta_2 + \beta_1\beta_3)$
3	$h((\alpha_1\alpha_2\alpha_4 + \beta_1\alpha_3\alpha_6) / (\alpha_1\alpha_2 + \beta_1\alpha_3))$	$\beta_1\beta_3\alpha_7\,(\alpha_1\alpha_2 + \beta_1\alpha_3)$
4	$h(\alpha_3\alpha_6 + \beta_3\beta_7)$	$\beta_1\alpha_3\beta_6\,\beta_1$
5	$h(\alpha_2\alpha_4 + \beta_2\alpha_5)$	$\beta_1\alpha_3\beta_9\,\alpha_1$
6	$h(\alpha_7)$	$\beta_1\alpha_3\alpha_6\,\beta_1\beta_3$
7	$h(\alpha_6)$	$\beta_1\alpha_3\alpha_6\,\beta_1\alpha_3$
8	$h(\alpha_5)$	$\beta_1\alpha_3\alpha_9\,\alpha_1\beta_2$
9	$h(\alpha_4)$	$\beta_1\alpha_3\alpha_9\,\alpha_1\alpha_2$
1'	$h(\alpha_1(\alpha_2\alpha_8 + \beta_2\alpha_5) + \beta_1(\alpha_3\alpha_9 + \beta_3\alpha_7))$	$\alpha_1\beta_2\beta_5$
2'	$h((\alpha_1\beta_2\alpha_5 + \beta_1\beta_3\alpha_7) / (\alpha_1\beta_2 + \beta_1\beta_3))$	$\alpha_1\beta_2\alpha_5\,(\alpha_1\beta_2 + \beta_1\beta_3)$
3'	$h((\alpha_1\alpha_2\alpha_8 + \beta_1\alpha_3\alpha_9) / (\alpha_1\alpha_2 + \beta_1\alpha_3))$	$\alpha_1\beta_2\alpha_5\,(\alpha_1\alpha_2 + \beta_1\alpha_3)$
4'	$h(\alpha_3\alpha_9 + \beta_3\alpha_7)$	$\alpha_1\alpha_2\beta_4\,\beta_1$
5'	$h(\alpha_2\alpha_8 + \beta_2\alpha_5)$	$\alpha_1\alpha_2\beta_8\,\alpha_1$
6'	$h(\alpha_7)$	$\alpha_1\alpha_2\alpha_4\,\beta_1\beta_3$
7'	$h(\alpha_9)$	$\alpha_1\alpha_2\alpha_4\,\beta_1\alpha_3$
8'	$h(\alpha_5)$	$\alpha_1\alpha_2\alpha_8\,\alpha_1\beta_2$
9'	$h(\alpha_8)$	$\alpha_1\alpha_2\alpha_8\,\alpha_1\alpha_2$

These numbers furthermore refer to the numbers used in the source code of the program `mems3`. In this way it is easy to determine what part of the source code belongs to what particular area of the unit square. The next paragraph will discuss the results we found for the depth 3, 4 and 5 strategies using different sorts of maximizing algorithms.

3.2 Results

3.2.1 Depth 3 coding strategy

The overall transmission rate of the depth 3 coding strategy, which is described in paragraph 3.1, is given by

$$R_j = \frac{I(\theta_j; Y_{k,2} \mid \theta_k) + I(\theta_j; Y_{k,3} \mid \theta_k, Y_{k,2}) + I(\theta_j; Y_{k,4} \mid \theta_k, Y_{k,3})}{4}, \quad j=1,2,\ k=1,2,\ j \neq k \qquad (19)$$

and is calculated by the program `mems3` of which the source code `mems3.p` can be found in appendix 1. In this program, and all the following programs, we used the array variable `a` and `b` to implement the different α's and β's respectively. This means that α_i is represented by `a[i]` and β_i by `b[i]`. The binary entropy function h is realized by the function `H` and the calculation of the average mutual information is performed by the function `V`, where the variable `avi`, denoting the average mutual information, is updated after each input symbol. In this function we furthermore placed the numbers of the areas, as shown in figure 6 and listed in table I, between brackets, so it is easy to determine what the average mutual information is for a particular area of the unit square.

To find the maximum of the average mutual information, we used several different methods to the determine the optimal parameters. The first approach, from now on called the *normal*-optimization, is listed within the source code of `mems3.p` in appendix 1. It optimizes the parameters one at a time, and repeats this step with increasing accuracy. It assumes that there is only one maximum, by terminating the optimization of a parameter of the present step, when a next value of this parameter decreases the average mutual information. The resulting optimal values of the nine parameters are listed in table II and give rise to the following overall transmission rate:

$$R_1 = R_2 = 0.55298 \qquad (20)$$

Table II: Optimal parameters for depth 3 coding strategy

α_1	0.50000	α_6	0.64000
α_2	0.75000	α_7	0.64000
α_3	0.75000	α_8	0.64000
α_4	0.64000	α_9	0.64000
α_5	0.64000		

As one can see in table II, the two parameters α_8 and α_9, which introduced feedback, are equal to α_4 and α_6 respectively. This means that with these parameters the feedback is actually not

used.
To check whether the order of optimizing the parameters would influence the maximum, we utilized a random generator, of which the source code `rndgen.p` is listed in appendix 2, to randomize the initial values of the parameters and the sequence by which the parameters will be optimized. This *random*-optimization is implemented in the program `mems3.rnd` and it's source code is included in appendix 3, but since the functions `H` and `V` are the same as in `mems3.p`, only the body of the program is listed. The resulting optimal values of the parameters were nevertheless the same as the ones we found earlier in table II.
The third maximization algorithm we implemented, the *full*-optimization, does not assume that there is only one maximum. It therefore runs each parameter through the whole [0,1] interval using the given accuracy and then sets it to the current optimal value. But also this approach, which is included in `mems3.ful`, appendix 4, gave the same results as the previous two algorithms.
To check whether the found results proved to be right, we also used the program Mathematica™ and found the same maximum achievable transmission rate as (20). So, assuming that the depth 3 coding strategy is, just as the K_2 strategy in [4], not "deep" enough to exceed Shannon's inner bound, we will proceed with the depth 4 coding strategy.

3.2.2 Depth 4 coding strategy

Following the same analogy as with the depth 3 coding strategy, we now extend this strategy with one more input. This means that the output and input sequences as defined in (9) and (10) become:

$$Y_{j,4} = (y_{j,2}, y_{j,3}, y_{j,4}, y_{j,5}),\ j = 1,2 \tag{21}$$

and

$$X_{j,4} = (x_{j,1}, x_{j,2}, \ldots, x_{j,4}, 1),\ j = 1,2 \tag{22}$$

respectively. The average mutual information after the first three input symbols remains the same as with the depth 3 coding strategy, but the average mutual information in the $j \rightarrow k$ direction, after user k has sent it's fourth input symbol $x_{k,4}$, becomes:

$$I(\theta_j; Y_{k,4} \mid \theta_k, Y_{k,3}) = P[x_{k,4} = 1 \wedge X_{k,3} \wedge Y_{k,3}]\ I(\theta_j; Y_{k,4} \mid x_{k,4} = 1, X_{k,3}, Y_{k,3}),\ j = 1,2,\ k = 1,2,\ j \neq k \tag{23}$$

where we used the fact that $I(\theta_j; Y_{k,4} \mid x_{k,4} = 0, X_{k,3}, Y_{k,3}) = 0$. This step is depicted in figure 7, in which we, because of clarity, indicated the parameters α_i only by their number i, $i \in \{1..27\}$. The more specific description of (23) can be found in the procedure `V` of the program `mems4.p`, of which the source code is listed in appendix 5. In this procedure `V` we have included the extra pair of numbers $\{4,b\}$, where the 4 indicates that this part is related to the fourth input symbol and where b refers to the number of the area which is depicted in figure 6.

After both users have sent their fourth input symbol, they will send their last input symbols $x_{1,5}$ and $x_{2,5}$, which are, according to (22), equal to one. This step, of which the average mutual

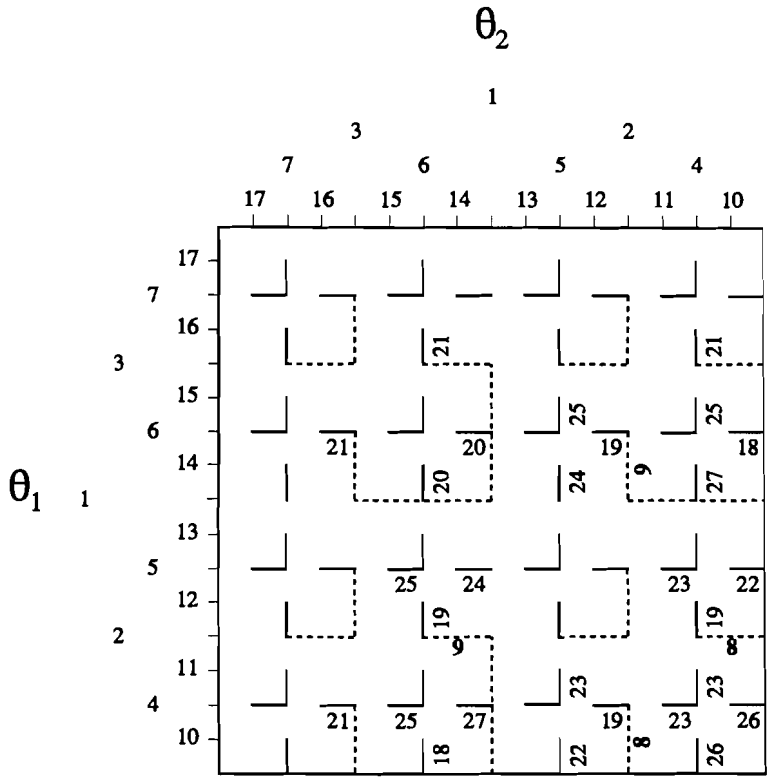

Figure 7: The unit square for the depth 4 coding strategy after 4 input symbols

information for user 2 is given by $I(\theta_1; Y_{2,5} \mid \theta_2, Y_{2,4})$ is depicted in figure 8 with the additional areas that have the same conditional average mutual information, which again can be found in appendix 5 indicated by the pair of numbers $\{5,b\}$, where $b \in \{1..27, 1'..27'\}$.

In the search for the maximum overall transmission rate, we again implemented the three optimizing methods as used with the depth 3 coding strategy. The *normal*-optimization, implemented in `mems4`, appendix 5, resulted in the following overall transmission rate:

$$R_1 = R_2 = 0.56526 \tag{24}$$

Compared with the parameters of the depth 3 coding strategy, this time the parameters as listed in table III are not always similar to each other, which could suggest that this time the feedback has been used with this strategy. But both the *random* and the *full*-optimization, listed in appendix 6 and 7 respectively, gave the same overall transmission rate as found in (24), but with different values for the parameters α_{10}, α_{12}, α_{14}, α_{16}, α_{22} and α_{24}. After inspection of the found

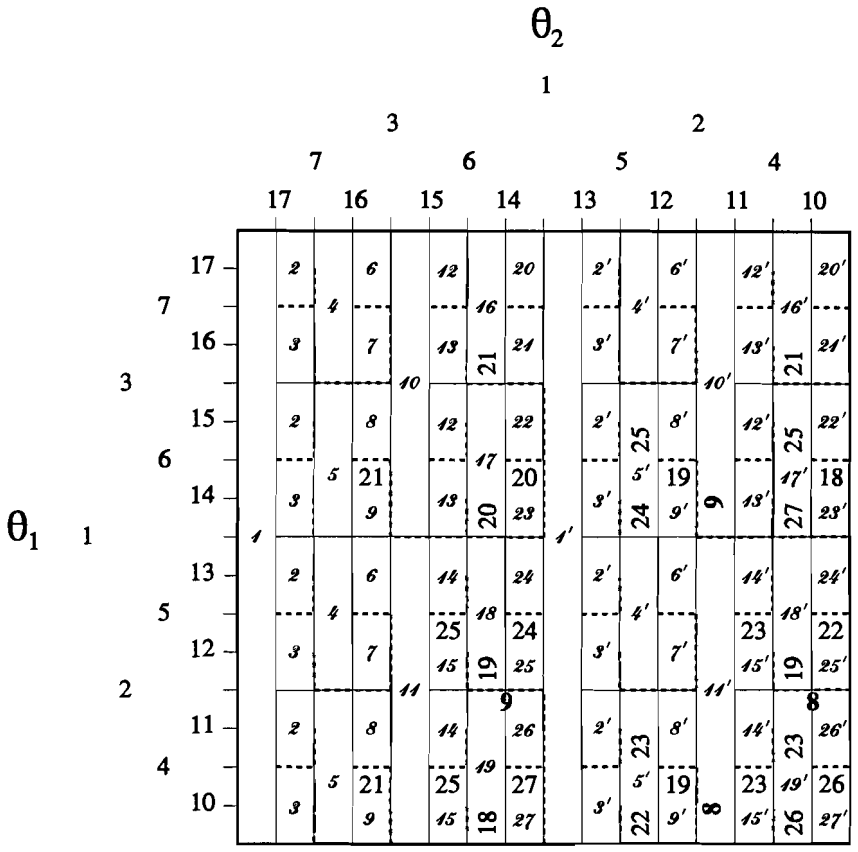

Figure 8: The unit square for the depth 4 coding strategy after 5 input symbols

Table III: Optimal parameters for the depth 4 coding strategy

1	0.50000	12	0.78798
2,3	0.72553	14	0.46469
4,5,6,7,8,9	0.71308	16	0.50305
11,13,15,17,18,19, 20,21,23,25,26,27	0.64551	22	0.80587
10	0.82634	24	0.48516

results, it turned out that these parameters are dependent in the following way:

$$\begin{aligned}(\alpha_{10}+\alpha_{14})/2 &= 0.64551\\ (\alpha_{12}+\alpha_{16})/2 &= 0.64551\\ (\alpha_{22}+\alpha_{24})/2 &= 0.64551\end{aligned} \tag{25}$$

This means that, just as with the depth 3 coding strategy, the result found in (24) doesn't make use of feedback. To examine whether this dependency would disappear if one would consider asymmetric rates, we rewrote `mems4` in such a way that the parameters of both users are independent. The source code `mems4.asm.p` can be found in 8, where the variables `a` and `b` correspond with the parameters of user 2 and `c` and `d` with the parameters of user 1. The average overall transmission rate is given by:

$$R=\frac{1}{10}\Big(I(\theta_1;Y_{2,2}\,|\,\theta_2)+I(\theta_1;Y_{2,3}\,|\,\theta_2,Y_{2,2})+I(\theta_1;Y_{2,4}\,|\,\theta_2,Y_{2,3})+I(\theta_1;Y_{2,5}\,|\,\theta_2,Y_{2,4})+ I(\theta_2;Y_{1,2}\,|\,\theta_1)+I(\theta_2;Y_{1,3}\,|\,\theta_1,Y_{1,2})+I(\theta_2;Y_{1,4}\,|\,\theta_1,Y_{1,3})+I(\theta_2;Y_{1,5}\,|\,\theta_1,Y_{1,4})\Big) \tag{26}$$

To determine the conditional average mutual information for area j after user i has sent it's kth input symbol, one must look in the source code of `mems4.asm` for the triplet $\{i,j,k\}$.

Although the resulting optimal parameters `c` were not equal to `a`, the same dependency as stated in (25) holds for the parameters of both users, leading to the similar overall transmission rate in both directions as found in (24).
This time it was not possible to check the results with Mathematica™, because the determination of the average mutual information had become to complex. We therefore continued with the depth 5 coding strategy.

3.2.3 Depth 5 coding strategy

The last simultaneous coding strategy we implemented was of depth 5. The output and input sequences become:

$$Y_{j,5}=(y_{j,2},y_{j,3},\ldots,y_{j,6}),\ j=1,2 \tag{27}$$

and

$$X_{j,5}=(x_{j,1},x_{j,2},\ldots,x_{j,5},1),\ j=1,2 \tag{28}$$

respectively. When both users have sent their fifth input symbol, the unit square is divided into subrectangles as depicted in figure 9. In appendix 9 one can look up for a particular area the corresponding conditional average mutual information, where the area numbers correspond with the numbers used in figure 8. After user 2 has sent it's sixth and last input symbol $x_{2,6}=1$ to clean up the channel, the resulting unit square consists of 162 areas with each a different conditional average mutual information. This can be seen in figure 10.
To optimize the overall transmission rate, which is given by:

$$R_j=\frac{1}{6}\Big(I(\theta_j;Y_{k,2}\,|\,\theta_k)+I(\theta_j;Y_{k,3}\,|\,\theta_k,Y_{k,2})+I(\theta_j;Y_{k,4}\,|\,\theta_k,Y_{k,3})+ I(\theta_j;Y_{k,5}\,|\,\theta_k,Y_{k,4})+I(\theta_j;Y_{k,6}\,|\,\theta_k,Y_{k,5})\Big),\ j=1,2,\ k=1,2,\ j\neq k \tag{29}$$

we need to optimize 81 parameters. Because of this large number we only implemented the

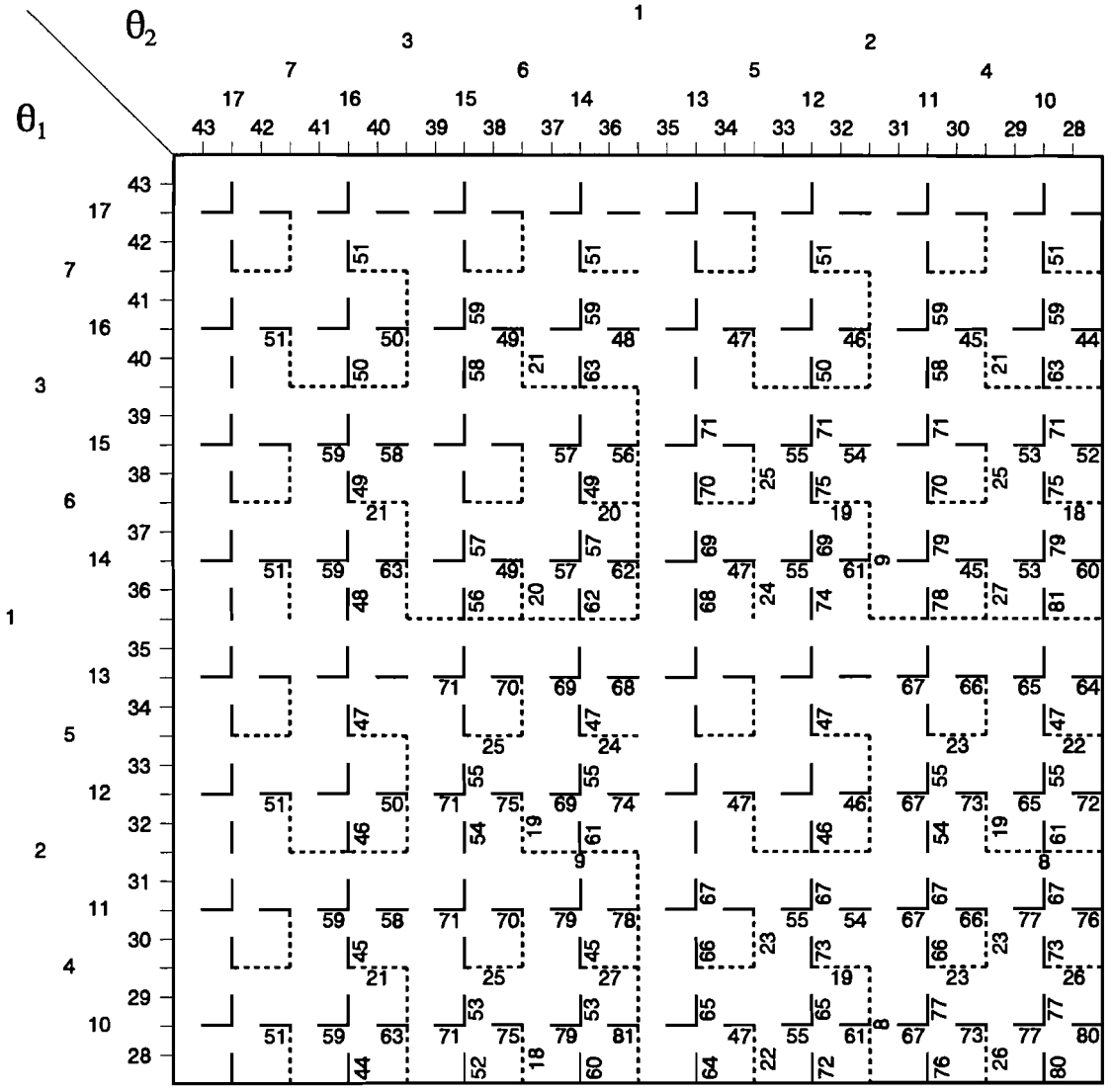

Figure 9: The unit square for the depth 5 coding strategy after 5 input symbols

normal and the *random*-optimization, listed in appendix 9 and 10 respectively. And just as with the depth 4 coding strategy we found for varying parameters the same maximum overall transmission rate, which indicates that, just as in (25), some parameters are dependent. The maximum overall transmission rate the two optimizing algorithms found was:

$$R_1 = R_2 = 0.57399 \tag{30}$$

and still makes no use of feedback.

Since the simultaneous depth 6 coding strategy will, according to (18), contain 243 parameters and 486 areas, we do not implement this coding strategy. Instead of that we will first investigate how fast the overall transmission rate would approach Shannon's inner bound without using feedback.

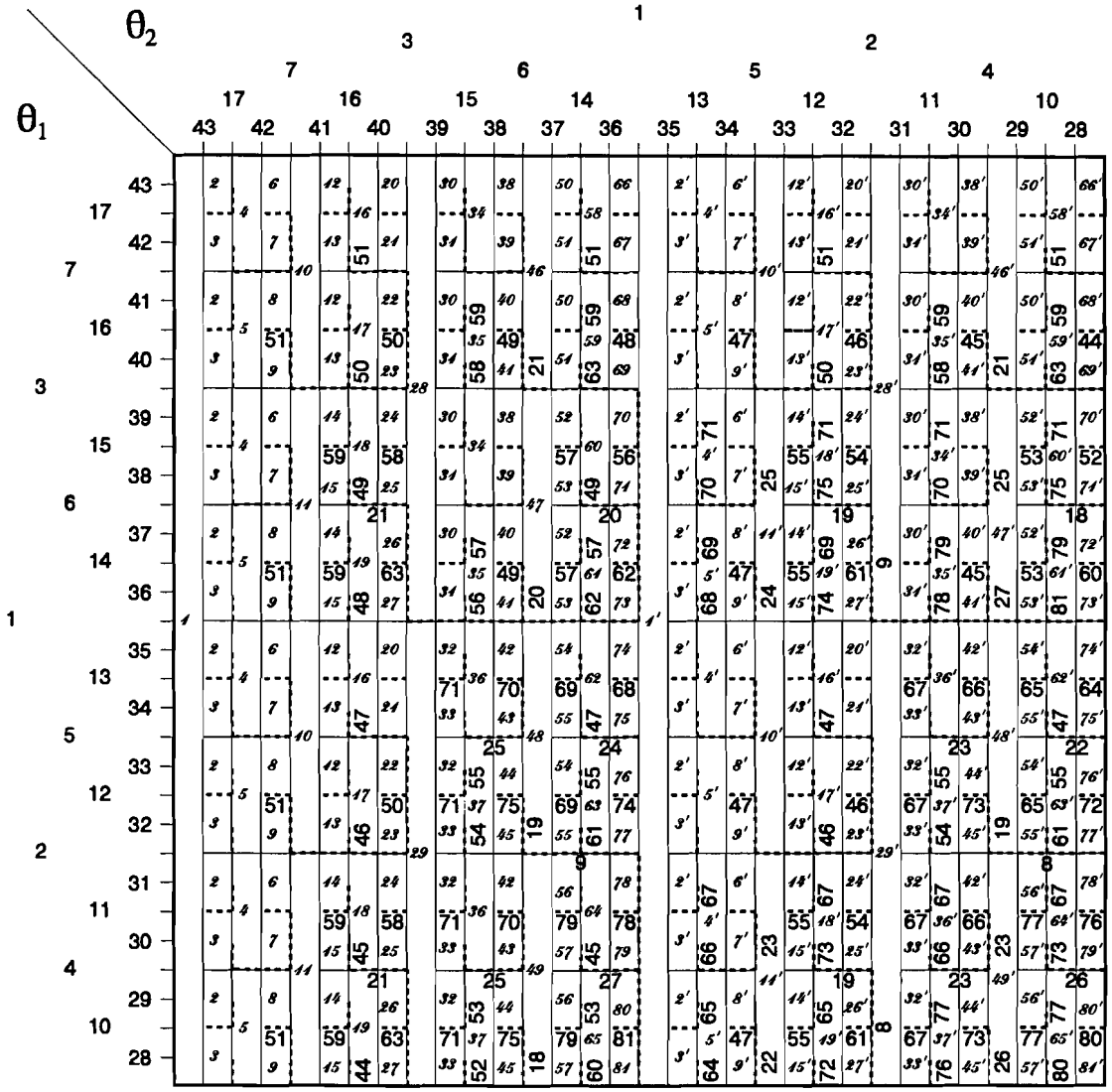

Figure 10: The unit square for the depth 5 coding strategy after 6 input symbols

3.2.4 Coding without feedback

The results we found thus far could not benefit from the available feedback. In figure 11 we have plotted the overall transmission rate for increasing depth of the coding strategies together with Shannon's inner bound, which is equal to 0.61695.

To verify how fast coding without using feedback would approach this inner bound, we wrote the program `memi`, which could systematically compute the overall transmission rate for unlimited length of codewords, but with the limitations of computer power.

The source code `memi.p` can be found in appendix 11 and the results for codewords of length up to 17 are listed in table IV. The results are also plotted in figure 12, and, as one can see, they slowly tend to the inner bound. This can be an indication that the depth of the coding strategies

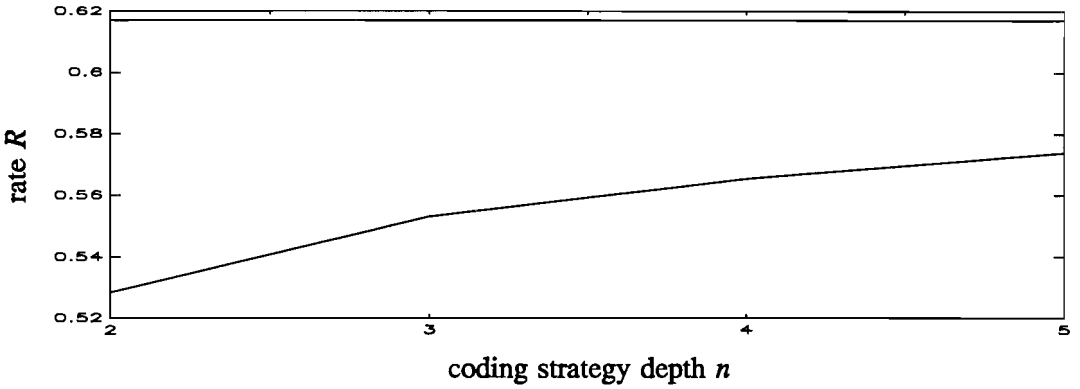

Figure 11: Plot of the overall transmission rate for depth n coding strategy

Table IV: The overall transmission rate for codewords of length n

2	0.52832	6	0.58010	10	0.59350	14	0.59976
3	0.55298	7	0.58471	11	0.59546	15	0.60083
4	0.56526	8	0.58829	12	0.59711	16	0.60178
5	0.57399	9	0.59116	13	0.59853	17	0.60262

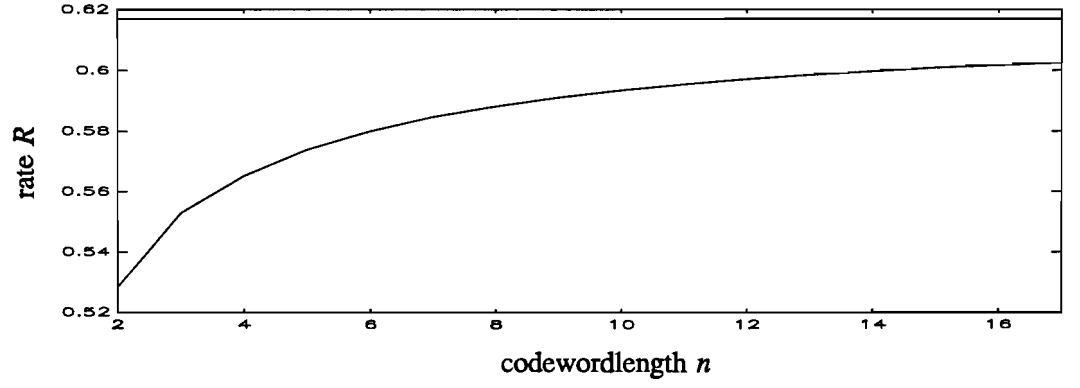

Figure 12: Plot of the overall transmission rate for codewords of length n

used so far, are not deep enough. We therefore started looking for a different coding strategy that could be implemented for greater depth. This so called sequential coding strategy is discussed in the following chapter.

4 The sequential coding strategy

4.1 Strategy description

The simultaneous coding strategies as described in the previous chapter, have used the coding strategies for the normal BMC as a starting-point, by displaying all the inputs and outputs of both users simultaneously in the unit square. In order to reduce the number of parameters, we need to look at the Binary Multiplying Delay Channel against the time axis. One can see in figure 13, that the communication between the two users consists of two independent 'zigzags', representing two *sequential* communication paths.

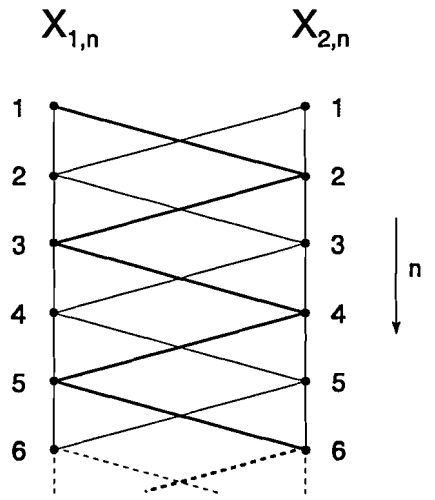

Figure 13: The BMDC plotted against the time axis

This means that when we want to determine the maximum overall transmission rate, we only have to compute the average mutual information for one zigzag. Without loss of generality we will always investigate the zigzag that starts at user 1. In figure 13 we have highlighted this zigzag.

Just as the simultaneous coding strategy, will this so called sequential coding strategy use the first input symbol of user 1 to initialize the channel. Depending on whether the depth n of the coding strategy is even or odd, user 1 or 2 respectively, will send a last input symbol equal to one to clean up the channel.

For even depth n, this leads to the following definitions for the output and input sequences:

$$\begin{aligned} Y_{1,n} &= (y_{1,3}, y_{1,5}, \cdots, y_{1,n+1}) \\ Y_{2,n} &= (y_{2,2}, y_{2,4}, \cdots, y_{2,n}) \end{aligned} \tag{31}$$

and

$$X_{1,n} = (x_{1,1}, x_{1,3}, \cdots, x_{1,n-1}, 1)$$
$$X_{2,n} = (x_{2,2}, x_{2,4}, \cdots, x_{2,n}) \tag{32}$$

For odd n, the output and input sequences are defined by:

$$Y_{1,n} = (y_{1,3}, y_{1,5}, \cdots, y_{1,n})$$
$$Y_{2,n} = (y_{2,2}, y_{2,4}, \cdots, y_{2,n+1}) \tag{33}$$

and

$$X_{1,n} = (x_{1,1}, x_{1,3}, \cdots, x_{1,n})$$
$$X_{2,n} = (x_{2,2}, x_{2,4}, \cdots, x_{2,n-1}, 1) \tag{34}$$

respectively.

Since the description of the sequential coding strategy shows many similarities with the simultaneous coding strategy, we will not describe the sequential coding strategy down to the smallest detail. We will therefore assume that the reader is well-known with the description of the simultaneous coding strategy as described in chapter 3.
We start the description with figure 14, where we display the unit square when $n = 3$, so user 1 has sent the input symbols $x_{1,1}$ and $x_{1,3}$, and user 2 has sent $x_{2,2}$.

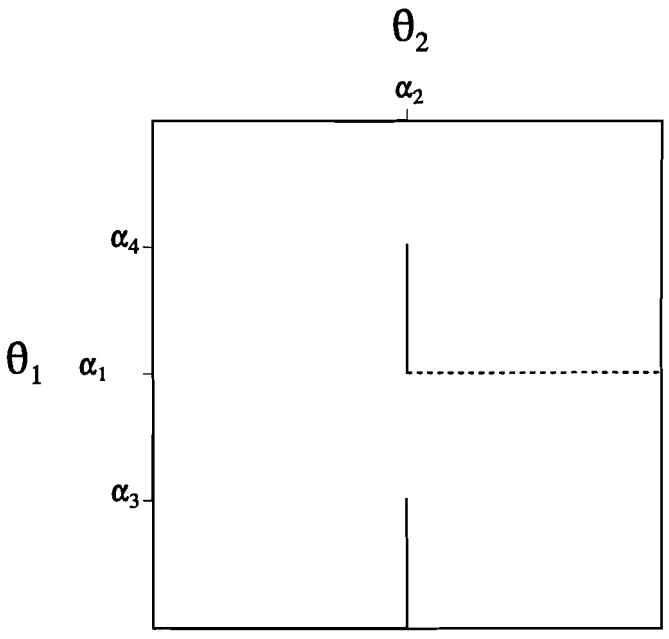

Figure 14: The unit square after three input symbols

As one can see in this figure, no user could use feedback so far, in contrast to the simultaneous coding strategy. Since $x_{1,1}$ is used to initialize the channel, no information is given to user 1.

After user 2 has sent $x_{2,2}$, the average mutual information in the $1 \rightarrow 2$ is given by:

$$I(\theta_1; Y_{2,2} \mid \theta_2) = P[x_{2,2}=1]\, I(\theta_1; Y_{2,2} \mid x_{2,2}=1) = \alpha_2 h(\alpha_1) \tag{35}$$

where we again used the fact that $I(\theta_1; Y_{2,2} \mid x_{2,2}=0) = 0$. After the third input symbol, when user 1 has sent $x_{1,3}$, the average mutual information in the $2 \rightarrow 1$ direction becomes:

$$I(\theta_2; Y_{1,3} \mid \theta_1) = P[x_{1,3}=1 \wedge X_{1,1}]\, I(\theta_2; Y_{1,3} \mid x_{1,3}=1, X_{1,1}) = (\alpha_1\alpha_3+\beta_1\alpha_4)h(\alpha_2) \tag{36}$$

Just as with the simultaneous coding strategy, one must repeat the last two steps, before the last input symbol is sent by user 1 or 2, depending on whether n is even or odd. In figure 15 is this step depicted for the depth 3 coding strategy.

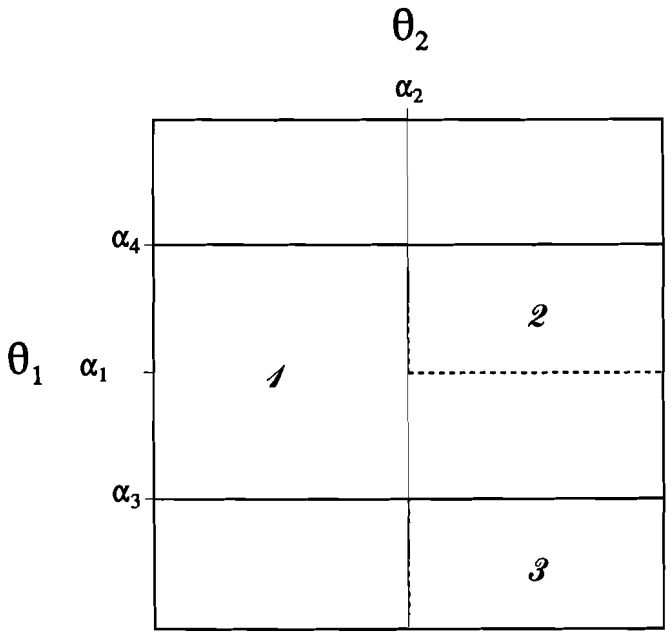

Figure 15: The unit square for the depth 3 coding strategy after 4 input symbols

For each indicated area, table V lists the corresponding conditional average mutual information and the probability for that area, so one can determine $I(\theta_1; Y_{2,4} \mid \theta_2, Y_{2,2})$.

Table V: The average mutual information for each area in figure 15

area	$I(\theta_1; Y_{2,4} \mid X_{2,2}, Y_{2,2})$	$P[X_{2,2} \wedge Y_{2,2}]$
1	$h(\alpha_1\alpha_3 + \beta_1\beta_4)$	β_2
2	$h(\alpha_4)$	$\alpha_2\beta_1$
3	$h(\alpha_3)$	$\alpha_2\alpha_1$

The overall information rate for the sequential depth 3 coding strategy is given by:

$$R = \frac{I(\theta_1; Y_{2,2} \mid \theta_2) + I(\theta_2; Y_{1,3} \mid \theta_1) + I(\theta_1; Y_{2,4} \mid \theta_2, Y_{2,2})}{4} \tag{37}$$

In order to cancel out the loss one makes by sending the initializing first input symbol and the finishing last input symbol, we also calculated the limiting overall information rate for a particular depth of a coding strategy. We did this by coupling the last input symbol with the first input symbol, thus creating a loop, where the last input symbol of a coding strategy is also used as the first input symbol of the next coding strategy. In figure 16 we have depicted this loop for the depth 3 coding strategy.

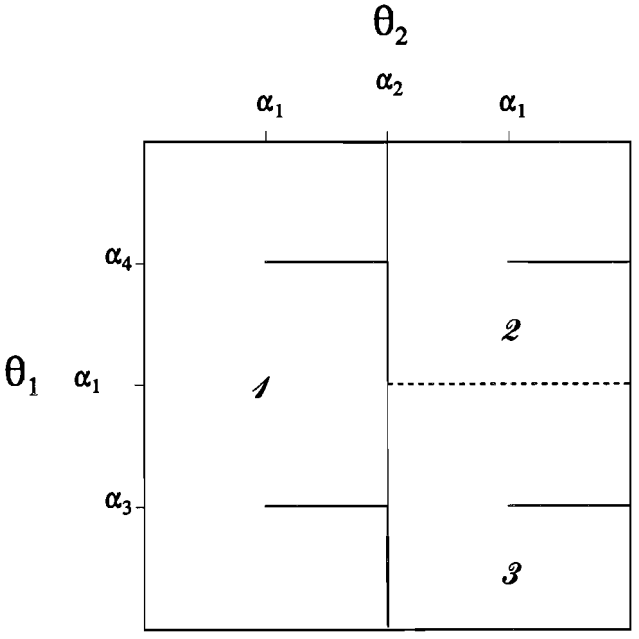

Figure 16: The unit square after coupling $x_{2,4}$ with $x_{2,1}$

This means that instead of setting $P[x_{2,4} \mid X_{2,2}, Y_{2,2}] = 1$, we now use $P[x_{2,4} \mid X_{2,2}, Y_{2,2}] = \alpha_1$. The average mutual information after this last input symbol then becomes:

$$\begin{aligned} I_{\text{lim}}(\theta_1; Y_{2,4} \mid \theta_2, Y_{2,2}) = {} & \alpha_1\ \beta_2\ h(\alpha_1\alpha_3 + \beta_1\alpha_4) + \\ & \alpha_1\ \alpha_2\beta_1\ h(\alpha_4) + \\ & \alpha_1\ \alpha_2\alpha_1\ h(\alpha_3) \end{aligned} \tag{38}$$

The limiting overall transmission rate is therefore defined by:

$$R_{\text{lim}} = \frac{I(\theta_1; Y_{2,2} \mid \theta_2) + I(\theta_2; Y_{1,3} \mid \theta_1) + I_{\text{lim}}(\theta_1; Y_{2,4} \mid \theta_2, Y_{2,2})}{3} \tag{39}$$

To determine the number of parameters we need for optimizing a depth n coding strategy, we must make a difference for n is even or odd. When n is odd, user 1 has $3^{\left(\frac{n-1}{2}\right)}$ parameters and the number of parameters for user 2 are $\frac{1}{2}\cdot 3^{\left(\frac{n-1}{2}\right)}-\frac{1}{2}$. For even n, the number of parameters for user 1 equals $3^{\left(\frac{n-2}{2}\right)}$, and for user 2 we find $\frac{1}{2}\cdot 3^{\left(\frac{n}{2}\right)}-\frac{1}{2}$ parameters. Therefore the total number of parameters p_n for coding strategy depth n is given by:

$$p_n=\begin{cases}\frac{1}{2}\cdot 3^{\left(\frac{n+1}{2}\right)}-\frac{1}{2}, & n=odd\\ \\ \frac{5}{6}\cdot 3^{\left(\frac{n}{2}\right)}-\frac{1}{2}, & n=even\end{cases} \tag{40}$$

which is almost a quadratic decrease compared with the number of parameters we found for the simultaneous coding strategy in (18).
The next paragraph will present the results we found for the depth 4, 6 and 7 strategies.

4.2 Results

4.2.1 Depth 4 coding strategy

To check whether the sequential coding strategy would give the same results as the simultaneous coding strategy, we first implemented the depth 4 coding strategy, which is already done for the simultaneous coding strategy in paragraph 3.2.2.
Since this is an even depth coding strategy, the output and input sequences for the zigzag that starts at user 1, are defined by:

$$\begin{aligned}Y_{1,4}&=(y_{1,3},y_{1,5})\\ Y_{2,4}&=(y_{2,2},y_{2,4})\end{aligned} \tag{41}$$

and

$$\begin{aligned}X_{1,4}&=(x_{1,1},x_{1,3},1)\\ X_{2,4}&=(x_{2,2},x_{2,4})\end{aligned} \tag{42}$$

respectively. The total number of parameters is, according to (40), equal to 7, which can be seen in figure 17. This figure shows the unit square after user 2 has sent his last input symbol $x_{2,4}$. This is the first time feedback is realized when $y_{2,2}=1$, by replacing the parameter α_5 by α_7. The corresponding average mutual information is given by:

$$I(\theta_1;Y_{2,4}\,|\,\theta_2,Y_{2,2}) = P[x_{2,4}=1\wedge X_{2,2}\wedge Y_{2,2}]\; I(\theta_1;Y_{2,4}\,|\,x_{2,4}=1,X_{2,2},Y_{2,2}) \tag{43}$$

and can be found in appendix 12, where we have listed the source code of the program `memseq4`.
The finishing input symbol $x_{1,5}$ from user 1 leads to the unit square as depicted in figure 18, which contains the additional areas that have the same conditional average mutual information.

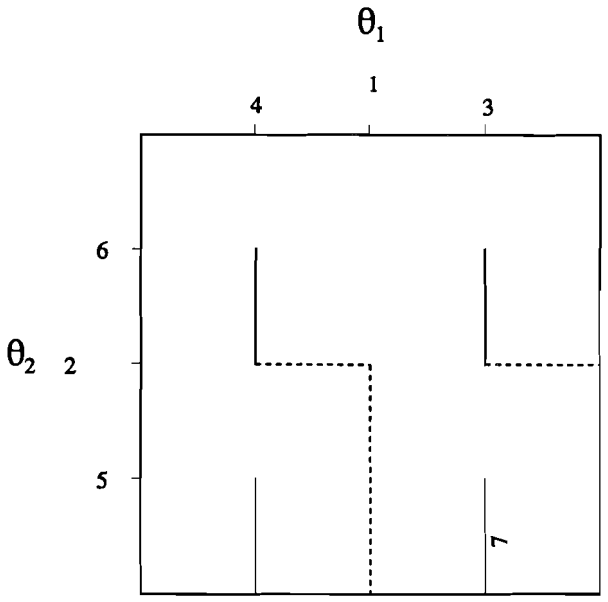

Figure 17: The unit square for the depth 4 coding strategy after 4 input symbols

The average mutual information $I(\theta_2; Y_{1,5} \mid \theta_1, Y_{1,3})$ can be looked up in appendix 12.

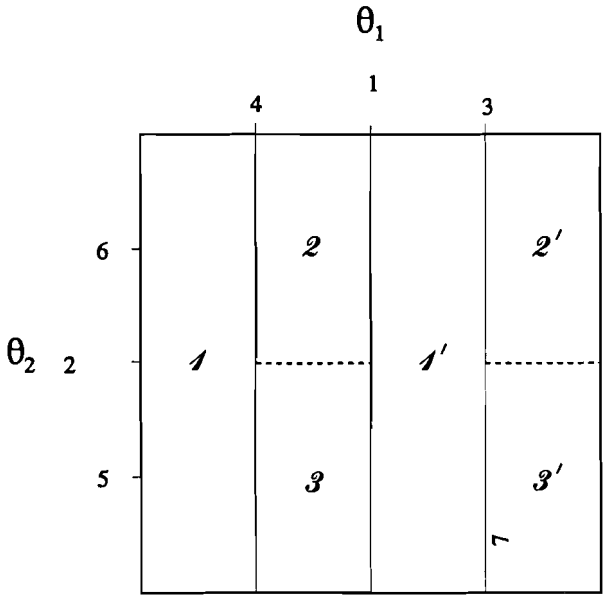

Figure 18: The unit square for the depth 4 coding strategy after user 1 has sent $x_{1,5}$

As far as the results are concerned, we found with `memseq4`, which uses the so called *normal*-optimization, the same overall information rate as with the simultaneous depth 4 coding strategy:

$$R = 0.56526 \tag{44}$$

The accompanying optimal parameters are listed in table VI, and as one can see, the dependency we found with the simultaneous coding strategy does not apply for the sequential depth 4 coding strategy.

Table VI: Optimal parameters for the depth 4 coding strategy

α_1	0.50000	α_3,α_4	0.71315
α_2	0.72548	$\alpha_5,\alpha_6,\alpha_7$	0.64551

To calculate the limiting overall information rate, we changed the program `memseq4` into the program `memloop4`, of which the source code can be found in appendix 13. In figure 19 we show the resulting unit square when input symbol $x_{1,5}$ is coupled with $x_{1,1}$.

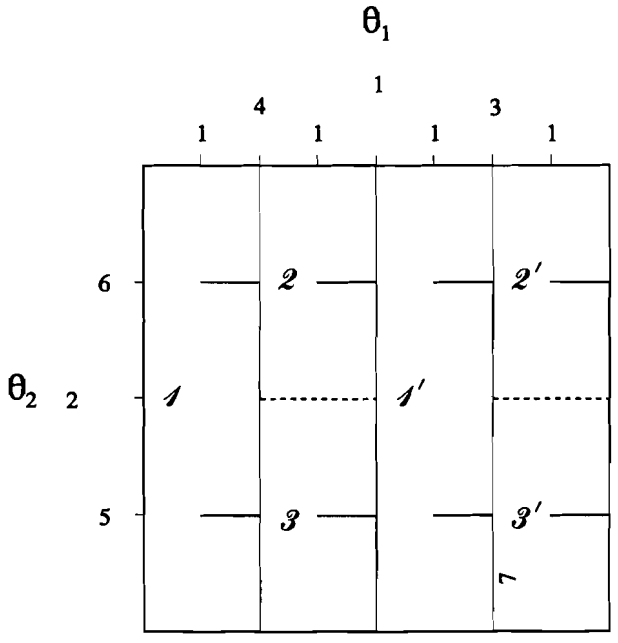

Figure 19: The unit square after coupling $x_{1,5}$ with $x_{1,1}$

The limiting overall information rate this program found is:

$$R_{\text{lim}} = 0.61695 \tag{45}$$

which is equal to Shannon's inner bound. This is of no surprise, since the parameters in table VI already indicated that no feedback had been used. For the limiting case, the resulting optimal parameters α_i, $i = 1..7$, were all equal to 0.70351, which is the same optimum Schalkwijk found in [2] for the parameter α of the *i*(nnerbound)-situation.

4.2.2 Depth 6 coding strategy

In comparison with the 243 parameters the simultaneous depth 6 coding strategy would consist of, we now can implement the sequential depth 6 coding strategy with only 22 parameters. This can be seen in figure 20, where user 2 has sent his last input symbol $x_{2,6}$. To determine the

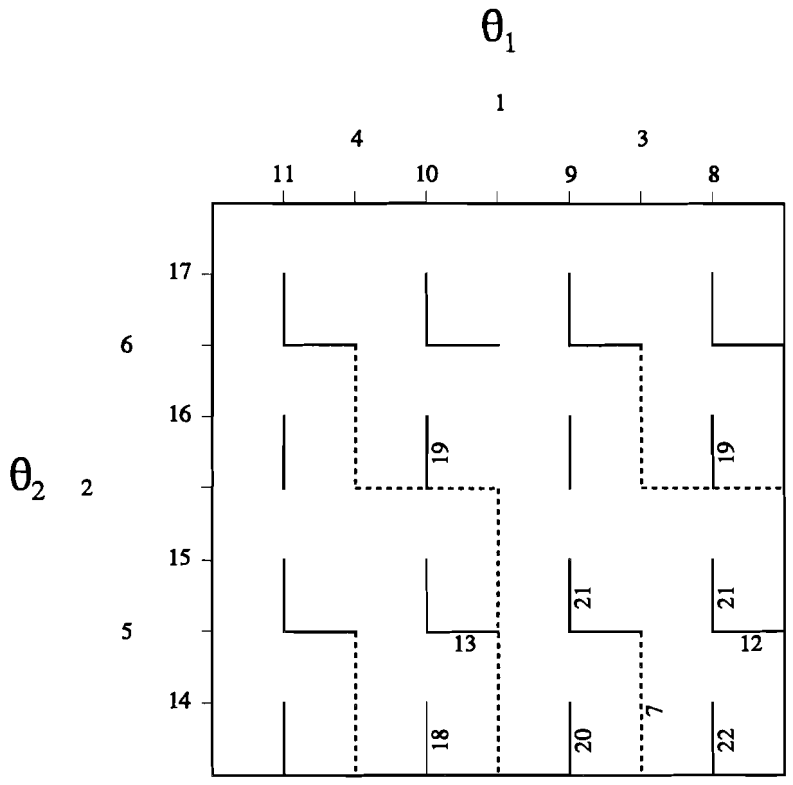

Figure 20: The unit square for the depth 6 coding strategy after input symbol $x_{2,6}$

average mutual information $I(\theta_1; Y_{2,6} \mid \theta_2, Y_{2,4})$, we have indicated in appendix 14 which part of the procedure V belongs to this input symbol. To make this determination more easier, we also placed the number of the concerning parameter between brackets.

The finishing input symbol $x_{1,7}$ of user 1, which leads to $I(\theta_2; Y_{1,7} \mid \theta_1, Y_{1,5})$, divides the unit square into the subrectangles as depicted in figure 21. To prevent confusion with the above-mentioned numbers which represent a particular parameter, we have placed in appendix 14 an extra apostrophe in front of these numbers. So the average mutual information for area *5'* is marked as `{'5'}`.

The overall transmission rate is defined as:

$$R = \frac{1}{7}\Big(I(\theta_1; Y_{2,2} \mid \theta_2) + I(\theta_2; Y_{1,3} \mid \theta_1) + I(\theta_1; Y_{2,4} \mid \theta_2, Y_{2,2}) + I(\theta_2; Y_{1,5} \mid \theta_1, Y_{1,3}) + I(\theta_1; Y_{2,6} \mid \theta_2, Y_{2,4}) + I(\theta_2; Y_{1,7} \mid \theta_1, Y_{1,5})\Big) \tag{46}$$

Using the *full*-optimization, implemented with the program `memseq6`, we could not exceed the

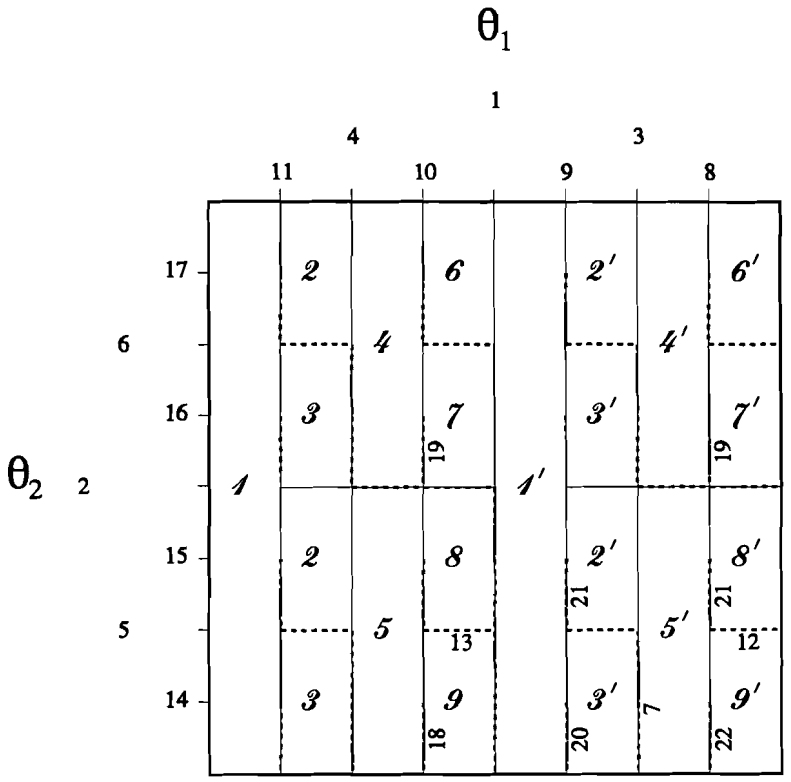

Figure 21: The unit square for the depth 6 coding strategy after 7 input symbols

overall transmission rate we found earlier for the case where we did not have feedback:

$$R = 0.58010 \tag{47}$$

which is the same as listed in table IV. Using different initial values for the 22 parameters, yielded the same overall transmission rate for varying values of the parameters. This indicates that, just as with the simultaneous coding strategy, some parameters are dependent.

To calculate the limiting overall transmission rate, of which the unit square is depicted in figure 22, we first implemented the *full*-optimization with the adjusted program `memloop6`, of which the source code can be found in appendix 15. Since (47) already indicated that no feedback has been used, it is of no surprise that the resulting limiting overall transmission rate equals Shannon's inner bound.

In our disbelief that the optimum can not exceed the inner bound, we implemented three different methods to verify this optimum. The first method is accomplished by the program `memeps6`, of which the program body can be found in appendix 16. At first it initializes all parameters by setting them to 0.70351, which gives result to Shannon's inner bound. After that, it starts to fluctuate the parameters with a given constant `eps`, in an attempt to find a maximum that is located close to the optimum yielding the inner bound.

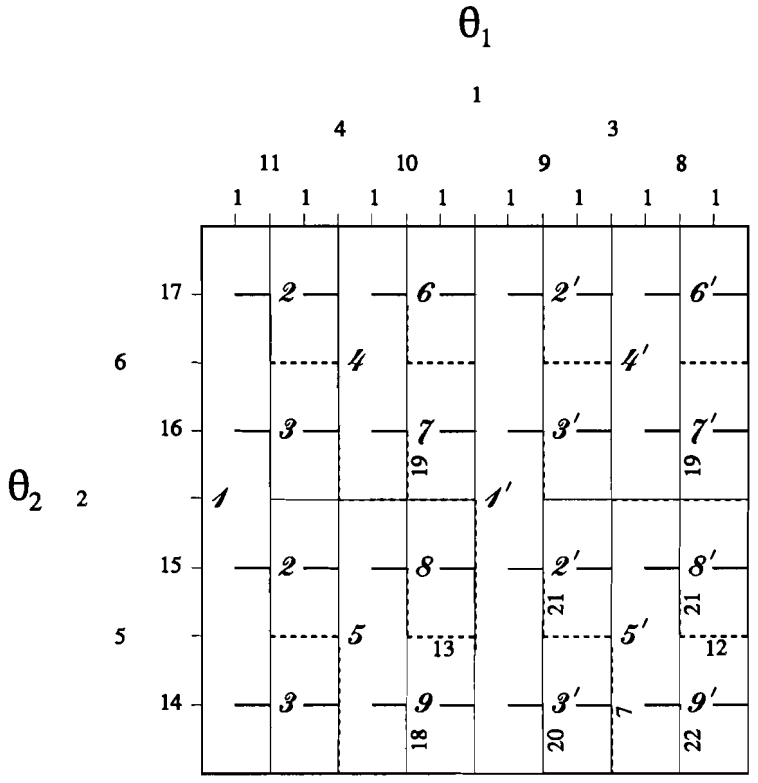

Figure 22: The unit square for the depth 6 coding strategy after $x_{1,7}$ has been coupled with $x_{1,1}$

The second method tries to reduce the effect of optimizing the parameters one by one, by linking particular parameters with each other. The program `memlink6`, appendix 17, has performed this for different combinations of linked parameters.

The last method combines the *full*-optimization with the *random*-optimization, so that the order by which the parameters are optimized is randomized. The program body of this program, `memloop6.rnd`, is listed in appendix 18.

Nevertheless provided all three methods no better optimum than Shannon's inner bound, which made us decide to implement also the depth 7 coding strategy.

4.2.3 Depth 7 coding strategy

The last sequential coding strategy we implemented is of depth 7. Describing the strategy that starts at user 1, the output and input sequences are defined as follows:

$$\begin{aligned} Y_{1,7} &= (y_{1,3}, y_{1,5}, y_{1,7}) \\ Y_{2,7} &= (y_{2,2}, y_{2,4}, y_{2,6}, y_{2,8}) \end{aligned} \tag{48}$$

and

$$\begin{aligned} X_{1,7} &= (x_{1,1}, x_{1,3}, x_{1,5}, x_{1,7}) \\ X_{2,7} &= (x_{2,2}, x_{2,4}, x_{2,6}, 1) \end{aligned} \tag{49}$$

respectively. The subdivision of the unit square after user 1 has sent input symbol $x_{1,7}$ is depicted in figure 23, and, as one can see in this figure, this step leads to 40 parameters. The average

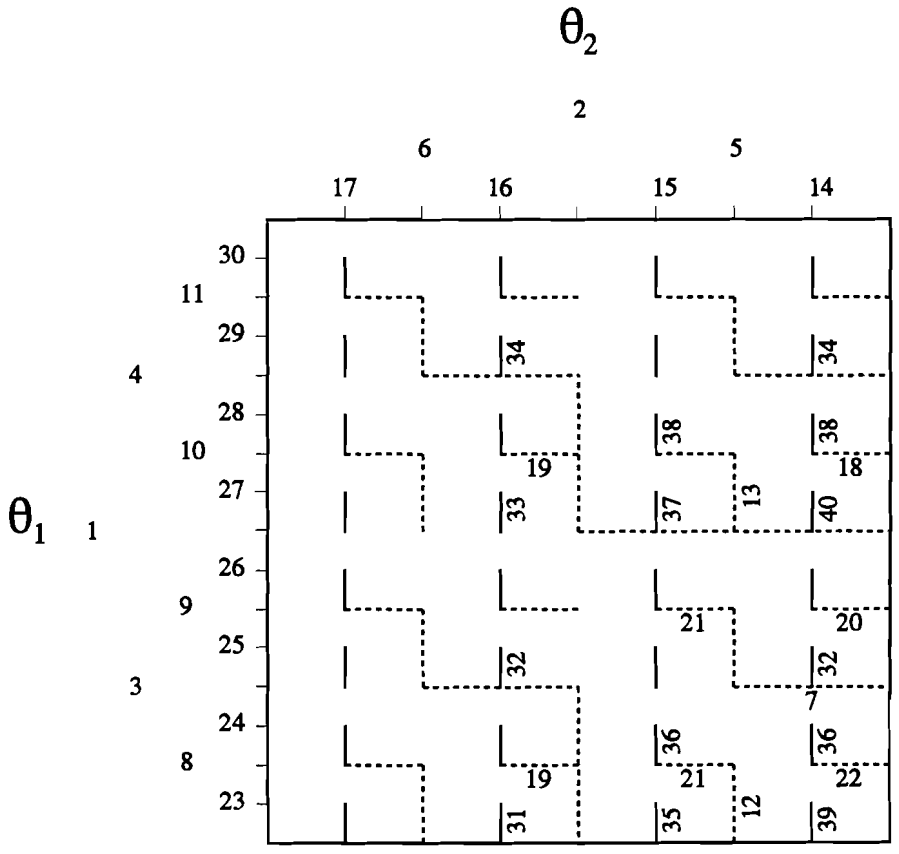

Figure 23: Subdivision of the unit square for the depth 7 coding strategy after user 1 has sent $x_{1,7}$

mutual information which is transferred in the 2 → 1 direction is:

$$I(\theta_2; Y_{1,7} \mid \theta_1, Y_{1,5}) = P[x_{1,7}=1 \wedge X_{1,5} \wedge Y_{1,5}]\, I(\theta_2; Y_{1,7} \mid x_{1,7}=1, X_{1,5}, Y_{1,5}) \tag{50}$$

In appendix 19 we have listed the source code of the program memseq7, so one can determine the average mutual information that corresponds with parameter α_{23} through α_{40}, indicated by {23} and {40} respectively.

When the last input symbol of user 2, $x_{2,8}$, is not linked with the first input symbol $x_{2,1}$, the unit square is divided into the 27 areas as shown in figure 24. For each separate area, appendix 19 gives the corresponding average mutual information, where the number of the areas are marked by {'1} through {'27}. The resulting average mutual information $I(\theta_1; Y_{2,8} \mid \theta_2, Y_{1,6})$ leads to

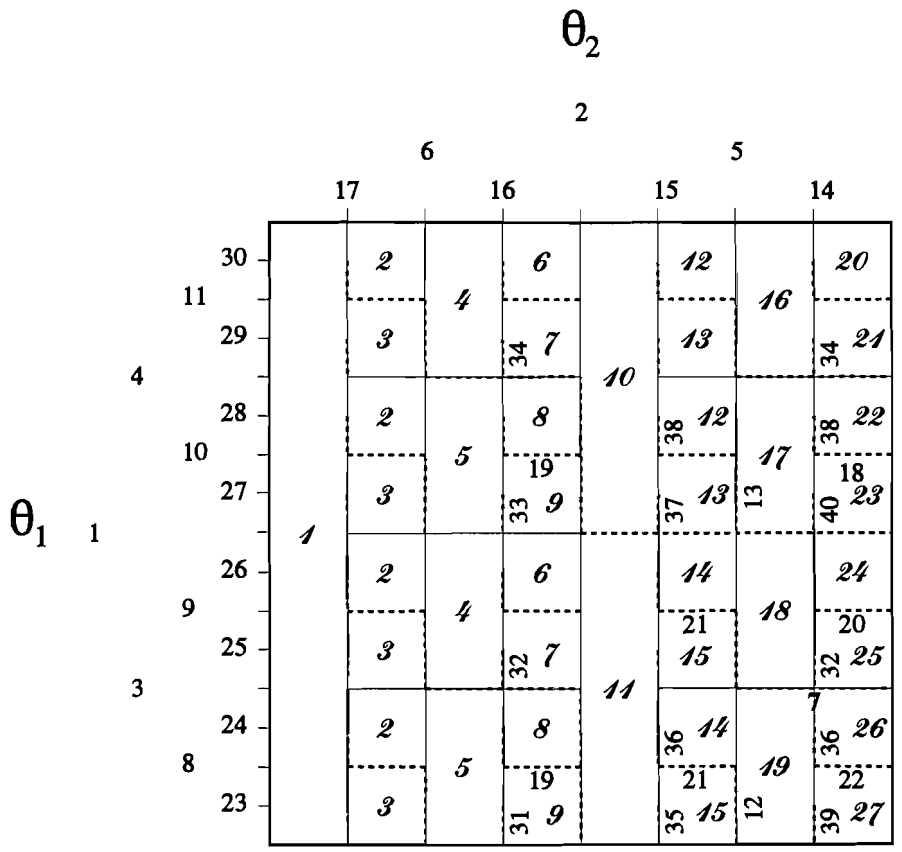

Figure 24: Subdivision of the unit square for the depth 7 coding strategy after user 2 has sent input symbol $x_{2,8} = 1$

the following overall transmission rate:

$$R = \frac{1}{8}\Big(I(\theta_1; Y_{2,2} \mid \theta_2) + I(\theta_2; Y_{1,3} \mid \theta_1) + I(\theta_1; Y_{2,4} \mid \theta_2, Y_{2,2}) + I(\theta_2; Y_{1,5} \mid \theta_1, Y_{1,3}) + I(\theta_1; Y_{2,6} \mid \theta_2, Y_{2,4}) + I(\theta_2; Y_{1,7} \mid \theta_1, Y_{1,5}) + I(\theta_1; Y_{2,8} \mid \theta_2, Y_{2,6})\Big) \tag{51}$$

In case $x_{2,8}$ is linked with $x_{2,1}$, the unit square is subdivided into the areas as shown in figure 25. The program memloop7, of which the source code can be found in appendix 20, contains the procedure V to calculate the corresponding limiting average mutual information $I_{\lim}(\theta_1; Y_{2,8} \mid \theta_2, Y_{1,6})$. The limiting overall transmission rate is therefore defined by:

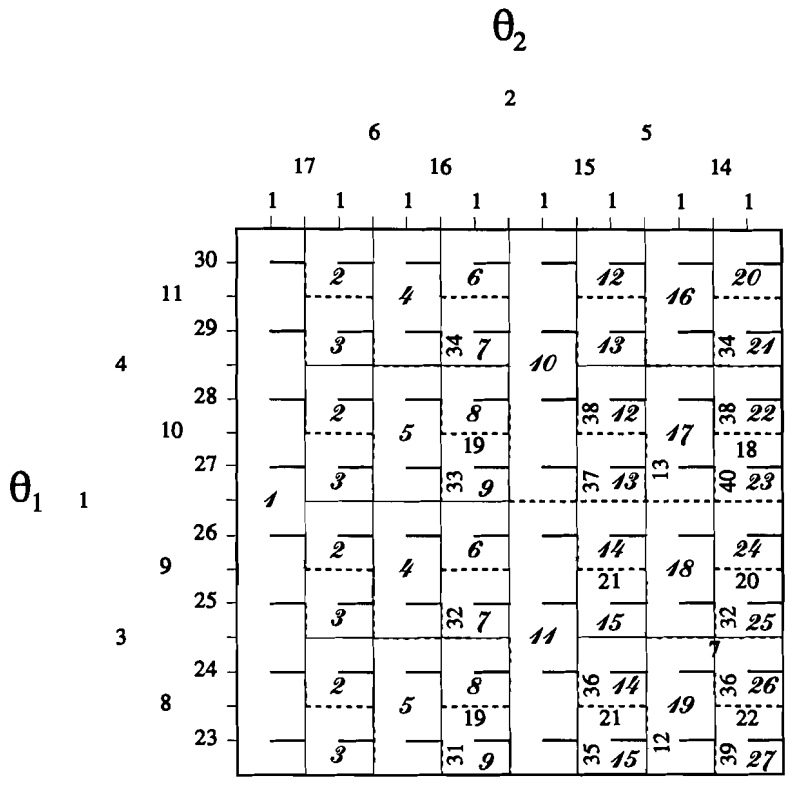

Figure 25: The unit square for the depth 7 coding strategy after linking $x_{2,8}$ with $x_{2,1}$

$$R_{\lim} = \frac{1}{7}\Big(I(\theta_1; Y_{2,2} \mid \theta_2) + I(\theta_2; Y_{1,3} \mid \theta_1) + I(\theta_1; Y_{2,4} \mid \theta_2, Y_{2,2}) + I(\theta_2; Y_{1,5} \mid \theta_1, Y_{1,3}) + I(\theta_1; Y_{2,6} \mid \theta_2, Y_{2,4}) + I(\theta_2; Y_{1,7} \mid \theta_1, Y_{1,5}) + I_{\lim}(\theta_1; Y_{2,8} \mid \theta_2, Y_{2,6})\Big) \tag{52}$$

Initially we only implemented the *full*-optimization with both `memseq7` and `memloop7`. The maximum overall transmission rate these two programs found were respectively:

$$R = 0.58471$$
$$R_{\lim} = 0.61695 \tag{53}$$

which again means that this maximum does not use the present feedback. But after extending the *full*-optimization with the *random*-optimization, we finally could use the feedback to increase the maximum overall transmission rate.

Without linking the last input symbol with the first input symbol, we implemented the program `memseq7.rnd`, of which the program body is listed in appendix 21. This program gave rise to the following maximum overall transmission rate:

$$R = 0.58470 \tag{54}$$

which is better than the rate listed in table IV, where we did not use feedback.
To compute the limiting overall transmission rate we wrote the program `memloop7.rnd`, listed in appendix 22. Since both the initial values of the parameters as well as the order of optimizing the parameters influenced the resulting maximum rate, we repeated the optimizing for numerous times, which finally resulted in the following limiting overall transmission rate:

$$R_{\text{lim}} = 0.61852 \tag{55}$$

which exceeds Shannon's inner bound. The corresponding values of the parameters, which can be found in table VII, indicate indeed that they differ when feedback has been implemented. Whether this maximum is optimal is a non-trivial problem, and lays beyond the scope of this thesis. But since we completed our main goal to prove that it is possible to construct a coding strategy for the BMDC that can exceed Shannon's inner bound, we will we now finish this thesis with the last chapter containing the conclusions and recommendations.

Table VII: The parameters leading to the maximum rate in (55)

1	0.707	11	0.710	21	0.675	31	0.754
2	0.695	12	0.706	22	0.706	32	0.705
3	0.707	13	0.706	23	0.367	33	0.409
4	0.706	14	0.857	24	0.955	34	0.704
5	0.699	15	0.675	25	1.000	35	0.589
6	0.563	16	0.352	26	0.696	36	0.705
7	0.699	17	0.896	27	0.526	37	0.589
8	0.563	18	0.706	28	0.000	38	0.705
9	0.710	19	0.852	29	1.000	39	0.702
10	0.689	20	0.857	30	0.696	40	0.702

5 Conclusions and recommendations

In this thesis we proved that after adding some delay to the Binary Multiplying Channel, in this way creating the Binary Multiplying Delay Channel, it is still possible to construct a continuous coding strategy for which the limiting overall transmission rate exceeds the Shannon inner bound. With a high degree of certainty we showed that the depth 7 coding strategy is the first coding strategy that could benefit from the present feedback, resulting to an overall transmission rate of 0.61852, whereas coding strategies of lesser depth could only achieve the Shannon inner bound.

Finding the maximum overall transmission rate is a non-trivial problem and was not our main goal, but we have several suggestions to make an improvement on the results we found thus far. One could for example apply higher accuracy with the computations we carried out or one could construct coding strategies of depth 8 or more, but these approaches cause a considerable increase of complexity. Another approach could be gaining more insight in the equations that lead to the resulting overall transmission rate, so one gets a better understanding where the global maximum is located, because up to now we have only used brute force techniques to determine the maximum overall transmission rate.

One can also improve the results, by extending the existing coding strategies with bootstrapping, in order to reach higher rates and with the possibility to exceed the inner bound for coding strategies of depth 6 or less.

References

[1] Bacchiani, M.A.U.
Realisation of a communication system using a Binary Multiplying Channel. Master's thesis, Eindhoven University of Technology, Dept. of Electrical Engineering, Research Group on Information and Communication Theory, March 1993.

[2] Schalkwijk, J.P.M.
The binary multiplying channel - a coding scheme that operates beyond the Shannon inner bound. *IEEE Transactions on Information Theory*, IT-28(1):107-110, Jan 1982.

[3] Shannon, C.E.
Two way communication channels. In: *Proceedings 4th Berkeley Symposium on Mathematics, Statistics and Probability*, 1961, Volume 1, pages 611-644, University of California Press. Reprinted in: *Key papers in the Development of Information Theory*, 1974, pages 339-372, Ed. by D. Slepian, New York: IEEE Press.

[4] Smeets, B.J.M
Shannon's and other strategies for the BMC. Master's thesis, Eindhoven University of Technology, Dept. of Electrical Engineering, Research Group on Information and Communication Theory, May 1983.

Appendix 1: Source code mems3.p

```
PROGRAM MEMS3 (input, output);

{A1..A9}

TYPE AR = ArraY [1..9] of Longreal;
VAR i,j,teller                  : Integer;
    old, oldprev, new, max      : Longreal;
    a                           : AR;
    out                         : Text;

FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a : AR) : Longreal;
  VAR i                   : Integer;
      arg, ent, snt, avi : Longreal;
      b                   : AR;

  BEGIN

  FOR i := 1 TO 9 do b[i] := 1 - a[i];

{input symbol 2}
  arg := a[1];
  ent := H(arg);
  avi := (b[1]*a[3] + a[1]*a[2]) * ent;

{input symbol 3}
  arg := a[1]*a[2] + b[1]*a[3];
  ent := H(arg);
  snt := (b[1]*b[3]*a[7] + a[1]*b[2]*a[5]) * ent;
  arg := a[3];
  ent := H(arg);
  snt := snt + b[1] * (b[1]*a[3]*a[6] + a[1]*a[2]*a[4]) * ent;
  arg := a[2];
  ent := H(arg);
  snt := snt + a[1] * (b[1]*a[3]*a[9] + a[1]*a[2]*a[8]) * ent;
  avi := avi + snt;

{input symbol 4}
{1}
  arg := a[1] * (a[2]*a[4] + b[2]*a[5]) + b[1] * (a[3]*a[6] + b[3]*a[7]);
  ent := H(arg);
  snt := b[1]*b[3]*b[7] * ent;
{2}
  arg := (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) / (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7] * (a[1]*b[2] + b[1]*b[3]) * ent;
{3}
  arg := (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]) / (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7] * (a[1]*a[2] + b[1]*a[3]) * ent;
{4}
  arg := a[3]*a[6] + b[3]*a[7];
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6] * b[1] * ent;
{5}
  arg := a[2]*a[4] + b[2]*a[5];
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9] * a[1] * ent;
{6}
  arg := a[7];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6] * b[1]*b[3] * ent;
{7}
  arg := a[6];
  ent := H(arg);
```

```
  snt := snt + b[1]*a[3]*a[6] * b[1]*a[3] * ent;
{8}
  arg := a[5];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9] * a[1]*b[2] * ent;
{9}
  arg := a[4];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9] * a[1]*a[2] * ent;

  avi := avi + snt;


{1'}
  arg := a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7]);
  ent := H(arg);
  snt := a[1]*b[2]*b[5] * ent;
{2'}
  arg := (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) / (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5] * (a[1]*b[2] + b[1]*b[3]) * ent;
{3'}
  arg := (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]) / (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5] * (a[1]*a[2] + b[1]*a[3]) * ent;
{4'}
  arg := a[3]*a[9] + b[3]*a[7];
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4] * b[1] * ent;
{5'}
  arg := a[2]*a[8] + b[2]*a[5];
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8] * a[1] * ent;
{6'}
  arg := a[7];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4] * b[1]*b[3] * ent;
{7'}
  arg := a[9];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4] * b[1]*a[3] * ent;

{8'}
  arg := a[5];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8] * a[1]*b[2] * ent;

{9'}
  arg := a[8];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8] * a[1]*a[2] * ent;

  avi := avi + snt;
  avi := avi/4;
  V := avi;

END; {of V(a)}


BEGIN {of prog}

Rewrite (out,'mems3.out');

  FOR i := 1 TO 9 DO a[i] := 0.1;
  ap := a;
  am := a;

  old := V(a);

  oldprev := 1.0;
  i := 0;
  WHILE (oldprev <> old) AND (i < 10000) DO BEGIN
    oldprev := old;
    i := i + 1;
    FOR j := 1 TO 9 DO BEGIN
```

```
      s := 1;
      FOR k := 1 TO 9 DO BEGIN
        s := s / 10;
        endflag := FALSE;
        REPEAT
          ap := a;
          IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
          new := V(ap);

          IF new > old THEN BEGIN
            a[j] := ap[j];
            old := new;
          END ELSE
          BEGIN
            am := a;
            IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
            new := V(am);

            IF new > old THEN BEGIN
              a[j] := am[j];
              old := new;
            END ELSE endflag := TRUE;
          END;

        UNTIL endflag;

      END; {of k}

    END; {of jj}

    WritelN (     'i=', i:3, ' R=', old:18:16);
    WritelN (out, 'i=', i:3, ' R=', old:18:16)

  END; {of i}

  FOR i := 1 TO 9 DO Write   (     'a[',i:2,']=',a[i]:12:10);
  FOR i := 1 TO 9 DO WritelN (out, 'a[',i:2,']=',a[i]:12:10);

END.
```

Appendix 2: Source code `rndgen.p`

```
MODULE Rndgen ;
EXPORT
procedure InitRndm ;
function Rndm : Longreal ;
function Normal : Longreal ;
IMPLEMENT
type longbint = record
                  case boolean of
                    true : (i : longint ) ;
                    false : ( b : set of 0..63 ) ;
                end ;
type bint = record
            case boolean of
            true : (i : integer ) ;
            false : ( b : set of 0..31 ) ;
            end ;
var anmin, an : integer ;
    table : array[0..15] of integer ;
function Number : integer ;
var anhelp, lanmin, lan : longint ;
    a, b, c : longbint ;
begin
  a.i := hex('000000007FFFFFFF') ;
  lanmin := anmin ; lan := an ;
  anhelp := lanmin + lan ;
  anmin := lan ;
  b.i := anhelp ;
  c.b := a.b * b.b ;
  an := c.i ;
  Number := an ;
end ; { end of Number }
function Rndm : Longreal ;
const Maxx = 2147483648.0 ;    {2**31}
var  adr, num, tab, i : integer ;
     a, b, c : bint ;
begin
  a.i := hex('0000000F') ;
  for i := 1 to 5 do tab := Number ;
  b.i := anmin ;
  c.b := a.b * b.b ;
  adr := c.i ;
  num := table [ adr ] ;
  table [ adr ] := an ;
  Rndm := num / Maxx ;
end ; { end of Rndm }
procedure InitRndm ;
var  i : integer ;
     tab : Longreal ;
begin
  table[0] := 1 ; table[1] := 1 ;
  anmin := 1 ; an := 1 ;
  for i := 2 to 15 do table[i] := Number ;
  for i := 1 to 500 do tab := Rndm ;
end ; { end of InitRndm }
function Normal : Longreal ;
(* Abramowitz and Stegun, 26.2.23, p.933,  abs.error < .000045   *)
(* if Normal is multiplied with a constant c, the variance becomes sqr(c) *)
const
    c0 = 2.515517 ;
    c1 = 0.802853 ;
    c2 = 0.010328 ;
    d1 = 1.432788 ;
    d2 = 0.189269 ;
    d3 = 0.001308 ;
var
    uniform, t, tt : Longreal ;
begin
  uniform := Rndm ;
  if uniform <= 0.5 then
  begin
```

```
    tt := - 2.0 * ln ( uniform ) ;
    t := sqrt ( tt ) ;
    Normal := (c0+c1*t+c2*tt)/(1.0+d1*t+d2*tt+d3*t*tt) - t ;
  end
  else
  begin
    tt := - 2.0 * ln ( 1.0 - uniform ) ;
    t  := sqrt ( tt ) ;
    Normal := t - (c0+c1*t+c2*tt)/(1.0+d1*t+d2*tt+d3*t*tt) ;
  end ;
end ; { end of Normal }

end. { end of MODULE Rndgen }
```

Appendix 3: Program body `mems3.rnd.p`

```
PROGRAM MEMS3RND (input, output);
{A1..A9}

$SEARCH 'rndgen.o'$
IMPORT rndgen;

TYPE AR = ArraY [1..9] of Longreal;
VAR i,j,jj,k,l,teller          : Integer;
    old, oldprev, new, s, Rbest : Longreal;
    endflag                     : Boolean;
    a, ap ,am                   : AR;
    out                         : Text;
    verz                        : SET OF 1..9;


BEGIN {of prog}

Rewrite (out,'mems3rnd.out');
InitRndm;
Rbest := 0;
teller := 0;

REPEAT

  FOR i := 1  TO 9 DO a[i] := Rndm;
  ap := a;
  am := a;

  old := V(a);

  oldprev := 1.0;
  i := 0;

  WHILE (oldprev <> old) AND (i < 10000) DO BEGIN
    oldprev := old;
    i := i + 1;
    write (i:1, ' ');
    verz := [1..9];
    FOR jj := 1 TO 9 DO BEGIN
      j := TrunC (9 * Rndm + 1);
      WhilE NOT (j IN verz) DO j := TrunC (9 * Rndm + 1);
      verz := verz - [j];
      s := 1;
      FOR k := 1 TO 9 DO BEGIN
        s := s / 10;
        endflag := FALSE;
        REPEAT
          ap := a;
          IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
          new := V(ap);

          IF new > old THEN BEGIN
            a[j] := ap[j];
            old := new;
          END ELSE
          BEGIN
            am := a;
            IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
            new := V(am);

            IF new > old THEN BEGIN
              a[j] := am[j];
              old := new;
            END ELSE endflag := TRUE;
          END;

        UNTIL endflag;

      END; {of k}
```

```
    END; {of jj}

  END; {of i}

  teller := teller + 1;
  Write (teller:1,' ');
  WritelN (out);
  WritelN (out, 'teller=',teller:1);
  WritelN (out, 'i=', i:3, ' R=', old:18:16);
  WritelN (out);
  FOR i := 1 TO 9 DO BEGIN
    WritE (out, 'a[',i:2,']=',a[i]:12:10, ' ');
    IF i IN [1,3,7,9] THEN WritelN (out);
  END;

UNTIL FALSE

END.
```

Appendix 4: Program body `mems3.ful.p`

```
PROGRAM MEMS3FUL (input, output);

{A1..A9}

TYPE AR = ArraY [1..9] of Longreal;
VAR i,j,teller                  : Integer;
    old, oldprev, new, max      : Longreal;
    greater                     : Boolean;
    a                           : AR;
    out                         : Text;

BEGIN {of prog}

  Rewrite (out,'mems3.ful.out');
  teller := 0;

  FOR i := 1 TO 9 DO a[i] := 0.01;
  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WritelN;
    WritelN (teller:1, ' R=', old:18:16);

    FOR i := 1 TO 9 DO BEGIN

      Write (i:1,' ');
      a[i] := 0.01;
      old := V(a);
      max := a[i];

      FOR j := 1 TO 99 DO BEGIN

        greater := TRUE;
        a[i] := a[i] + 0.01;
        IF a[i] > 1 THEN a[i] := 1;
        new := V(a);
        IF new > old THEN BEGIN
          IF NOT greater THEN Write (' multiple maxima');
          old := new;
          max := a[i];
        END ELSE
          greater := FALSE;

      END; {of j}

      a[i] := max;

    END; {of i}

  END; {of while}

  WritelN (out, 'Teller=',teller:1);
  WritelN;
  WritelN (     ' R=', old:18:16);
  WritelN (out, ' R=', old:18:16);

  FOR i := 1 TO 9 DO BEGIN
    WritelN (     'a[',i:2,']=',a[i]:18:16);
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  END;

END.
```

Appendix 5: Source code mems4.p

```
PROGRAM MEMS4 (input, output);

{A1..A27}

TYPE AR = ArraY [1..27] of Longreal;
VAR i,j,teller                  : Integer;
    old, oldprev, new, max      : Longreal;
    greater                     : Boolean;
    a                           : AR;
    out                         : Text;

FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a : AR) : Longreal;
  VAR i                     : Integer;
      arg, ent, snt, avi    : Longreal;
      b                     : AR;

  BEGIN

  FOR i := 1 TO 27 do b[i] := 1 - a[i];

{input symbol 2}
  arg := a[1];
  ent := H(arg);
  avi := (b[1]*a[3] + a[1]*a[2]) * ent;

{input symbol 3}
  arg := a[1]*a[2] + b[1]*a[3];
  ent := H(arg);
  snt := (b[1]*b[3]*a[7] + a[1]*b[2]*a[5]) * ent;
  arg := a[3];
  ent := H(arg);
  snt := snt + b[1] * (b[1]*a[3]*a[6] + a[1]*a[2]*a[4]) * ent;
  arg := a[2];
  ent := H(arg);
  snt := snt + a[1] * (b[1]*a[3]*a[9] + a[1]*a[2]*a[8]) * ent;
  avi := avi + snt;

{input symbol 4}
{4,1}
  arg := a[1] * (a[2]*a[4] + b[2]*a[5]) + b[1] * (a[3]*a[6] + b[3]*a[7]);
  ent := H(arg);
  snt := b[1]*b[3]*b[7]*a[17] * ent;
{4,2}
  arg := (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) / (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16] * (a[1]*b[2] + b[1]*b[3]) * ent;
{4,3}
  arg := (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]) / (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21] * (a[1]*a[2] + b[1]*a[3]) * ent;
{4,4}
  arg := a[3]*a[6] + b[3]*a[7];
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15] * b[1] * ent;
{4,5}
  arg := a[2]*a[4] + b[2]*a[5];
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25] * a[1] * ent;
{4,6}
  arg := a[7];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14] * b[1]*b[3] * ent;
{4,7}
  arg := a[6];
```

```
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20] * b[1]*a[3] * ent;
{4,8}
  arg := a[5];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24] * a[1]*b[2] * ent;
{4,9}
  arg := a[4];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27] * a[1]*a[2] * ent;

  avi := avi + snt;


{4,1'}
  arg := a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7]);
  ent := H(arg);
  snt := a[1]*b[2]*b[5]*a[13] * ent;
{4,2'}
  arg := (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) / (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12] * (a[1]*b[2] + b[1]*b[3]) * ent;
{4,3'}
  arg := (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]) / (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19] * (a[1]*a[2] + b[1]*a[3]) * ent;
{4,4'}
  arg := a[3]*a[9] + b[3]*a[7];
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11] * b[1] * ent;
{4,5'}
  arg := a[2]*a[8] + b[2]*a[5];
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23] * a[1] * ent;
{4,6'}
  arg := a[7];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10] * b[1]*b[3] * ent;
{4,7'}
  arg := a[9];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18] * b[1]*a[3] * ent;

{4,8'}
  arg := a[5];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22] * a[1]*b[2] * ent;

{4,9'}
  arg := a[8];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26] * a[1]*a[2] * ent;

  avi := avi + snt;


{input symbol 5}
{5,1}
  arg := a[1] * (a[2] * (a[4]*a[10] + b[4]*a[11]) +
                 b[2] * (a[5]*a[12] + b[5]*a[13])) +
         b[1] * (a[3] * (a[6]*a[14] + b[6]*a[15]) +
                 b[3] * (a[7]*a[16] + b[7]*a[17]));
  ent := H(arg);
  snt := b[1]*b[3]*b[7]*b[17] * ent;
{5,2}
  arg := (a[1] * (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]) +
          b[1] * (a[3]*b[6]*a[15] + b[3]*b[7]*a[17])) /
         (a[1] * (a[2]*b[4] + b[2]*b[5]) + b[1] * (a[3]*b[6] + b[3]*b[7]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17] *
         (a[1] * (a[2]*b[4] + b[2]*b[5]) +
          b[1] * (a[3]*b[6] + b[3]*b[7])) * ent;
{5,3}
  arg := (a[1] * (a[2]*a[4]*a[10] + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[6]*a[14] + b[3]*a[7]*a[16])) /
```

```
        (a[1] * (a[2]*a[4] + b[2]*a[5]) + b[1] * (a[3]*a[6] + b[3]*a[7]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17] *
        (a[1] * (a[2]*a[4] + b[2]*a[5]) +
         b[1] * (a[3]*a[6] + b[3]*a[7])) * ent;
{5,4}
  arg := (a[1]*b[2] * (a[5]*a[12] + b[5]*a[13]) +
         b[1]*b[3] * (a[7]*a[16] + b[7]*a[17])) /
        (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[16] * (a[1]*b[2] + b[1]*b[3]) * ent;
{5,5}
  arg := (a[1]*a[2] * (a[4]*a[10] + b[4]*a[11]) +
         b[1]*a[3] * (a[6]*a[14] + b[6]*a[15])) /
        (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[21] * (a[1]*a[2] + b[1]*a[3]) * ent;
{5,6}
  arg := (a[1]*b[2]*b[5]*a[13] + b[1]*b[3]*b[7]*a[17]) /
        (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16] *
        (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]) * ent;
{5,7}
  arg := (a[1]*b[2]*a[5]*a[12] + b[1]*b[3]*a[7]*a[16]) /
        (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16] *
        (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) * ent;
{5,8}
  arg := (a[1]*a[2]*b[4]*a[11] + b[1]*a[3]*b[6]*a[15]) /
        (a[1]*a[2]*b[4] + b[1]*a[3]*b[6]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21] *
        (a[1]*a[2]*b[4] + b[1]*a[3]*b[6]) * ent;
{5,9}
  arg := (a[1]*a[2]*a[4]*a[10] + b[1]*a[3]*a[6]*a[14]) /
        (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21] *
        (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]) * ent;
{5,10}
  arg := a[3] * (a[6]*a[20] + b[6]*a[15]) + b[3] * (a[7]*a[21] + b[7]*a[17]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*b[15] * b[1] * ent;
{5,11}
  arg := a[2] * (a[4]*a[18] + b[4]*a[11]) + b[2] * (a[5]*a[19] + b[5]*a[13]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*b[25] * a[1] * ent;
{5,12}
  arg := (a[3]*b[6]*a[15] + b[3]*b[7]*a[17]) / (a[3]*b[6] + b[3]*b[7]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15] *
        b[1] * (a[3]*b[6] + b[3]*b[7]) * ent;
{5,13}
  arg := (a[3]*a[6]*a[20] + b[3]*a[7]*a[21]) / (a[3]*a[6] + b[3]*a[7]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15] *
        b[1] * (a[3]*a[6] + b[3]*a[7]) * ent;
{5,14}
  arg := (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]) / (a[2]*b[4] + b[2]*b[5]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25] *
        a[1] * (a[2]*b[4] + b[2]*b[5]) * ent;
{5,15}
  arg := (a[2]*a[4]*a[18] + b[2]*a[5]*a[19]) /  (a[2]*a[4] + b[2]*a[5]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25] *
        a[1] *  (a[2]*a[4] + b[2]*a[5]) * ent;
{5,16}
  arg := a[7]*a[21] + b[7]*a[17];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[14] * b[1]*b[3] * ent;
{5,17}
  arg := a[6]*a[20] + b[6]*a[15];
```

```
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[20] * b[1]*a[3] * ent;
{5,18}
  arg := a[5]*a[19] + b[5]*a[13];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[24] * a[1]*b[2] * ent;
{5,19}
  arg := a[4]*a[18] + b[4]*a[11];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[27] * a[1]*a[2] * ent;
{5,20}
  arg := a[17];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14] * b[1]*b[3]*b[7] * ent;
{5,21}
  arg := a[21];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14] * b[1]*b[3]*a[7] * ent;
{5,22}
  arg := a[15];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20] * b[1]*a[3]*b[6] * ent;
{5,23}
  arg := a[20];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20] * b[1]*a[3]*a[6] * ent;
{5,24}
  arg := a[13];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24] * a[1]*b[2]*b[5] * ent;
{5,25}
  arg := a[19];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24] * a[1]*b[2]*a[5] * ent;
{5,26}
  arg := a[11];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27] * a[1]*a[2]*b[4] * ent;
{5,27}
  arg := a[18];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27] * a[1]*a[2]*a[4] * ent;

  avi := avi + snt;


{5,1'}
  arg := a[1] * (a[2] * (a[8]*a[22] + b[8]*a[23]) +
                 b[2] * (a[5]*a[12] + b[5]*a[13])) +
         b[1] * (a[3] * (a[9]*a[24] + b[9]*a[25]) +
                 b[3] * (a[7]*a[16] + b[7]*a[17]));
  ent := H(arg);
  snt := a[1]*b[2]*b[5]*b[13] * ent;
{5,2'}
  arg := (a[1] * (a[2]*b[8]*a[23] + b[2]*b[5]*a[13]) +
          b[1] * (a[3]*b[9]*a[25] + b[3]*b[7]*a[17])) /
         (a[1] * (a[2]*b[8] + b[2]*b[5]) + b[1] * (a[3]*b[9] + b[3]*b[7]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13] *
         (a[1] * (a[2]*b[8] + b[2]*b[5]) + b[1] * (a[3]*b[9] + b[3]*b[7])) *
         ent;
{5,3'}
  arg := (a[1] * (a[2]*a[8]*a[22] + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[9]*a[24] + b[3]*a[7]*a[16])) /
         (a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13] *
         (a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7])) *
         ent;
{5,4'}
  arg := (a[1]*b[2] * (a[5]*a[12] + b[5]*a[13]) +
          b[1]*b[3] * (a[7]*a[16] + b[7]*a[17])) /
         (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[12] * (a[1]*b[2] + b[1]*b[3]) * ent;
```

```
{5,5'}
  arg := (a[1]*a[2] * (a[8]*a[22] + b[8]*a[23]) +
          b[1]*a[3] * (a[9]*a[24] + b[9]*a[25])) /
         (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[19] * (a[1]*a[2] + b[1]*a[3]) * ent;
{5,6'}
  arg := (a[1]*b[2]*b[5]*a[13] + b[1]*b[3]*b[7]*a[17]) /
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12] *
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]) * ent;
{5,7'}
  arg := (a[1]*b[2]*a[5]*a[12] + b[1]*b[3]*a[7]*a[16]) /
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12] *
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) * ent;
{5,8'}
  arg := (a[1]*a[2]*b[8]*a[23] + b[1]*a[3]*b[9]*a[25]) /
         (a[1]*a[2]*b[8] + b[1]*a[3]*b[9]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19] *
         (a[1]*a[2]*b[8] + b[1]*a[3]*b[9]) * ent;
{5,9'}
  arg := (a[1]*a[2]*a[8]*a[22] +b[1]*a[3]*a[9]*a[24]) /
         (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19] *
         (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]) * ent;
{5,10'}
  arg := a[3] * (a[9]*a[27] + b[9]*a[25]) +
         b[3] * (a[7]*a[21] + b[7]*a[17]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*b[11] * b[1] * ent;
{5,11'}
  arg := a[2] * (a[8]*a[26] + b[8]*a[23]) +
         b[2] * (a[5]*a[19] + b[5]*a[13]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*b[23] * a[1] * ent;
{5,12'}
  arg := (a[3]*b[9]*a[25] + b[3]*b[7]*a[17]) / (a[3]*b[9] + b[3]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11] *
         b[1] * (a[3]*b[9] + b[3]*b[7]) * ent;
{5,13'}
  arg := (a[3]*a[9]*a[27] + b[3]*a[7]*a[21]) /
         (a[3]*a[9] + b[3]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11] *
         b[1] * (a[3]*a[9] + b[3]*a[7]) * ent;
{5,14'}
  arg := (a[2]*b[8]*a[23] + b[2]*b[5]*a[13]) /
         (a[2]*b[8] + b[2]*b[5]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23] *
         a[1] * (a[2]*b[8] + b[2]*b[5]) * ent;
{5,15'}
  arg := (a[2]*a[8]*a[26] + b[2]*a[5]*a[19]) /
         (a[2]*a[8] + b[2]*a[5]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23] *
         a[1] * (a[2]*a[8] + b[2]*a[5]) * ent;
{5,16'}
  arg := a[7]*a[21] + b[7]*a[17];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[10] * b[1]*b[3] * ent;
{5,17'}
  arg := a[9]*a[27] + b[9]*a[25];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[18] * b[1]*a[3] * ent;
{5,18'}
  arg := a[5]*a[19] + b[5]*a[13];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[22] * a[1]*b[2] * ent;
```

```
{5,19'}
  arg := a[8]*a[26] + b[8]*a[23];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[26] * a[1]*a[2] * ent;
{5,20'}
  arg := a[17];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10] * b[1]*b[3]*b[7] * ent;
{5,21'}
  arg := a[21];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10] * b[1]*b[3]*a[7] * ent;
{5,22'}
  arg := a[25];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18] * b[1]*a[3]*b[9] * ent;
{5,23'}
  arg := a[27];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18] * b[1]*a[3]*a[9] * ent;
{5,24'}
  arg := a[13];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22] * a[1]*b[2]*b[5] * ent;
{5,25'}
  arg := a[19];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22] * a[1]*b[2]*a[5] * ent;
{5,26'}
  arg := a[23];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26] * a[1]*a[2]*b[8] * ent;
{5,27'}
  arg := a[26];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26] * a[1]*a[2]*a[8] * ent;

  avi := avi + snt;

  avi := avi/5;
  V := avi;

END; {of V(a)}


BEGIN {of prog}

Rewrite (out,'mems4.out');

  FOR i := 1  TO 27 DO a[i] := 0.1;
  ap := a;
  am := a;

  old := V(a);

  oldprev := 1.0;
  i := 0;
  WHILE (oldprev <> old) AND (i < 10000) DO BEGIN
    oldprev := old;
    i := i + 1;
    FOR j := 1 TO 27 DO BEGIN
      s := 1;
      FOR k := 1 TO 9 DO BEGIN
        s := s / 10;
        endflag := FALSE;
        REPEAT
          ap := a;
          IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
          new := V(ap);

          IF new > old THEN BEGIN
            a[j] := ap[j];
            old := new;
          END ELSE
          BEGIN
```

```
            am := a;
            IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
            new := V(am);

            IF new > old THEN BEGIN
              a[j] := am[j];
              old := new;
            END ELSE endflag := TRUE;
          END;

        UNTIL endflag;

      END; {of k}

    END; {of jj}

    WritelN (     'i=', i:3, ' R=', old:18:16);
    WritelN (out, 'i=', i:3, ' R=', old:18:16)

  END; {of i}

  FOR i := 1 TO 27 DO Write   (     'a[',i:2,']=',a[i]:12:10);
  FOR i := 1 TO 27 DO WritelN (out, 'a[',i:2,']=',a[i]:12:10);

END.
```

Appendix 6: Program body `mems4.rnd.p`

```
PROGRAM MEMS4RND (input, output);

{A1..A27}

$SEARCH 'rndgen.o'$
IMPORT rndgen;

TYPE AR = ArraY [1..27] of Longreal;
VAR i,j,jj,k,l,teller           : Integer;
    old, oldprev, new, s, Rbest : Longreal;
    endflag, min, plus          : Boolean;
    a, ap ,am                   : AR;
    out                         : Text;
    verz                        : SET OF 1..27;


BEGIN {of prog}

Rewrite (out,'mems42rnd.out');
InitRndm;
Rbest := 0;
teller := 0;

REPEAT

  FOR i := 1 TO 27 DO a[i] := Rndm;

  ap := a;
  am := a;

  old := V(a);

  oldprev := 1.0;
  i := 0;
   WHILE (oldprev <> old) AND (i < 10000) DO BEGIN
    oldprev := old;
    i := i + 1;
    Write (i:1, ' ');
    verz := [1..27];
    FOR jj := 1 TO 27 DO BEGIN
      j := TrunC (27 * Rndm + 1);
      WhilE NOT (j IN verz) DO j := TrunC (27 * Rndm + 1);
      verz := verz - [j];
       s := 1 / EXP(LN(10) * i);
      endflag := FALSE;
      plus    := FALSE;
      min     := FALSE;
      REPEAT
        IF NOT (min) THEN
        BEGIN
          ap := a;
          IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
          new := V(ap);

          IF new > old THEN BEGIN
            plus := TRUE;
            a[j] := ap[j];
            old := new;
          END ELSE min := TRUE;
        END ELSE
        BEGIN
          IF plus THEN endflag := TRUE ELSE
          BEGIN
            am := a;
            IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
            new := V(am);

            IF new > old THEN BEGIN
              a[j] := am[j];
              old := new;
```

```
            END ELSE endflag := TRUE;
          END;
        END;

      UNTIL endflag;

    END; {of jj}

  END; {of i}

  teller := teller + 1;
  WritelN;
  WritelN (teller:1,' ');

  IF old > Rbest THEN BEGIN
    WritelN;
    WritelN (     'i=', i:1, ' R=', old:18:16);
    WritelN (out);
    WritelN (out, 'teller=',teller:1);
    WritelN (out, 'i=', i:1, ' R=', old:18:16);
    WritelN (out);
    FOR i := 1 TO 27 DO
      WritelN (out, 'a[',i:2,']=',a[i]:18:16);
    Rbest := old;
  END;

UNTIL FALSE

END.
```

Appendix 7: Program body mems4.ful.p

```
PROGRAM MEMS4FUL (input, output);

{A1..A27}

TYPE AR = ArraY [1..27] of Longreal;
VAR i,j,teller                  : Integer;
    old, oldprev, new, max      : Longreal;
    greater                     : Boolean;
    a                           : AR;
    out                         : Text;


BEGIN {of prog}

  Rewrite (out,'mems4.ful.out');
  teller := 0;

  FOR i := 1 TO 27 DO a[i] := 0.01;
  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WritelN;
    WritelN (teller:1, ' R=', old:18:16);

    FOR i := 1 TO 27 DO BEGIN

      Write (i:1,' ');
      a[i] := 0.01;
      old := V(a);
      max := a[i];

      FOR j := 1 TO 99 DO BEGIN

        greater := TRUE;
        a[i] := a[i] + 0.01082749356256743;
        IF a[i] > 1 THEN a[i] := 1;
        new := V(a);
        IF new > old THEN BEGIN
          IF NOT greater THEN Write (' Oeps !!! ');
          old := new;
          max := a[i];
        END ELSE
          greater := FALSE;

      END; {of j}

      a[i] := max;

    END; {of i}

  END; {of while}

  WritelN (out, 'Teller=',teller:1);
  WritelN;
  WritelN (     ' R=', old:18:16);
  WritelN (out, ' R=', old:18:16);

  FOR i := 1 TO 27 DO BEGIN
    WritelN (     'a[',i:2,']=',a[i]:18:16);
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  END;

END.
```

Appendix 8: Program body mems4.asm.p

```
PROGRAM MEMS4ASM (input, output);
{A1..A27}


TYPE AR = ArraY [1..27] of Longreal;

VAR i,j,k                 : Integer;
    old, oldprev, new, s  : Longreal;
    endflag, min, plus    : Boolean;
    a, c, ap, am, cp, cm  : AR;
    out                   : Text;

FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a, c : AR) : Longreal;
  VAR i                   : Integer;
      arg, ent, snt, avi  : Longreal;
      b, d                : AR;

  BEGIN

  FOR i := 1 TO 27 do b[i] := 1 - a[i];
  FOR i := 1 TO 27 do d[i] := 1 - c[i];

{input symbol 2 user 1}
  arg := a[1];
  ent := H(arg);
  avi := (d[1]*c[3] + c[1]*c[2]) * ent;

{input symbol 3 user 1}
  arg := a[1]*a[2] + b[1]*a[3];
  ent := H(arg);
  snt := (d[1]*d[3]*c[7] + c[1]*d[2]*c[5]) * ent;
  arg := a[3];
  ent := H(arg);
  snt := snt + b[1] * (d[1]*c[3]*c[6] + c[1]*c[2]*c[4]) * ent;
  arg := a[2];
  ent := H(arg);
  snt := snt + a[1] * (d[1]*c[3]*c[9] + c[1]*c[2]*c[8]) * ent;
  avi := avi + snt;

{input symbol 4 user 1}
{1,4,1}
  arg := a[1] * (a[2]*a[4] + b[2]*a[5]) + b[1] * (a[3]*a[6] + b[3]*a[7]);
  ent := H(arg);
  snt := d[1]*d[3]*d[7]*c[17] * ent;
{1,4,2}
  arg := (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) / (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + d[1]*d[3]*c[7]*c[16] * (a[1]*b[2] + b[1]*b[3]) * ent;
{1,4,3}
  arg := (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]) / (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + d[1]*d[3]*c[7]*c[21] * (a[1]*a[2] + b[1]*a[3]) * ent;
{1,4,4}
  arg := a[3]*a[6] + b[3]*a[7];
  ent := H(arg);
  snt := snt + d[1]*c[3]*d[6]*c[15] * b[1] * ent;
{1,4,5}
  arg := a[2]*a[4] + b[2]*a[5];
  ent := H(arg);
  snt := snt + d[1]*c[3]*d[9]*c[25] * a[1] * ent;
{1,4,6}
  arg := a[7];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[6]*c[14] * b[1]*b[3] * ent;
```

```
{1,4,7}
  arg := a[6];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[6]*c[20] * b[1]*a[3] * ent;
{1,4,8}
  arg := a[5];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[9]*c[24] * a[1]*b[2] * ent;
{1,4,9}
  arg := a[4];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[9]*c[27] * a[1]*a[2] * ent;

  avi := avi + snt;


{1,4,1'}
  arg := a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7]);
  ent := H(arg);
  snt := c[1]*d[2]*d[5]*c[13] * ent;
{1,4,2'}
  arg := (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) / (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + c[1]*d[2]*c[5]*c[12] * (a[1]*b[2] + b[1]*b[3]) * ent;
{1,4,3'}
  arg := (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]) / (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + c[1]*d[2]*c[5]*c[19] * (a[1]*a[2] + b[1]*a[3]) * ent;
{1,4,4'}
  arg := a[3]*a[9] + b[3]*a[7];
  ent := H(arg);
  snt := snt + c[1]*c[2]*d[4]*c[11] * b[1] * ent;
{1,4,5'}
  arg := a[2]*a[8] + b[2]*a[5];
  ent := H(arg);
  snt := snt + c[1]*c[2]*d[8]*c[23] * a[1] * ent;
{1,4,6'}
  arg := a[7];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[4]*c[10] * b[1]*b[3] * ent;
{1,4,7'}
  arg := a[9];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[4]*c[18] * b[1]*a[3] * ent;

{1,4,8'}
  arg := a[5];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[8]*c[22] * a[1]*b[2] * ent;

{1,4,9'}
  arg := a[8];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[8]*c[26] * a[1]*a[2] * ent;


  avi := avi + snt;


{input symbol 5 user 1}
{1,5,1}
  arg := a[1] * (a[2] * (a[4]*a[10] + b[4]*a[11]) +
                 b[2] * (a[5]*a[12] + b[5]*a[13])) +
         b[1] * (a[3] * (a[6]*a[14] + b[6]*a[15]) +
                 b[3] * (a[7]*a[16] + b[7]*a[17]));
  ent := H(arg);
  snt := d[1]*d[3]*d[7]*d[17] * ent;
{1,5,2}
  arg := (a[1] * (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]) +
          b[1] * (a[3]*b[6]*a[15] + b[3]*b[7]*a[17])) /
         (a[1] * (a[2]*b[4] + b[2]*b[5]) + b[1] * (a[3]*b[6] + b[3]*b[7]));
  ent := H(arg);
  snt := snt + d[1]*d[3]*d[7]*c[17] *
         (a[1] * (a[2]*b[4] + b[2]*b[5]) +
          b[1] * (a[3]*b[6] + b[3]*b[7])) * ent;
```

```
{1,5,3}
  arg := (a[1] * (a[2]*a[4]*a[10] + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[6]*a[14] + b[3]*a[7]*a[16])) /
         (a[1] * (a[2]*a[4] + b[2]*a[5]) + b[1] * (a[3]*a[6] + b[3]*a[7]));
  ent := H(arg);
  snt := snt + d[1]*d[3]*d[7]*c[17] *
         (a[1] * (a[2]*a[4] + b[2]*a[5]) +
          b[1] * (a[3]*a[6] + b[3]*a[7])) * ent;
{1,5,4}
  arg := (a[1]*b[2] * (a[5]*a[12] + b[5]*a[13]) +
          b[1]*b[3] * (a[7]*a[16] + b[7]*a[17])) /
         (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + d[1]*d[3]*c[7]*d[16] * (a[1]*b[2] + b[1]*b[3]) * ent;
{1,5,5}
  arg := (a[1]*a[2] * (a[4]*a[10] + b[4]*a[11]) +
          b[1]*a[3] * (a[6]*a[14] + b[6]*a[15])) /
         (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + d[1]*d[3]*c[7]*d[21] * (a[1]*a[2] + b[1]*a[3]) * ent;
{1,5,6}
  arg := (a[1]*b[2]*b[5]*a[13] + b[1]*b[3]*b[7]*a[17]) /
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]);
  ent := H(arg);
  snt := snt + d[1]*d[3]*c[7]*c[16] *
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]) * ent;
{1,5,7}
  arg := (a[1]*b[2]*a[5]*a[12] + b[1]*b[3]*a[7]*a[16]) /
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]);
  ent := H(arg);
  snt := snt + d[1]*d[3]*c[7]*c[16] *
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) * ent;
{1,5,8}
  arg := (a[1]*a[2]*b[4]*a[11] + b[1]*a[3]*b[6]*a[15]) /
         (a[1]*a[2]*b[4] + b[1]*a[3]*b[6]);
  ent := H(arg);
  snt := snt + d[1]*d[3]*c[7]*c[21] *
         (a[1]*a[2]*b[4] + b[1]*a[3]*b[6]) * ent;
{1,5,9}
  arg := (a[1]*a[2]*a[4]*a[10] + b[1]*a[3]*a[6]*a[14]) /
         (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]);
  ent := H(arg);
  snt := snt + d[1]*d[3]*c[7]*c[21] *
         (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]) * ent;
{1,5,10}
  arg := a[3] * (a[6]*a[20] + b[6]*a[15]) + b[3] * (a[7]*a[21] + b[7]*a[17]);
  ent := H(arg);
  snt := snt + d[1]*c[3]*d[6]*d[15] * b[1] * ent;
{1,5,11}
  arg := a[2] * (a[4]*a[18] + b[4]*a[11]) + b[2] * (a[5]*a[19] + b[5]*a[13]);
  ent := H(arg);
  snt := snt + d[1]*c[3]*d[9]*d[25] * a[1] * ent;
{1,5,12}
  arg := (a[3]*b[6]*a[15] + b[3]*b[7]*a[17]) / (a[3]*b[6] + b[3]*b[7]);
  ent := H(arg);
  snt := snt + d[1]*c[3]*d[6]*c[15] *
         b[1] * (a[3]*b[6] + b[3]*b[7]) * ent;
{1,5,13}
  arg := (a[3]*a[6]*a[20] + b[3]*a[7]*a[21]) / (a[3]*a[6] + b[3]*a[7]);
  ent := H(arg);
  snt := snt + d[1]*c[3]*d[6]*c[15] *
         b[1] * (a[3]*a[6] + b[3]*a[7]) * ent;
{1,5,14}
  arg := (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]) / (a[2]*b[4] + b[2]*b[5]);
  ent := H(arg);
  snt := snt + d[1]*c[3]*d[9]*c[25] *
         a[1] * (a[2]*b[4] + b[2]*b[5]) * ent;
{1,5,15}
  arg := (a[2]*a[4]*a[18] + b[2]*a[5]*a[19]) /  (a[2]*a[4] + b[2]*a[5]);
  ent := H(arg);
  snt := snt + d[1]*c[3]*d[9]*c[25] *
         a[1] *  (a[2]*a[4] + b[2]*a[5]) * ent;
{1,5,16}
  arg := a[7]*a[21] + b[7]*a[17];
  ent := H(arg);
```

```
  snt := snt + d[1]*c[3]*c[6]*d[14] * b[1]*b[3] * ent;
{1,5,17}
  arg := a[6]*a[20] + b[6]*a[15];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[6]*d[20] * b[1]*a[3] * ent;
{1,5,18}
  arg := a[5]*a[19] + b[5]*a[13];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[9]*d[24] * a[1]*b[2] * ent;
{1,5,19}
  arg := a[4]*a[18] + b[4]*a[11];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[9]*d[27] * a[1]*a[2] * ent;
{1,5,20}
  arg := a[17];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[6]*c[14] * b[1]*b[3]*b[7] * ent;
{1,5,21}
  arg := a[21];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[6]*c[14] * b[1]*b[3]*a[7] * ent;
{1,5,22}
  arg := a[15];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[6]*c[20] * b[1]*a[3]*b[6] * ent;
{1,5,23}
  arg := a[20];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[6]*c[20] * b[1]*a[3]*a[6] * ent;
{1,5,24}
  arg := a[13];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[9]*c[24] * a[1]*b[2]*b[5] * ent;
{1,5,25}
  arg := a[19];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[9]*c[24] * a[1]*b[2]*a[5] * ent;
{1,5,26}
  arg := a[11];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[9]*c[27] * a[1]*a[2]*b[4] * ent;
{1,5,27}
  arg := a[18];
  ent := H(arg);
  snt := snt + d[1]*c[3]*c[9]*c[27] * a[1]*a[2]*a[4] * ent;

  avi := avi + snt;


{1,5,1'}
  arg := a[1] * (a[2] * (a[8]*a[22] + b[8]*a[23]) +
                 b[2] * (a[5]*a[12] + b[5]*a[13])) +
         b[1] * (a[3] * (a[9]*a[24] + b[9]*a[25]) +
                 b[3] * (a[7]*a[16] + b[7]*a[17]));
  ent := H(arg);
  snt := c[1]*d[2]*d[5]*d[13] * ent;
{1,5,2'}
  arg := (a[1] * (a[2]*b[8]*a[23] + b[2]*b[5]*a[13]) +
          b[1] * (a[3]*b[9]*a[25] + b[3]*b[7]*a[17])) /
         (a[1] * (a[2]*b[8] + b[2]*b[5]) + b[1] * (a[3]*b[9] + b[3]*b[7]));
  ent := H(arg);
  snt := snt + c[1]*d[2]*d[5]*c[13] *
         (a[1] * (a[2]*b[8] + b[2]*b[5]) + b[1] * (a[3]*b[9] + b[3]*b[7])) *
         ent;
{1,5,3'}
  arg := (a[1] * (a[2]*a[8]*a[22] + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[9]*a[24] + b[3]*a[7]*a[16])) /
         (a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7]));
  ent := H(arg);
  snt := snt + c[1]*d[2]*d[5]*c[13] *
         (a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7])) *
         ent;
{1,5,4'}
  arg := (a[1]*b[2] * (a[5]*a[12] + b[5]*a[13]) +
          b[1]*b[3] * (a[7]*a[16] + b[7]*a[17])) /
```

```
          (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + c[1]*d[2]*c[5]*d[12] * (a[1]*b[2] + b[1]*b[3]) * ent;
{1,5,5'}
  arg := (a[1]*a[2] * (a[8]*a[22] + b[8]*a[23]) +
          b[1]*a[3] * (a[9]*a[24] + b[9]*a[25])) /
          (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + c[1]*d[2]*c[5]*d[19] * (a[1]*a[2] + b[1]*a[3]) * ent;
{1,5,6'}
  arg := (a[1]*b[2]*b[5]*a[13] + b[1]*b[3]*b[7]*a[17]) /
          (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]);
  ent := H(arg);
  snt := snt + c[1]*d[2]*c[5]*c[12] *
          (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]) * ent;
{1,5,7'}
  arg := (a[1]*b[2]*a[5]*a[12] + b[1]*b[3]*a[7]*a[16]) /
          (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]);
  ent := H(arg);
  snt := snt + c[1]*d[2]*c[5]*c[12] *
          (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) * ent;
{1,5,8'}
  arg := (a[1]*a[2]*b[8]*a[23] + b[1]*a[3]*b[9]*a[25]) /
          (a[1]*a[2]*b[8] + b[1]*a[3]*b[9]);
  ent := H(arg);
  snt := snt + c[1]*d[2]*c[5]*c[19] *
          (a[1]*a[2]*b[8] + b[1]*a[3]*b[9]) * ent;
{1,5,9'}
  arg := (a[1]*a[2]*a[8]*a[22] +b[1]*a[3]*a[9]*a[24]) /
          (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]);
  ent := H(arg);
  snt := snt + c[1]*d[2]*c[5]*c[19] *
          (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]) * ent;
{1,5,10'}
  arg := a[3] * (a[9]*a[27] + b[9]*a[25]) +
          b[3] * (a[7]*a[21] + b[7]*a[17]);
  ent := H(arg);
  snt := snt + c[1]*c[2]*d[4]*d[11] * b[1] * ent;
{1,5,11'}
  arg := a[2] * (a[8]*a[26] + b[8]*a[23]) +
          b[2] * (a[5]*a[19] + b[5]*a[13]);
  ent := H(arg);
  snt := snt + c[1]*c[2]*d[8]*d[23] * a[1] * ent;
{1,5,12'}
  arg := (a[3]*b[9]*a[25] + b[3]*b[7]*a[17]) / (a[3]*b[9] + b[3]*b[7]);
  ent := H(arg);
  snt := snt + c[1]*c[2]*d[4]*c[11] *
          b[1] * (a[3]*b[9] + b[3]*b[7]) * ent;
{1,5,13'}
  arg := (a[3]*a[9]*a[27] + b[3]*a[7]*a[21]) /
          (a[3]*a[9] + b[3]*a[7]);
  ent := H(arg);
  snt := snt + c[1]*c[2]*d[4]*c[11] *
          b[1] * (a[3]*a[9] + b[3]*a[7]) * ent;
{1,5,14'}
  arg := (a[2]*b[8]*a[23] + b[2]*b[5]*a[13]) /
          (a[2]*b[8] + b[2]*b[5]);
  ent := H(arg);
  snt := snt + c[1]*c[2]*d[8]*c[23] *
          a[1] * (a[2]*b[8] + b[2]*b[5]) * ent;
{1,5,15'}
  arg := (a[2]*a[8]*a[26] + b[2]*a[5]*a[19]) /
          (a[2]*a[8] + b[2]*a[5]);
  ent := H(arg);
  snt := snt + c[1]*c[2]*d[8]*c[23] *
          a[1] * (a[2]*a[8] + b[2]*a[5]) * ent;
{1,5,16'}
  arg := a[7]*a[21] + b[7]*a[17];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[4]*d[10] * b[1]*b[3] * ent;
{1,5,17'}
  arg := a[9]*a[27] + b[9]*a[25];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[4]*d[18] * b[1]*a[3] * ent;
{1,5,18'}
```

```
  arg := a[5]*a[19] + b[5]*a[13];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[8]*d[22] * a[1]*b[2] * ent;
{1,5,19'}
  arg := a[8]*a[26] + b[8]*a[23];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[8]*d[26] * a[1]*a[2] * ent;
{1,5,20'}
  arg := a[17];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[4]*c[10] * b[1]*b[3]*b[7] * ent;
{1,5,21'}
  arg := a[21];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[4]*c[10] * b[1]*b[3]*a[7] * ent;
{1,5,22'}
  arg := a[25];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[4]*c[18] * b[1]*a[3]*b[9] * ent;
{1,5,23'}
  arg := a[27];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[4]*c[18] * b[1]*a[3]*a[9] * ent;
{1,5,24'}
  arg := a[13];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[8]*c[22] * a[1]*b[2]*b[5] * ent;
{1,5,25'}
  arg := a[19];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[8]*c[22] * a[1]*b[2]*a[5] * ent;
{1,5,26'}
  arg := a[23];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[8]*c[26] * a[1]*a[2]*b[8] * ent;
{1,5,27'}
  arg := a[26];
  ent := H(arg);
  snt := snt + c[1]*c[2]*c[8]*c[26] * a[1]*a[2]*a[8] * ent;

  avi := avi + snt;


{input symbol 2 user 2}
  arg := c[1];
  ent := H(arg);
  avi := avi + (b[1]*a[3] + a[1]*a[2]) * ent;

{input symbol 3 user 2}
  arg := c[1]*c[2] + d[1]*c[3];
  ent := H(arg);
  snt := (b[1]*b[3]*a[7] + a[1]*b[2]*a[5]) * ent;
  arg := c[3];
  ent := H(arg);
  snt := snt + d[1] * (b[1]*a[3]*a[6] + a[1]*a[2]*a[4]) * ent;
  arg := c[2];
  ent := H(arg);
  snt := snt + c[1] * (b[1]*a[3]*a[9] + a[1]*a[2]*a[8]) * ent;
  avi := avi + snt;

{input symbol 4 user 2}
{2,4,1}
  arg := c[1] * (c[2]*c[4] + d[2]*c[5]) + d[1] * (c[3]*c[6] + d[3]*c[7]);
  ent := H(arg);
  snt := b[1]*b[3]*b[7]*a[17] * ent;
{2,4,2}
  arg := (c[1]*d[2]*c[5] + d[1]*d[3]*c[7]) / (c[1]*d[2] + d[1]*d[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16] * (c[1]*d[2] + d[1]*d[3]) * ent;
{2,4,3}
  arg := (c[1]*c[2]*c[4] + d[1]*c[3]*c[6]) / (c[1]*c[2] + d[1]*c[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21] * (c[1]*c[2] + d[1]*c[3]) * ent;
{2,4,4}
  arg := c[3]*c[6] + d[3]*c[7];
```

```
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15] * d[1] * ent;
{2,4,5}
  arg := c[2]*c[4] + d[2]*c[5];
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25] * c[1] * ent;
{2,4,6}
  arg := c[7];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14] * d[1]*d[3] * ent;
{2,4,7}
  arg := c[6];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20] * d[1]*c[3] * ent;
{2,4,8}
  arg := c[5];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24] * c[1]*d[2] * ent;
{2,4,9}
  arg := c[4];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27] * c[1]*c[2] * ent;

  avi := avi + snt;


{2,4,1'}
  arg := c[1] * (c[2]*c[8] + d[2]*c[5]) + d[1] * (c[3]*c[9] + d[3]*c[7]);
  ent := H(arg);
  snt := a[1]*b[2]*b[5]*a[13] * ent;
{2,4,2'}
  arg := (c[1]*d[2]*c[5] + d[1]*d[3]*c[7]) / (c[1]*d[2] + d[1]*d[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12] * (c[1]*d[2] + d[1]*d[3]) * ent;
{2,4,3'}
  arg := (c[1]*c[2]*c[8] + d[1]*c[3]*c[9]) / (c[1]*c[2] + d[1]*c[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19] * (c[1]*c[2] + d[1]*c[3]) * ent;
{2,4,4'}
  arg := c[3]*c[9] + d[3]*c[7];
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11] * d[1] * ent;
{2,4,5'}
  arg := c[2]*c[8] + d[2]*c[5];
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23] * c[1] * ent;
{2,4,6'}
  arg := c[7];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10] * d[1]*d[3] * ent;
{2,4,7'}
  arg := c[9];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18] * d[1]*c[3] * ent;

{2,4,8'}
  arg := c[5];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22] * c[1]*d[2] * ent;

{2,4,9'}
  arg := c[8];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26] * c[1]*c[2] * ent;

  avi := avi + snt;

{input symbol 5 user 2}
{2,5,1}
  arg := c[1] * (c[2] * (c[4]*c[10] + d[4]*c[11]) +
                 d[2] * (c[5]*c[12] + d[5]*c[13])) +
         d[1] * (c[3] * (c[6]*c[14] + d[6]*c[15]) +
                 d[3] * (c[7]*c[16] + d[7]*c[17]));
  ent := H(arg);
  snt := b[1]*b[3]*b[7]*b[17] * ent;
```

```
{2,5,2}
  arg := (c[1] * (c[2]*d[4]*c[11] + d[2]*d[5]*c[13]) +
          d[1] * (c[3]*d[6]*c[15] + d[3]*d[7]*c[17])) /
         (c[1] * (c[2]*d[4] + d[2]*d[5]) + d[1] * (c[3]*d[6] + d[3]*d[7]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17] *
         (c[1] * (c[2]*d[4] + d[2]*d[5]) +
          d[1] * (c[3]*d[6] + d[3]*d[7])) * ent;
{2,5,3}
  arg := (c[1] * (c[2]*c[4]*c[10] + d[2]*c[5]*c[12]) +
          d[1] * (c[3]*c[6]*c[14] + d[3]*c[7]*c[16])) /
         (c[1] * (c[2]*c[4] + d[2]*c[5]) + d[1] * (c[3]*c[6] + d[3]*c[7]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17] *
         (c[1] * (c[2]*c[4] + d[2]*c[5]) +
          d[1] * (c[3]*c[6] + d[3]*c[7])) * ent;
{2,5,4}
  arg := (c[1]*d[2] * (c[5]*c[12] + d[5]*c[13]) +
          d[1]*d[3] * (c[7]*c[16] + d[7]*c[17])) /
         (c[1]*d[2] + d[1]*d[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[16] * (c[1]*d[2] + d[1]*d[3]) * ent;
{2,5,5}
  arg := (c[1]*c[2] * (c[4]*c[10] + d[4]*c[11]) +
          d[1]*c[3] * (c[6]*c[14] + d[6]*c[15])) /
         (c[1]*c[2] + d[1]*c[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[21] * (c[1]*c[2] + d[1]*c[3]) * ent;
{2,5,6}
  arg := (c[1]*d[2]*d[5]*c[13] + d[1]*d[3]*d[7]*c[17]) /
         (c[1]*d[2]*d[5] + d[1]*d[3]*d[7]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16] *
         (c[1]*d[2]*d[5] + d[1]*d[3]*d[7]) * ent;
{2,5,7}
  arg := (c[1]*d[2]*c[5]*c[12] + d[1]*d[3]*c[7]*c[16]) /
         (c[1]*d[2]*c[5] + d[1]*d[3]*c[7]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16] *
         (c[1]*d[2]*c[5] + d[1]*d[3]*c[7]) * ent;
{2,5,8}
  arg := (c[1]*c[2]*d[4]*c[11] + d[1]*c[3]*d[6]*c[15]) /
         (c[1]*c[2]*d[4] + d[1]*c[3]*d[6]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21] *
         (c[1]*c[2]*d[4] + d[1]*c[3]*d[6]) * ent;
{2,5,9}
  arg := (c[1]*c[2]*c[4]*c[10] + d[1]*c[3]*c[6]*c[14]) /
         (c[1]*c[2]*c[4] + d[1]*c[3]*c[6]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21] *
         (c[1]*c[2]*c[4] + d[1]*c[3]*c[6]) * ent;
{2,5,10}
  arg := c[3] * (c[6]*c[20] + d[6]*c[15]) + d[3] * (c[7]*c[21] + d[7]*c[17]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*b[15] * d[1] * ent;
{2,5,11}
  arg := c[2] * (c[4]*c[18] + d[4]*c[11]) + d[2] * (c[5]*c[19] + d[5]*c[13]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*b[25] * c[1] * ent;
{2,5,12}
  arg := (c[3]*d[6]*c[15] + d[3]*d[7]*c[17]) / (c[3]*d[6] + d[3]*d[7]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15] *
         d[1] * (c[3]*d[6] + d[3]*d[7]) * ent;
{2,5,13}
  arg := (c[3]*c[6]*c[20] + d[3]*c[7]*c[21]) / (c[3]*c[6] + d[3]*c[7]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15] *
         d[1] * (c[3]*c[6] + d[3]*c[7]) * ent;
{2,5,14}
  arg := (c[2]*d[4]*c[11] + d[2]*d[5]*c[13]) / (c[2]*d[4] + d[2]*d[5]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25] *
         c[1] * (c[2]*d[4] + d[2]*d[5]) * ent;
```

```
{2,5,15}
  arg := (c[2]*c[4]*c[18] + d[2]*c[5]*c[19]) /  (c[2]*c[4] + d[2]*c[5]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25] *
         c[1] *  (c[2]*c[4] + d[2]*c[5]) * ent;
{2,5,16}
  arg := c[7]*c[21] + d[7]*c[17];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[14] * d[1]*d[3] * ent;
{2,5,17}
  arg := c[6]*c[20] + d[6]*c[15];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[20] * d[1]*c[3] * ent;
{2,5,18}
  arg := c[5]*c[19] + d[5]*c[13];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[24] * c[1]*d[2] * ent;
{2,5,19}
  arg := c[4]*c[18] + d[4]*c[11];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[27] * c[1]*c[2] * ent;
{2,5,20}
  arg := c[17];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14] * d[1]*d[3]*d[7] * ent;
{2,5,21}
  arg := c[21];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14] * d[1]*d[3]*c[7] * ent;
{2,5,22}
  arg := c[15];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20] * d[1]*c[3]*d[6] * ent;
{2,5,23}
  arg := c[20];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20] * d[1]*c[3]*c[6] * ent;
{2,5,24}
  arg := c[13];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24] * c[1]*d[2]*d[5] * ent;
{2,5,25}
  arg := c[19];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24] * c[1]*d[2]*c[5] * ent;
{2,5,26}
  arg := c[11];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27] * c[1]*c[2]*d[4] * ent;
{2,5,27}
  arg := c[18];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27] * c[1]*c[2]*c[4] * ent;

  avi := avi + snt;


{2,5,1'}
  arg := c[1] * (c[2] * (c[8]*c[22] + d[8]*c[23]) +
                 d[2] * (c[5]*c[12] + d[5]*c[13])) +
         d[1] * (c[3] * (c[9]*c[24] + d[9]*c[25]) +
                 d[3] * (c[7]*c[16] + d[7]*c[17]));
  ent := H(arg);
  snt := a[1]*b[2]*b[5]*b[13] * ent;
{2,5,2'}
  arg := (c[1] * (c[2]*d[8]*c[23] + d[2]*d[5]*c[13]) +
          d[1] * (c[3]*d[9]*c[25] + d[3]*d[7]*c[17])) /
         (c[1] * (c[2]*d[8] + d[2]*d[5]) + d[1] * (c[3]*d[9] + d[3]*d[7]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13] *
         (c[1] * (c[2]*d[8] + d[2]*d[5]) + d[1] * (c[3]*d[9] + d[3]*d[7])) *
         ent;
{2,5,3'}
  arg := (c[1] * (c[2]*c[8]*c[22] + d[2]*c[5]*c[12]) +
          d[1] * (c[3]*c[9]*c[24] + d[3]*c[7]*c[16])) /
```

```
          (c[1] * (c[2]*c[8] + d[2]*c[5]) + d[1] * (c[3]*c[9] + d[3]*c[7]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13] *
          (c[1] * (c[2]*c[8] + d[2]*c[5]) + d[1] * (c[3]*c[9] + d[3]*c[7])) *
          ent;
{2,5,4'}
  arg := (c[1]*d[2] * (c[5]*c[12] + d[5]*c[13]) +
           d[1]*d[3] * (c[7]*c[16] + d[7]*c[17])) /
          (c[1]*d[2] + d[1]*d[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[12] * (c[1]*d[2] + d[1]*d[3]) * ent;
{2,5,5'}
  arg := (c[1]*c[2] * (c[8]*c[22] + d[8]*c[23]) +
           d[1]*c[3] * (c[9]*c[24] + d[9]*c[25])) /
          (c[1]*c[2] + d[1]*c[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[19] * (c[1]*c[2] + d[1]*c[3]) * ent;
{2,5,6'}
  arg := (c[1]*d[2]*d[5]*c[13] + d[1]*d[3]*d[7]*c[17]) /
          (c[1]*d[2]*d[5] + d[1]*d[3]*d[7]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12] *
          (c[1]*d[2]*d[5] + d[1]*d[3]*d[7]) * ent;
{2,5,7'}
  arg := (c[1]*d[2]*c[5]*c[12] + d[1]*d[3]*c[7]*c[16]) /
          (c[1]*d[2]*c[5] + d[1]*d[3]*c[7]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12] *
          (c[1]*d[2]*c[5] + d[1]*d[3]*c[7]) * ent;
{2,5,8'}
  arg := (c[1]*c[2]*d[8]*c[23] + d[1]*c[3]*d[9]*c[25]) /
          (c[1]*c[2]*d[8] + d[1]*c[3]*d[9]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19] *
          (c[1]*c[2]*d[8] + d[1]*c[3]*d[9]) * ent;
{2,5,9'}
  arg := (c[1]*c[2]*c[8]*c[22] +d[1]*c[3]*c[9]*c[24]) /
          (c[1]*c[2]*c[8] + d[1]*c[3]*c[9]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19] *
          (c[1]*c[2]*c[8] + d[1]*c[3]*c[9]) * ent;
{2,5,10'}
  arg := c[3] * (c[9]*c[27] + d[9]*c[25]) +
          d[3] * (c[7]*c[21] + d[7]*c[17]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*b[11] * d[1] * ent;
{2,5,11'}
  arg := c[2] * (c[8]*c[26] + d[8]*c[23]) +
          d[2] * (c[5]*c[19] + d[5]*c[13]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*b[23] * c[1] * ent;
{2,5,12'}
  arg := (c[3]*d[9]*c[25] + d[3]*d[7]*c[17]) / (c[3]*d[9] + d[3]*d[7]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11] *
          d[1] * (c[3]*d[9] + d[3]*d[7]) * ent;
{2,5,13'}
  arg := (c[3]*c[9]*c[27] + d[3]*c[7]*c[21]) /
          (c[3]*c[9] + d[3]*c[7]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11] *
          d[1] * (c[3]*c[9] + d[3]*c[7]) * ent;
{2,5,14'}
  arg := (c[2]*d[8]*c[23] + d[2]*d[5]*c[13]) /
          (c[2]*d[8] + d[2]*d[5]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23] *
          c[1] * (c[2]*d[8] + d[2]*d[5]) * ent;
{2,5,15'}
  arg := (c[2]*c[8]*c[26] + d[2]*c[5]*c[19]) /
          (c[2]*c[8] + d[2]*c[5]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23] *
          c[1] * (c[2]*c[8] + d[2]*c[5]) * ent;
{2,5,16'}
```

```
  arg := c[7]*c[21] + d[7]*c[17];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[10] * d[1]*d[3] * ent;
{2,5,17'}
  arg := c[9]*c[27] + d[9]*c[25];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[18] * d[1]*c[3] * ent;
{2,5,18'}
  arg := c[5]*c[19] + d[5]*c[13];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[22] * c[1]*d[2] * ent;
{2,5,19'}
  arg := c[8]*c[26] + d[8]*c[23];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[26] * c[1]*c[2] * ent;
{2,5,20'}
  arg := c[17];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10] * d[1]*d[3]*d[7] * ent;
{2,5,21'}
  arg := c[21];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10] * d[1]*d[3]*c[7] * ent;
{2,5,22'}
  arg := c[25];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18] * d[1]*c[3]*d[9] * ent;
{2,5,23'}
  arg := c[27];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18] * d[1]*c[3]*c[9] * ent;
{2,5,24'}
  arg := c[13];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22] * c[1]*d[2]*d[5] * ent;
{2,5,25'}
  arg := c[19];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22] * c[1]*d[2]*c[5] * ent;
{2,5,26'}
  arg := c[23];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26] * c[1]*c[2]*d[8] * ent;
{2,5,27'}
  arg := c[26];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26] * c[1]*c[2]*c[8] * ent;

  avi := avi + snt;

  avi := avi/10;
  V := avi;

  END; {of V(a)}


BEGIN {of prog}

Rewrite (out,'mems4asm.out');

  FOR i := 1 TO 27 DO BEGIN
    a[i] := 0.1;
    c[i] := 0.1;
  END;

  ap := a;
  am := a;

  old := V(a, c);

  oldprev := 1.0;
  i := 0;
  WHILE (Round (100000 * oldprev) <> Round (100000 * old))  DO BEGIN
    oldprev := old;
    i := i + 1;
```

```
  FOR j := 1 TO 27 DO BEGIN
    Write (j:1,' ');
    s := 1 / EXP(LN(10) * i);
    endflag := FALSE;
    plus    := FALSE;
    min     := FALSE;
    REPEAT
      IF NOT (min) THEN
      BEGIN
        ap := a;
        IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
        new := V(ap, c);

        IF new > old THEN BEGIN
          plus := TRUE;
          a[j] := ap[j];
          old := new;
        END ELSE min := TRUE;
      END ELSE
      BEGIN
        IF plus THEN endflag := TRUE ELSE
        BEGIN
          am := a;
          IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
          new := V(am, c);

          IF new > old THEN BEGIN
            a[j] := am[j];
            old := new;
          END ELSE endflag := TRUE;
        END;
      END;

    UNTIL endflag;

    endflag := FALSE;
    plus    := FALSE;
    min     := FALSE;
    REPEAT
      IF NOT (min) THEN
      BEGIN
        cp := c;
        IF c[j] + s < 1 THEN cp[j] := c[j] + s ELSE cp[j] := 1;
        new := V(a, cp);

        IF new > old THEN BEGIN
          plus := TRUE;
          c[j] := cp[j];
          old := new;
        END ELSE min := TRUE;
      END ELSE
      BEGIN
        IF plus THEN endflag := TRUE ELSE
        BEGIN
          cm := c;
          IF c[j] - s > 0 THEN cm[j] := c[j] - s ELSE cm[j] := 0;
          new := V(a, cm);

          IF new > old THEN BEGIN
            c[j] := cm[j];
            old := new;
          END ELSE endflag := TRUE;
        END;
      END;

    UNTIL endflag;

  END; {of j}

  WriteLN;
  WriteLN (out);
  WriteLN (     'i=', i:1, ' R=', old:18:16);
  WriteLN (out, 'i=', i:1, ' R=', old:18:16)

END; {of i}
```

```
  WritelN;
  WritelN (out);
  FOR i := 1 TO 27  DO
    WritelN (     'a[',i:2,']=',a[i]:18:16,' ','c[',i:2,']=',c[i]:18:16);
  FOR i := 1 TO 27  DO
    WritelN (out, 'a[',i:2,']=',a[i]:18:16,' ','c[',i:2,']=',c[i]:18:16);
  WritelN;
  WritelN (out);


END.
```

Appendix 9: Source code mems5.p

```
PROGRAM MEMS5 (input, output);
{A1..A81}

TYPE AR = ArraY [1..81] of Longreal;
VAR i,j,k,l              : Integer;
    old, oldprev, new, s : Longreal;
    endflag              : Boolean;
    a, ap ,am            : AR;
    out                  : Text;

FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a : AR) : Longreal;
  VAR i                  : Integer;
      arg, ent, snt, avi : Longreal;
      b                  : AR;

  BEGIN

  FOR i := 1 TO 81 do b[i] := 1 - a[i];

{input symbol 2}
  arg := a[1];
  ent := H(arg);
  avi := (b[1]*a[3] + a[1]*a[2]) * ent;

{input symbol 3}
  arg := a[1]*a[2] + b[1]*a[3];
  ent := H(arg);
  snt := (b[1]*b[3]*a[7] + a[1]*b[2]*a[5]) * ent;
  arg := a[3];
  ent := H(arg);
  snt := snt + b[1] * (b[1]*a[3]*a[6] + a[1]*a[2]*a[4]) * ent;
  arg := a[2];
  ent := H(arg);
  snt := snt + a[1] * (b[1]*a[3]*a[9] + a[1]*a[2]*a[8]) * ent;
  avi := avi + snt;

{input symbol 4}
{4,1}
  arg := a[1] * (a[2]*a[4] + b[2]*a[5]) + b[1] * (a[3]*a[6] + b[3]*a[7]);
  ent := H(arg);
  snt := b[1]*b[3]*b[7]*a[17] * ent;
{4,2}
  arg := (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) / (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16] * (a[1]*b[2] + b[1]*b[3]) * ent;
{4,3}
  arg := (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]) / (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21] * (a[1]*a[2] + b[1]*a[3]) * ent;
{4,4}
  arg := a[3]*a[6] + b[3]*a[7];
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15] * b[1] * ent;
{4,5}
  arg := a[2]*a[4] + b[2]*a[5];
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25] * a[1] * ent;
{4,6}
  arg := a[7];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14] * b[1]*b[3] * ent;
{4,7}
  arg := a[6];
  ent := H(arg);
```

```
  snt := snt + b[1]*a[3]*a[6]*a[20] * b[1]*a[3] * ent;
{4,8}
  arg := a[5];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24]*a[1]*b[2] * ent;
{4,9}
  arg := a[4];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27]*a[1]*a[2] * ent;

  avi := avi + snt;


{4,1'}
  arg := a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7]);
  ent := H(arg);
  snt := a[1]*b[2]*b[5]*a[13] * ent;
{4,2'}
  arg := (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) / (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12] * (a[1]*b[2] + b[1]*b[3]) * ent;
{4,3'}
  arg := (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]) / (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19] * (a[1]*a[2] + b[1]*a[3]) * ent;
{4,4'}
  arg := a[3]*a[9] + b[3]*a[7];
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11] * b[1] * ent;
{4,5'}
  arg := a[2]*a[8] + b[2]*a[5];
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23] * a[1] * ent;
{4,6'}
  arg := a[7];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10] * b[1]*b[3] * ent;
{4,7'}
  arg := a[9];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18] * b[1]*a[3] * ent;

{4,8'}
  arg := a[5];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22] * a[1]*b[2] * ent;

{4,9'}
  arg := a[8];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26] * a[1]*a[2] * ent;


  avi := avi + snt;

{input symbol 5}
{5,1}
  arg := a[1] * (a[2] * (a[4]*a[10] + b[4]*a[11]) +
                 b[2] * (a[5]*a[12] + b[5]*a[13])) +
         b[1] * (a[3] * (a[6]*a[14] + b[6]*a[15]) +
                 b[3] * (a[7]*a[16] + b[7]*a[17]));
  ent := H(arg);
  snt := b[1]*b[3]*b[7]*b[17]*a[43] * ent;
{5,2}
  arg := (a[1] * (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]) +
          b[1] * (a[3]*b[6]*a[15] + b[3]*b[7]*a[17])) /
         (a[1] * (a[2]*b[4] + b[2]*b[5]) + b[1] * (a[3]*b[6] + b[3]*b[7]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17]*a[42] *
         (a[1] * (a[2]*b[4] + b[2]*b[5]) +
          b[1] * (a[3]*b[6] + b[3]*b[7])) * ent;
{5,3}
  arg := (a[1] * (a[2]*a[4]*a[10] + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[6]*a[14] + b[3]*a[7]*a[16])) /
         (a[1] * (a[2]*a[4] + b[2]*a[5]) + b[1] * (a[3]*a[6] + b[3]*a[7]));
```

```
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17]*a[51] *
         (a[1] * (a[2]*a[4] + b[2]*a[5]) +
          b[1] * (a[3]*a[6] + b[3]*a[7])) * ent;
{5,4}
  arg := (a[1]*b[2] * (a[5]*a[12] + b[5]*a[13]) +
          b[1]*b[3] * (a[7]*a[16] + b[7]*a[17])) /
         (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[16]*a[41] * (a[1]*b[2] + b[1]*b[3]) * ent;
{5,5}
  arg := (a[1]*a[2] * (a[4]*a[10] + b[4]*a[11]) +
          b[1]*a[3] * (a[6]*a[14] + b[6]*a[15])) /
         (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[21]*a[59] * (a[1]*a[2] + b[1]*a[3]) * ent;
{5,6}
  arg := (a[1]*b[2]*b[5]*a[13] + b[1]*b[3]*b[7]*a[17]) /
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16]*a[40] *
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]) * ent;
{5,7}
  arg := (a[1]*b[2]*a[5]*a[12] + b[1]*b[3]*a[7]*a[16]) /
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16]*a[50] *
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) * ent;
{5,8}
  arg := (a[1]*a[2]*b[4]*a[11] + b[1]*a[3]*b[6]*a[15]) /
         (a[1]*a[2]*b[4] + b[1]*a[3]*b[6]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21]*a[58] *
         (a[1]*a[2]*b[4] + b[1]*a[3]*b[6]) * ent;
{5,9}
  arg := (a[1]*a[2]*a[4]*a[10] + b[1]*a[3]*a[6]*a[14]) /
         (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21]*a[63] *
         (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]) * ent;
{5,10}
  arg := a[3] * (a[6]*a[20] + b[6]*a[15]) + b[3] * (a[7]*a[21] + b[7]*a[17]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*b[15]*a[39] * b[1] * ent;
{5,11}
  arg := a[2] * (a[4]*a[18] + b[4]*a[11]) + b[2] * (a[5]*a[19] + b[5]*a[13]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*b[25]*a[71] * a[1] * ent;
{5,12}
  arg := (a[3]*b[6]*a[15] + b[3]*b[7]*a[17]) / (a[3]*b[6] + b[3]*b[7]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15]*a[38] *
         b[1] * (a[3]*b[6] + b[3]*b[7]) * ent;
{5,13}
  arg := (a[3]*a[6]*a[20] + b[3]*a[7]*a[21]) / (a[3]*a[6] + b[3]*a[7]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15]*a[49] *
         b[1] * (a[3]*a[6] + b[3]*a[7]) * ent;
{5,14}
  arg := (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]) / (a[2]*b[4] + b[2]*b[5]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25]*a[70] *
         a[1] * (a[2]*b[4] + b[2]*b[5]) * ent;
{5,15}
  arg := (a[2]*a[4]*a[18] + b[2]*a[5]*a[19]) /  (a[2]*a[4] + b[2]*a[5]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25]*a[75] *
         a[1] *  (a[2]*a[4] + b[2]*a[5]) * ent;
{5,16}
  arg := a[7]*a[21] + b[7]*a[17];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[14]*a[37] * b[1]*b[3] * ent;
{5,17}
  arg := a[6]*a[20] + b[6]*a[15];
  ent := H(arg);
```

```
  snt := snt + b[1]*a[3]*a[6]*b[20]*a[57] * b[1]*a[3] * ent;
{5,18}
  arg := a[5]*a[19] + b[5]*a[13];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[24]*a[69] * a[1]*b[2] * ent;
{5,19}
  arg := a[4]*a[18] + b[4]*a[11];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[27]*a[79] * a[1]*a[2] * ent;
{5,20}
  arg := a[17];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14]*a[36] * b[1]*b[3]*b[7] * ent;
{5,21}
  arg := a[21];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14]*a[48] * b[1]*b[3]*a[7] * ent;
{5,22}
  arg := a[15];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20]*a[56] * b[1]*a[3]*b[6] * ent;
{5,23}
  arg := a[20];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20]*a[62] * b[1]*a[3]*a[6] * ent;
{5,24}
  arg := a[13];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24]*a[68] * a[1]*b[2]*b[5] * ent;
{5,25}
  arg := a[19];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24]*a[74] * a[1]*b[2]*a[5] * ent;
{5,26}
  arg := a[11];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27]*a[78] * a[1]*a[2]*b[4] * ent;
{5,27}
  arg := a[18];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27]*a[81] * a[1]*a[2]*a[4] * ent;

avi := avi + snt;


{5,1'}
  arg := a[1] * (a[2] * (a[8]*a[22] + b[8]*a[23]) +
                 b[2] * (a[5]*a[12] + b[5]*a[13])) +
         b[1] * (a[3] * (a[9]*a[24] + b[9]*a[25]) +
                 b[3] * (a[7]*a[16] + b[7]*a[17]));
  ent := H(arg);
  snt := a[1]*b[2]*b[5]*b[13]*a[35] * ent;
{5,2'}
  arg := (a[1] * (a[2]*b[8]*a[23] + b[2]*b[5]*a[13]) +
          b[1] * (a[3]*b[9]*a[25] + b[3]*b[7]*a[17])) /
         (a[1] * (a[2]*b[8] + b[2]*b[5]) + b[1] * (a[3]*b[9] + b[3]*b[7]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13]*a[34] *
         (a[1] * (a[2]*b[8] + b[2]*b[5]) + b[1] * (a[3]*b[9] + b[3]*b[7])) *
         ent;
{5,3'}
  arg := (a[1] * (a[2]*a[8]*a[22] + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[9]*a[24] + b[3]*a[7]*a[16])) /
         (a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13]*a[47] *
         (a[1] * (a[2]*a[8] + b[2]*a[5]) + b[1] * (a[3]*a[9] + b[3]*a[7])) *
         ent;
{5,4'}
  arg := (a[1]*b[2] * (a[5]*a[12] + b[5]*a[13]) +
          b[1]*b[3] * (a[7]*a[16] + b[7]*a[17])) /
         (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[12]*a[33] * (a[1]*b[2] + b[1]*b[3]) * ent;
{5,5'}
```

```
  arg := (a[1]*a[2] * (a[8]*a[22] + b[8]*a[23]) +
          b[1]*a[3] * (a[9]*a[24] + b[9]*a[25])) /
         (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[19]*a[55] * (a[1]*a[2] + b[1]*a[3]) * ent;
{5,6'}
  arg := (a[1]*b[2]*b[5]*a[13] + b[1]*b[3]*b[7]*a[17]) /
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12]*a[32] *
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]) * ent;
{5,7'}
  arg := (a[1]*b[2]*a[5]*a[12] + b[1]*b[3]*a[7]*a[16]) /
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12]*a[46] *
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) * ent;
{5,8'}
  arg := (a[1]*a[2]*b[8]*a[23] + b[1]*a[3]*b[9]*a[25]) /
         (a[1]*a[2]*b[8] + b[1]*a[3]*b[9]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19]*a[54] *
         (a[1]*a[2]*b[8] + b[1]*a[3]*b[9]) * ent;
{5,9'}
  arg := (a[1]*a[2]*a[8]*a[22] +b[1]*a[3]*a[9]*a[24]) /
         (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19]*a[61] *
         (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]) * ent;
{5,10'}
  arg := a[3] * (a[9]*a[27] + b[9]*a[25]) +
         b[3] * (a[7]*a[21] + b[7]*a[17]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*b[11]*a[31] * b[1] * ent;
{5,11'}
  arg := a[2] * (a[8]*a[26] + b[8]*a[23]) +
         b[2] * (a[5]*a[19] + b[5]*a[13]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*b[23]*a[67] * a[1] * ent;
{5,12'}
  arg := (a[3]*b[9]*a[25] + b[3]*b[7]*a[17]) / (a[3]*b[9] + b[3]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11]*a[30] *
         b[1] * (a[3]*b[9] + b[3]*b[7]) * ent;
{5,13'}
  arg := (a[3]*a[9]*a[27] + b[3]*a[7]*a[21]) /
         (a[3]*a[9] + b[3]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11]*a[45] *
         b[1] * (a[3]*a[9] + b[3]*a[7]) * ent;
{5,14'}
  arg := (a[2]*b[8]*a[23] + b[2]*b[5]*a[13]) /
         (a[2]*b[8] + b[2]*b[5]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23]*a[66] *
         a[1] * (a[2]*b[8] + b[2]*b[5]) * ent;
{5,15'}
  arg := (a[2]*a[8]*a[26] + b[2]*a[5]*a[19]) /
         (a[2]*a[8] + b[2]*a[5]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23]*a[73] *
         a[1] * (a[2]*a[8] + b[2]*a[5]) * ent;
{5,16'}
  arg := a[7]*a[21] + b[7]*a[17];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[10]*a[29] * b[1]*b[3] * ent;
{5,17'}
  arg := a[9]*a[27] + b[9]*a[25];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[18]*a[53] * b[1]*a[3] * ent;
{5,18'}
  arg := a[5]*a[19] + b[5]*a[13];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[22]*a[65] * a[1]*b[2] * ent;
{5,19'}
```

```
  arg := a[8]*a[26] + b[8]*a[23];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[26]*a[77] * a[1]*a[2] * ent;
{5,20'}
  arg := a[17];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10]*a[28] * b[1]*b[3]*b[7] * ent;
{5,21'}
  arg := a[21];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10]*a[44] * b[1]*b[3]*a[7] * ent;
{5,22'}
  arg := a[25];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18]*a[52] * b[1]*a[3]*b[9] * ent;
{5,23'}
  arg := a[27];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18]*a[60] * b[1]*a[3]*a[9] * ent;
{5,24'}
  arg := a[13];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22]*a[64] * a[1]*b[2]*b[5] * ent;
{5,25'}
  arg := a[19];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22]*a[72] * a[1]*b[2]*a[5] * ent;
{5,26'}
  arg := a[23];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26]*a[76] * a[1]*a[2]*b[8] * ent;
{5,27'}
  arg := a[26];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26]*a[80] * a[1]*a[2]*a[8] * ent;

  avi := avi + snt;

{input symbol 6}
{6,1}
  arg := a[1] * (a[2] * (a[4] * (a[10]*a[28] + b[10]*a[29]) +
                         b[4] * (a[11]*a[30] + b[11]*a[31])) +
                 b[2] * (a[5] * (a[12]*a[32] + b[12]*a[33]) +
                         b[5] * (a[13]*a[34] + b[13]*a[35]))) +
         b[1] * (a[3] * (a[6] * (a[14]*a[36] + b[14]*a[37]) +
                         b[6] * (a[15]*a[38] + b[15]*a[39])) +
                 b[3] * (a[7] * (a[16]*a[40] + b[16]*a[41]) +
                         b[7] * (a[17]*a[42] + b[17]*a[43])));
  ent := H(arg);
  snt := b[1]*b[3]*b[7]*b[17]*b[43] * ent;
{6,2}
  arg := (a[1] * (a[2] * (a[4]*b[10]*a[29] + b[4]*b[11]*a[31]) +
                  b[2] * (a[5]*b[12]*a[33] + b[5]*b[13]*a[35])) +
          b[1] * (a[3] * (a[6]*b[14]*a[37] + b[6]*b[15]*a[39]) +
                  b[3] * (a[7]*b[16]*a[41] + b[7]*b[17]*a[43]))) /
         (a[1] * (a[2] * (a[4]*b[10] + b[4]*b[11]) +
                  b[2] * (a[5]*b[12] + b[5]*b[13])) +
          b[1] * (a[3] * (a[6]*b[14] + b[6]*b[15]) +
                  b[3] * (a[7]*b[16] + b[7]*b[17]))));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*b[17]*a[43] *
         (a[1] * (a[2] * (a[4]*b[10] + b[4]*b[11]) +
                  b[2] * (a[5]*b[12] + b[5]*b[13])) +
          b[1] * (a[3] * (a[6]*b[14] + b[6]*b[15]) +
                  b[3] * (a[7]*b[16] + b[7]*b[17]))) * ent;
{6,3}
  arg := (a[1] * (a[2] * (a[4]*a[10]*a[28] + b[4]*a[11]*a[30]) +
                  b[2] * (a[5]*a[12]*a[32] + b[5]*a[13]*a[34])) +
          b[1] * (a[3] * (a[6]*a[14]*a[36] + b[6]*a[15]*a[38]) +
                  b[3] * (a[7]*a[16]*a[40] + b[7]*a[17]*a[42]))) /
         (a[1] * (a[2] * (a[4]*a[10] + b[4]*a[11]) +
                  b[2] * (a[5]*a[12] + b[5]*a[13])) +
          b[1] * (a[3] * (a[6]*a[14] + b[6]*a[15]) +
                  b[3] * (a[7]*a[16] + b[7]*a[17]))));
  ent := H(arg);
```

```
  snt := snt + b[1]*b[3]*b[7]*b[17]*a[43] *
         (a[1] * (a[2] * (a[4]*a[10] + b[4]*a[11]) +
                  b[2] * (a[5]*a[12] + b[5]*a[13])) +
          b[1] * (a[3] * (a[6]*a[14] + b[6]*a[15]) +
                  b[3] * (a[7]*a[16] + b[7]*a[17]))) * ent;
{6,4}
  arg := (a[1] * (a[2]*b[4] * (a[11]*a[30] + b[11]*a[31]) +
                  b[2]*b[5] * (a[13]*a[34] + b[13]*a[35])) +
          b[1] * (a[3]*b[6] * (a[15]*a[38] + b[15]*a[39]) +
                  b[3]*b[7] * (a[17]*a[42] + b[17]*a[43]))) /
         (a[1] * (a[2]*b[4] + b[2]*b[5]) + b[1] * (a[3]*b[6] + b[3]*b[7]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17]*b[42] *
         (a[1] * (a[2]*b[4] + b[2]*b[5]) + b[1] * (a[3]*b[6] + b[3]*b[7])) *
         ent;
{6,5}
  arg := (a[1] * (a[2]*a[4] * (a[10]*a[28] + b[10]*a[29]) +
                  b[2]*a[5] * (a[12]*a[32] + b[12]*a[33])) +
          b[1] * (a[3]*a[6] * (a[14]*a[36] + b[14]*a[37]) +
                  b[3]*a[7] * (a[16]*a[40] + b[16]*a[41]))) /
         (a[1] * (a[2]*a[4] + b[2]*a[5]) + b[1] * (a[3]*a[6] + b[3]*a[7]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17]*b[51] *
         (a[1] * (a[2]*a[4] + b[2]*a[5]) + b[1] * (a[3]*a[6] + b[3]*a[7])) *
         ent;
{6,6}
  arg := (a[1] * (a[2]*b[4]*b[11]*a[31] + b[2]*b[5]*b[13]*a[35]) +
          b[1] * (a[3]*b[6]*b[15]*a[39] + b[3]*b[7]*b[17]*a[43])) /
         (a[1] * (a[2]*b[4]*b[11] + b[2]*b[5]*b[13]) +
          b[1] * (a[3]*b[6]*b[15] + b[3]*b[7]*b[17]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17]*a[42] *
         (a[1] * (a[2]*b[4]*b[11] + b[2]*b[5]*b[13]) +
          b[1] * (a[3]*b[6]*b[15] + b[3]*b[7]*b[17])) * ent;
{6,7}
  arg := (a[1] * (a[2]*b[4]*a[11]*a[30] + b[2]*b[5]*a[13]*a[34]) +
          b[1] * (a[3]*b[6]*a[15]*a[38] + b[3]*b[7]*a[17]*a[42])) /
         (a[1] * (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]) +
          b[1] * (a[3]*b[6]*a[15] + b[3]*b[7]*a[17]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17]*a[42] *
         (a[1] * (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]) +
          b[1] * (a[3]*b[6]*a[15] + b[3]*b[7]*a[17])) * ent;
{6,8}
  arg := (a[1] * (a[2]*a[4]*b[10]*a[29] + b[2]*a[5]*b[12]*a[33]) +
          b[1] * (a[3]*a[6]*b[14]*a[37] + b[3]*a[7]*b[16]*a[41])) /
         (a[1] * (a[2]*a[4]*b[10] + b[2]*a[5]*b[12]) +
          b[1] * (a[3]*a[6]*b[14] + b[3]*a[7]*b[16]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17]*a[51] *
         (a[1] * (a[2]*a[4]*b[10] + b[2]*a[5]*b[12]) +
          b[1] * (a[3]*a[6]*b[14] + b[3]*a[7]*b[16])) * ent;
{6,9}
  arg := (a[1] * (a[2]*a[4]*a[10]*a[28] + b[2]*a[5]*a[12]*a[32]) +
          b[1] * (a[3]*a[6]*a[14]*a[36] + b[3]*a[7]*a[16]*a[40])) /
         (a[1] * (a[2]*a[4]*a[10] + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[6]*a[14] + b[3]*a[7]*a[16]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*b[7]*a[17]*a[51] *
         (a[1] * (a[2]*a[4]*a[10] + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[6]*a[14] + b[3]*a[7]*a[16])) * ent;
{6,10}
  arg := (a[1]*b[2] * (a[5] * (a[12]*a[46] + b[12]*a[33]) +
                       b[5] * (a[13]*a[47] + b[13]*a[35])) +
          b[1]*b[3] * (a[7] * (a[16]*a[50] + b[16]*a[41]) +
                       b[7] * (a[17]*a[51] + b[17]*a[43]))) /
          (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[16]*b[41] * (a[1]*b[2] + b[1]*b[3]) * ent;
{6,11}
  arg := (a[1]*a[2] * (a[4] * (a[10]*a[44] + b[10]*a[29]) +
                       b[4] * (a[11]*a[45] + b[11]*a[31])) +
          b[1]*a[3] * (a[6] * (a[14]*a[48] + b[14]*a[37]) +
                       b[6] * (a[15]*a[49] + b[15]*a[39]))) /
          (a[1]*a[2] + b[1]*a[3]);
```

```
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[21]*b[59] * (a[1]*a[2] + b[1]*a[3]) * ent;
{6,12}
  arg := (a[1]*b[2] * (a[5]*b[12]*a[33] + b[5]*b[13]*a[35]) +
          b[1]*b[3] * (a[7]*b[16]*a[41] + b[7]*b[17]*a[43])) /
         (a[1]*b[2] * (a[5]*b[12] + b[5]*b[13]) +
          b[1]*b[3] * (a[7]*b[16] + b[7]*b[17]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[16]*a[41] *
         (a[1]*b[2] * (a[5]*b[12] + b[5]*b[13]) +
          b[1]*b[3] * (a[7]*b[16] + b[7]*b[17])) * ent;
{6,13}
  arg := (a[1]*b[2] * (a[5]*a[12]*a[46] + b[5]*a[13]*a[47]) +
          b[1]*b[3] * (a[7]*a[16]*a[50] + b[7]*a[17]*a[51])) /
         (a[1]*b[2] * (a[5]*a[12] + b[5]*a[13]) +
          b[1]*b[3] * (a[7]*a[16] + b[7]*a[17]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[16]*a[41] *
         (a[1]*b[2] * (a[5]*a[12] + b[5]*a[13]) +
          b[1]*b[3] * (a[7]*a[16] + b[7]*a[17])) * ent;
{6,14}
  arg := (a[1]*a[2] * (a[4]*b[10]*a[29] + b[4]*b[11]*a[31]) +
          b[1]*a[3] * (a[6]*b[14]*a[37] + b[6]*b[15]*a[39])) /
         (a[1]*a[2] * (a[4]*b[10] + b[4]*b[11]) +
          b[1]*a[3] * (a[6]*b[14] + b[6]*b[15]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[21]*a[59] *
         (a[1]*a[2] * (a[4]*b[10] + b[4]*b[11]) +
          b[1]*a[3] * (a[6]*b[14] + b[6]*b[15])) * ent;
{6,15}
  arg := (a[1]*a[2] * (a[4]*a[10]*a[44] + b[4]*a[11]*a[45]) +
          b[1]*a[3] * (a[6]*a[14]*a[48] + b[6]*a[15]*a[49])) /
         (a[1]*a[2] * (a[4]*a[10] + b[4]*a[11]) +
          b[1]*a[3] * (a[6]*a[14] + b[6]*a[15]));
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*b[21]*a[59] *
         (a[1]*a[2] * (a[4]*a[10] + b[4]*a[11]) +
          b[1]*a[3] * (a[6]*a[14] + b[6]*a[15])) * ent;
{6,16}
  arg := (a[1]*b[2]*b[5] * (a[13]*a[47] + b[13]*a[35]) +
          b[1]*b[3]*b[7] * (a[17]*a[51] + b[17]*a[43])) /
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16]*b[40] *
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]) * ent;
{6,17}
  arg := (a[1]*b[2]*a[5] * (a[12]*a[46] + b[12]*a[33]) +
          b[1]*b[3]*a[7] * (a[16]*a[50] + b[16]*a[41])) /
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16]*b[50] *
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) * ent;
{6,18}
  arg := (a[1]*a[2]*b[4] * (a[11]*a[45] + b[11]*a[31]) +
          b[1]*a[3]*b[6] * (a[15]*a[49] + b[15]*a[39])) /
         (a[1]*a[2]*b[4] + b[1]*a[3]*b[6]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21]*b[58] *
         (a[1]*a[2]*b[4] + b[1]*a[3]*b[6]) * ent;
{6,19}
  arg := (a[1]*a[2]*a[4] * (a[10]*a[44] + b[10]*a[29]) +
          b[1]*a[3]*a[6] * (a[14]*a[48] + b[14]*a[37])) /
         (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21]*b[63] *
         (a[1]*a[2]*a[4] + b[1]*a[3]*a[6]) * ent;
{6,20}
  arg := (a[1]*b[2]*b[5]*b[13]*a[35] + b[1]*b[3]*b[7]*b[17]*a[43]) /
         (a[1]*b[2]*b[5]*b[13] + b[1]*b[3]*b[7]*b[17]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16]*a[40] *
         (a[1]*b[2]*b[5]*b[13] + b[1]*b[3]*b[7]*b[17]) * ent;
{6,21}
  arg := (a[1]*b[2]*b[5]*a[13]*a[47] + b[1]*b[3]*b[7]*a[17]*a[51]) /
         (a[1]*b[2]*b[5]*a[13] + b[1]*b[3]*b[7]*a[17]);
```

```
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16]*a[40] *
         (a[1]*b[2]*b[5]*a[13] + b[1]*b[3]*b[7]*a[17]) * ent;
{6,22}
  arg := (a[1]*b[2]*a[5]*b[12]*a[33] + b[1]*b[3]*a[7]*b[16]*a[41]) /
         (a[1]*b[2]*a[5]*b[12] + b[1]*b[3]*a[7]*b[16]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16]*a[50] *
         (a[1]*b[2]*a[5]*b[12] + b[1]*b[3]*a[7]*b[16]) * ent;
{6,23}
  arg := (a[1]*b[2]*a[5]*a[12]*a[46] + b[1]*b[3]*a[7]*a[16]*a[50]) /
         (a[1]*b[2]*a[5]*a[12] + b[1]*b[3]*a[7]*a[16]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[16]*a[50] *
         (a[1]*b[2]*a[5]*a[12] + b[1]*b[3]*a[7]*a[16]) * ent;
{6,24}
  arg := (a[1]*a[2]*b[4]*b[11]*a[31] + b[1]*a[3]*b[6]*b[15]*a[39]) /
         (a[1]*a[2]*b[4]*b[11] + b[1]*a[3]*b[6]*b[15]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21]*a[58] *
         (a[1]*a[2]*b[4]*b[11] + b[1]*a[3]*b[6]*b[15]) * ent;
{6,25}
  arg := (a[1]*a[2]*b[4]*a[11]*a[45] + b[1]*a[3]*b[6]*a[15]*a[49]) /
         (a[1]*a[2]*b[4]*a[11] + b[1]*a[3]*b[6]*a[15]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21]*a[58] *
         (a[1]*a[2]*b[4]*a[11] + b[1]*a[3]*b[6]*a[15]) * ent;
{6,26}
  arg := (a[1]*a[2]*a[4]*b[10]*a[29] + b[1]*a[3]*a[6]*b[14]*a[37]) /
         (a[1]*a[2]*a[4]*b[10] + b[1]*a[3]*a[6]*b[14]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21]*a[63] *
         (a[1]*a[2]*a[4]*b[10] + b[1]*a[3]*a[6]*b[14]) * ent;
{6,27}
  arg := (a[1]*a[2]*a[4]*a[10]*a[44] + b[1]*a[3]*a[6]*a[14]*a[48]) /
         (a[1]*a[2]*a[4]*a[10] + b[1]*a[3]*a[6]*a[14]);
  ent := H(arg);
  snt := snt + b[1]*b[3]*a[7]*a[21]*a[63] *
         (a[1]*a[2]*a[4]*a[10] + b[1]*a[3]*a[6]*a[14]) * ent;
{6,28}
  arg := a[3] * (a[6] * (a[20]*a[56] + b[20]*a[57]) +
                 b[6] * (a[15]*a[38] + b[15]*a[39])) +
         b[3] * (a[7] * (a[21]*a[58] + b[21]*a[59]) +
                 b[7] * (a[17]*a[42] + b[17]*a[43]));
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*b[15]*b[39] * b[1] * ent;
{6,29}
  arg := a[2] * (a[4] * (a[18]*a[52] + b[18]*a[53]) +
                 b[4] * (a[11]*a[30] + b[11]*a[31])) +
         b[2] * (a[5] * (a[19]*a[54] + b[19]*a[55]) +
                 b[5] * (a[13]*a[34] + b[13]*a[35]));
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*b[25]*b[71] * a[1] * ent;
{6,30}
  arg := (a[3] * (a[6]*b[20]*a[57] + b[6]*b[15]*a[39]) +
          b[3] * (a[7]*b[21]*a[59] + b[7]*b[17]*a[43])) /
         (a[3] * (a[6]*b[20] + b[6]*b[15]) +
          b[3] * (a[7]*b[21] + b[7]*b[17]));
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*b[15]*a[39] *
         b[1] * (a[3] * (a[6]*b[20] + b[6]*b[15]) +
                 b[3] * (a[7]*b[21] + b[7]*b[17])) * ent;
{6,31}
  arg := (a[3] * (a[6]*a[20]*a[56] + b[6]*a[15]*a[38]) +
          b[3] * (a[7]*a[21]*a[58] + b[7]*a[17]*a[42])) /
         (a[3] * (a[6]*a[20] + b[6]*a[15]) +
          b[3] * (a[7]*a[21] + b[7]*a[17]));
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*b[15]*a[39] * b[1] *
         (a[3] * (a[6]*a[20] + b[6]*a[15]) +
          b[3] * (a[7]*a[21] + b[7]*a[17])) * ent;
{6,32}
  arg := (a[2] * (a[4]*b[18]*a[53] + b[4]*b[11]*a[31]) +
          b[2] * (a[5]*b[19]*a[55] + b[5]*b[13]*a[35])) /
         (a[2] * (a[4]*b[18] + b[4]*b[11]) +
```

```
          b[2] * (a[5]*b[19] + b[5]*b[13]));
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*b[25]*a[71] * a[1] *
         (a[2] * (a[4]*b[18] + b[4]*b[11]) +
          b[2] * (a[5]*b[19] + b[5]*b[13])) * ent;
{6,33}
  arg := (a[2] * (a[4]*a[18]*a[52] + b[4]*a[11]*a[30]) +
          b[2] * (a[5]*a[19]*a[54] + b[5]*a[13]*a[34])) /
         (a[2] * (a[4]*a[18] + b[4]*a[11]) +
          b[2] * (a[5]*a[19] + b[5]*a[13]));
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*b[25]*a[71] * a[1] *
         (a[2] * (a[4]*a[18] + b[4]*a[11]) +
          b[2] * (a[5]*a[19] + b[5]*a[13])) * ent;
{6,34}
  arg := (a[3]*b[6] * (a[15]*a[38] + b[15]*a[39]) +
          b[3]*b[7] * (a[17]*a[42] + b[17]*a[43])) /
         (a[3]*b[6] + b[3]*b[7]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15]*b[38] * b[1] *
         (a[3]*b[6] + b[3]*b[7]) * ent;
{6,35}
  arg := (a[3]*a[6] * (a[20]*a[56] + b[20]*a[57]) +
          b[3]*a[7] * (a[21]*a[58] + b[21]*a[59])) /
         (a[3]*a[6] + b[3]*a[7]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15]*b[49] * b[1] *
         (a[3]*a[6] + b[3]*a[7]) * ent;
{6,36}
  arg := (a[2]*b[4] * (a[11]*a[30] + b[11]*a[31]) +
          b[2]*b[5] * (a[13]*a[34] + b[13]*a[35])) /
         (a[2]*b[4] + b[2]*b[5]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25]*b[70] * a[1] *
         (a[2]*b[4] + b[2]*b[5]) * ent;
{6,37}
  arg := (a[2]*a[4] * (a[18]*a[52] + b[18]*a[53]) +
          b[2]*a[5] * (a[19]*a[54] + b[19]*a[55])) /
         (a[2]*a[4] + b[2]*a[5]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25]*b[75] * a[1] *
         (a[2]*a[4] + b[2]*a[5]) * ent;
{6,38}
  arg := (a[3]*b[6]*b[15]*a[39] + b[3]*b[7]*b[17]*a[43]) /
         (a[3]*b[6]*b[15] + b[3]*b[7]*b[17]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15]*a[38] * b[1] *
         (a[3]*b[6]*b[15] + b[3]*b[7]*b[17]) * ent;
{6,39}
  arg := (a[3]*b[6]*a[15]*a[38] + b[3]*b[7]*a[17]*a[42]) /
         (a[3]*b[6]*a[15] + b[3]*b[7]*a[17]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15]*a[38] * b[1] *
         (a[3]*b[6]*a[15] + b[3]*b[7]*a[17]) * ent;
{6,40}
  arg := (a[3]*a[6]*b[20]*a[57] + b[3]*a[7]*b[21]*a[59]) /
         (a[3]*a[6]*b[20] + b[3]*a[7]*b[21]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15]*a[49] * b[1] *
         (a[3]*a[6]*b[20] + b[3]*a[7]*b[21]) * ent;
{6,41}
  arg := (a[3]*a[6]*a[20]*a[56] + b[3]*a[7]*a[21]*a[58]) /
         (a[3]*a[6]*a[20] + b[3]*a[7]*a[21]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[6]*a[15]*a[49] * b[1] *
         (a[3]*a[6]*a[20] + b[3]*a[7]*a[21]) * ent;
{6,42}
  arg := (a[2]*b[4]*b[11]*a[31] + b[2]*b[5]*b[13]*a[35]) /
         (a[2]*b[4]*b[11] + b[2]*b[5]*b[13]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25]*a[70] * a[1] *
         (a[2]*b[4]*b[11] + b[2]*b[5]*b[13]) * ent;
{6,43}
  arg := (a[2]*b[4]*a[11]*a[30] + b[2]*b[5]*a[13]*a[34]) /
         (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]);
```

```
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25]*a[70] * a[1] *
         (a[2]*b[4]*a[11] + b[2]*b[5]*a[13]) * ent;
{6,44}
  arg := (a[2]*a[4]*b[18]*a[53] + b[2]*a[5]*b[19]*a[55]) /
         (a[2]*a[4]*b[18] + b[2]*a[5]*b[19]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25]*a[75] * a[1] *
         (a[2]*a[4]*b[18] + b[2]*a[5]*b[19]) * ent;
{6,45}
  arg := (a[2]*a[4]*a[18]*a[52] + b[2]*a[5]*a[19]*a[54]) /
         (a[2]*a[4]*a[18] + b[2]*a[5]*a[19]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*b[9]*a[25]*a[75] * a[1] *
         (a[2]*a[4]*a[18] + b[2]*a[5]*a[19]) * ent;
{6,46}
  arg := a[7] * (a[21]*a[63] + b[21]*a[59]) +
         b[7] * (a[17]*a[51] + b[17]*a[43]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[14]*b[37] * b[1]*b[3] * ent;
{6,47}
  arg := a[6] * (a[20]*a[62] + b[20]*a[57]) +
         b[6] * (a[15]*a[49] + b[15]*a[39]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[20]*b[57] * b[1]*a[3] * ent;
{6,48}
  arg := a[5] * (a[19]*a[61] + b[19]*a[55]) +
         b[5] * (a[13]*a[47] + b[13]*a[35]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[24]*b[69] * a[1]*b[2] * ent;
{6,49}
  arg := a[4] * (a[18]*a[60] + b[18]*a[53]) +
         b[4] * (a[11]*a[45] + b[11]*a[31]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[27]*b[79] * a[1]*a[2] * ent;
{6,50}
  arg := (a[7]*b[21]*a[59] + b[7]*b[17]*a[43]) / (a[7]*b[21] + b[7]*b[17]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[14]*a[37] * b[1]*b[3] *
         (a[7]*b[21] + b[7]*b[17]) * ent;
{6,51}
  arg := (a[7]*a[21]*a[63] + b[7]*a[17]*a[51]) / (a[7]*a[21] + b[7]*a[17]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[14]*a[37] * b[1]*b[3] *
         (a[7]*a[21] + b[7]*a[17]) * ent;
{6,52}
  arg := (a[6]*b[20]*a[57] + b[6]*b[15]*a[39]) / (a[6]*b[20] + b[6]*b[15]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[20]*a[57] * b[1]*a[3] *
          (a[6]*b[20] + b[6]*b[15]) * ent;
{6,53}
  arg := (a[6]*a[20]*a[62] + b[6]*a[15]*a[49]) / (a[6]*a[20] + b[6]*a[15]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*b[20]*a[57] * b[1]*a[3] *
          (a[6]*a[20] + b[6]*a[15]) * ent;
{6,54}
  arg := (a[5]*b[19]*a[55] + b[5]*b[13]*a[35]) / (a[5]*b[19] + b[5]*b[13]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[24]*a[69] * a[1]*b[2] *
         (a[5]*b[19] + b[5]*b[13]) * ent;
{6,55}
  arg := (a[5]*a[19]*a[61] + b[5]*a[13]*a[47]) / (a[5]*a[19] + b[5]*a[13]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[24]*a[69] * a[1]*b[2] *
         (a[5]*a[19] + b[5]*a[13]) * ent;
{6,56}
  arg := (a[4]*b[18]*a[53] + b[4]*b[11]*a[31]) / (a[4]*b[18] + b[4]*b[11]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[27]*a[79] * a[1]*a[2] *
         (a[4]*b[18] + b[4]*b[11]) * ent;
{6,57}
  arg := (a[4]*a[18]*a[60] + b[4]*a[11]*a[45]) / (a[4]*a[18] + b[4]*a[11]);
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*b[27]*a[79] * a[1]*a[2] *
         (a[4]*a[18] + b[4]*a[11]) * ent;
```

```
{6,58}
  arg := a[17]*a[51] + b[17]*a[43];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14]*b[36] * b[1]*b[3]*b[7] * ent;
{6,59}
  arg := a[21]*a[63] + b[21]*a[59];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14]*b[48] * b[1]*b[3]*a[7] * ent;
{6,60}
  arg := a[15]*a[49] + b[15]*a[39];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20]*b[56] * b[1]*a[3]*b[6] * ent;
{6,61}
  arg := a[20]*a[62] + b[20]*a[57];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20]*b[62] * b[1]*a[3]*a[6] * ent;
{6,62}
  arg := a[13]*a[47] + b[13]*a[35];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24]*b[68] * a[1]*b[2]*b[5] * ent;
{6,63}
  arg := a[19]*a[61] + b[19]*a[55];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24]*b[74] * a[1]*b[2]*a[5] * ent;
{6,64}
  arg := a[11]*a[45] + b[11]*a[31];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27]*b[78] * a[1]*a[2]*b[4] * ent;
{6,65}
  arg := a[18]*a[60] + b[18]*a[53];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27]*b[81] * a[1]*a[2]*a[4] * ent;
{6,66}
  arg := a[43];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14]*a[36] * b[1]*b[3]*b[7]*b[17] * ent;
{6,67}
  arg := a[51];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14]*a[36] * b[1]*b[3]*b[7]*a[17] * ent;
{6,68}
  arg := a[59];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14]*a[48] * b[1]*b[3]*a[7]*b[21] * ent;
{6,69}
  arg := a[63];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[14]*a[48] * b[1]*b[3]*a[7]*a[21] * ent;
{6,70}
  arg := a[39];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20]*a[56] * b[1]*a[3]*b[6]*b[15] * ent;
{6,71}
  arg := a[49];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20]*a[56] * b[1]*a[3]*b[6]*a[15] * ent;
{6,72}
  arg := a[57];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20]*a[62] * b[1]*a[3]*a[6]*b[20] * ent;
{6,73}
  arg := a[62];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[6]*a[20]*a[62] * b[1]*a[3]*a[6]*a[20] * ent;
{6,74}
  arg := a[35];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24]*a[68] * a[1]*b[2]*b[5]*b[13] * ent;
{6,75}
  arg := a[47];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24]*a[68] * a[1]*b[2]*b[5]*a[13] * ent;
{6,76}
  arg := a[55];
  ent := H(arg);
```

```
  snt := snt + b[1]*a[3]*a[9]*a[24]*a[74] * a[1]*b[2]*a[5]*b[19] * ent;
{6,77}
  arg := a[61];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[24]*a[74] * a[1]*b[2]*a[5]*a[19] * ent;
{6,78}
  arg := a[31];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27]*a[78] * a[1]*a[2]*b[4]*b[11] * ent;
{6,79}
  arg := a[45];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27]*a[78] * a[1]*a[2]*b[4]*a[11] * ent;
{6,80}
  arg := a[53];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27]*a[81] * a[1]*a[2]*a[4]*b[18] * ent;
{6,81}
  arg := a[60];
  ent := H(arg);
  snt := snt + b[1]*a[3]*a[9]*a[27]*a[81] * a[1]*a[2]*a[4]*a[18] * ent;

  avi := avi + snt;


{6,1'}
  arg := a[1] * (a[2] * (a[8] * (a[22]*a[64] + b[22]*a[65]) +
                         b[8] * (a[23]*a[66] + b[23]*a[67])) +
                 b[2] * (a[5] * (a[12]*a[32] + b[12]*a[33]) +
                         b[5] * (a[13]*a[34] + b[13]*a[35]))) +
         b[1] * (a[3] * (a[9] * (a[24]*a[68] + b[24]*a[69]) +
                         b[9] * (a[25]*a[70] + b[25]*a[71])) +
                 b[3] * (a[7] * (a[16]*a[40] + b[16]*a[41]) +
                         b[7] * (a[17]*a[42] + b[17]*a[43])));
  ent := H(arg);
  snt := a[1]*b[2]*b[5]*b[13]*b[35] * ent;
{6,2'}
  arg := (a[1] * (a[2] * (a[8]*b[22]*a[65] + b[8]*b[23]*a[67]) +
                  b[2] * (a[5]*b[12]*a[33] + b[5]*b[13]*a[35])) +
          b[1] * (a[3] * (a[9]*b[24]*a[69] + b[9]*b[25]*a[71]) +
                  b[3] * (a[7]*b[16]*a[41] + b[7]*b[17]*a[43]))) /
         (a[1] * (a[2] * (a[8]*b[22]       + b[8]*b[23]) +
                  b[2] * (a[5]*b[12]       + b[5]*b[13])) +
          b[1] * (a[3] * (a[9]*b[24]       + b[9]*b[25]) +
                  b[3] * (a[7]*b[16]       + b[7]*b[17])));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*b[13]*a[35] *
         (a[1] * (a[2] * (a[8]*b[22]       + b[8]*b[23]) +
                  b[2] * (a[5]*b[12]       + b[5]*b[13])) +
          b[1] * (a[3] * (a[9]*b[24]       + b[9]*b[25]) +
                  b[3] * (a[7]*b[16]       + b[7]*b[17]))) * ent;
{6,3'}
  arg := (a[1] * (a[2] * (a[8]*a[22]*a[64] + b[8]*a[23]*a[66]) +
                  b[2] * (a[5]*a[12]*a[32] + b[5]*a[13]*a[34])) +
          b[1] * (a[3] * (a[9]*a[24]*a[68] + b[9]*a[25]*a[70]) +
                  b[3] * (a[7]*a[16]*a[40] + b[7]*a[17]*a[42]))) /
         (a[1] * (a[2] * (a[8]*a[22]       + b[8]*a[23]) +
                  b[2] * (a[5]*a[12]       + b[5]*a[13])) +
          b[1] * (a[3] * (a[9]*a[24]       + b[9]*a[25]) +
                  b[3] * (a[7]*a[16]       + b[7]*a[17])));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*b[13]*a[35] *
         (a[1] * (a[2] * (a[8]*a[22]       + b[8]*a[23]) +
                  b[2] * (a[5]*a[12]       + b[5]*a[13])) +
          b[1] * (a[3] * (a[9]*a[24]       + b[9]*a[25]) +
                  b[3] * (a[7]*a[16]       + b[7]*a[17]))) * ent;
{6,4'}
  arg := (a[1] * (a[2]*b[8] * (a[23]*a[66] + b[23]*a[67]) +
                  b[2]*b[5] * (a[13]*a[34] + b[13]*a[35])) +
          b[1] * (a[3]*b[9] * (a[25]*a[70] + b[25]*a[71]) +
                  b[3]*b[7] * (a[17]*a[42] + b[17]*a[43]))) /
         (a[1] * (a[2]*b[8] +  b[2]*b[5]) +
          b[1] * (a[3]*b[9] +  b[3]*b[7]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13]*b[34] *
```

```
         (a[1] * (a[2]*b[8] +  b[2]*b[5]) +
          b[1] * (a[3]*b[9] +  b[3]*b[7])) * ent;
{6,5'}
  arg := (a[1] * (a[2]*a[8] * (a[22]*a[64] + b[22]*a[65]) +
                   b[2]*a[5] * (a[12]*a[32] + b[12]*a[33])) +
          b[1] * (a[3]*a[9] * (a[24]*a[68] + b[24]*a[69]) +
                   b[3]*a[7] * (a[16]*a[40] + b[16]*a[41]))) /
         (a[1] * (a[2]*a[8] +  b[2]*a[5]) +
          b[1] * (a[3]*a[9] +  b[3]*a[7]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13]*b[47] *
         (a[1] * (a[2]*a[8] +  b[2]*a[5]) +
          b[1] * (a[3]*a[9] +  b[3]*a[7])) * ent;
{6,6'}
  arg :=(a[1] * (a[2]*b[8]*b[23]*a[67] + b[2]*b[5]*b[13]*a[35]) +
         b[1] * (a[3]*b[9]*b[25]*a[71] + b[3]*b[7]*b[17]*a[43])) /
        (a[1] * (a[2]*b[8]*b[23]       + b[2]*b[5]*b[13]) +
         b[1] * (a[3]*b[9]*b[25]       + b[3]*b[7]*b[17]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13]*a[34] *
         (a[1] * (a[2]*b[8]*b[23]       + b[2]*b[5]*b[13]) +
          b[1] * (a[3]*b[9]*b[25]       + b[3]*b[7]*b[17])) * ent;
{6,7'}
  arg :=(a[1] * (a[2]*b[8]*a[23]*a[66] + b[2]*b[5]*a[13]*a[34]) +
         b[1] * (a[3]*b[9]*a[25]*a[70] + b[3]*b[7]*a[17]*a[42])) /
        (a[1] * (a[2]*b[8]*a[23]       + b[2]*b[5]*a[13]) +
         b[1] * (a[3]*b[9]*a[25]       + b[3]*b[7]*a[17]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13]*a[34] *
         (a[1] * (a[2]*b[8]*a[23]       + b[2]*b[5]*a[13]) +
          b[1] * (a[3]*b[9]*a[25]       + b[3]*b[7]*a[17])) * ent;
{6,8'}
  arg := (a[1] * (a[2]*a[8]*b[22]*a[65] + b[2]*a[5]*b[12]*a[33]) +
          b[1] * (a[3]*a[9]*b[24]*a[69] + b[3]*a[7]*b[16]*a[41])) /
         (a[1] * (a[2]*a[8]*b[22]       + b[2]*a[5]*b[12]) +
          b[1] * (a[3]*a[9]*b[24]       + b[3]*a[7]*b[16]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13]*a[47] *
         (a[1] * (a[2]*a[8]*b[22]       + b[2]*a[5]*b[12]) +
          b[1] * (a[3]*a[9]*b[24]       + b[3]*a[7]*b[16])) * ent;
{6,9'}
  arg := (a[1] * (a[2]*a[8]*a[22]*a[64] + b[2]*a[5]*a[12]*a[32]) +
          b[1] * (a[3]*a[9]*a[24]*a[68] + b[3]*a[7]*a[16]*a[40])) /
         (a[1] * (a[2]*a[8]*a[22]       + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[9]*a[24]       + b[3]*a[7]*a[16]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*b[5]*a[13]*a[47] *
         (a[1] * (a[2]*a[8]*a[22]       + b[2]*a[5]*a[12]) +
          b[1] * (a[3]*a[9]*a[24]       + b[3]*a[7]*a[16])) * ent;
{6,10'}
  arg:= (a[1]*b[2] * (a[5] * (a[12]*a[46] + b[12]*a[33]) +
                      b[5] * (a[13]*a[47] + b[13]*a[35])) +
         b[1]*b[3] * (a[7] * (a[16]*a[50] + b[16]*a[41]) +
                      b[7] * (a[17]*a[51] + b[17]*a[43]))) /
        (a[1]*b[2] + b[1]*b[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[12]*b[33] * (a[1]*b[2] + b[1]*b[3]) * ent;
{6,11'}
  arg := (a[1]*a[2] * (a[8] * (a[22]*a[72] + b[22]*a[65]) +
                       b[8] * (a[23]*a[73] + b[23]*a[67])) +
          b[1]*a[3] * (a[9] * (a[24]*a[74] + b[24]*a[69]) +
                       b[9] * (a[25]*a[75] + b[25]*a[71]))) /
         (a[1]*a[2] + b[1]*a[3]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[19]*b[55] * (a[1]*a[2] + b[1]*a[3]) * ent;
{6,12'}
  arg := (a[1]*b[2] * (a[5]*b[12]*a[33] + b[5]*b[13]*a[35]) +
          b[1]*b[3] * (a[7]*b[16]*a[41] + b[7]*b[17]*a[43])) /
         (a[1]*b[2] * (a[5]*b[12]       + b[5]*b[13]) +
          b[1]*b[3] * (a[7]*b[16]       + b[7]*b[17]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[12]*a[33] *
         (a[1]*b[2] * (a[5]*b[12]       + b[5]*b[13]) +
          b[1]*b[3] * (a[7]*b[16]       + b[7]*b[17])) * ent;
{6,13'}
```

```
  arg := (a[1]*b[2] * (a[5]*a[12]*a[46] + b[5]*a[13]*a[47]) +
          b[1]*b[3] * (a[7]*a[16]*a[50] + b[7]*a[17]*a[51])) /
         (a[1]*b[2] * (a[5]*a[12]       + b[5]*a[13]) +
          b[1]*b[3] * (a[7]*a[16]       + b[7]*a[17]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[12]*a[33] *
         (a[1]*b[2] * (a[5]*a[12]       + b[5]*a[13]) +
          b[1]*b[3] * (a[7]*a[16]       + b[7]*a[17])) * ent;
{6,14'}
  arg := (a[1]*a[2] * (a[8]*b[22]*a[65] + b[8]*b[23]*a[67]) +
          b[1]*a[3] * (a[9]*b[24]*a[69] + b[9]*b[25]*a[71])) /
         (a[1]*a[2] * (a[8]*b[22]       + b[8]*b[23]) +
          b[1]*a[3] * (a[9]*b[24]       + b[9]*b[25]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[19]*a[55] *
         (a[1]*a[2] * (a[8]*b[22]       + b[8]*b[23]) +
          b[1]*a[3] * (a[9]*b[24]       + b[9]*b[25])) * ent;
{6,15'}
  arg := (a[1]*a[2] * (a[8]*a[22]*a[72] + b[8]*a[23]*a[73]) +
          b[1]*a[3] * (a[9]*a[24]*a[74] + b[9]*a[25]*a[75])) /
         (a[1]*a[2] * (a[8]*a[22]       + b[8]*a[23]) +
          b[1]*a[3] * (a[9]*a[24]       + b[9]*a[25]));
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*b[19]*a[55] *
         (a[1]*a[2] * (a[8]*a[22]       + b[8]*a[23]) +
          b[1]*a[3] * (a[9]*a[24]       + b[9]*a[25])) * ent;
{6,16'}
  arg := (a[1]*b[2]*b[5] * (a[13]*a[47] + b[13]*a[35]) +
          b[1]*b[3]*b[7] * (a[17]*a[51] + b[17]*a[43])) /
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12]*b[32] *
         (a[1]*b[2]*b[5] + b[1]*b[3]*b[7]) * ent;
{6,17'}
  arg := (a[1]*b[2]*a[5] * (a[12]*a[46] + b[12]*a[33]) +
          b[1]*b[3]*a[7] * (a[16]*a[50] + b[16]*a[41])) /
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12]*b[46] *
         (a[1]*b[2]*a[5] + b[1]*b[3]*a[7]) * ent;
{6,18'}
  arg := (a[1]*a[2]*b[8] * (a[23]*a[73] + b[23]*a[67]) +
          b[1]*a[3]*b[9] * (a[25]*a[75] + b[25]*a[71])) /
         (a[1]*a[2]*b[8] + b[1]*a[3]*b[9]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19]*b[54] *
         (a[1]*a[2]*b[8] + b[1]*a[3]*b[9]) * ent;
{6,19'}
  arg := (a[1]*a[2]*a[8] * (a[22]*a[72] + b[22]*a[65]) +
          b[1]*a[3]*a[9] * (a[24]*a[74] + b[24]*a[69])) /
         (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19]*b[61] *
         (a[1]*a[2]*a[8] + b[1]*a[3]*a[9]) * ent;
{6,20'}
  arg := (a[1]*b[2]*b[5]*b[13]*a[35] + b[1]*b[3]*b[7]*b[17]*a[43]) /
         (a[1]*b[2]*b[5]*b[13]       + b[1]*b[3]*b[7]*b[17]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12]*a[32] *
         (a[1]*b[2]*b[5]*b[13]       + b[1]*b[3]*b[7]*b[17]) * ent;
{6,21'}
  arg := (a[1]*b[2]*b[5]*a[13]*a[47] + b[1]*b[3]*b[7]*a[17]*a[51]) /
         (a[1]*b[2]*b[5]*a[13]       + b[1]*b[3]*b[7]*a[17]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12]*a[32] *
         (a[1]*b[2]*b[5]*a[13] + b[1]*b[3]*b[7]*a[17]) * ent;
{6,22'}
  arg := (a[1]*b[2]*a[5]*b[12]*a[33] + b[1]*b[3]*a[7]*b[16]*a[41]) /
         (a[1]*b[2]*a[5]*b[12]       + b[1]*b[3]*a[7]*b[16]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12]*a[46] *
         (a[1]*b[2]*a[5]*b[12]       + b[1]*b[3]*a[7]*b[16]) * ent;
{6,23'}
  arg := (a[1]*b[2]*a[5]*a[12]*a[46] + b[1]*b[3]*a[7]*a[16]*a[50]) /
         (a[1]*b[2]*a[5]*a[12]       + b[1]*b[3]*a[7]*a[16]);
```

```
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[12]*a[46] *
         (a[1]*b[2]*a[5]*a[12]       + b[1]*b[3]*a[7]*a[16]) * ent;
{6,24'}
  arg := (a[1]*a[2]*b[8]*b[23]*a[67] + b[1]*a[3]*b[9]*b[25]*a[71]) /
         (a[1]*a[2]*b[8]*b[23]       + b[1]*a[3]*b[9]*b[25]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19]*a[54] *
         (a[1]*a[2]*b[8]*b[23]       + b[1]*a[3]*b[9]*b[25]) * ent;
{6,25'}
  arg := (a[1]*a[2]*b[8]*a[23]*a[73] + b[1]*a[3]*b[9]*a[25]*a[75]) /
         (a[1]*a[2]*b[8]*a[23]       + b[1]*a[3]*b[9]*a[25]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19]*a[54] *
         (a[1]*a[2]*b[8]*a[23]       + b[1]*a[3]*b[9]*a[25]) * ent;
{6,26'}
  arg := (a[1]*a[2]*a[8]*b[22]*a[65] + b[1]*a[3]*a[9]*b[24]*a[69]) /
         (a[1]*a[2]*a[8]*b[22]       + b[1]*a[3]*a[9]*b[24]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19]*a[61] *
         (a[1]*a[2]*a[8]*b[22]       + b[1]*a[3]*a[9]*b[24]) * ent;
{6,27'}
  arg := (a[1]*a[2]*a[8]*a[22]*a[72] + b[1]*a[3]*a[9]*a[24]*a[74]) /
         (a[1]*a[2]*a[8]*a[22]       + b[1]*a[3]*a[9]*a[24]);
  ent := H(arg);
  snt := snt + a[1]*b[2]*a[5]*a[19]*a[61] *
         (a[1]*a[2]*a[8]*a[22]       + b[1]*a[3]*a[9]*a[24]) * ent;
{6,28'}
  arg := a[3] * (a[9] * (a[27]*a[78] + b[27]*a[79]) +
                 b[9] * (a[25]*a[70] + b[25]*a[71])) +
         b[3] * (a[7] * (a[21]*a[58] + b[21]*a[59]) +
                 b[7] * (a[17]*a[42] + b[17]*a[43]));
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*b[11]*b[31] * b[1] * ent;
{6,29'}
  arg := a[2] * (a[8] * (a[26]*a[76] + b[26]*a[77]) +
                 b[8] * (a[23]*a[66] + b[23]*a[67])) +
         b[2] * (a[5] * (a[19]*a[54] + b[19]*a[55]) +
                 b[5] * (a[13]*a[34] + b[13]*a[35]));
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*b[23]*b[67] * a[1] * ent;
{6,30'}
  arg := (a[3] * (a[9]*b[27]*a[79] + b[9]*b[25]*a[71]) +
          b[3] * (a[7]*b[21]*a[59] + b[7]*b[17]*a[43])) /
         (a[3] * (a[9]*b[27]       + b[9]*b[25]) +
          b[3] * (a[7]*b[21]       + b[7]*b[17]));
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*b[11]*a[31] * b[1] *
         (a[3] * (a[9]*b[27]       + b[9]*b[25]) +
          b[3] * (a[7]*b[21]       + b[7]*b[17])) * ent;
{6,31'}
  arg := (a[3] * (a[9]*a[27]*a[78] + b[9]*a[25]*a[70]) +
          b[3] * (a[7]*a[21]*a[58] + b[7]*a[17]*a[42])) /
         (a[3] * (a[9]*a[27]       + b[9]*a[25]) +
          b[3] * (a[7]*a[21]       + b[7]*a[17]));
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*b[11]*a[31] * b[1] *
         (a[3] * (a[9]*a[27]       + b[9]*a[25]) +
          b[3] * (a[7]*a[21]       + b[7]*a[17])) * ent;
{6,32'}
  arg := (a[2] * (a[8]*b[26]*a[77] + b[8]*b[23]*a[67]) +
          b[2] * (a[5]*b[19]*a[55] + b[5]*b[13]*a[35])) /
         (a[2] * (a[8]*b[26]       + b[8]*b[23]) +
          b[2] * (a[5]*b[19]       + b[5]*b[13]));
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*b[23]*a[67] * a[1] *
         (a[2] * (a[8]*b[26]       + b[8]*b[23]) +
          b[2] * (a[5]*b[19]       + b[5]*b[13])) * ent;
{6,33'}
  arg := (a[2] * (a[8]*a[26]*a[76] + b[8]*a[23]*a[66]) +
          b[2] * (a[5]*a[19]*a[54] + b[5]*a[13]*a[34])) /
         (a[2] * (a[8]*a[26]       + b[8]*a[23]) +
          b[2] * (a[5]*a[19]       + b[5]*a[13]));
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*b[23]*a[67] * a[1] *
```

```
          (a[2] * (a[8]*a[26]       + b[8]*a[23]) +
           b[2] * (a[5]*a[19]       + b[5]*a[13])) * ent;
{6,34'}
  arg := (a[3]*b[9] * (a[25]*a[70] + b[25]*a[71]) +
          b[3]*b[7] * (a[17]*a[42] + b[17]*a[43])) /
         (a[3]*b[9] + b[3]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11]*b[30] * b[1] *
         (a[3]*b[9] + b[3]*b[7]) * ent;
{6,35'}
  arg := (a[3]*a[9] * (a[27]*a[78] + b[27]*a[79]) +
          b[3]*a[7] * (a[21]*a[58] + b[21]*a[59])) /
         (a[3]*a[9] + b[3]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11]*b[45] * b[1] *
         (a[3]*a[9] + b[3]*a[7]) * ent;
{6,36'}
  arg := (a[2]*b[8] * (a[23]*a[66] + b[23]*a[67]) +
          b[2]*b[5] * (a[13]*a[34] + b[13]*a[35])) /
         (a[2]*b[8] + b[2]*b[5]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23]*b[66] * a[1] *
         (a[2]*b[8] + b[2]*b[5]) * ent;
{6,37'}
  arg := (a[2]*a[8] * (a[26]*a[76] + b[26]*a[77]) +
          b[2]*a[5] * (a[19]*a[54] + b[19]*a[55])) /
         (a[2]*a[8] + b[2]*a[5]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23]*b[73] * a[1] *
         (a[2]*a[8] + b[2]*a[5]) * ent;
{6,38'}
  arg := (a[3]*b[9]*b[25]*a[71] + b[3]*b[7]*b[17]*a[43]) /
         (a[3]*b[9]*b[25]       + b[3]*b[7]*b[17]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11]*a[30] * b[1] *
         (a[3]*b[9]*b[25]       + b[3]*b[7]*b[17]) * ent;
{6,39'}
  arg := (a[3]*b[9]*a[25]*a[70] + b[3]*b[7]*a[17]*a[42]) /
         (a[3]*b[9]*a[25]       + b[3]*b[7]*a[17]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11]*a[30] * b[1] *
         (a[3]*b[9]*a[25]       + b[3]*b[7]*a[17]) * ent;
{6,40'}
  arg := (a[3]*a[9]*b[27]*a[79] + b[3]*a[7]*b[21]*a[59]) /
         (a[3]*a[9]*b[27]       + b[3]*a[7]*b[21]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11]*a[45] * b[1] *
         (a[3]*a[9]*b[27]       + b[3]*a[7]*b[21]) * ent;
{6,41'}
  arg := (a[3]*a[9]*a[27]*a[78] + b[3]*a[7]*a[21]*a[58]) /
         (a[3]*a[9]*a[27]       + b[3]*a[7]*a[21]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[4]*a[11]*a[45] * b[1] *
         (a[3]*a[9]*a[27]       + b[3]*a[7]*a[21]) * ent;
{6,42'}
  arg := (a[2]*b[8]*b[23]*a[67] + b[2]*b[5]*b[13]*a[35]) /
         (a[2]*b[8]*b[23]       + b[2]*b[5]*b[13]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23]*a[66] * a[1] *
         (a[2]*b[8]*b[23]       + b[2]*b[5]*b[13]) * ent;
{6,43'}
  arg := (a[2]*b[8]*a[23]*a[66] + b[2]*b[5]*a[13]*a[34]) /
         (a[2]*b[8]*a[23]       + b[2]*b[5]*a[13]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23]*a[66] * a[1] *
         (a[2]*b[8]*a[23]       + b[2]*b[5]*a[13]) * ent;
{6,44'}
  arg := (a[2]*a[8]*b[26]*a[77] + b[2]*a[5]*b[19]*a[55]) /
         (a[2]*a[8]*b[26]       + b[2]*a[5]*b[19]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23]*a[73] * a[1] *
         (a[2]*a[8]*b[26]       + b[2]*a[5]*b[19]) * ent;
{6,45'}
  arg := (a[2]*a[8]*a[26]*a[76] + b[2]*a[5]*a[19]*a[54]) /
         (a[2]*a[8]*a[26]       + b[2]*a[5]*a[19]);
```

```
  ent := H(arg);
  snt := snt + a[1]*a[2]*b[8]*a[23]*a[73] * a[1] *
         (a[2]*a[8]*a[26]       + b[2]*a[5]*a[19]) * ent;
{6,46'}
  arg := a[7] * (a[21]*a[63] + b[21]*a[59]) +
         b[7] * (a[17]*a[51] + b[17]*a[43]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[10]*b[29] * b[1]*b[3] * ent;
{6,47'}
  arg := a[9] * (a[27]*a[81] + b[27]*a[79]) +
         b[9] * (a[25]*a[75] + b[25]*a[71]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[18]*b[53] * b[1]*a[3] * ent;
{6,48'}
  arg := a[5] * (a[19]*a[61] + b[19]*a[55]) +
         b[5] * (a[13]*a[47] + b[13]*a[35]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[22]*b[65] * a[1]*b[2] * ent;
{6,49'}
  arg := a[8] * (a[26]*a[80] + b[26]*a[77]) +
         b[8] * (a[23]*a[73] + b[23]*a[67]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[26]*b[77] * a[1]*a[2] * ent;
{6,50'}
  arg := (a[7]*b[21]*a[59] + b[7]*b[17]*a[43]) /
         (a[7]*b[21]       + b[7]*b[17]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[10]*a[29] * b[1]*b[3] *
         (a[7]*b[21]       + b[7]*b[17]) * ent;
{6,51'}
  arg := (a[7]*a[21]*a[63] + b[7]*a[17]*a[51]) /
         (a[7]*a[21]       + b[7]*a[17]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[10]*a[29] * b[1]*b[3] *
         (a[7]*a[21]       + b[7]*a[17]) * ent;
{6,52'}
  arg := (a[9]*b[27]*a[79] + b[9]*b[25]*a[71]) /
         (a[9]*b[27]       + b[9]*b[25]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[18]*a[53] * b[1]*a[3] *
         (a[9]*b[27]       + b[9]*b[25]) * ent;
{6,53'}
  arg := (a[9]*a[27]*a[81] + b[9]*a[25]*a[75]) /
         (a[9]*a[27]       + b[9]*a[25]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*b[18]*a[53] * b[1]*a[3] *
         (a[9]*a[27]       + b[9]*a[25]) * ent;
{6,54'}
  arg := (a[5]*b[19]*a[55] + b[5]*b[13]*a[35]) /
         (a[5]*b[19]       + b[5]*b[13]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[22]*a[65] * a[1]*b[2] *
         (a[5]*b[19]       + b[5]*b[13]) * ent;
{6,55'}
  arg := (a[5]*a[19]*a[61] + b[5]*a[13]*a[47]) /
         (a[5]*a[19]       + b[5]*a[13]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[22]*a[65] * a[1]*b[2] *
         (a[5]*a[19]       + b[5]*a[13]) * ent;
{6,56'}
  arg := (a[8]*b[26]*a[77] + b[8]*b[23]*a[67]) /
         (a[8]*b[26]       + b[8]*b[23]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[26]*a[77] * a[1]*a[2] *
         (a[8]*b[26]       + b[8]*b[23]) * ent;
{6,57'}
  arg := (a[8]*a[26]*a[80] + b[8]*a[23]*a[73]) /
         (a[8]*a[26]       + b[8]*a[23]);
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*b[26]*a[77] * a[1]*a[2] *
         (a[8]*a[26]       + b[8]*a[23]) * ent;
{6,58'}
  arg := a[17]*a[51] + b[17]*a[43];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10]*b[28] * b[1]*b[3]*b[7] * ent;
```

```
{6,59'}
  arg := a[21]*a[63] + b[21]*a[59];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10]*b[44] * b[1]*b[3]*a[7] * ent;
{6,60'}
  arg := a[25]*a[75] + b[25]*a[71];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18]*b[52] * b[1]*a[3]*b[9] * ent;
{6,61'}
  arg := a[27]*a[81] + b[27]*a[79];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18]*b[60] * b[1]*a[3]*a[9] * ent;
{6,62'}
  arg := a[13]*a[47] + b[13]*a[35];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22]*b[64] * a[1]*b[2]*b[5] * ent;
{6,63'}
  arg := a[19]*a[61] + b[19]*a[55];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22]*b[72] * a[1]*b[2]*a[5] * ent;
{6,64'}
  arg := a[23]*a[73] + b[23]*a[67];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26]*b[76] * a[1]*a[2]*b[8] * ent;
{6,65'}
  arg := a[26]*a[80] + b[26]*a[77];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26]*b[80] * a[1]*a[2]*a[8] * ent;
{6,66'}
  arg := a[43];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10]*a[28] * b[1]*b[3]*b[7]*b[17] * ent;
{6,67'}
  arg := a[51];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10]*a[28] * b[1]*b[3]*b[7]*a[17] * ent;
{6,68'}
  arg := a[59];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10]*a[44] * b[1]*b[3]*a[7]*b[21] * ent;
{6,69'}
  arg := a[63];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[10]*a[44] * b[1]*b[3]*a[7]*a[21] * ent;
{6,70'}
  arg := a[71];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18]*a[52] * b[1]*a[3]*b[9]*b[25] * ent;
{6,71'}
  arg := a[75];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18]*a[52] * b[1]*a[3]*b[9]*a[25] * ent;
{6,72'}
  arg := a[79];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18]*a[60] * b[1]*a[3]*a[9]*b[27] * ent;
{6,73'}
  arg := a[81];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[4]*a[18]*a[60] * b[1]*a[3]*a[9]*a[27] * ent;
{6,74'}
  arg := a[35];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22]*a[64] * a[1]*b[2]*b[5]*b[13] * ent;
{6,75'}
  arg := a[47];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22]*a[64] * a[1]*b[2]*b[5]*a[13] * ent;
{6,76'}
  arg := a[55];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[22]*a[72] * a[1]*b[2]*a[5]*b[19] * ent;
{6,77'}
  arg := a[61];
  ent := H(arg);
```

```
  snt := snt + a[1]*a[2]*a[8]*a[22]*a[72] * a[1]*b[2]*a[5]*a[19] * ent;
{6,78'}
  arg := a[67];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26]*a[76] * a[1]*a[2]*b[8]*b[23] * ent;
{6,79'}
  arg := a[73];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26]*a[76] * a[1]*a[2]*b[8]*a[23] * ent;
{6,80'}
  arg := a[77];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26]*a[80] * a[1]*a[2]*a[8]*b[26] * ent;
{6,81'}
  arg := a[80];
  ent := H(arg);
  snt := snt + a[1]*a[2]*a[8]*a[26]*a[80] * a[1]*a[2]*a[8]*a[26] * ent;

  avi := avi + snt;
  avi := avi/6;
  V := avi;

  END; {of V(a)}


BEGIN {of prog}

Rewrite (out,'mems5.out');

  FOR i := 1  TO 81 DO a[i] := 0.1;
  ap := a;
  am := a;

  old := V(a);

  oldprev := 1.0;
  i := 0;
  WHILE (oldprev <> old) AND (i < 10000) DO BEGIN
    oldprev := old;
    i := i + 1;
    FOR j := 1 TO 81 DO BEGIN
      s := 1;
      FOR k := 1 TO 9 DO BEGIN
        s := s / 10;
        endflag := FALSE;
        REPEAT
          ap := a;
          IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
          new := V(ap);

          IF new > old THEN BEGIN
            a[j] := ap[j];
            old := new;
          END ELSE
          BEGIN
            am := a;
            IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
            new := V(am);

            IF new > old THEN BEGIN
              a[j] := am[j];
              old := new;
            END ELSE endflag := TRUE;
          END;

        UNTIL endflag;

      END; {of k}

    END; {of jj}

    WritelN (     'i=', i:3, ' R=', old:18:16);
    WritelN (out, 'i=', i:3, ' R=', old:18:16)

  END; {of i}
```

Appendix 10: Program body mems5.rnd.p

```
PROGRAM MEMS5RND (input, output);
{A1..A81}

$SEARCH 'rndgen.o'$
IMPORT rndgen;

TYPE AR = ArraY [1..81] of Longreal;
VAR i,j,jj,k,l,teller          : Integer;
    old, oldprev, new, s, Rbest : Longreal;
    endflag                     : Boolean;
    a, ap ,am                   : AR;
    out                         : Text;
    verz                        : SET OF 1..81;


BEGIN {of prog}

Rewrite (out,'mems5rnd.out');
InitRndm;
Rbest := 0;
teller := 0;

REPEAT

  FOR i := 1  TO 81 DO a[i] := Rndm;
  ap := a;
  am := a;

  old := V(a);

  oldprev := 1.0;
  i := 0;
  WHILE (oldprev <> old) AND (i < 10000) DO BEGIN
    oldprev := old;
    i := i + 1;
    verz := [1..81];
    FOR jj := 1 TO 81 DO BEGIN
      j := TrunC (81 * Rndm + 1);
      WhilE NOT (j IN verz) DO j := TrunC (81 * Rndm + 1);
      verz := verz - [j];
      s := 1;
      FOR k := 1 TO 9 DO BEGIN
        s := s / 10;
        endflag := FALSE;
        REPEAT
          ap := a;
          IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
          new := V(ap);

          IF new > old THEN BEGIN
            a[j] := ap[j];
            old := new;
          END ELSE
          BEGIN
            am := a;
            IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
            new := V(am);

            IF new > old THEN BEGIN
              a[j] := am[j];
              old := new;
            END ELSE endflag := TRUE;
          END;

        UNTIL endflag;

      END; {of k}

    END; {of jj}
```

```
  END; {of i}

  teller := teller + 1;
  Write (teller:1,' ');
  WritelN (out);
  WritelN (out, 'teller=',teller:1);
  WritelN (out, 'i=', i:3,   ' R =   ', old:18:16);
  WritelN (out, '     ', ' Rbest=', Rbest:18:16);
  WritelN (out);
  FOR i := 1 TO 81 DO BEGIN
    WritE (out, 'a[',i:2,']=',a[i]:12:10, ' ');
    IF i IN [1,3,7,9,13,17,21,25,27,31,35,39,43,47,51,55,59,63,67,71,75,79,81]
    THEN WritelN (out);
  END;

UNTIL FALSE

END.
```

Appendix 11: Source code `memi.p`

```
PROGRAM MEMi (input, output);

TYPE AR = ArraY [1..16] of Longreal;
TYPE AAR = ArraY [1..65536] of Longreal;

VAR i,j,k,m,
    onder,boven          : Integer; {prec}
    old, oldprev, new, s : Longreal;
    endflag,min,plus     : Boolean;
    a, ap ,am            : AR;
    aa                   : AAR;
    out                  : Text;


FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION TWO (x : Integer) : Integer;
  BEGIN
    TWO := ROUND (EXP(LN(2) * x));
  END;

FUNCTION V (a : AR) : Longreal;

  VAR i,new_i,j,new_j,jj,k,l,p,q,
      var0,var1,var2,var3,var4,
      var5,var6,var7              : Integer;
      arg, ent, snt, avi          : Longreal;
      tot                         : ArraY [0..15,0..32767] OF Longreal;

BEGIN

  FOR i := 1 TO TWO(m - 1) DO BEGIN

    k := i;
    aa[i] := 1;
    FOR j := 1 TO m - 1 DO BEGIN

      var0 := TWO(m - j - 1);
      IF var0 <= k - 1 THEN BEGIN
        aa[i] := aa[i] * a[j];
        k := k - var0;
      END
      ELSE
        aa[i] := aa[i] * (1 - a[j]);

    END; {of j}
  END; {of i}

  tot [0,0] := 1;

  FOR j := 1 TO m - 2 DO BEGIN

    var0 := TWO(m - j - 1);
    FOR jj := 0 TO TWO(j) - 1 DO BEGIN

      tot [j,jj] := 0;
      FOR k := 1 TO var0 DO
        tot [j,jj] := tot [j,jj] + aa[var0 * jj + k];

    END; {of jj}
  END; {of j}


  {0,0}
  avi := 0;
  FOR i := TWO(m - 2) + 1 TO TWO(m - 1) DO
```

```
    avi := avi + aa[i];

  {i,j}
  FOR i := 1 TO m - 1 DO BEGIN

    FOR j := 0 TO i - 1 DO BEGIN

      var1 := TWO(i - j - 1);
      var2 := TWO(m - i - 1);
      var3 := TWO(m - j - 1);
      var4 := TWO(m - i);

      snt := 0;
      FOR jj := 0 TO TWO(j) - 1 DO BEGIN

        arg := 0;
        FOR k := 0 TO var1 - 1 DO BEGIN
          FOR l := 1 TO var2 DO BEGIN

            arg := arg + aa[var3 * jj + var4 * k + l] / tot [j,jj];
          END; {of l}

        END; {of k}

        ent := H(arg);
        snt := snt + tot [j,jj] * ent;
      END; {of jj}


      IF i = m - 1 THEN new_i := m - 2 ELSE new_i := i;
      IF j = 0     THEN new_j := 1     ELSE new_j := j;

      var5 := TWO(m - new_i - 2);
      var6 := TWO(m - new_j - 1);
      var7 := TWO(m - j - 2);


      FOR p := 1 TO TWO(new_j) DO BEGIN

        FOR q := 1 TO var5 DO BEGIN

          avi := avi + (aa[var6 * p - var7 + var2 - var5 + q]) * snt;

        END; {of q}

      END; {of p}

    END; {of j}
  END; {of i}

  V := avi / (m + 1);

END; {of V}

BEGIN {of prog}

Write ('Ondergrens=');
Readln (onder);
Write ('Bovengrens=');
Readln (boven);
{Write ('Nauwkeurigheid=');}
{Readln (prec);}

Rewrite (out,'memi.out');

FOR m := onder TO boven DO
BEGIN

  Writeln (     'm=',m:2);
  Writeln (out, 'm=',m:2);
  FOR i := 1 TO m - 1 DO a[i] := 0.7;

  ap := a;
  am := a;
```

```
  old := V(a);

  oldprev := 1.0;
  i := 0;
  WHILE (Round (1000000 * oldprev) <> Round (1000000 * old))  DO BEGIN
    oldprev := old;
    i := i + 1;
    FOR j := 1 TO m - 1 DO BEGIN
      Write (j:1,' ');
      s := 1 / EXP(LN(10) * i);
      endflag := FALSE;
      plus    := FALSE;
      min     := FALSE;
      REPEAT
        IF NOT (min) THEN
        BEGIN
          ap := a;
          IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
          new := V(ap);

          IF new > old THEN BEGIN
            plus := TRUE;
            a[j] := ap[j];
            old := new;
          END ELSE min := TRUE;
        END ELSE
        BEGIN
          IF plus THEN endflag := TRUE ELSE
          BEGIN
            am := a;
            IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
            new := V(am);

            IF new > old THEN BEGIN
              a[j] := am[j];
              old := new;
            END ELSE endflag := TRUE;
          END;
        END;

      UNTIL endflag;

    END; {of j}

    WritelN (     'i=', i:1, ' R=', old:18:16);
    WritelN (out, 'i=', i:1, ' R=', old:18:16)

  END; {of i}

  WritelN;
  WritelN (out);
  FOR i := 1 TO m - 1  DO WritelN (     'a[',i:2,']=',a[i]:18:16);
  FOR i := 1 TO m - 1  DO WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  WritelN;
  WritelN (out);

END; {of m}
END.
```

Appendix 12: Source code memseq4.p

```
PROGRAM MEMSEQ4 (input, output);

TYPE AR = ArraY [1..7] of Longreal;

VAR i,j,k                  : Integer;
    old, oldprev, new, s : Longreal;
    endflag, min, plus   : Boolean;
    a, ap, am            : AR;
    out                  : Text;

FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a : AR)       : Longreal;
  VAR i                   : Integer;
      arg, ent, snt, avi : Longreal;
      b                   : AR;

  BEGIN

  FOR i := 1 TO 7 do b[i] := 1 - a[i];

{input symbol x_2,2}
  arg := a[1];
  ent := H(arg);
  avi := a[2] * ent;

{input symbol x_1,3}
  arg := a[2];
  ent := H(arg);
  snt := (b[1]*a[4] + a[1]*a[3]) * ent;

{input symbol x_2,4}
  arg := b[1]*a[4] + a[1]*a[3];
  ent := H(arg);
  snt := snt + b[2]*a[6] * ent;
  arg := a[4];
  ent := H(arg);
  snt := snt + b[1] * a[2]*a[5] * ent;
  arg := a[3];
  ent := H(arg);
  snt := snt + a[1] * a[2]*a[7] * ent;
  avi := avi + snt;

{input symbol x_1,5}
  arg := b[2]*a[6] + a[2]*a[5];
  ent := H(arg);
  snt := b[1]*b[4] * ent;
  arg := a[6];
  ent := H(arg);
  snt := snt + b[2] * b[1]*a[4] * ent;
  arg := a[5];
  ent := H(arg);
  snt := snt + a[2] * b[1]*a[4] * ent;

  arg := b[2]*a[6] + a[2]*a[7];
  ent := H(arg);
  snt := snt + a[1]*b[3] * ent;
  arg := a[6];
  ent := H(arg);
  snt := snt + b[2] * a[1]*a[3] * ent;
  arg := a[7];
  ent := H(arg);
  snt := snt + a[2] * a[1]*a[3] * ent;

  avi := avi + snt;
  avi := avi/5;
```

```
  V := avi;

  END; {of V(a)}


BEGIN {of prog}

Rewrite (out,'mem.seq.4.out');

  FOR i := 1 TO 7 DO BEGIN
    a[i] := 0.5;
  END;

  ap := a;
  am := a;

  old := V(a);

  oldprev := 1.0;
  i := 0;
  WriteLN;
  WriteLN (     'i=', i:1, ' R=', old:18:16);
  WriteLN (out, 'i=', i:1, ' R=', old:18:16);

  WHILE (Round (100000 * oldprev) <> Round (100000 * old))  DO BEGIN
    oldprev := old;
    i := i + 1;
    FOR j := 1 TO 7 DO BEGIN
      Write (j:1,' ');
      s := 1 / EXP(LN(10) * i);
      endflag := FALSE;
      plus    := FALSE;
      min     := FALSE;
      REPEAT
        IF NOT (min) THEN
        BEGIN
          ap := a;
          IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
          new := V(ap);

          IF new > old THEN BEGIN
            plus := TRUE;
            a[j] := ap[j];
            old := new;
          END ELSE min := TRUE;
        END ELSE
        BEGIN
          IF plus THEN endflag := TRUE ELSE
          BEGIN
            am := a;
            IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
            new := V(am);

            IF new > old THEN BEGIN
              a[j] := am[j];
              old := new;
            END ELSE endflag := TRUE;
          END;
        END;

      UNTIL endflag;

    END; {of j}

    WriteLN;
    WriteLN (     'i=', i:1, ' R=', old:18:16);
    WriteLN (out, 'i=', i:1, ' R=', old:18:16)

  END; {of i}

  WriteLN;
  WriteLN (out);
  FOR i := 1 TO 7  DO
    WriteLN (     'a[',i:2,']=',a[i]:18:16);
```

Appendix 13: Source code `memloop4.p`

```
PROGRAM MEMLOOP4 (input, output);

TYPE AR = ArraY [1..7] of Longreal;

VAR i,j,k                 : Integer;
    old, oldprev, new, s  : Longreal;
    endflag, min, plus    : Boolean;
    a, ap, am             : AR;
    out                   : Text;

FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a : AR)       : Longreal;
  VAR i                   : Integer;
      arg, ent, snt, avi  : Longreal;
      b                   : AR;

  BEGIN

  FOR i := 1 TO 7 do b[i] := 1 - a[i];

{input symbol x_2,2}
  arg := a[1];
  ent := H(arg);
  avi := a[2] * ent;

{input symbol x_1,3}
  arg := a[2];
  ent := H(arg);
  snt := (b[1]*a[4] + a[1]*a[3]) * ent;

{input symbol x_2,4}
  arg := b[1]*a[4] + a[1]*a[3];
  ent := H(arg);
  snt := snt + b[2]*a[6] * ent;
  arg := a[4];
  ent := H(arg);
  snt := snt + b[1] * a[2]*a[5] * ent;
  arg := a[3];
  ent := H(arg);
  snt := snt + a[1] * a[2]*a[7] * ent;
  avi := avi + snt;

{looping: input symbol x_1,5 = x_1,1}
  arg := b[2]*a[6] + a[2]*a[5];
  ent := H(arg);
  snt := b[1]*b[4]*a[1] * ent;
  arg := a[6];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[1] * b[2] * ent;
  arg := a[5];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[1] * a[2] * ent;
  arg := b[2]*a[6] + a[2]*a[7];
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[1] * ent;
  arg := a[6];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[1] * b[2] * ent;
  arg := a[7];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[1] * a[2] * ent;

  avi := avi + snt;
  avi := avi/4;
```

```
  V := avi;

  END; {of V(a)}


BEGIN {of prog}

Rewrite (out,'mem.loop.4.out');
  FOR i := 1 TO 7 DO BEGIN
    a[i] := 0.5;
  END;
  ap := a;
  am := a;

  old := V(a);
  oldprev := 1.0;
  i := 0;
  WritelN;
  WritelN (     'i=', i:1, ' R=', old:18:16);
  WritelN (out, 'i=', i:1, ' R=', old:18:16);
  WHILE i < 15 DO BEGIN
    oldprev := old;
    i := i + 1;
    FOR j := 1 TO 7 DO BEGIN
      Write (j:1,' ');
      s := 1 / EXP(LN(10) * i);
      endflag := FALSE;
      plus    := FALSE;
      min     := FALSE;
      REPEAT
        IF NOT (min) THEN
        BEGIN
          ap := a;
          IF a[j] + s < 1 THEN ap[j] := a[j] + s ELSE ap[j] := 1;
          new := V(ap);

          IF new > old THEN BEGIN
            plus := TRUE;
            a[j] := ap[j];
            old := new;
          END ELSE min := TRUE;
        END ELSE
        BEGIN
          IF plus THEN endflag := TRUE ELSE
          BEGIN
            am := a;
            IF a[j] - s > 0 THEN am[j] := a[j] - s ELSE am[j] := 0;
            new := V(am);

            IF new > old THEN BEGIN
              a[j] := am[j];
              old := new;
            END ELSE endflag := TRUE;
          END;
        END;

      UNTIL endflag;

    END; {of j}

    WritelN;
    WritelN (     'i=', i:1, ' R=', old:18:16);
    WritelN (out, 'i=', i:1, ' R=', old:18:16)

  END; {of i}

  WritelN;
  WritelN (out);
  FOR i := 1 TO 7  DO
    WritelN (     'a[',i:2,']=',a[i]:18:16);
  FOR i := 1 TO 7  DO
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  WritelN;
  WritelN (out);
END.
```

Appendix 14: Source code `memseq6.p`

```
PROGRAM MEMSEQ6 (input, output);

TYPE AR = ArraY [1..22] of Longreal;

VAR i,j,teller                  : Integer;
    old, oldprev, new, max      : Longreal;
    greater                     : Boolean;
    a                           : AR;
    out                         : Text;

FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a : AR)       : Longreal;
  VAR i                   : Integer;
      arg, ent, snt, avi : Longreal;
      b                   : AR;

  BEGIN

  FOR i := 1 TO 22 do b[i] := 1 - a[i];

{input symbol x_2,2}
{2}
  arg := a[1];
  ent := H(arg);
  avi := a[2] * ent;

{input symbol x_1,3}
{4 + 3}
  arg := a[2];
  ent := H(arg);
  snt := (b[1]*a[4] + a[1]*a[3]) * ent;

{input symbol x_2,4}
{6}
  arg := b[1]*a[4] + a[1]*a[3];
  ent := H(arg);
  snt := snt + b[2]*a[6] * ent;
{5}
  arg := a[4];
  ent := H(arg);
  snt := snt + b[1] * a[2]*a[5] * ent;
{7}
  arg := a[3];
  ent := H(arg);
  snt := snt + a[1] * a[2]*a[7] * ent;
  avi := avi + snt;

{input symbol x_1,5}
{11}
  arg := b[2]*a[6] + a[2]*a[5];
  ent := H(arg);
  snt := b[1]*b[4]*a[11] * ent;
{10}
  arg := a[6];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10] * b[2] * ent;
{13}
  arg := a[5];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13] * a[2] * ent;
{9}
  arg := b[2]*a[6] + a[2]*a[7];
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9] * ent;
{8}
```

```
  arg := a[6];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8] * b[2] * ent;
{12}
  arg := a[7];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12] * a[2] * ent;

{input symbol x_2,6}
{17}
  arg := b[1]*b[4]*a[11] + b[1]*a[4]*a[10] +
         a[1]*b[3]*a[9]  + a[1]*a[3]*a[8];
  ent := H(arg);
  snt := snt + b[2]*b[6]*a[17] * ent;
{16}
  arg := (b[1]*b[4]*a[11] + a[1]*b[3]*a[9]) /
         (b[1]*b[4]       + a[1]*b[3]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[16] * (b[1]*b[4] + a[1]*b[3]) * ent;
{19}
  arg := (b[1]*a[4]*a[10] + a[1]*a[3]*a[8]) /
         (b[1]*a[4]       + a[1]*a[3]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[19] * (b[1]*a[4] + a[1]*a[3]) * ent;
{15}
  arg := b[4]*a[11] + a[4]*a[13];
  ent := H(arg);
  snt := snt + a[2]*b[5]*a[15] * b[1] * ent;
{21}
  arg := b[3]*a[9] + a[3]*a[12];
  ent := H(arg);
  snt := snt + a[2]*b[7]*a[21] * a[1] * ent;
{14}
  arg := a[11];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[14] * b[1]*b[4] * ent;
{18}
  arg := a[13];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[18] * b[1]*a[4] * ent;
{20}
  arg := a[9];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[20] * a[1]*b[3] * ent;
{22}
  arg := a[12];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[22] * a[1]*a[3] * ent;


{input symbol x_1,7}
{'1}
  arg := b[2] * (b[6]*a[17] + a[6]*a[16]) +
         a[2] * (b[5]*a[15] + a[5]*a[14]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*b[11] * ent;
{'2}
  arg := (b[2]*b[6]*a[17] + a[2]*b[5]*a[15]) /
         (b[2]*b[6]       + a[2]*b[5]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*a[11] * (b[2]*b[6] + a[2]*b[5]) * ent;
{'3}
  arg := (b[2]*a[6]*a[16] + a[2]*a[5]*a[14]) /
         (b[2]*a[6]       + a[2]*a[5]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*a[11] * (b[2]*a[6] + a[2]*a[5]) * ent;
{'4}
  arg := b[6]*a[17] + a[6]*a[19];
  ent := H(arg);
  snt := snt + b[1]*a[4]*b[10] * b[2] * ent;
{'5}
  arg := b[5]*a[15] + a[5]*a[18];
  ent := H(arg);
  snt := snt + b[1]*a[4]*b[13] * a[2] * ent;
{'6}
```

```
  arg := a[17];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10] * b[2]*b[6] * ent;
{'7}
  arg := a[19];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10] * b[2]*a[6] * ent;
{'8}
  arg := a[15];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13] * a[2]*b[5] * ent;
{'9}
  arg := a[18];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13] * a[2]*a[5] * ent;

{'1'}
  arg := b[2] * (b[6]*a[17] + a[6]*a[16]) +
         a[2] * (b[7]*a[21] + a[7]*a[20]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*b[9] * ent;
{'2'}
  arg := (b[2]*b[6]*a[17] + a[2]*b[7]*a[21]) /
         (b[2]*b[6]       + a[2]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9] * (b[2]*b[6] + a[2]*b[7]) * ent;
{'3'}
  arg := (b[2]*a[6]*a[16] + a[2]*a[7]*a[20]) /
         (b[2]*a[6]       + a[2]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9] * (b[2]*a[6] + a[2]*a[7]) * ent;
{'4'}
  arg := b[6]*a[17] + a[6]*a[19];
  ent := H(arg);
  snt := snt + a[1]*a[3]*b[8] * b[2] * ent;
{'5'}
  arg := b[7]*a[21] + a[7]*a[22];
  ent := H(arg);
  snt := snt + a[1]*a[3]*b[12] * a[2] * ent;
{'6'}
  arg := a[17];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8] * b[2]*b[6] * ent;
{'7'}
  arg := a[19];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8] * b[2]*a[6] * ent;
{'8'}
  arg := a[21];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12] * a[2]*b[7] * ent;
{'9'}
  arg := a[22];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12] * a[2]*a[7] * ent;


  avi := avi + snt;

  V := avi/7;

  END; {of V(a)}


BEGIN {of prog}

Rewrite (out,'mem.seq.6.out');

  Writeln (out, 'Nauwkeurigheid = 0.00001');

  FOR i := 1 TO 22 DO BEGIN
    a[i] := 0.5;
  END;

  teller := 0;
```

```
  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WriteLN;
    WriteLN (teller:1, ' R=', old:18:16);

    FOR i := 1 TO 22 DO BEGIN

      Write (i:1,' ');
      a[i] := 0.00001;
      old := V(a);
      max := a[i];

      FOR j := 1 TO 99999 DO BEGIN

        a[i] := a[i] + 0.00001;
        new := V(a);
        IF new > old THEN BEGIN
          old := new;
          max := a[i];
        END

      END; {of j}

      a[i] := max;

    END; {of i}

  END; {of while}

  WriteLN (out, 'Teller=',teller:1);
  WriteLN;
  WriteLN (      ' R=', old:18:16);
  WriteLN (out, ' R=', old:18:16);

  FOR i := 1 TO 22 DO BEGIN
    WriteLN (      'a[',i:2,']=',a[i]:18:16);
    WriteLN (out, 'a[',i:2,']=',a[i]:18:16);
  END;

END.
```

Appendix 15: Source code memloop6.p

```
PROGRAM MEMLOOP6 (input, output);

TYPE AR = ArraY [1..22] of Longreal;

VAR i,j,teller                  : Integer;
    old, oldprev, new, max      : Longreal;
    greater                     : Boolean;
    a                           : AR;
    out                         : Text;


FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a : AR)       : Longreal;
  VAR i                   : Integer;
      arg, ent, snt, avi : Longreal;
      b                   : AR;

  BEGIN

  FOR i := 1 TO 22 do b[i] := 1 - a[i];

{input symbol x_2,2}
{2}
  arg := a[1];
  ent := H(arg);
  avi := a[2] * ent;

{input symbol x_1,3}
{4 + 3}
  arg := a[2];
  ent := H(arg);
  snt := (b[1]*a[4] + a[1]*a[3]) * ent;

{input symbol x_2,4}
{6}
  arg := b[1]*a[4] + a[1]*a[3];
  ent := H(arg);
  snt := snt + b[2]*a[6] * ent;
{5}
  arg := a[4];
  ent := H(arg);
  snt := snt + b[1] * a[2]*a[5] * ent;
{7}
  arg := a[3];
  ent := H(arg);
  snt := snt + a[1] * a[2]*a[7] * ent;
  avi := avi + snt;

{input symbol x_1,5}
{11}
  arg := b[2]*a[6] + a[2]*a[5];
  ent := H(arg);
  snt := b[1]*b[4]*a[11] * ent;
{10}
  arg := a[6];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10] * b[2] * ent;
{13}
  arg := a[5];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13] * a[2] * ent;
{9}
  arg := b[2]*a[6] + a[2]*a[7];
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9] * ent;
```

```
{8}
  arg := a[6];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8] * b[2] * ent;
{12}
  arg := a[7];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12] * a[2] * ent;

{input symbol x_2,6}
{17}
  arg := b[1]*b[4]*a[11] + b[1]*a[4]*a[10] +
         a[1]*b[3]*a[9]  + a[1]*a[3]*a[8];
  ent := H(arg);
  snt := snt + b[2]*b[6]*a[17] * ent;
{16}
  arg := (b[1]*b[4]*a[11] + a[1]*b[3]*a[9]) /
         (b[1]*b[4]       + a[1]*b[3]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[16] * (b[1]*b[4] + a[1]*b[3]) * ent;
{19}
  arg := (b[1]*a[4]*a[10] + a[1]*a[3]*a[8]) /
         (b[1]*a[4]       + a[1]*a[3]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[19] * (b[1]*a[4] + a[1]*a[3]) * ent;
{15}
  arg := b[4]*a[11] + a[4]*a[13];
  ent := H(arg);
  snt := snt + a[2]*b[5]*a[15] * b[1] * ent;
{21}
  arg := b[3]*a[9] + a[3]*a[12];
  ent := H(arg);
  snt := snt + a[2]*b[7]*a[21] * a[1] * ent;
{14}
  arg := a[11];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[14] * b[1]*b[4] * ent;
{18}
  arg := a[13];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[18] * b[1]*a[4] * ent;
{20}
  arg := a[9];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[20] * a[1]*b[3] * ent;
{22}
  arg := a[12];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[22] * a[1]*a[3] * ent;


{looping: input symbol x_1,7 = x_1,1}
{'1}
  arg := b[2] * (b[6]*a[17] + a[6]*a[16]) +
         a[2] * (b[5]*a[15] + a[5]*a[14]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*b[11]*a[1] * ent;
{'2}
  arg := (b[2]*b[6]*a[17] + a[2]*b[5]*a[15]) /
         (b[2]*b[6]       + a[2]*b[5]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*a[11]*a[1] * (b[2]*b[6] + a[2]*b[5]) * ent;
{'3}
  arg := (b[2]*a[6]*a[16] + a[2]*a[5]*a[14]) /
         (b[2]*a[6]       + a[2]*a[5]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*a[11]*a[1] * (b[2]*a[6] + a[2]*a[5]) * ent;
{'4}
  arg := b[6]*a[17] + a[6]*a[19];
  ent := H(arg);
  snt := snt + b[1]*a[4]*b[10]*a[1] * b[2] * ent;
{'5}
  arg := b[5]*a[15] + a[5]*a[18];
  ent := H(arg);
  snt := snt + b[1]*a[4]*b[13]*a[1] * a[2] * ent;
```

```
{'6}
  arg := a[17];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10]*a[1] * b[2]*b[6] * ent;
{'7}
  arg := a[19];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10]*a[1] * b[2]*a[6] * ent;
{'8}
  arg := a[15];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13]*a[1] * a[2]*b[5] * ent;
{'9}
  arg := a[18];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13]*a[1] * a[2]*a[5] * ent;


{'1'}
  arg := b[2] * (b[6]*a[17] + a[6]*a[16]) +
         a[2] * (b[7]*a[21] + a[7]*a[20]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*b[9]*a[1] * ent;
{'2'}
  arg := (b[2]*b[6]*a[17] + a[2]*b[7]*a[21]) /
         (b[2]*b[6]       + a[2]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9]*a[1] * (b[2]*b[6] + a[2]*b[7]) * ent;
{'3'}
  arg := (b[2]*a[6]*a[16] + a[2]*a[7]*a[20]) /
         (b[2]*a[6]       + a[2]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9]*a[1] * (b[2]*a[6] + a[2]*a[7]) * ent;
{'4'}
  arg := b[6]*a[17] + a[6]*a[19];
  ent := H(arg);
  snt := snt + a[1]*a[3]*b[8]*a[1] * b[2] * ent;
{'5'}
  arg := b[7]*a[21] + a[7]*a[22];
  ent := H(arg);
  snt := snt + a[1]*a[3]*b[12]*a[1] * a[2] * ent;
{'6'}
  arg := a[17];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8]*a[1] * b[2]*b[6] * ent;
{'7'}
  arg := a[19];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8]*a[1] * b[2]*a[6] * ent;
{'8'}
  arg := a[21];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12]*a[1] * a[2]*b[7] * ent;
{'9'}
  arg := a[22];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12]*a[1] * a[2]*a[7] * ent;

  avi := avi + snt;

  V := avi;

  END; {of V(a)}


BEGIN {of prog}

Rewrite (out,'mem.loop.6.out');

  Writeln (out, 'Nauwkeurigheid = 0.00001');

  FOR i := 1 TO 22 DO BEGIN
    a[i] := 0.5;
  END;
```

```
  teller := 0;

  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WritelN;
    WritelN (teller:1, ' R=', old:18:16);

    FOR i := 1 TO 22 DO BEGIN

      Write (i:1,' ');
      a[i] := 0.00001;
      old := V(a);
      max := a[i];

      FOR j := 1 TO 99999 DO BEGIN

        a[i] := a[i] + 0.00001;
        new := V(a);
        IF new > old THEN BEGIN
          old := new;
          max := a[i];
        END

      END; {of j}

      a[i] := max;

    END; {of i}

  END; {of while}

  WritelN (out, 'Teller=',teller:1);
  WritelN;
  WritelN (     ' R=', old:18:16);
  WritelN (out, ' R=', old:18:16);

  FOR i := 1 TO 22 DO BEGIN
    WritelN (     'a[',i:2,']=',a[i]:18:16);
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  END;

END.
```

Appendix 16: Program body memeps6.p

```
PROGRAM MEMEPS6 (input, output);

CONST shan = 0.703506;
      eps  = 0.001;
      init = shan - eps;

TYPE AR = ArraY [1..22] of Longreal;
TYPE ARINT = ArraY [1..22] of Integer;

VAR i,j,teller                  : Integer;
    old, oldprev, new, max      : Longreal;
    next, einde                 : Boolean;
    a,c                         : AR;
    f                           : ARINT;
    setje                       : SET OF Integer;
    out                         : Text;


BEGIN {of prog}

  Append (out, 'mem.eps.6.out');
  WritelN (out);
  WritelN (out,'****************************************');
  WritelN (out, 'eps=',eps:8:6);
  WritelN (out, 'init=',init:8:6);

  FOR i := 1 TO 22 DO BEGIN
    a[i] := shan;
  END;

  setje := [1,2,3,4,5,6];

  Write (out,'setje=');
  FOR i := 1 TO 22 DO
  IF i IN setje THEN BEGIN
                  Write (out,i:1,' ');
                  a[i] := init;
                  f[i] := 0;
                  c[i] := 0;
                END;
  WritelN (out);
  WritelN (out);
  old := V(a);
  WritelN (out, 'R=', old:18:16);
  einde := false;
  j := 0;
  WHILE NOT (einde) DO BEGIN
    i := 1;
    next := true;
    WHILE next DO BEGIN
       IF f[i] <> 3 THEN BEGIN
                        a[i] := init + f[i]*eps;

                        new := V(a);
                        IF new > old THEN BEGIN
                          old := new;
                          c := a;
                          WritelN (out);
                          WritelN;
                          WritelN (out, 'R=', old:18:16);
                          WritelN (     'R=', old:18:16);
                        END;

                        next := false;
                        f[i] := f[i] + 1;

                     END ELSE
                        IF i = 6 THEN BEGIN
                                    einde := true;
                                    next  := false;
```

```
                                    END ELSE BEGIN

                                      IF i > j THEN BEGIN
                                        j := i;
                                        WriteLN (j:1,' ');
                                        IF j = 5 THEN j := 0;
                                      END ;

                                      a[i] := init;
                                      f[i] := 0;
                                      i := i + 1;
                                      WHILE NOT (i IN setje) DO i := i + 1;
                                    END;
    END;

  END;

  WriteLN;
  WriteLN (out);
  FOR i := 1 TO 22 DO
    IF i IN setje THEN BEGIN
      WriteLN (      'c[',i:2,']=',c[i]:8:6);
      WriteLN (out, 'c[',i:2,']=',c[i]:8:6);
    END;


END.
```

Appendix 17: Program body `memlink6.p`

```
PROGRAM MEMLINK6 (input, output);

TYPE AR = ArraY [1..22] of Longreal;

VAR i,j,teller                  : Integer;
    old, oldprev, new, max      : Longreal;
    greater                     : Boolean;
    a                           : AR;
    out                         : Text;

CONST
    eps = 0.01;


BEGIN {of prog}

  Append (out, 'mem.link.6.out');
  Writeln (out);
  WritelN (out,'****************************************');
  Writeln (out, 'Nauwkeurigheid = 0.001');
  Writeln (out, 'eps = ' ,eps:8:6);
  WritelN (out,'a[1] aan a[2] gekoppeld');

  FOR i := 1 TO 22 DO BEGIN
    a[i] := 0.7;
  END;

  teller := 0;

  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WritelN;
    WritelN (teller:1, ' R=', old/6:18:16);

    FOR i := 2 TO 22 DO BEGIN
      Write (i:1,' ');
      a[i] := 0.001;
      old := V(a);
      max := a[i];

      FOR j := 1 TO 999 DO BEGIN
        a[i] := a[i] + 0.001;
        a[1] := a[2] + eps;
        new := V(a);
        IF new > old THEN BEGIN
          old := new;
          max := a[i];
        END
      END; {of j}

      a[i] := max;
    END; {of i}

  END; {of while}

  WritelN (out, 'Teller=',teller:1);
  WritelN;
  WritelN (     ' R=', old/6:18:16);
  WritelN (out, ' R=', old/6:18:16);

  FOR i := 1 TO 22 DO BEGIN
    WritelN (     'a[',i:2,']=',a[i]:18:16);
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  END;
END.
```

Appendix 18: Program body memloop6.rnd.p

```
PROGRAM MEMLOOP6.RND (input, output);

$SEARCH 'rndgen.o'$
IMPORT rndgen;

TYPE AR = ArraY [1..22] of Longreal;

VAR i,ii,j,teller                          : Longint;
    old, oldprev, new, max,dummy           : Longreal;
    greater                                : Boolean;
    a                                      : AR;
    out                                    : Text;
    verz                                   : SET OF 1..22;


BEGIN {of prog}

Append (out,'memloop6.rnd.out');
Writeln (out, 'Nauwkeurigheid = 0.0001');

InitRndm;
FOR i := 1 TO loopnr DO dummy := Rndm;

  FOR i := 1 TO 22 DO BEGIN
    a[i] := Rndm;
  END;

  teller := 0;

  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WritelN;
    WritelN (teller:1, ' R=', old:18:16);

    verz := [1..22];
    FOR ii := 1 TO 22 DO BEGIN
      i := TrunC (22 * Rndm + 1);
      WhilE NOT (i IN verz) DO i := TrunC (22 * Rndm + 1);
      verz := verz - [i];

        WritelN (i:1,' ');
        a[i] := 0;
        old := V(a);
        max := a[i];

        FOR j := 1 TO 10000 DO BEGIN

          a[i] := a[i] + 0.0001;
          if a[i] > 1 then a[i] := 1;
          new := V(a);
          IF new > old THEN BEGIN
            old := new;
            max := a[i];
          END

        END; {of j}

        a[i] := max;

    END; {of ii}

  END; {of while}

  WritelN (out, '**********************************');
  WritelN ('**********************************');
```

```
  WritelN (out, ' R=', old:18:16);
  FOR i := 1 TO 22 DO BEGIN
    WritelN (      'a[',i:2,']=',a[i]:18:16);
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  END;

END.
```

Appendix 19: Source code memseq7.p

```
PROGRAM MEMSEQ7 (input, output);

TYPE AR = ArraY [1..40] of Longreal;

VAR i,j,teller                  : Integer;
    old, oldprev, new, max      : Longreal;
    greater                     : Boolean;
    a                           : AR;
    out                         : Text;

FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a : AR)       : Longreal;
  VAR i                   : Integer;
      arg, ent, snt, avi : Longreal;
      b                   : AR;

  BEGIN

  FOR i := 1 TO 40 do b[i] := 1 - a[i];

{input symbol x_2,2}
{2}
  arg := a[1];
  ent := H(arg);
  avi := a[2] * ent;

{input symbol x_1,3}
{4 + 3}
  arg := a[2];
  ent := H(arg);
  snt := (b[1]*a[4] + a[1]*a[3]) * ent;

{input symbol x_2,4}
{6}
  arg := b[1]*a[4] + a[1]*a[3];
  ent := H(arg);
  snt := snt + b[2]*a[6] * ent;
{5}
  arg := a[4];
  ent := H(arg);
  snt := snt + b[1] * a[2]*a[5] * ent;
{7}
  arg := a[3];
  ent := H(arg);
  snt := snt + a[1] * a[2]*a[7] * ent;
  avi := avi + snt;

{input symbol x_1,5}
{11}
  arg := b[2]*a[6] + a[2]*a[5];
  ent := H(arg);
  snt := b[1]*b[4]*a[11] * ent;
{10}
  arg := a[6];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10] * b[2] * ent;
{13}
  arg := a[5];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13] * a[2] * ent;
{9}
  arg := b[2]*a[6] + a[2]*a[7];
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9] * ent;
{8}
```

```
  arg := a[6];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8] * b[2] * ent;
{12}
  arg := a[7];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12] * a[2] * ent;

{input symbol x_2,6}
{17}
  arg := b[1]*b[4]*a[11] + b[1]*a[4]*a[10] +
         a[1]*b[3]*a[9]  + a[1]*a[3]*a[8];
  ent := H(arg);
  snt := snt + b[2]*b[6]*a[17] * ent;
{16}
  arg := (b[1]*b[4]*a[11] + a[1]*b[3]*a[9]) /
         (b[1]*b[4]       + a[1]*b[3]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[16] * (b[1]*b[4] + a[1]*b[3]) * ent;
{19}
  arg := (b[1]*a[4]*a[10] + a[1]*a[3]*a[8]) /
         (b[1]*a[4]       + a[1]*a[3]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[19] * (b[1]*a[4] + a[1]*a[3]) * ent;
{15}
  arg := b[4]*a[11] + a[4]*a[13];
  ent := H(arg);
  snt := snt + a[2]*b[5]*a[15] * b[1] * ent;
{21}
  arg := b[3]*a[9] + a[3]*a[12];
  ent := H(arg);
  snt := snt + a[2]*b[7]*a[21] * a[1] * ent;
{14}
  arg := a[11];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[14] * b[1]*b[4] * ent;
{18}
  arg := a[13];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[18] * b[1]*a[4] * ent;
{20}
  arg := a[9];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[20] * a[1]*b[3] * ent;
{22}
  arg := a[12];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[22] * a[1]*a[3] * ent;
  avi := avi + snt;

{input symbol x_1,7}
{23}
  arg := a[17];
  ent := H(arg);
  snt := a[1]*a[3]*a[8]*a[23] * b[2]*b[6] * ent;
{24}
  arg := b[6]*a[17] + a[6]*a[16];
  ent := H(arg);
  snt := snt + a[1]*a[3]*b[8]*a[24] * b[2] * ent;
{25}
  arg := (b[2]*b[6]*a[17] + a[2]*b[7]*a[21]) /
         (b[2]*b[6]       + a[2]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9]*a[25] * (b[2]*b[6] + a[2]*b[7]) * ent;
{26}
  arg := b[2] * (b[6]*a[17] + a[6]*a[16]) +
         a[2] * (b[7]*a[21] + a[7]*a[20]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*b[9]*a[26] * ent;
{27}
  arg := a[17];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10]*a[27] * b[2]*b[6] * ent;
{28}
  arg := b[6]*a[17] + a[6]*a[19];
```

```
  ent := H(arg);
  snt := snt + b[1]*a[4]*b[10]*a[28] * b[2] * ent;
{29}
  arg := (b[2]*b[6]*a[17] + a[2]*b[5]*a[15]) /
         (b[2]*b[6]       + a[2]*b[5]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*a[11]*a[29] * (b[2]*b[6] + a[2]*b[5]) * ent;
{30}
  arg := b[2] * (b[6]*a[17] + a[6]*a[16]) +
         a[2] * (b[5]*a[15] + a[5]*a[14]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*b[11]*a[30] * ent;
{31}
  arg := a[19];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8]*a[31] * b[2]*a[6] * ent;
{32}
  arg := (b[2]*a[6]*a[16] + a[2]*a[7]*a[20]) /
         (b[2]*a[6]       + a[2]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9]*a[32] * (b[2]*a[6] + a[2]*a[7]) * ent;
{33}
  arg := a[19];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10]*a[33] * b[2]*a[6] * ent;
{34}
  arg := (b[2]*a[6]*a[16] + a[2]*a[5]*a[14]) /
         (b[2]*a[6]       + a[2]*a[5]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*a[11]*a[34] * (b[2]*a[6] + a[2]*a[5]) * ent;
{35}
  arg := a[21];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12]*a[35] * a[2]*b[7] * ent;
{36}
  arg := b[7]*a[21] + a[7]*a[22];
  ent := H(arg);
  snt := snt + a[1]*a[3]*b[12]*a[36] * a[2] * ent;
{37}
  arg := a[15];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13]*a[37] * a[2]*b[5] * ent;
{38}
  arg := b[5]*a[15] + a[5]*a[18];
  ent := H(arg);
  snt := snt + b[1]*a[4]*b[13]*a[38] * a[2] * ent;
{39}
  arg := a[22];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12]*a[39] * a[2]*a[7] * ent;
{40}
  arg := a[18];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13]*a[40] * a[2]*a[5] * ent;

  avi := avi + snt;

{input symbol x_2,8}
{'1}
  arg := a[1] * (a[3] * (a[8]*a[23] + b[8]*a[24]) +
                 b[3] * (a[9]*a[25] + b[9]*a[26])) +
         b[1] * (a[4] * (a[10]*a[27] + b[10]*a[28]) +
                 b[4] * (a[11]*a[29] + b[11]*a[30]));
  ent := H(arg);
  snt := b[2]*b[6]*b[17] * ent;
{'2}
  arg := (a[1] * (a[3]*b[8]*a[24] + b[3]*b[9]*a[26]) +
          b[1] * (a[4]*b[10]*a[28] + b[4]*b[11]*a[30])) /
         (a[1] * (a[3]*b[8]        + b[3]*b[9]) +
          b[1] * (a[4]*b[10]       + b[4]*b[11]));
  ent := H(arg);
  snt := snt + b[2]*b[6]*a[17] *
               (a[1] * (a[3]*b[8]  + b[3]*b[9]) +
                b[1] * (a[4]*b[10] + b[4]*b[11])) * ent;
{'3}
```

```
  arg := (a[1] * (a[3]*a[8]*a[23] + b[3]*a[9]*a[25]) +
          b[1] * (a[4]*a[10]*a[27] + b[4]*a[11]*a[29])) /
         (a[1] * (a[3]*a[8]        + b[3]*a[9]) +
          b[1] * (a[4]*a[10]       + b[4]*a[11]));
  ent := H(arg);
  snt := snt + b[2]*b[6]*a[17] *
               (a[1] * (a[3]*a[8]  + b[3]*a[9]) +
                b[1] * (a[4]*a[10] + b[4]*a[11])) * ent;
{'4}
  arg := (a[1]*b[3] * (a[9]*a[32] + b[9]*a[26]) +
          b[1]*b[4] * (a[11]*a[34] + b[11]*a[30])) /
         (a[1]*b[3] + b[1]*b[4]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*b[16] * (a[1]*b[3] + b[1]*b[4]) * ent;
{'5}
  arg := (a[1]*a[3] * (a[8]*a[31] + b[8]*a[24]) +
          b[1]*a[4] * (a[10]*a[33] + b[10]*a[28])) /
         (a[1]*a[3] + b[1]*a[4]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*b[19] * (a[1]*a[3] + b[1]*a[4]) * ent;
{'6}
  arg := (a[1]*b[3]*b[9]*a[26] + b[1]*b[4]*b[11]*a[30]) /
         (a[1]*b[3]*b[9]       + b[1]*b[4]*b[11]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[16] * (a[1]*b[3]*b[9] + b[1]*b[4]*b[11]) * ent;
{'7}
  arg := (a[1]*b[3]*a[9]*a[32] + b[1]*b[4]*a[11]*a[34]) /
         (a[1]*b[3]*a[9]       + b[1]*b[4]*a[11]);

  ent := H(arg);
  snt := snt + b[2]*a[6]*a[16] * (a[1]*b[3]*a[9] + b[1]*b[4]*a[11]) * ent;
{'8}
  arg := (a[1]*a[3]*b[8]*a[24] + b[1]*a[4]*b[10]*a[28]) /
         (a[1]*a[3]*b[8]       + b[1]*a[4]*b[10]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[19] * (a[1]*a[3]*b[8] + b[1]*a[4]*b[10]) * ent;
{'9}
  arg := (a[1]*a[3]*a[8]*a[31] + b[1]*a[4]*a[10]*a[33]) /
         (a[1]*a[3]*a[8]       + b[1]*a[4]*a[10]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[19] * (a[1]*a[3]*a[8] + b[1]*a[4]*a[10]) * ent;
{'10}
  arg := a[4] * (a[13]*a[37] + b[13]*a[38]) +
         b[4] * (a[11]*a[29] + b[11]*a[30]);
  ent := H(arg);
  snt := snt + a[2]*b[5]*b[15] * b[1] * ent;
{'11}
  arg := a[3] * (a[12]*a[35] + b[12]*a[36]) +
         b[3] * (a[9]*a[25] + b[9]*a[26]);
  ent := H(arg);
  snt := snt + a[2]*b[7]*b[21] * a[1] * ent;
{'12}
  arg := (a[4]*b[13]*a[38] + b[4]*b[11]*a[30]) /
         (a[4]*b[13] + b[4]*b[11]);
  ent := H(arg);
  snt := snt + a[2]*b[5]*a[15] * b[1]*(a[4]*b[13] + b[4]*b[11]) * ent;
{'13}
  arg := (a[4]*a[13]*a[37] + b[4]*a[11]*a[29]) /
         (a[4]*a[13] + b[4]*a[11]);
  ent := H(arg);
  snt := snt + a[2]*b[5]*a[15] * b[1]*(a[4]*a[13] + b[4]*a[11]) * ent;
{'14}
  arg := (a[3]*b[12]*a[36] + b[3]*b[9]*a[26]) /
         (a[3]*b[12] + b[3]*b[9]);
  ent := H(arg);
  snt := snt + a[2]*b[7]*a[21] * a[1]*(a[3]*b[12] + b[3]*b[9]) * ent;
{'15}
  arg := (a[3]*a[12]*a[35] + b[3]*a[9]*a[25]) /
         (a[3]*a[12] + b[3]*a[9]);
  ent := H(arg);
  snt := snt + a[2]*b[7]*a[21] * a[1]*(a[3]*a[12] + b[3]*a[9]) * ent;
{'16}
  arg := a[11]*a[34] + b[11]*a[30];
  ent := H(arg);
  snt := snt + a[2]*a[5]*b[14] * b[1]*b[4] * ent;
```

```
{'17}
  arg := a[13]*a[40] + b[13]*a[38];
  ent := H(arg);
  snt := snt + a[2]*a[5]*b[18] * b[1]*a[4] * ent;
{'18}
  arg := a[9]*a[32] + b[9]*a[26];
  ent := H(arg);
  snt := snt + a[2]*a[7]*b[20] * a[1]*b[3] * ent;
{'19}
  arg := a[12]*a[39] + b[12]*a[36];
  ent := H(arg);
  snt := snt + a[2]*a[7]*b[22] * a[1]*a[3] * ent;
{'20}
  arg := a[30];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[14] * b[1]*b[4]*b[11] * ent;
{'21}
  arg := a[34];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[14] * b[1]*b[4]*a[11] * ent;
{'22}
  arg := a[38];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[18] * b[1]*a[4]*b[13] * ent;
{'23}
  arg := a[40];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[18] * b[1]*a[4]*a[13] * ent;
{'24}
  arg := a[26];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[20] * a[1]*b[3]*b[9] * ent;
{'25}
  arg := a[32];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[20] * a[1]*b[3]*a[9] * ent;
{'26}
  arg := a[36];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[22] * a[1]*a[3]*b[12] * ent;
{'27}
  arg := a[39];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[22] * a[1]*a[3]*a[12] * ent;

  avi := avi + snt;

  V := avi/8;

  END; {of V(a)}


BEGIN {of prog}

Append (out,'memseq7.out');

  FOR i := 1 TO 40 DO BEGIN
    a[i] := 0.5;
  END;

  teller := 0;

  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WritelN;
    WritelN (teller:1, ' R=', old:18:16);

    FOR i := 1 TO 40 DO BEGIN

      WritelN (i:1,' ');
```

```
      a[i] := 0;
      old := V(a);
      max := a[i];

      FOR j := 1 TO 10000 DO BEGIN

        a[i] := a[i] + 0.0001;
        new := V(a);
        IF new > old THEN BEGIN
          old := new;
          max := a[i];
        END

      END; {of j}

      a[i] := max;

    END; {of i}

  END; {of while}

  WritelN (out, '************************');
  Writeln (out, 'Nauwkeurigheid = 0.0001');

  WritelN (out, 'Teller=',teller:1);
  WritelN;
  WritelN (     ' R=', old:18:16);
  WritelN (out, ' R=', old:18:16);

  FOR i := 1 TO 40 DO BEGIN
    WritelN (     'a[',i:2,']=',a[i]:18:16);
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  END;

END.
```

Appendix 20: Source code memloop7.p

```
PROGRAM MEMLOOP7 (input, output);

TYPE AR = ArraY [1..40] of Longreal;

VAR i,j,teller                                  : Longint;
    old, oldprev, new, max                      : Longreal;
    greater                                     : Boolean;
    a                                           : AR;
    out                                         : Text;

FUNCTION H (x : Longreal) : Longreal;
  BEGIN
    IF (x >= 1) OR (x <= 0) THEN H := 0
    ELSE H := -(x * Ln (x) + (1 - x) * Ln (1 - x)) / Ln(2)
  END;

FUNCTION V (a : AR)       : Longreal;
  VAR i                   : Integer;
      arg, ent, snt, avi  : Longreal;
      b                   : AR;

  BEGIN

  FOR i := 1 TO 40 do b[i] := 1 - a[i];

{input symbol x_2,2}
{2}
  arg := a[1];
  ent := H(arg);
  avi := a[2] * ent;

{input symbol x_1,3}
{4 + 3}
  arg := a[2];
  ent := H(arg);
  snt := (b[1]*a[4] + a[1]*a[3]) * ent;

{input symbol x_2,4}
{6}
  arg := b[1]*a[4] + a[1]*a[3];
  ent := H(arg);
  snt := snt + b[2]*a[6] * ent;
{5}
  arg := a[4];
  ent := H(arg);
  snt := snt + b[1] * a[2]*a[5] * ent;
{7}
  arg := a[3];
  ent := H(arg);
  snt := snt + a[1] * a[2]*a[7] * ent;
  avi := avi + snt;

{input symbol x_1,5}
{11}
  arg := b[2]*a[6] + a[2]*a[5];
  ent := H(arg);
  snt := b[1]*b[4]*a[11] * ent;
{10}
  arg := a[6];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10] * b[2] * ent;
{13}
  arg := a[5];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13] * a[2] * ent;
{9}
  arg := b[2]*a[6] + a[2]*a[7];
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9] * ent;
{8}
```

```
  arg := a[6];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8] * b[2] * ent;
{12}
  arg := a[7];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12] * a[2] * ent;

{input symbol x_2,6}
{17}
  arg := b[1]*b[4]*a[11] + b[1]*a[4]*a[10] +
         a[1]*b[3]*a[9]  + a[1]*a[3]*a[8];
  ent := H(arg);
  snt := snt + b[2]*b[6]*a[17] * ent;
{16}
  arg := (b[1]*b[4]*a[11] + a[1]*b[3]*a[9]) /
         (b[1]*b[4]       + a[1]*b[3]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[16] * (b[1]*b[4] + a[1]*b[3]) * ent;
{19}
  arg := (b[1]*a[4]*a[10] + a[1]*a[3]*a[8]) /
         (b[1]*a[4]       + a[1]*a[3]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[19] * (b[1]*a[4] + a[1]*a[3]) * ent;
{15}
  arg := b[4]*a[11] + a[4]*a[13];
  ent := H(arg);
  snt := snt + a[2]*b[5]*a[15] * b[1] * ent;
{21}
  arg := b[3]*a[9] + a[3]*a[12];
  ent := H(arg);
  snt := snt + a[2]*b[7]*a[21] * a[1] * ent;
{14}
  arg := a[11];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[14] * b[1]*b[4] * ent;
{18}
  arg := a[13];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[18] * b[1]*a[4] * ent;
{20}
  arg := a[9];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[20] * a[1]*b[3] * ent;
{22}
  arg := a[12];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[22] * a[1]*a[3] * ent;
  avi := avi + snt;

{input symbol x_1,7}
{23}
  arg := a[17];
  ent := H(arg);
  snt := a[1]*a[3]*a[8]*a[23] * b[2]*b[6] * ent;
{24}
  arg := b[6]*a[17] + a[6]*a[16];
  ent := H(arg);
  snt := snt + a[1]*a[3]*b[8]*a[24] * b[2] * ent;
{25}
  arg := (b[2]*b[6]*a[17] + a[2]*b[7]*a[21]) /
         (b[2]*b[6]       + a[2]*b[7]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9]*a[25] * (b[2]*b[6] + a[2]*b[7]) * ent;
{26}
  arg := b[2] * (b[6]*a[17] + a[6]*a[16]) +
         a[2] * (b[7]*a[21] + a[7]*a[20]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*b[9]*a[26] * ent;
{27}
  arg := a[17];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10]*a[27] * b[2]*b[6] * ent;
{28}
  arg := b[6]*a[17] + a[6]*a[19];
```

```
  ent := H(arg);
  snt := snt + b[1]*a[4]*b[10]*a[28] * b[2] * ent;
{29}
  arg := (b[2]*b[6]*a[17] + a[2]*b[5]*a[15]) /
         (b[2]*b[6]       + a[2]*b[5]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*a[11]*a[29] * (b[2]*b[6] + a[2]*b[5]) * ent;
{30}
  arg := b[2] * (b[6]*a[17] + a[6]*a[16]) +
         a[2] * (b[5]*a[15] + a[5]*a[14]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*b[11]*a[30] * ent;
{31}
  arg := a[19];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[8]*a[31] * b[2]*a[6] * ent;
{32}
  arg := (b[2]*a[6]*a[16] + a[2]*a[7]*a[20]) /
         (b[2]*a[6]       + a[2]*a[7]);
  ent := H(arg);
  snt := snt + a[1]*b[3]*a[9]*a[32] * (b[2]*a[6] + a[2]*a[7]) * ent;
{33}
  arg := a[19];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[10]*a[33] * b[2]*a[6] * ent;
{34}
  arg := (b[2]*a[6]*a[16] + a[2]*a[5]*a[14]) /
         (b[2]*a[6]       + a[2]*a[5]);
  ent := H(arg);
  snt := snt + b[1]*b[4]*a[11]*a[34] * (b[2]*a[6] + a[2]*a[5]) * ent;
{35}
  arg := a[21];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12]*a[35] * a[2]*b[7] * ent;
{36}
  arg := b[7]*a[21] + a[7]*a[22];
  ent := H(arg);
  snt := snt + a[1]*a[3]*b[12]*a[36] * a[2] * ent;
{37}
  arg := a[15];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13]*a[37] * a[2]*b[5] * ent;
{38}
  arg := b[5]*a[15] + a[5]*a[18];
  ent := H(arg);
  snt := snt + b[1]*a[4]*b[13]*a[38] * a[2] * ent;
{39}
  arg := a[22];
  ent := H(arg);
  snt := snt + a[1]*a[3]*a[12]*a[39] * a[2]*a[7] * ent;
{40}
  arg := a[18];
  ent := H(arg);
  snt := snt + b[1]*a[4]*a[13]*a[40] * a[2]*a[5] * ent;

  avi := avi + snt;

{input symbol x_2,8}
{'1}
  arg := a[1] * (a[3] * (a[8]*a[23] + b[8]*a[24]) +
                 b[3] * (a[9]*a[25] + b[9]*a[26])) +
         b[1] * (a[4] * (a[10]*a[27] + b[10]*a[28]) +
                 b[4] * (a[11]*a[29] + b[11]*a[30]));
  ent := H(arg);
  snt := b[2]*b[6]*b[17]*a[1] * ent;
{'2}
  arg := (a[1] * (a[3]*b[8]*a[24] + b[3]*b[9]*a[26]) +
          b[1] * (a[4]*b[10]*a[28] + b[4]*b[11]*a[30])) /
         (a[1] * (a[3]*b[8]        + b[3]*b[9]) +
          b[1] * (a[4]*b[10]       + b[4]*b[11]));
  ent := H(arg);
  snt := snt + b[2]*b[6]*a[17]*a[1] *
              (a[1] * (a[3]*b[8]  + b[3]*b[9]) +
               b[1] * (a[4]*b[10] + b[4]*b[11])) * ent;
{'3}
```

```
  arg := (a[1] * (a[3]*a[8]*a[23] + b[3]*a[9]*a[25]) +
          b[1] * (a[4]*a[10]*a[27] + b[4]*a[11]*a[29])) /
         (a[1] * (a[3]*a[8]        + b[3]*a[9]) +
          b[1] * (a[4]*a[10]       + b[4]*a[11]));
  ent := H(arg);
  snt := snt + b[2]*b[6]*a[17]*a[1] *
              (a[1] * (a[3]*a[8]  + b[3]*a[9]) +
               b[1] * (a[4]*a[10] + b[4]*a[11])) * ent;
{'4}
  arg := (a[1]*b[3] * (a[9]*a[32] + b[9]*a[26]) +
          b[1]*b[4] * (a[11]*a[34] + b[11]*a[30])) /
         (a[1]*b[3] + b[1]*b[4]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*b[16]*a[1] * (a[1]*b[3] + b[1]*b[4]) * ent;
{'5}
  arg := (a[1]*a[3] * (a[8]*a[31] + b[8]*a[24]) +
          b[1]*a[4] * (a[10]*a[33] + b[10]*a[28])) /
         (a[1]*a[3] + b[1]*a[4]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*b[19]*a[1] * (a[1]*a[3] + b[1]*a[4]) * ent;
{'6}
  arg := (a[1]*b[3]*b[9]*a[26] + b[1]*b[4]*b[11]*a[30]) /
         (a[1]*b[3]*b[9]       + b[1]*b[4]*b[11]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[16]*a[1] * (a[1]*b[3]*b[9] + b[1]*b[4]*b[11]) * ent;
{'7}
  arg := (a[1]*b[3]*a[9]*a[32] + b[1]*b[4]*a[11]*a[34]) /
         (a[1]*b[3]*a[9]       + b[1]*b[4]*a[11]);

  ent := H(arg);
  snt := snt + b[2]*a[6]*a[16]*a[1] * (a[1]*b[3]*a[9] + b[1]*b[4]*a[11]) * ent;
{'8}
  arg := (a[1]*a[3]*b[8]*a[24] + b[1]*a[4]*b[10]*a[28]) /
         (a[1]*a[3]*b[8]       + b[1]*a[4]*b[10]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[19]*a[1] * (a[1]*a[3]*b[8] + b[1]*a[4]*b[10]) * ent;
{'9}
  arg := (a[1]*a[3]*a[8]*a[31] + b[1]*a[4]*a[10]*a[33]) /
         (a[1]*a[3]*a[8]       + b[1]*a[4]*a[10]);
  ent := H(arg);
  snt := snt + b[2]*a[6]*a[19]*a[1] * (a[1]*a[3]*a[8] + b[1]*a[4]*a[10]) * ent;
{'10}
  arg := a[4] * (a[13]*a[37] + b[13]*a[38]) +
         b[4] * (a[11]*a[29] + b[11]*a[30]);
  ent := H(arg);
  snt := snt + a[2]*b[5]*b[15]*a[1] * b[1] * ent;
{'11}
  arg := a[3] * (a[12]*a[35] + b[12]*a[36]) +
         b[3] * (a[9]*a[25] + b[9]*a[26]);
  ent := H(arg);
  snt := snt + a[2]*b[7]*b[21]*a[1] * a[1] * ent;
{'12}
  arg := (a[4]*b[13]*a[38] + b[4]*b[11]*a[30]) /
         (a[4]*b[13] + b[4]*b[11]);
  ent := H(arg);
  snt := snt + a[2]*b[5]*a[15]*a[1] * b[1]*(a[4]*b[13] + b[4]*b[11]) * ent;
{'13}
  arg := (a[4]*a[13]*a[37] + b[4]*a[11]*a[29]) /
         (a[4]*a[13] + b[4]*a[11]);
  ent := H(arg);
  snt := snt + a[2]*b[5]*a[15]*a[1] * b[1]*(a[4]*a[13] + b[4]*a[11]) * ent;
{'14}
  arg := (a[3]*b[12]*a[36] + b[3]*b[9]*a[26]) /
         (a[3]*b[12] + b[3]*b[9]);
  ent := H(arg);
  snt := snt + a[2]*b[7]*a[21]*a[1] * a[1]*(a[3]*b[12] + b[3]*b[9]) * ent;
{'15}
  arg := (a[3]*a[12]*a[35] + b[3]*a[9]*a[25]) /
         (a[3]*a[12] + b[3]*a[9]);
  ent := H(arg);
  snt := snt + a[2]*b[7]*a[21]*a[1] * a[1]*(a[3]*a[12] + b[3]*a[9]) * ent;
{'16}
  arg := a[11]*a[34] + b[11]*a[30];
  ent := H(arg);
  snt := snt + a[2]*a[5]*b[14]*a[1] * b[1]*b[4] * ent;
```

```
{'17}
  arg := a[13]*a[40] + b[13]*a[38];
  ent := H(arg);
  snt := snt + a[2]*a[5]*b[18]*a[1] * b[1]*a[4] * ent;
{'18}
  arg := a[9]*a[32] + b[9]*a[26];
  ent := H(arg);
  snt := snt + a[2]*a[7]*b[20]*a[1] * a[1]*b[3] * ent;
{'19}
  arg := a[12]*a[39] + b[12]*a[36];
  ent := H(arg);
  snt := snt + a[2]*a[7]*b[22]*a[1] * a[1]*a[3] * ent;
{'20}
  arg := a[30];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[14]*a[1] * b[1]*b[4]*b[11] * ent;
{'21}
  arg := a[34];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[14]*a[1] * b[1]*b[4]*a[11] * ent;
{'22}
  arg := a[38];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[18]*a[1] * b[1]*a[4]*b[13] * ent;
{'23}
  arg := a[40];
  ent := H(arg);
  snt := snt + a[2]*a[5]*a[18]*a[1] * b[1]*a[4]*a[13] * ent;
{'24}
  arg := a[26];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[20]*a[1] * a[1]*b[3]*b[9] * ent;
{'25}
  arg := a[32];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[20]*a[1] * a[1]*b[3]*a[9] * ent;
{'26}
  arg := a[36];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[22]*a[1] * a[1]*a[3]*b[12] * ent;
{'27}
  arg := a[39];
  ent := H(arg);
  snt := snt + a[2]*a[7]*a[22]*a[1] * a[1]*a[3]*a[12] * ent;

  avi := avi + snt;

  V := avi/7;

  END; {of V(a)}


BEGIN {of prog}

Append (out,'memloop7.out');
Writeln (out, 'Nauwkeurigheid = 0.0001');

  FOR i := 1 TO 40 DO BEGIN
    a[i] := 0.5;
  END;

  teller := 0;

  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WritelN;
    WritelN (teller:1, ' R=', old:18:16);

    FOR i := 1 TO 40 DO BEGIN
        WritelN (i:1,' ');
```

```
        a[i] := 0;
        old := V(a);
        max := a[i];

        FOR j := 1 TO 10000 DO BEGIN

          a[i] := a[i] + 0.0001;
          if a[i] > 1 then a[i] := 1;
          new := V(a);
          IF new > old THEN BEGIN
            old := new;
            max := a[i];
          END

        END; {of j}

        a[i] := max;

    END; {of i}

  END; {of while}

  WritelN (out, ' R=', old:18:16);
  FOR i := 1 TO 40 DO BEGIN
    WritelN (     'a[',i:2,']=',a[i]:18:16);
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  END;

END.
```

Appendix 21: Program body memseq7.rnd.p

```
PROGRAM MEMSEQ7.RND (input, output);

$SEARCH 'rndgen.o'$
IMPORT rndgen;

TYPE AR = ArraY [1..40] of Longreal;

VAR i,ii,j,teller               : Integer;
    old, oldprev, new, max      : Longreal;
    greater                     : Boolean;
    a                           : AR;
    out                         : Text;
    verz                        : SET OF 1..40;


BEGIN {of prog}

Append (out,'memseq7.rnd.out');
InitRndm;

  FOR i := 1 TO 40 DO BEGIN
    a[i] := Rndm;
  END;

  teller := 0;

  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WritelN;
    WritelN (teller:1, ' R=', old:18:16);

    verz := [1..40];
    FOR ii := 1 TO 40 DO BEGIN
      i := TrunC (40 * Rndm + 1);
      WhilE NOT (i IN verz) DO i := TrunC (40 * Rndm + 1);
      verz := verz - [i];

        WritelN (i:1,' ');
        a[i] := 0;
        old := V(a);
        max := a[i];

        FOR j := 1 TO 10000 DO BEGIN

          a[i] := a[i] + 0.0001;
          new := V(a);
          IF new > old THEN BEGIN
            old := new;
            max := a[i];
          END

        END; {of j}

        a[i] := max;

    END; {of ii}

  END; {of while}

  WritelN (out, '************************');
  Writeln (out, 'Nauwkeurigheid = 0.0001');

  WritelN (out, 'Teller=',teller:1);
  WritelN;
  WritelN (     ' R=', old:18:16);
```

```
  WritelN (out, ' R=', old:18:16);

  FOR i := 1 TO 40 DO BEGIN
    WritelN (     'a[',i:2,']=',a[i]:18:16);
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  END;

END.
```

Appendix 22: Program body memloop7.rnd.p

```
PROGRAM MEMLOOP7.RND (input, output);

$SEARCH 'rndgen.o'$
IMPORT rndgen;

TYPE AR = ArraY [1..40] of Longreal;

VAR i,ii,j,teller,loopnr                    : Longint;
    old, oldprev, new, max,dummy,loop_rate  : Longreal;
    greater                                 : Boolean;
    a                                       : AR;
    out                                     : Text;
    verz                                    : SET OF 1..40;


BEGIN {of prog}

Append (out,'memloop7.rnd.out');
loopnr := 0;
loop_rate := 0;
Writeln (out, 'Nauwkeurigheid = 0.0001');

REPEAT
InitRndm;
FOR i := 1 TO loopnr DO dummy := Rndm;

  FOR i := 1 TO 40 DO BEGIN
    a[i] := Rndm;
  END;

  teller := 0;

  old := V(a);
  oldprev := 1.0;

  WHILE (oldprev <> old) DO BEGIN

    oldprev := old;
    teller := teller + 1;
    WritelN;
    WritelN (teller:1, ' R=', old:18:16);

    verz := [1..40];
    FOR ii := 1 TO 40 DO BEGIN
      i := TrunC (40 * Rndm + 1);
      WhilE NOT (i IN verz) DO i := TrunC (40 * Rndm + 1);
      verz := verz - [i];

        WritelN (i:1,' ');
        a[i] := 0;
        old := V(a);
        max := a[i];

        FOR j := 1 TO 10000 DO BEGIN

          a[i] := a[i] + 0.0001;
          if a[i] > 1 then a[i] := 1;
          new := V(a);
          IF new > old THEN BEGIN
            old := new;
            max := a[i];
          END

        END; {of j}

        a[i] := max;

    END; {of ii}

  END; {of while}
```

```
  IF old > loop_rate then begin
    loop_rate := old;
    WritelN (out, '**********************************');
    WritelN ('**********************************');
  END;

  WritelN (out, 'loopnr=',loopnr:1);
  WritelN (out, ' R=', old:18:16);
  FOR i := 1 TO 40 DO BEGIN
    WritelN (     'a[',i:2,']=',a[i]:18:16);
    WritelN (out, 'a[',i:2,']=',a[i]:18:16);
  END;

loopnr := loopnr + 1;
UNTIL FALSE

END.
```