The Spectrality Decision Problem

E.V. Dubrova J.C. Muzio

VLSI Design and Test Group
Department of Computer Science
University of Victoria,
P.O.Box 3055, Victoria, B.C.,
Canada, V8W 3P6

Abstract

An efficient algorithm for deciding whether a given integer vector is the spectrum of some Boolean function is presented. The algorithm performs a step-by-step spectral decomposition of the input vector and checks at each step a set of necessary conditions for spectrality for the resulting vectors. The algorithm concludes that the input vector cannot lead to a valid Boolean function as soon as a vector not satisfying the conditions is found, which, as proved in the paper, for almost all cases happens after the first step of the decomposition.

1 Introduction

A Boolean function $f(x_1, \ldots, x_n)$ can be transformed from the usual domain ($\{0,1\}$) into the spectral domain by the transformation $T \cdot Y = S$, where T is a $2^n \times 2^n$ orthogonal transform matrix, Y is the two-valued truth table vector of $f(x_1, \ldots, x_n)$, and S is the resultant vector of integers (spectral coefficients) uniquely defining $f(x_1, \ldots, x_n)$. The original two-valued data is fully recoverable from S by application of the inverse transform $[T]^{-1}$.

The exploitation of the spectral data provides a sound mathematical basis for many applications in digital logic design. Some of them include Boolean function classification [1], [4], [15], logic network synthesis [4], [5], [6], [7], [8], [9], [11], [16], fault diagnosis [12], [13], [14], and others. Spectral based methodologies have also proved very effective in other digital areas such as signal processing [3] and transmission of information [2]. In many applications the spectral data is first generated, then manipulated in accordance with the application, and finally transformed back to the Boolean domain.

There are a number of situations where we are faced with a set of integers, and need to know if they represent the spectrum of a Boolean function. For example, when considering the spectrum of an incompletely specified function, such a situation always arises (see [15]). Since the number of all possible two-valued functions of n variables is 2^{2^n}, there are also only 2^{2^n} integer vectors of length 2^n defining some Boolean function $f(x_1, \ldots, x_n)$. An interesting theoretical problem is: "Given an integer vector of length 2^n, how to check efficiently whether this vector is the spectrum of some Boolean function $f(x_1, \ldots, x_n)$?". We call this problem "The spectrality decision problem".

One way to solve the spectrality decision problem is to verify the classical property of spectral coefficients - idempotenicity with respect to autoconvolution [5]. This can be done in $O(N^2)$ time, where $N = 2^n$ is the length of the vector under consideration. A faster way ($O(N \log_2 N)$) is to perform a fast inverse transform, and to check whether the resulting vector consists of entries from $\{0, 1\}$ (or $\{-1, 1\}$, depending on the encoding used). However, the disadvantage of both approaches is that the same amount of time is spent to decide spectrality in all cases. Since the number of vectors which are not spectra is considerably larger than the number of those which are, an algorithm which needs less time on deciding spectrality negatively would be more effective. These considerations have motivated us in the development of the algorithm for deciding spectrality, presented in this paper. It spends asymptotically the same time as a fast inverse transform to decide spectrality with a positive answer. However, in almost all cases it needs only $O(N)$ time to conclude that the input vector cannot lead to a valid Boolean function.

The paper is organized as follows. In Section 2, the relationship between the spectrum of a Boolean function and the spectra of all subfunctions involved in a general Shannon decomposition of that function is described. This relationship provides the mathematical basis for the algorithm for deciding spectrality, presented in Section 3. In Section 4 the evaluation of the algorithm is given. In the final Section, some conclusions are drawn and a topic for further research is proposed.

2 Spectral Decomposition

In this section we describe the relationship between the spectrum of a Boolean function and the spectra of all subfunctions involved in a general Shannon decomposition of that function [15], which we call spectral decomposition. We show that an inverse transform can be appreciated as a step-by-step application of this decomposition. This view of the inverse transform gives rise to the approach to deciding spectrality, presented here.

We employ spectra generated from the $\{+1, -1\}$ encoding of two-valued data by applying the Hadamard transform matrix, defined inductively by:

$$\begin{aligned} T^0 &= [1] \\ T^n &= \begin{bmatrix} T^{n-1} & T^{n-1} \\ T^{n-1} & -T^{n-1} \end{bmatrix} \end{aligned}$$

Let S be the spectrum of a Boolean function $f(x_1, \ldots, x_n)$, and S_0 and S_1 be the spectra of the subfunctions $f(x_1, \ldots, x_{n-1}, 0)$ and $f(x_1, \ldots, x_{n-1}, 1)$ involved in a general Shannon decomposition of that function. It is shown in [15], that we can express S_0 and S_1 as:

$$[S_0 \; S_1] = \frac{1}{2}\left\{[S^0 \; S^1]T^1\right\} \tag{1}$$

where S^0 and S^1 is the ordered partition of S into two proper halves, for which

$$S = \begin{matrix} S^0 \\ S^1 \end{matrix} \Bigg]$$

Further we refer to equation (1) as the *spectral decomposition* of S in variable x_n. Notice that, similarly to Shannon decomposition, spectral decomposition of the above type can be performed with respect to any variable x_i, $i \in \{1, \ldots, n\}$, with the appropriate reordering and partitioning of S into S^0 and S^1. Then (1) generates the spectra S_0 and S_1 of the two subfunctions $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$ and $f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$, respectively.

Equation (1) may readily be extended to higher orders. The general result, the formal proof of which is given by Tokmen [10], is

$$[S_0\ S_1\ \ldots\ S_\beta] = \frac{1}{2^{n-m}} \left\{ [S^0\ S^1\ \ldots\ S^\beta] T^{n-m} \right\} \tag{2}$$

where $\beta = 2^{n-m} - 1$. There are $n - m$ detached variables in the Shannon decomposition corresponding to (2). $S^0, S^1, \ldots, S^\beta$ is the ordered partition of the given spectrum S of $f(x_1, \ldots, x_n)$.

Equation (2) may be appreciated as being a partial application of the inverse transform. In the extreme case of $m = 0$, the right-hand side of (2) becomes the inverse transform on the individual spectral coefficients, in row rather than column order, and the left-hand side becomes a truth table row vector Y representing $f(x_1, \ldots, x_n)$. Alternatively, the inverse transform can be comprehended as a step by step application of spectral decomposition. After n steps decomposition in all n variables is performed and the truth table vector Y is obtained.

An immediate advantage of viewing the inverse transform as a step-by-step application of spectral decomposition is the following. It gives the information that the vectors obtained after each step of the transform are spectra of some Boolean functions, provided the original vector S is a spectrum of a Boolean function. The contrapositive of this statement gives that, if at least one of the vectors obtained after each step of the transform is not a spectrum, then S is a not a spectrum. So, if while performing the inverse transform we in addition check at each step some *necessary conditions* for spectrality, we can find that S cannot lead to a valid Boolean function as soon as a vector not satisfying some of the conditions is reached, possibly far before the end of the transform. This is the idea on which the algorithm described in the next section is based.

3 An algorithm for deciding spectrality

The algorithm described in this section estimates whether a given vector is the spectrum of some Boolean function by performing a step-by-step spectral decomposition in a given variable and checking, at each step, a set C of necessary conditions for spectrality for the resulting vectors. Since the Shannon decomposition may be performed in any variable x_i, $i \in \{1, \ldots, n\}$, there are also n different ways to decompose a spectrum S of a given function of n variables into two spectra of the subfunctions of $n - 1$ variables. So, at each step $i, i \in \{1, \ldots, n-1\}$ the decision should be made as to which of the remaining $n - i + 1$ variables is to be chosen for the spectral decomposition. The variable ordering may have some effect on how soon the vector not satisfying the necessary conditions is found. However, in this paper we do not consider the problem of the optimal choice of the variable ordering. For convenience, throughout the paper, we assume that the spectral decomposition is

always performed with respect to the lowest order variable x_n.

Let $S_{u_1 \ldots u_{n-m}}$ denote the spectrum of subfunction $f(x_1, \ldots, x_m, u_1, \ldots, u_{n-m})$ of a Boolean function $f(x_1, \ldots, x_n)$, where $u_i \in \{0, 1\}$, and let C be a set of conditions which are satisfied whenever a given vector is the spectrum of some Boolean function.

A step-by-step spectral decomposition of a vector S with length 2^n can be implemented as a binary tree breadth-first traversal with S as the root (at level 0), S_0 and S_1 its left and right children nodes (at level 1), and further the nodes $S_{0u_1u_2 \ldots u_k}$ and $S_{1u_1u_2 \ldots u_k}$ being the left and right children nodes of the node $S_{u_1u_2 \ldots u_k}$ (at level $k+1$), $k \in \{1, 2, \ldots n-1\}$. The tree is generated level by level starting from S. For each node the conditions from C are checked. Whenever the conditions are not simultaneously satisfied, the algorithm stops with a negative answer. Otherwise the left and right children of the node under consideration are constructed and placed in a queue Q of vectors. The use of a queue allows us to retrieve the nodes in the same order as they are generated so that the items can be processed level by level. The algorithm is given in pseudocode on Figure 1. The procedure *spectrum(S, n)* returns TRUE if and only if the input vector S is the spectrum of some n variable Boolean function.

```
Boolean spectrum(S, n) {
        QUEUE_OF_VECTORS Q;
        Q.Initialize;
        Q.Enqueue(S);
        while (Q.NotEmpty) {
                S = Q.Dequeue;
                CheckConditions(S);
                if (one of the conditions from C does not hold for S)
                        return(FALSE);
                if (length of S is greater than 1) {
                        S^0 = first half of S;
                        S^1 = second half of S;
                        S_0 = 1/2(S^0 + S^1);
                        S_1 = 1/2(S^0 - S^1);
                        Q.Enqueue(S_0);
                        Q.Enqueue(S_1);
                }
        }
        return(TRUE);
}
```

Figure 1: Pseudocode of the algorithm for deciding spectrality.

The algorithm described is a decision procedure, i.e. it just make a qualitative analysis for spectrality. However, it can be easily modified to output the corresponding Boolean function in cases

when the original vector is a spectrum, by keeping track of the last 2^n nodes of the tree (the leaves).

The choice of a set of conditions C is crucial to the algorithm's performance. From the pseudocode of the algorithm one may see that it is determined by two factors:

1. The number of levels of the tree to be checked before contradiction with necessary conditions is found;

2. The time required to check conditions for a node on a given level.

On one hand, the more properties of spectral coefficients are checked, the more probable it is that violation of a necessary condition is found earlier. On the other hand, more properties means more time to be spent on each level for checking them. By analyzing different properties of spectral coefficients, we have found that the set consisting of following two properties gives a good balance between these conflicting factors:

C1. The sum of all spectral coefficients of S for any fully defined function is $\pm 2^n$

C2. Each individual coefficient belongs to the set $R \in \{-2^n, -2^n + 2, \ldots, 0, \ldots, 2^n - 2, 2^n\}$

We show in the next section that the properties C1 and C2 guarantee that, provided the input vector in not a spectrum, in almost all cases the algorithm terminates at the first level of the tree.

4 Evaluation of the algorithm

In this section we evaluate the complexity of the algorithm using the set $C = \{C1, C2\}$ of necessary conditions. Although we do not exclude the possibility of a better choice for the set of necessary conditions, the above set seems to provide a good balance between the number of levels of the tree usually traversed before termination and the time required to check the conditions at each level. We perform the analysis separately for the two cases when the input vector is a spectrum and when it is not.

4.1 Input vector is a spectrum

If the input vector is a spectrum, the algorithm always traverses all n levels of the tree and terminates after examining the last mostright leaf. So, the complexity of the algorithm in this case is $c_1 N \log_2 N$, where $N = 2^n$ and c_1 is a constant equal to the total number of operations executed by the algorithm to check conditions C1 and C2 for all nodes at one level of the tree.

4.2 Input vector is not a spectrum

If the input vector is not a spectrum, the complexity of the algorithm depends on the number of levels of the tree traversed before the algorithm terminates. We evaluate the probability of the algorithm terminating on the ith level of the tree for a given i.

An exact formula for the probability for the case than the input vector is not a spectrum is a rather complicated one. It is much simpler to consider a general case when all vectors (spectra and not) are possible as input space of the algorithm. Since input vectors which are spectra always lead to traversing all n levels of the tree, the probability for the general case yields a lower bound on the the probability for the case than the input vector is not a spectrum. As shown bellow, using the lower bound instead of an exact formula gives sufficient accuracy for our analysis.

We perform the analysis under the following two assumptions:

1. The original input to the algorithm is a vector satisfying properties C1 and C2.
2. All possible vectors are equally likely.

Assumption (1) is rather restrictive; however, we chose it deliberately because it is reasonable to assume that in practical applications the input space will be larger, and hence the algorithm will perform even better in practice.

The notation $N(l,s,r)$ used in the Lemma 1 below is defined inductively as follows:

$$N(1,s,r) := \begin{cases} 1 & \text{if } -r \le s \le r \\ 0 & \text{otherwise} \end{cases}$$

$$N(l,s,r) := \sum_{i=-r/2}^{r/2} N(l-1, s-2i, r),$$

where l is a positive integer, and s and r are even positive integers such that $r \le s$. It is shown in the Appendix that $N(l,s,r)$ gives the number of all possible ways of assigning values from $R \in \{-r, -r+2, \ldots, r-2, r\}$ to l items so that they sum up to s.

Lemma 1 *Assuming the inputs of the algorithm are random vectors of length 2^n satisfying C1 and C2, the probability $\overline{P}(n,i)$ that the algorithm terminates on the ith level of the tree is:*

$$\overline{P}(n,i) = 1 - \prod_{j=1}^{i} \left(\frac{2N(2^{n-j}, 2^{n-j}, 2^{n-j})^2}{N(2^{n-j+1}, 2^{n-j+1}, 2^{n-j+1})} \right)^{2^{j-1}}$$

where $1 \le i \le n$.

Proof: given in the Appendix.

For example, the values for the probability $\overline{P}(n,i)$ for $n = 3$ are the following:

$$\overline{P}(3,0) = 0$$
$$\overline{P}(3,1) = 0.99539$$
$$\overline{P}(3,2) = 0.99993$$

We see that for $n = 3$ the probability that the algorithm terminates on the first level is more than 99%. Since the formula for $\overline{P}(n,i)$ is recursive, it doesn't give us any insight as to how $\overline{P}(n,i)$ changes with the change of n. We have proved that for a fixed i, $\overline{P}(n,i)$ increases exponentially as n increases. This result is stated in Lemma 2.

Lemma 2 *The probability* $\overline{P}(n,i)$ *is such that:*

$$\overline{P}(n,i) > 1 - \frac{1}{2^{h(n,i)}}$$

where $h(n,i) := \prod_{j=1}^{i} 2^{j-1}(n-j-1)$, *and* $1 \leq i \leq n$.

Proof: given in the Appendix.

The main theorem of the paper follows directly from Lemmata 1 and 2.

Theorem *If the inputs of the algorithm are random vectors of length* 2^n *satisfying C1 and C2, then the probability that that algorithm terminates on the first level of the tree is larger than 99%.*

From the above result, we can make a conclusion that, if the input vector is not a spectrum, in more than 99% of cases the algorithm traverses the zero and first levels of the tree only and terminates. So, the complexity of the algorithm for more than 99% of the vectors which are not spectra is $c_2 N$, where $N = 2^n$ and c_2 is a constant equal to the total number of operations executed by the algorithm to check conditions C1 and C2 for the input vector S and its left and right children nodes S_0 and S_1.

5 Conclusion

This paper presents an algorithm for deciding whether a given integer vector is the spectrum of some Boolean function. The algorithm performs a step-by-step spectral decomposition of the input vector S and checks at each step a set of necessary conditions for spectrality for the resulting vectors. The algorithm concludes that S cannot lead to a valid Boolean function as soon as a vector not satisfying the conditions is found, which, as proved in the paper, for almost all cases happens after the first step of the decomposition.

The algorithm presented is a "negative" one, i.e. it is more efficient when terminating with a negative answer. For more than 99% of vectors which are not spectra, it terminates in $O(N)$ time ($N = 2^n$) which is asymptotically better as compared to $O(N \log_2 N)$ of a fast inverse transform. However, it always takes $O(N \log_2 N)$ time to terminate with a positive answer. So, the algorithm would be more efficient in applications where the probability that the input vector is not a spectrum is larger than the probability that the input vector is a spectrum.

The result obtained for the probability of the algorithm terminating after performing the first step of spectral decomposition is based the assumption that the input of the algorithm are vectors satisfying properties C1 and C2. This assumption is rather restrictive. However, we chose it deliberately because it is reasonable to assume that in practical applications the input space will be larger, and hence the algorithm will perform even better.

Although throughout the paper we assumed that the spectrum S is generated by applying the Hadamard transform matrix, it can obviously also be used for spectra generated using other transformation matrices such as Rademacher-Walsh with an additional reordering of spectral coefficients.

Further research needs to be done to estimate the best selection of variables with respect to which the spectral decomposition is performed, and its effect on the performance of the algorithm.

Acknowledgment

The authors would like to acknowledge the kind assistance of the editor in the preparation of the final version of the paper.

References

[1] C. K. Chow, On the characterization of threshold functions, *IEE Special Pub.* S. 134 (1961), 34-38.

[2] H. F. Harmuth, Transmission of Information by Orthogonal Series, Springer-Verlag, New York, 1972.

[3] N. Ahmed, K. R. Rao, Orthogonal Transforms for Digital Signal Processing, Springer-Verlag, New York, 1975.

[4] C. R. Edwards, The application of Rademacher-Walsh transform to Boolean function classification and threshold synthesis, *IEEE Trans. on Computers* **C-24** (1975), 48-62.

[5] M. G. Karpovsky, Finite Orthogonal Series in the Design of Digital Devices, New York: Wiley, 1976.

[6] C. R. Edwards, The design of easily tested circuits using mapping and spectral techniques, *Radio Electron. Eng.* **47** No. 7 (1977), 321-342.

[7] A. M. Lloyd, Spectral addition techniques for the synthesis of multivariable logic networks, *IEE Comp. Digital Tech.* **1** (1978), 152-164.

[8] S. L. Hurst, The Logical Processing of Digital Signals, Crane-Russak, New York and Edward Arnold, London, 1978.

[9] A. M. Lloyd, Design of multiplexor universal-logic-module networks using spectral techniques, *Proc. IEE* **127 E** (1980), 31-36.

[10] V. H. Tokmen, Disjoint decomposability of multi-valued functions by spectral means, *Proc. IEEE 10th Internat. Symp. Multiple-Valued Logic* (1980), 88-93.

[11] S. L. Hurst, The Haar transform in digital network synthesis, *Proc. IEEE 11th Internat. Symp. Multiple-Valued Logic* (1981), 10-18.

[12] A. K. Susskind, Testing by verifying Walsh coefficients, *IEEE Trans. on Computers* **C-32** (1983), 198-201.

[13] D. M. Miller, J. C. Muzio, Spectral fault signatures for single stuck-at faults in combinational networks, *IEEE Trans. on Computers* **C-33** (1984), 765-768.

[14] T. C. Hsiao, S. C. Seth, An analysis of the use of Rademacher-Walsh spectrum in compact testing, *IEEE Trans. on Computers* **C-33** (1984), 934-937.

[15] S. L. Hurst, D. M. Miller, J. C. Muzio, Spectral Techniques in Digital Logic, Academic Press Inc., 1985.

[16] M. A. Thornton, V. S. S. Nair, An Iterative combinational logic synthesis technique using spectral information, in *Proc. of Europ. Design Automat. Conf.* (1993), 358-363.

APPENDIX

1. Proof of Lemma 1

We divide the proof into two parts. First we show, that if the inputs of the algorithm are random vectors of length 2^n satisfying a set of conditions C, then the probability $\overline{P}(n,i)$ that the algorithm terminates on the ith level of the tree for $1 \leq i \leq n$ is:

$$\overline{P}(n,i) = 1 - \prod_{j=1}^{i} \left(\frac{|K_C(n-j)|^2}{|K_C(n-j+1)|} \right)^{2^{j-1}} \tag{3}$$

where $K_C(j)$, $j \in \{0,\ldots,n\}$, is a set of all vectors of length 2^j satisfying a set conditions C. Then we give the proof for the size of $K_C(j)$, for $C = \{C1, C2\}$, showing that $|K_{\{C1,C2\}}(j)| = 2N(2^j, 2^j, 2^j)$.

PART 1. Proof of (3)

Let $U(n)$ be the set of all possible vectors of length 2^n which are the inputs for our algorithm, and let $K_C(n)$ be the subset of $U(n)$ consisting of all vectors satisfying some set of conditions C. The probability that a random vector $V \in U(n)$ is in $K_C(n)$ is $\frac{|K_C(n)|}{|U(n)|}$. If $V \in K_C(n)$, then the algorithm continues, otherwise it terminates. So, the probability $P(n,0)$ that the algorithm passes level 0 of the tree is:

$$P(n,0) = \frac{|K_C(n)|}{|U(n)|}$$

We assumed $U(n) = K_C(n)$, so $P(n,0)$ is always 1. Further we derive the formula for $P(n,i)$ with $1 \leq i \leq n$.

Since the tree is binary, the number of nodes at any level $i \in \{0,\ldots,n\}$ is 2^i. On the other hand, on each subsequent level the length of the vectors is halved, so on level i the nodes are vectors of length 2^{n-i}. Consider a node V on level $i-1$ and assume that it satisfies conditions C, i.e. it is in $K_C(n-i+1)$. The probability that both its children nodes V_0 and V_1 are in $K_C(n-i)$ is N_1/N_2, where N_2 is the number of possible pairs (V_0, V_1) which can be constructed from vectors in $K_C(n-i+1)$, and N_1 is the number of possible pairs (V_0, V_1) which can be constructed from vectors in $K_C(n-i+1)$ and are in $K_C(n-i)$.

The mapping which maps V onto a pair of vectors (V_0, V_1) is bijective, so $N_2 = |K_C(n-i+1)|$. On the other hand, $K_C(n-i) \times K_C(n-i)$ is always a subset of the set consisting of all possible pairs (V_0, V_1) constructed from vectors in $K_C(n-i+1)$, because the properties C1 and C2 always hold for any V constructed form any pair $(V_0, V_1) \in K_C(n-i) \times K_C(n-i)$. So, $N_1 = |K_C(n-i)|^2$. Thus, the fraction $\frac{|K_C(n-i)|^2}{|K_C(n-i+1)|}$ gives the probability of one node on level $i-1$ which is in $K_C(n-i+1)$ to have both children nodes in $K_C(n-i)$.

The algorithm doesn't terminate on level i only if passed successfully level $i-1$ and *all* nodes on level $i-1$ have both children in $K_C(n-i)$. Hence the probability $P(n,i)$ that the algorithm passes the ith level of the tree for $1 \leq i \leq n$ is:

$$
\begin{aligned}
P(n,i) &= P(n,i-1)\cdot\left(\tfrac{|K_C(n-i)|^2}{|K_C(n-i+1)|}\right)^{2^{i-1}} \\
&= P(n,0)\cdot\prod_{j=1}^{i}\left(\frac{|K_C(n-j)|^2}{|K_C(n-j+1)|}\right)^{2^{j-1}} \\
&= \prod_{j=1}^{i}\left(\frac{|K_C(n-j)|^2}{|K_C(n-j+1)|}\right)^{2^{j-1}}
\end{aligned}
$$

And thus the probability $\overline{P}(n,i)$ that the algorithm terminates on the ith level of the tree for $1 \le i \le n$ is:

$$
\overline{P}(n,i) = 1 - P(n,i) = 1 - \prod_{j=1}^{i}\left(\frac{|K_C(n-j)|^2}{|K_C(n-j+1)|}\right)^{2^{j-1}}
$$

□

PART 2. Proof for the size of $K_{\{C1,C2\}}(j)$

In a general setting, the problem of finding the number of all possible vectors of a given length satisfying conditions $C1$ and $C2$ can be formulated as follows. Having a list of l "items" and a set of "values" $R \in \{-r, -r+2, \ldots, r-2, r\}$, find the number of all possible ways of assigning values from R to l items so that they sum up to s. Lemma 3 gives the solution to this problem. The notation $N(l,s,r)$ used in the Lemma 3 is defined inductively as follows:

$$
\begin{aligned}
N(1,s,r) &:= \begin{cases} 1 & \text{if } -r \le s \le r \\ 0 & \text{otherwise} \end{cases} \\
N(l,s,r) &:= \sum_{i=-r/2}^{r/2} N(l-1, s-2i, r),
\end{aligned}
$$

where l is a positive integer, and s and r are even positive integers such that $r \le s$.

Lemma 3 *The number of all possible ways of assigning values from $R \in \{-r, -r+2, \ldots, r-2, r\}$ to l items so that they sum up to s is $N(l,s,r)$.*

Proof: By induction on l.
1) Basis: $l = 1$. Suppose $s \in R$. Then obviously $N(1,s,r) = 1$, since the only assignment is s itself. On the other hand, if $s \notin R$, then no assignment exists, and thus $N(1,s,r) = 0$. So, the statement holds for $l = 1$.
2) Hypothesis: Assume the statement holds for l. We may break $N(l+1,s,r)$ into a choice of values for the first item and, for each choice of the first value, an assignment of values for the remaining l items. There are $2^n + 1$ choices for the value of the first item, namely $j = 2i$, where $i \in \{-r/2, -r/2+2, \ldots, r/2-2, r/2\}$. For each such choice j by the inductive hypothesis there are $N(l, s-j, r)$ assignments of values for the remaining l items. The total number of assignments is thus

$$
N(l+1,s,r) = \sum_{i=-r/2}^{r/2} N(l, s-2i, r).
$$

Hence the statement holds for $l+1$.

□

In our case the length of the list is always a power of two, i.e. $l = 2^j$, for some $j \in \{0, \ldots, n\}$, the range is $R \in \{-2^j, -2^j+2, \ldots, 2^j-2, 2^j\}$ and the sum s is either -2^j or 2^j. Therefore, the number of all possible vectors of a length 2^j satisfying $C1$ and $C2$ is $|K_{\{C1,C2\}}(j)| = N(2^j, 2^j, 2^j) + N(2^j, -2^j, 2^j) = 2N(2^j, 2^j, 2^j)$, since obviously $N(2^j, 2^j, 2^j) = N(2^j, -2^j, 2^j)$. Lemma 1 follows.

2. Proof of Lemma 2

First, we show that for any even positive integer $l \geq 2$:

$$\frac{N(l/2, s/2, r/2)^2}{N(l, s, r)} < 2/l$$

We may break $N(l, s, r)$ into a choice of values for the $l/2$ items and, for each choice of the first value, an assignment of values for the remaining $l/2$ items. Obviously, $N(l, s, r)$ is at least $N(l/2, s/2, r/2)^2$, since for each of $N(l/2, s/2, r/2)$ choices for the value of the first $l/2$ items, there are $N(l/2, s/2, r/2)$ assignments of values for the remaining $l/2$ items.

Let $A = \{a_1, a_2, \ldots, a_l\}$ denotes the values assigned to l items by one of these $N(l/2, s/2, r/2)^2$ combinations. Since any a_i is in the range $\{-r/2, -r/2+2, \ldots, r/2-2, r/2\}$, the values $(a_i + r/2)$ as well as $(a_i - r/2)$ are in the range $\{-r, -r+2, \ldots, r-2, r\}$. Further, since the sum $\sum_{i=1}^{l} a_i = s$, if any two a_i and a_j are replaced by either $(a_i + r/2)$ and $(a_j - r/2)$ or by $(a_i - r/2)$ and $(a_j + r/2)$, consequently, the sum $\sum_{i=1}^{l} a_i$ remains s. These two replacements can be performed for any of $\binom{l}{2}$ possible choices of pairs (a_i, a_j) in A and for any of $N(l/2, s/2, r/2)^2$ possible assignments A. The total number of assignments $N(l, s, r)$ is thus at least:

$$N(l, s, r) > 2 \binom{l}{2} N(l/2, s/2, r/2)^2$$

Therefore:

$$\frac{N(l/2, s/2, r/2)^2}{N(l, s, r)} < \frac{1}{l(l-1)} < 2/l, \quad \text{for } l \geq 2 \tag{4}$$

Now we express $\overline{P}(n, i)$ as follows:

$$\begin{aligned}
\overline{P}(n, i) &= 1 - \prod_{j=1}^{i} \left(\frac{2N(2^{n-j}, 2^{n-j}, 2^{n-j})^2}{N(2^{n-j+1}, 2^{n-j+1}, 2^{n-j+1})} \right)^{2^{j-1}} && \text{Lemma 1} \\
&> 1 - \prod_{j=1}^{i} \left(\frac{4}{2^{n-j+1}} \right)^{2^{j-1}} && (4), \text{for } n \geq j \text{ (i.e. always hold)}) \\
&> 1 - \frac{1}{2^{h(n,i)}} && \text{where } h(n, i) := \prod_{j=1}^{i} 2^{j-1}(n-j-1)
\end{aligned}$$

□