# Resource Bounded Frequency Computations with Three Errors

Ulrich Hertrampf[1] and Christoph Minnameier[2]

[1] Abt. Theor. Informatik, University of Stuttgart, D-70569 Stuttgart, Germany
Hertrampf@informatik.uni-stuttgart.de
[2] Lst. Prakt. Informatik II, University of Mannheim, D-68131 Mannheim, Germany
cmm@informatik.uni-mannheim.de

**Abstract.** We deal with frequency computations in polynomial time, or more generally with resource bounded frequency computations. We investigate the first non-trivial case of the Hinrichs-Wechsung conjecture, which states that as soon as we have at least $2^d + d$ inputs to be queried, it does not become harder to get an answer with at most $d$ errors, if we increase the number of inputs to be queried. This conjecture can easily be seen to hold for cases $d < 3$, and it seems very hard to prove in general. We solve the problem affirmatively in the case $d = 3$ by a combination of theoretical reasoning with a highly optimized computer search.

## 1 Introduction

The concept of frequency computation goes back to G. F. Rose [14]. The idea was as follows: If a function $f$ is not computable in the usual sense it may still be possible to compute it in the following relaxed way: On $n$ (pairwise different) inputs $x_1, \ldots, x_n$, a sequence of outputs $y_1, \ldots, y_n$ shall be produced, which approximates the function in such a way that at least $m$ of the output values are correct, i.e. $\|\{i \in \{1, \ldots, n\} \mid y_i = f(x_i)\}\| \geq m$. The class of functions computable in this way is denoted by $(m, n)$. The same idea can also be applied to sets instead of functions.

Essentially there are three versions of the frequency computation model: For the recursion theoretic setting, see e.g. [4],[9],[12],[16],[3]. The resource bounded case was investigated in [9],[5], and for the finite state model setting, see [8],[2],[1]. A lot of related work can be found in the literature (e.g. [6],[7],[11],[10],[13],[15]).

As in many other areas it turned out that the recursion theoretic world much resembles the finite state machine world, whereas the resource bounded (for convenience we will in this paper always speak of polynomially time bounded) world looks completely different.

In all models it is clear that $(m + 1, n + 1) \subseteq (m, n)$, because given an $(m + 1, n + 1)$-algorithm one can obtain an $(m, n)$-algorithm as follows: On input $x_1, \ldots, x_n$ choose any element not equal to any of the $x_i$ and call it $x_{n+1}$. Now query $x_1, \ldots, x_{n+1}$ to the given $(m+1, n+1)$-algorithm and ignore its $(n+1)$-th output. Of the remaining $n$ outputs no more than $(n + 1) - (m + 1) = n - m$ can be erroneous.

X. Hu and J. Wang (Eds.): COCOON 2008, LNCS 5092, pp. 72–81, 2008.

In [5] it was shown that in the resource bounded case for all $m < 2^d$ (where $d = n-m$), the class $(m+1, n+1)$ is a proper subset of $(m, n)$, thus especially for polynomial time $(m+1, n+1)\mathrm{P} \subsetneq (m, n)\mathrm{P}$. Furthermore Hinrichs and Wechsung conjectured that to the contrary, one always obtains equality for $m \geq 2^d$. In this paper we will call that conjecture *the Hinrichs-Wechsung conjecture.*

As a hint on the general computation power of frequency computations we should remark here, that even in the finite state machine setting the classes $(m, n)$ with $2m \leq n$ contain non-countably many languages, i.e. especially they contain non-r.e. sets.

For $d < 3$ the Hinrichs-Wechsung conjecture can easily be proven (see the full paper for a proof), or it can be deduced from [9]:

**Proposition 1.** *Let $m > 2$. Then $(m, m+1)P = (2, 3)P$. Let $m > 4$. Then $(m, m+2)P = (4, 6)P$.*

However, for $d = 3$, the claim would be: Let $m > 8$. Then $(m, m+3)\mathrm{P} = (8, 11)\mathrm{P}$. No similar proof for this case is known. To provide a proof is the subject of this paper. We reach this goal using computer-aided search, where the search space is drastically reduced using theoretical arguments.

## 2 The Problem and Some Preliminaries

**Definition 1.** *The class $(m, n)P$ consists of all languages $L$, such that a polynomial time algorithm exists, which on every set of $n$ different inputs produces an $n$-bit vector, one output bit for each of the $n$ inputs, such that these outputs coincide with the characteristic function of $L$ on at least $m$ of the inputs.*

The problem we want to address is the following: Let $m \geq 9$. Let $L$ be a language from $(m, m+3)\mathrm{P}$, witnessed by the algorithm $A$. Assume we have a set of $m+4$ different inputs $x_1, \ldots, x_{m+4}$, and assume we have obtained $m+4$ results by querying every combination of $m+3$ of these inputs to $A$. We want to show that, no matter what these results look like, we can find an $(m+4)$-bit vector, which also makes at most 3 errors with respect to the characteristic function of $L$. Since our result will be obtained from the $m+4$ result vectors of the queries without further access to the inputs themselves, we can view the whole procedure as a polynomial time algorithm itself, thus proving that $(m, m+3)\mathrm{P} \subseteq (m+1, m+4)\mathrm{P}$. In other words, this will prove the Hinrichs-Wechsung conjecture for the case $d = 3$.

**Remark 1:** In fact this only proves $(9, 12)\mathrm{P} = (10, 13)\mathrm{P} = \ldots$ (due to the condition $m \geq 9$ above), whereas the conjecture starts with $(8, 11)\mathrm{P} = (9, 12)\mathrm{P}$. However, starting with the class $(9, 12)\mathrm{P}$ is necessary to obtain the extendability of the investigation to higher values (c.f. the discussion in the proof). But, the case $(8, 11)\mathrm{P} = (9, 12)\mathrm{P}$ can be obtained by a simple adaptation of our proof, so we will state this case as a corollary.

**Remark 2:** Of course, it is well known that for any fixed value of $m$, the question whether $(m, m+3)\mathrm{P} = (m+1, m+4)\mathrm{P}$ can be considered to be a finite problem

(by the characterization of Kummer and Stephan [9]). But, first of all the instance size already for $m = 9$ is rather big, and it is not at all clear how to investigate this case systematically. Moreover, as soon as we speak of variable $m$, we would have to consider infinitely many cases, and this considerably increases the difficulty in finding a solution.

The objects we deal with will be $13 \times 13$-matrices with entries in $\{0, 1, *\}$, asterisks being exactly in those positions, where row index plus column index equals 14, i.e. in the diagonal from bottom left to top right. All other entries will be 0 or 1. The rows of these matrices shall represent the results of the queries on 12 of the 13 inputs, in the $i$-th column the resulting bit for input $x_i$. The asterisk indicates that we have no answer for that input. This means, the first row represents the result on $x_1, \ldots, x_{12}$, the second row the results on $x_1, \ldots, x_{11}, x_{13}$, and so on.

Our proof is structured as follows: We present an algorithm, which checks that for all matrices of the described type, one of the following two properties will hold:

1. The matrix is not consistent, meaning: There is no possibility to obtain the answers from the matrix for any true characteristic sequence without making more than three errors in one row.

   Take e.g. a matrix with first row `(0,0,0,0,0,0,0,0,0,0,0,0,*)` and second row `(1,1,1,1,1,1,1,0,0,0,0,*,0)` . No matter what the real characteristic sequence should be, one of the two rows has to make at least 4 errors on the first 7 inputs, because the rows differ on all seven.

   Of course, these matrices will never appear in a computation of the kind we are interested in, as long as the underlying algorithm is a correct $(9, 12)$P-algorithm for the language $L$ to be decided.
2. We will be able to produce a 13-bit output, which coincides with the real characteristic vector on the 13 inputs in at least 10 components. Such an output will in the sequel always be called a *solution*.

In fact we will not try to enumerate all such matrices and investigate them – this would last by far too long. Instead, we consider partial matrices consisting of a few rows (always less than 13) and check, whether we can already show that one of the cases given will be true without knowing the other rows of the matrix. Because of this, the second possibility (proving the ability to produce a solution) will be split in two cases: Either we can explicitly give the solution already, or we can prove for a given $i$, that $x_i$ is definitely in (or definitely not in, resp.) the language $L$. Then, using at most one more row of the given matrix (namely row $14 - i$), we can obtain a solution.

Moreover our case inspection will take advantage of occuring symmetries: Rearranging the input vector by application of a permutation results in (simultaneous) permutations of rows and columns, without changing the properties investigated: An inconsistent matrix will still be inconsistent after such a permutation, and a solution will be transformed to a solution of the new matrix, if we apply the same permutation to its bits. The second symmetry operation will be what we call bit-flipping: If we change all bits of a given column of our

matrix (thus mapping 0 to 1 and 1 to 0, but leaving an asterisk unchanged), we also can see that the properties of inconsistency or having a solution remain the same (only in a solution, the according bit has to be flipped too). We will give details in Section 4.

Finally, we will show (inductively) that, whenever we know that $(9+k, 12+k)\mathrm{P} = (10+k, 13+k)\mathrm{P}$, then the same proof can be used to show that also $(10+k, 13+k)\mathrm{P} = (11+k, 14+k)\mathrm{P}$, which will complete the proof of our general result.

## 3 The Maxdist Technique

Dealing with the Hinrichs-Wechsung conjecture for quite some time, we found that a combination of theoretical arguments and machine power has to be used to make considerable progress in the direction of an affirmative solution. This means, we have to find a method of dividing the problem in subproblems, which may be easy enough to solve. Our approach does that by partitioning the set of all matrices to be considered according to a parameter called *maxdist*, the maximum distance between any two rows of the matrix. More formally:

**Definition 2.** *The* distance *of two (equal-sized) vectors over* $\{0, 1, *\}$ *is the number of places, where one vector carries value 0 and the other carries value 1.*

**Remark 3:** The distance between two rows of our matrices will not be changed, if a permutation as described above is applied. The same holds for bit-flipping, because we always flip all bits of a given column.

**Definition 3.** *The value* maxdist *for a* $13 \times 13$*-matrix of the kind considered is the maximum distance that appears between any two rows of the matrix.*

**Lemma 1.** *If a* $13 \times 13$*-matrix of the considered kind has a maxdist value of* 0 *or a maxdist value greater than* 4*, it always satisfies one of our desired properties (being not consistent or allowing a solution).*

**Proof:** We first look at maxdist at least 7. Then there are two rows in our matrix, which on 7 indices $i$ give different answers to the question "$x_i \in L$ ?". As in a consistent matrix both should err on at most 3 inputs, this is a contradiction. Thus the matrix has to be inconsistent.

Now, look at maxdist either 5 or 6. We pick two rows with distance maxdist, say the $i$-th and the $j$-th row. Perform a permutation that maps $i$ to 1 and $j$ to 2 and the columns in such a way that the differences between these two rows appear in the first 5 (or 6) columns. Now, the first row is $(a_1, \ldots, a_{12}, *)$, the second is $(b_1, \ldots, b_{11}, *, b_{13})$, and $a_i = b_i$ for $i \in \{7, \ldots, 11\}$, while $a_i \neq b_i$ for $i \in \{1, \ldots, 5\}$. Then, on the first 5 inputs, both rows together make exactly 5 errors with respect to the real characteristic sequence on $x_1, \ldots, x_5$. It follows that all of $a_7, \ldots, a_{11}$ coincide with the characteristic sequence of $x_7, \ldots, x_{11}$, because otherwise the two given rows would make at least 7 errors in the sum, contradicting the assumption that each makes at most 3 errors. So we can choose

for example row 3, where all but $x_{11}$ are queried, to obtain answers for the other 12 inputs, of which at most 3 are erroneous. With the answer $a_{11} = b_{11}$ for $x_{11}$ we have the desired solution.

Finally, let the matrix have maxdist value 0. Then all rows giving answer for $x_i$ agree on that input (otherwise their distance would be greater than 0). We call the according answer $a_i$. If more than three of the $a_i$s would be wrong, we could choose a row that answers at least four of them. Then, this row would make 4 errors, in contradiction to our assumptions. Thus the vector $a_1, \ldots, a_{13}$ is a solution. □

The case of maxdist 1 is solved in the same way as the cases 2, 3, and 4. However, the number of cases to be considered is rather small. Thus, it is a good opportunity to get some insight, how our computer program has to work. That is why we now show, how to perform this case:

**Lemma 2.** *If a* $13 \times 13$*-matrix of the considered kind has a maxdist value of* 1*, then it always satisfies one of our desired properties (being not consistent or allowing a solution).*

**Proof:** First observe that our matrix has the following property: all pairs of rows have either distance 0 or distance 1.

By a suitable rearrangement (applying a permutation), we make sure that the first two rows have distance 1, and that the difference occurs in the first component. Now, by bit-flipping we change the matrix in such a way that the first two rows get the following form:

```
(0,0,0,0,0,0,0,0,0,0,0,0,*)
(1,0,0,0,0,0,0,0,0,0,0,*,0)
```

We call this partial matrix 1-2-1, which means: maxdist is 1, depth (number of rows) is 2, and it is the first (in this case the only) matrix with these values of maxdist and depth, which we have to consider. For the third row we get the following possible values:

```
a) (0,0,0,0,0,0,0,0,0,0,*,0,0)
b) (1,0,0,0,0,0,0,0,0,0,*,0,0)
c) (0,0,0,0,0,0,0,0,0,0,*,1,0)
d) (1,0,0,0,0,0,0,0,0,0,*,0,1)
```

All other possible rows would contradict our assumption of maxdist 1. Now, case b) can be transformed to case a) by permuting the first two rows (and thus also the last two columns) and bit-flipping in the first column. The same holds for case d) to case c). So we only need to consider two cases of partial matrices with 3 rows. Thus we now have two matrices to consider on depth 3, namely

```
(0,0,0,0,0,0,0,0,0,0,0,0,*)
(1,0,0,0,0,0,0,0,0,0,0,*,0)
(0,0,0,0,0,0,0,0,0,0,*,0,0)
```

(denoted as matrix 1-3-1), and

```
(0,0,0,0,0,0,0,0,0,0,0,0,*)
(1,0,0,0,0,0,0,0,0,0,0,*,0)
(0,0,0,0,0,0,0,0,0,0,*,1,0)
```

(denoted as matrix 1-3-2).

We proceed with matrix 1-3-1 and find the following three cases of possible next rows:

a) (0,0,0,0,0,0,0,0,0,∗,0,0,0)
b) (1,0,0,0,0,0,0,0,0,∗,0,0,0)
c) (0,0,0,0,0,0,0,0,0,∗,0,1,0)

All three cases are different and will lead to matrices 1-4-1, 1-4-2, and 1-4-3. But, from matrix 1-3-2 we only have two possible next rows:

a) (0,0,0,0,0,0,0,0,0,∗,0,0,0)
b) (0,0,0,0,0,0,0,0,0,∗,0,1,0)

Here, case a) can be transformed to matrix 1-4-3 (which was case c) above), by swapping rows 3 and 4 (and of course also columns 11 and 10). And also case b) can be transformed to that matrix by a somewhat more complex operation: A circular swap of rows 1, 3, and 4, followed by a bit-flipping in column 12.

Now, we have to consider depth 4. However, as the principle should be clear now, we invite interested readers to perform the rest of the proof themselves.

## 4 Our Algorithm

Now we want to complete the proof of $(9, 12)\mathrm{P} = (10, 13)\mathrm{P}$. Therefor we have to consider the cases of maxdist value 2, 3, or 4. This now is definitely a case for the computer. But, in order to reduce computation time, we want to reduce the problem as far as possible.

In Section 2 we already mentioned that we will use transformations, and we did apply this technique in Section 3 for the easy cases. We justify this by the following discussion.

Let $x_1, \ldots, x_{13}$ be any sequence of (pairwise different) inputs, which are to be checked for membership in $L$, the given language from $(9, 12)\mathrm{P}$. By 13 queries to the given $(9, 12)\mathrm{P}$-algorithm we will obtain 13 times 12 output bits for the possible combinations of 12 out of the 13 inputs. We will know that at least 9 output bits in each answer sequence are correct. (But of course, we do not a priori know which ones!)

Thus the output could look like:

(0,1,∗,0,1,1,0,0,1,1,0,0,1)
(0,0,1,0,1,0,0,0,1,0,0,1,∗)
(0,0,0,0,0,1,∗,0,1,0,0,0,1)
(0,0,1,0,1,1,1,∗,1,1,1,0,1)
(0,0,0,0,0,1,1,1,1,0,0,∗,1)
(0,0,1,0,1,1,1,1,1,0,∗,1,1)
(0,0,1,0,1,0,1,0,1,∗,0,1,1)
(0,0,1,0,0,0,1,0,∗,0,0,1,1)

and so on. Clearly, we can choose to write any row first, then any other row second, and so forth. But it is obvious that there is only one way to arrange the rows, where the asterisks appear in the diagonal from top right to bottom left:

(0,0,1,0,1,0,0,0,1,0,0,1,∗)
(0,0,0,0,0,1,1,1,1,0,0,∗,1)
(0,0,1,0,1,1,1,1,1,0,∗,1,1)

and so on. We always choose the sequence of rows in this way and call that a normalization. Now, whenever we swap two rows, or more generally whenever

we perform a permutation on the rows, we have to swap columns too, in order to keep our normalized form. Note, that swapping of rows $i$ and $j$ has to be followed by swapping of columns $14-i$ and $14-j$ (for $1 \leq i < j \leq 13$), and similarly for general permutations. However, while the ordering of the rows technically means nothing, the ordering of the columns refers to the ordering of the queried inputs $x_1, \ldots, x_{13}$. Thus, one should keep in mind that such an operation always means a rearrangement of the inputs.

**Lemma 3.** *If the normalized matrix $A$ can be transformed into the normalized matrix $B$ by a sequence of row and column permutations, then $B$ is consistent if and only if $A$ is, and $B$ allows a solution, if and only if $A$ does.*

**Proof:** (omitted).

**Corollary 1.** *If the normalized partial matrix $A$ (i.e. an upper part of a normalized complete matrix) can be transformed into the normalized partial matrix $B$ by a sequence of row and column permutations, then $B$ can be extended to a consistent matrix if and only if $A$ can, and $B$ already allows for a solution, if and only if $A$ does.*

We will normalize the matrices further: We want to consider only such matrices, where the first row is `(0,0,0,0,0,0,0,0,0,0,0,0,*)` , and the second row has a 0 in the 13-th place. This can be arranged by bit-flipping of all columns, where the first row had a 1, resp. of column 13, if the second row had a 1 there. This operation may be done because of the following lemma:

**Lemma 4.** *If the normalized matrix $A$ can be transformed to $B$ by bit-flipping on one or more columns, then $B$ is consistent if and only if $A$ is, and $B$ allows a solution, if and only if $A$ does.*

**Proof:** It suffices to prove the claim for bit-flipping on one column. Let column $i$ be flipped. If $a_1, \ldots, a_{13}$ is a possible truth for the original matrix $A$, then the sequence obtained by flipping $a_i$ is a possible truth for the new matrix $B$, and vice versa. Thus, $A$ is consistent if and only if $B$ is. If a solution for $A$ is given, then it can be changed to a solution for $B$ by flipping the $i$-th bit, and also vice versa. Thus, a solution for $A$ exists, if and only if a solution for $B$ exists. □

**Notation:** The relation between (partial) matrices, given in such a way that $A$ is related to $B$ if and only if $A$ can be transformed to $B$ by the above introduced transformation rules, is an equivalence relation. We will often call matrices related in this way to be *symmetric* to each other.

By the above normalization rules we will always be able to transform a given matrix to the following form:

- The asterisk of row $i$ is in column $14-i$.
- The first row consists only of 0 entries (and the asterisk in column 13).
- The second row has a 0 in column 13.
- The distance between rows 1 and 2 is exactly the value of maxdist.

- In each row, the following holds: If columns $j$ and $j+1$ are exactly equal in all rows above the current one, then in the current row we may not have a 0 in column $j$ and a 1 in column $j+1$. (We call that the monotonicity rule.)
- The monotonicity rule, together with the rule about the distance between rows 1 and 2 implies for row 2 that there are 1-entries exactly in columns 1 to maxdist, and all other entries in row 2 are 0 (except for the asterisk in column 12).

The discussion proved that in order to show that all possible (that is consistent) matrices have solutions, it is sufficient to only consider matrices in the described form.

Now we are ready to introduce our algorithm for the cases of maxdist values 2, 3, and 4. A pseudo-code formulation of the algorithm, as well as an exe-file with our implementation (and the source code in ADA) can be found on the web page

`http://134.155.88.3/main/chair_de/03/cmm_download/index_de.html`

Essentially, the algorithm works exactly as the explicit procedure for the case of maxdist value 1 in Lemma 2. For a given value of maxdist (2, 3, or 4) it starts with the first two rows, which are uniquely determined by the above described form and the value of maxdist. We call that stage "depth 2" and we initialize a list of matrices to be considered with that one matrix of two rows. Moreover we compute the set of possible truths, i.e. the set of all 13-bit vectors which have distance at most 3 to both rows of that matrix, and we attach that set to the matrix.

Now for a given depth (starting from depth 2), we take all matrices from the list, compute all candidates for next rows, which have a distance less than or equal to maxdist to all rows already in the matrix. For each of these candidate rows, we check, whether the row would obey the monotonicity rule. If it does, it is output as a new row to be considered.

The algorithm has to cancel out all so far possible truths, which contradict the new row (by having distance greater than 3 to it), and then examine the remaining set of possible truths, in order to find out, whether one of the following cases happens:

(a) The (partial) matrix becomes inconsistent (set of possible truths is empty), or
(b) All possible truths have the same value in one position (this will lead to a solution), or
(c) We can find a 13-bit vector, which has distance less than or equal to 3 for all possible truths, or
(d) None of these cases, but in our list for the next depth, we can already find a matrix which can be obtained from the current one by row permutation and bit-flipping operations, or
(e) None of the other cases, so we have to append the new matrix to the list for the next depth.

The most complex test to be performed here is for case d). We could try to apply all possible transformations in order to get any of the matrices already in the list. However, we do it the other way round: We feed all pairs, built of the current matrix and one matrix of the list, into a symmetry detecting procedure called *the matcher*, which uses additional structural properties of the matrices to speed up detection of nonsymmetry in many cases: The mainly used property is the vector of numbers of pairs of a given distance. See the full paper for details.

Now, if one of cases a), b), c) or d) happens, the algorithm produces an output telling exactly which case happened, and moreover in case b) the special position and the value that all truths have in that position, in case c) the solution, and in case d) the row permutation that leads to the other matrix, and the number of that matrix in our list (note, that the according column permutation and the bit-flipping operations are implicitly given by our normalization conditions). When all partial matrices of a given depth are done, the algorithm starts to examine the list for the next depth. If that is empty, the algorithm terminates.

The algorithm can be performed on maxdist values 2, 3 and 4, and it runs very fast. The output can also be found on the above named web page. By inspection of the output one can see that in all three cases the maximum depth to be considered is depth 8. At the maximum depth the algorithm terminates in each case, meaning there are no more matrices to consider. Thus we obtain:

**Theorem 1.** $(9, 12)P = (10, 13)P$.

One can observe that all (partial) matrices occuring in the execution of the algorithm have at least one column of all zeroes. If we omit one such column in every considered matrix, the whole procedure does exactly the same, only it now works on only 12 inputs. Thus, as a corollary we get:

**Corollary 2.** $(8, 11)P = (9, 12)P$.

## 5 Extendability

As we observed at the end of Section 4, every matrix used in the execution of our algorithm has at least one column of all zeroes. To be more exact: In cases maxdist value 3 or 4, we always have at least one such column, and in case maxdist value 2 (and maxdist value 1 as well, as investigated in Lemma 2), we always have at least two such columns. These zero columns will be the key in the proof, that the result of Theorem 1 can be extended to all cases with more inputs and up to three errors. Together with Corollary 2 this will complete the solution of the $d = 3$ instance of the Hinrichs-Wechsung conjecture. Unfortunately, the rigorous space restrictions in these lecture notes only allow us to state the final result. We refer the reader to the full paper for details.

**Theorem 2.** *For all* $k > 0$*, we have* $(8 + k, 11 + k)P = (8, 11)P$.

## 6 Further Work

One obvious next step would be an adaptation of the proof to case $d = 4$, which might be done by a simple extension of our algorithm. Or, certainly more interesting, a (theoretical) proof should be given for the following conjecture:

**Conjecture:** For any $m < n$ we have:

$$(m, n)\mathrm{P} = (m + 1, n + 1)\mathrm{P} \Longrightarrow (m + 1, n + 1)\mathrm{P} = (m + 2, n + 2)\mathrm{P}$$

## References


1. Austinat, H., Diekert, V., Hertrampf, U.: A Structural Property of Regular Frequency Computations. Theor. Comp. Sc. 292(1), 33–43 (2003)
2. Austinat, H., Diekert, V., Hertrampf, U., Petersen, H.: Regular Frequency Computations. In: RIMS Symposium on Algebraic Systems, Formal Languages and Computation, Kyoto, Japan, pp. 35–42 (2000)
3. Beigel, R., Gasarch, W.I., Kinber, E.B.: Frequency Computation and Bounded Queries. Theor. Comp. Sc. 163(1–2), 177–192 (1996)
4. Degtev, A.N.: On $(m, n)$-Computable Sets. Algebraic Systems, 88–99 (1981) (in Russian)
5. Hinrichs, M., Wechsung, G.: Time Bounded Frequency Computations. Information and Computation 139(2), 234–257 (1997)
6. Kinber, E.B.: Frequency Calculations of General Recursive Predicates and Frequency Enumeration of Sets. Soviet Mathematics Doklady 13, 873–876 (1972)
7. Kinber, E.B.: On Frequency-Enumerable Sets. Algebra i Logika 13, 398–419 (1974) (in Russian); English translation in Algebra and Logic 13, 226–237 (1974)
8. Kinber, E.B.: Frequency Computations in Finite Automata. Kibernetika 2, 7–15 (1976) (in Russian); English translation in Cybernetics 12, 179–187 (1976)
9. Kummer, M., Stephan, F.: The Power of Frequency Computation. In: Reichel, H. (ed.) FCT 1995. LNCS, vol. 969, pp. 323–332. Springer, Heidelberg (1995)
10. Kummer, M., Stephan, F.: Recursion Theoretic Properties of Frequency Computation and Bounded Queries. Information and Computation 120, 59–77 (1995)
11. Kummer, M.: A Proof of Beigel's Cardinality Conjecture. J. of Symbolic Logic 57(2), 677–681 (1992)
12. McNaughton, R.: The Theory of Automata, a Survey. Advances in Computers 2, 379–421 (1961)
13. McNicholl, T.: The Inclusion Problem for Generalized Frequency Classes. PhD thesis, George Washington University, Washington (1995)
14. Rose, G.F.: An Extended Notion of Computability, Abstracts Int. Congress for Logic, Methodology, and Philosophy of Science. Stanford, California, p. 14 (1960)
15. Tantau, T.: Towards a Cardinality Theorem for Finite Automata. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 625–636. Springer, Heidelberg (2002)
16. Trakhtenbrot, B.A.: On the Frequency Computation of Functions. Algebra i Logika 2, 25–32 (1963) (in Russian)