Lower Bound for Evaluation of $\mu\nu$ Fixpoint

Paweł Parys*
Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw
Banacha 2, PL-02-097 Warszawa, Poland
parys@mimuw.edu.pl

Abstract

We consider a fixpoint expressions $\mu y.\nu x.f(x,y)$ over the lattice $\{0,1\}^n$, where $f\colon \{0,1\}^{2n} \to \{0,1\}^n$ is any monotone function. We study only algorithms for calculating these expressions using f only as a black-box: they may only ask for the value of f for given arguments. We show that any such algorithm has to do at least about n^2 queries to the function f, namely $\Omega\left(\frac{n^2}{\log n}\right)$ queries.

1 Introduction

Fast evaluation of fixpoint expressions is a key problem in the fixpoint theory. We consider a special form of expressions:

$$\mu x_d.\nu x_{d-1}\ldots\mu x_2.\nu x_1.f(x_1,\ldots,x_d)$$

(when d is even, and starting from νx_d when d is odd). We call such expression $\mu\nu(d,f)$. Moreover we consider these expressions only over the lattice $L=\{0,1\}^n$ with the order defined by $a_1\ldots a_n \le b_1\ldots b_n$ when $a_i \le b_i$ for all i. The function f is an arbitrary monotone function $f\colon L^d \to L$. Calculating the value of $\mu\nu(d,f)$ is already a very general problem. The problem of finding winning positions in a parity game may be reduced to it in polynomial time (where n corresponds to the game graph size and d to the number of priorities). The problem of solving parity games is polynomial time equivalent to the non-emptiness problem of automata on infinite trees with the parity acceptance conditions [4], and to the model checking problem of the modal μ-calculus (modal fixpoint logic) [3, 6].

Although these are very important problems and many people were working on them, no one could show any polynomial time algorithm. Our goal is the opposite—to prove some lower bound. It may be very difficult to show any algorithmic lower bound, especially because it is known that the problems are in NP∩co-NP. In such situation the only possibility is to reformulate the problem slightly, so that it becomes combinatorial. To achieve that we use a black-box model (or an oracle model) introduced in [1]. Instead of arbitrary algorithms, which could analyze for example a formula defining f, we consider only algorithms, that can only ask for values of f for given arguments. Moreover we are not interested in their exact complexity, only in the number of queries to the function f. In other words we consider decision trees: each internal node of the tree is labeled by an argument, for which the function f should be checked, and each its child corresponds to a possible value of f for that argument. The tree has to determine the value of the fixpoint expression $\mu\nu(d,f)$: for each path from the root to a leaf there is at most one possible value of $\mu\nu(d,f)$ for all functions which are consistent with the answers on that path. We are interested in the height of such trees, which justifies the following definition.

Definition 1. For any natural number d and finite lattice L we define $num(d,L)$ as the minimal number of queries, which has to be asked by any algorithm correctly calculating expression $\mu\nu(d,f)$ basing only on queries to the function $f\colon L^d \to L$.

*Partly supported by Polish government grant N206 008 32/0810.

The most basic method of evaluating fixpoint expression is to use the observation that $\mu x.g(x) = g^n(\bot)$; so it is enough to evaluate n times g on the previous result, starting from the minimal element $\bot$. To get ν instead of μ, one should start from $\top$ instead of $\bot$. This generalizes to d nested fixpoints $\mu\nu(d,f)$ and requires $O(n^d)$ queries to f; see [5]. For some time no better algorithm was known. Then an algorithm using only $O(n^{\lfloor d/2\rfloor+1})$ queries to f was shown in [8] and [1], which was rather a surprise. Recently some better algorithms for the modal μ-calculus and parity games were discovered, like [9] working in time $O(n^{d/3})$ or [7] working in time $n^{O(\sqrt{n})}$. However these two algorithms use parity games framework and do not translate to the black-box model. Here we see one of the limitations of our model: there may exist fast algorithm, which uses a definition of f in some tricky manner, but is unable to work when it can only evaluate f. The other limitation is that the number of monotone functions definable by a short formula is only single exponential, while the number of all monotone functions $f\colon \{0,1\}^{nd} \to \{0,1\}^n$ is double exponential. When we restrict only to functions definable by a short formula it is possible that less queries would be needed.[1] Beside of that, the following are very important questions (following [1]): how good may we do using f only as a black-box? Is the complexity of about $n^{d/2}$ queries optimal? What is the optimal number of queries? If the answer will be rather high, we will know that any fast algorithm for parity games and modal μ-calculus has to use different techniques. If the answer will be rather low, it may also give some fast algorithm. We write „may" because there is no implication in a formal sense: the decision tree with small number of queries may be very irregular and it may take a lot of time to compute what the next query should be.

In this paper we consider only the case $d = 2$. We show that $\Omega\left(\frac{n^2}{\log n}\right)$ queries are necessary in that case (which is almost n^2). Our result is the following.

Theorem 2. *For any natural n it holds $num(2,\{0,1\}^n) = \Omega\left(\frac{n^2}{\log n}\right)$.*

This result is a first step towards solving the general question, for any d. It shows that in the black-box model something may be proved. Earlier it was unknown even if for any d there are needed more than nd queries. Note that $num(1,\{0,1\}^n)$ is n and that in the case when all d fixpoint operators are μ (instead of alternating μ and ν) it is enough to do n queries. So the result gives an example of a situation where the alternation of fixpoint quantifiers μ and ν is provably more difficult than just one type of quantifiers μ or ν. Although it is widely believed that the alternation should be a source of algorithmic complexity, the author is not aware of any other result showing this phenomenon, except the result in [2].

The paper is organized as follows. In Section 2 we reduce the problem from the lattice $\{0,1\}^n$ to some more convenient lattice. In Section 3 we define a family of difficult functions f. In Section 4 we finish the proof of Theorem 2.

Acknowledgment. The author would like to thank Igor Walukiewicz for suggesting this topic and many useful comments.

2 Changing the Lattice

Instead of the lattice $\{0,1\}^n$ it is convenient to use a better one. Take the alphabet Γ_n consisting of letters a_i for $1 \le i \le \frac{n(n+1)}{2}+1$ and the alphabet $\Sigma_n = \{0,1\} \cup \Gamma_n$. We introduce the following partial order on it: the letters a_i are incomparable; the letter 0 is smaller than all other letters; the letter 1 is bigger than all other letters. We will be considering sequences of n such letters, i.e. the lattice is Σ_n^n. The order on the sequences is defined as previously: $a_1 \dots a_n \le b_1 \dots b_n$ when $a_i \le b_i$ for all i.

[1] This is not the case for $d = 2$; functions used in our lower bound proof are all definable by a boolean formula of size polynomial in n.

We formulate a general lemma, which allows to change a lattice in our problem. For any two lattices L_1, L_2 we say that $h\colon L_1 \to L_2$ is a homomorphism, when it preserves the order, i.e. $x \leq y$ implies $h(x) \leq h(y)$.

Lemma 3. *Let L_1, L_2 be two finite lattices and $enc\colon L_1 \to L_2$ and $dec\colon L_2 \to L_1$ two homomorphisms such that $dec \circ enc = id_{L_1}$. Then $num(d, L_1) \leq num(d, L_2)$.*

Proof
In other words we should be able to use any algorithm calculating $\mu\nu(d, f)$ in L_2 to calculate $\mu\nu(d, f)$ in L_1. Let $f_1\colon L_1^d \to L_1$ be the unknown function in L_1. We define $f_2\colon L_2^d \to L_2$ as $f_2(x_1, \ldots, x_d) = enc(f_1(dec(x_1), \ldots, dec(x_d)))$. Note that f_2 is a monotone function if f_1 was monotone, since *enc* and *dec* preserve the order.

Let $\bot_1, \bot_2, \top_1, \top_2$ be the minimal and maximal elements in L_1 and L_2. For any $x \in L_1$ we have $\bot_2 \leq enc(x)$, so $dec(\bot_2) \leq dec(enc(x)) = x$, which means that $dec(\bot_2) = \bot_1$. Similarly $dec(\top_2) = \top_1$.

See that $dec(\mu\nu(d, f_2)) = \mu\nu(d, f_1)$. This is true, because these fixpoint expressions may be replaced by a term containing applications of f and minimal and maximal elements. This is done in a classic way, we replace the fixpoint operators by a iterated nesting. The minimal required number of iterations depends on the structure. Here we have only two structures, L_1 and L_2, so we may take the bigger of the two minimal numbers. Hence we may use the same term in L_1 and L_2, the difference is if we use f_1 or f_2, $\bot_1$ or $\bot_2$, $\top_1$ or $\top_2$. Then easy induction on the term structure shows that $dec(\mu\nu(d, f_2)) = \mu\nu(d, f_1)$, because $dec(\bot_2) = \bot_1$, $dec(\top_2) = \top_1$, $dec(f_2(x_1, \ldots, x_d)) = f_1(dec(x_1), \ldots, dec(x_d))$. So to find $\mu\nu(d, f_1)$ it is enough to find $\mu\nu(d, f_2)$, which may be found for any f_2 in $num(d, L_2)$ queries to f_2. To evaluate f_2 in our case it is enough to do one query to f_1. Hence $\mu\nu(d, f_1)$ may be found in $num(d, L_2)$ queries (or maybe less queries in some other way). □

For the lattice Σ_n^n we have the following result, from which Theorem 2 follows:

Lemma 4. *For any natural n it holds $num(2, \Sigma_n^n) \geq \frac{n(n+1)}{2}$.*

Proof (Theorem 2)
We will show how Theorem 2 follows from this lemma. Take k such that $\binom{2k}{k} \geq \frac{n(n+1)}{2} + 1$. From the Stirling formula follows that $\binom{2k}{k}$ grows exponentially in k, so we may have $k = O(\log n)$. Take $m = \left\lfloor \frac{n}{2k} \right\rfloor$. From Lemma 4 for m we see that $num(2, \Sigma_m^m) \geq \frac{m(m+1)}{2} = \Omega\left(\frac{n^2}{\log n}\right)$.

Now it is enough to use Lemma 3 to see that $num(2, \{0,1\}^n) \geq num(2, \Sigma_m^m)$. We need to define functions $enc\colon \Sigma_m^m \to \{0,1\}^n$ and $dec\colon \{0,1\}^n \to \Sigma_m^m$. Each letter from Σ_m will be encoded in a sequence of $2k$ letters from $\{0,1\}$ in the following way: 0 is translated to the sequence of $2k$ zeroes, 1 to the sequence of $2k$ ones, any of the letters a_i is translated to some sequence of $2k$ bits, in which exactly k bits are equal to 1. Because $n \geq m$ we have $\binom{2k}{k} \geq \frac{m(m+1)}{2} + 1$, so there are enough different such sequences to encode all letters. We use this encoding to define $enc(x)$: an i-th letter of x is encoded in the i-th fragment of $2k$ bits and the final $n - 2km$ bits are set to zeroes. On the other hand to read an i-th letter of the value of $dec(y)$, we look at the i-th fragment of $2k$ bits: when it corresponds to one of the letters a_i, this a_i is the result; otherwise the result is 0 or 1 depending on whether there are less than k ones in the sequence or not. Note that *dec* is defined on all sequences, not only on results of *enc*. It is easy to see that $dec(enc(x)) = x$ for any $x \in \Sigma_m^m$ and that both functions are homomorphisms (mainly because encodings of different letters a_i are incomparable). □

3 Difficult Functions

In this section we define a family of functions used in a proof of Lemma 4. A function $f_{z,\sigma}\colon \Sigma_n^{2n} \to \Sigma_n^n$ is parametrized by a sequence $z \in \Gamma_n^n$ (which will be the result of $\mu y.\nu x.f_{z,\sigma}(x,y)$) and by a permutation $\sigma\colon \{1,\dots,n\} \to \{1,\dots,n\}$ (which is an order in which the letters of z are uncovered). Note that z is from Γ_n^n, not from Σ_n^n, so it can not contain 0 or 1, just the letters a_i. Whenever z and σ are clear from the context, we simply write f. In the following the i-th element of a sequence $x \in \Sigma_n^n$ is denoted by $x[i]$. A pair z,σ defines a sequence of values $y_0,\dots,y_n$:

$$y_k[i] = \begin{cases} z[i] & \text{for } \sigma^{-1}(i) \le k \\ 0 & \text{otherwise.} \end{cases}$$

In other words y_k is equal to z, but with some letters covered: they are 0 instead of the actual letter of z. In y_k there are k uncovered letters; the permutation σ defines the order, in which the letters are uncovered. Using this sequence of values we define the function. In some sense the values of the function are meaningful only for $y = y_k$, we define them first (assuming $y_{n+1} = y_n$):

$$f(x,y_k)[i] = \begin{cases} 0 & \text{if } \forall_{j>i} x[j] \le y_{k+1}[j] \text{ and } x[i] \not\ge y_{k+1}[i] \quad \text{(case 1)} \\ y_{k+1}[i] & \text{if } \forall_{j>i} x[j] \le y_{k+1}[j] \text{ and } x[i] \ge y_{k+1}[i] \quad \text{(case 2)} \\ x[i] & \text{if } \exists_{j>i} x[j] \not\le y_{k+1}[j] \quad \text{(case 3).} \end{cases}$$

For any other node y we look for the lowest possible k such that $y \le y_k$ and we put $f(x,y) = f(x,y_k)$. When such k does not exists ($y \not\le z$), we put $f(x,y)[i] = 1$.

Lemma 5. *The function f is monotone and $\mu y.\nu x.f(x,y) = z$.*

Proof
First see what happens when we increase x: take $x' \ge x$. We want to have $f(x',y)[i] \ge f(x,y)[i]$ for each i. Whenever for x and x' we are in the same case of the function definition, it is OK. Also when for x we have an earlier case than for x' it is OK (in particular when for x we have case 2, it holds $x'[i] \ge x[i] \ge y_{k+1}[i]$). On the other hand it is impossible, that for x' we get an earlier case than for x (it is easy to see looking at the conditions for choosing a case). Also when $y \not\le z$, for both x and x' we get the same result 1.

Now see what happens, when we increase y: take $y' \ge y$. When for y' there is $y' \not\le z$, we get a result 1, which is bigger than anything else. Otherwise the values y_k and $y_{k'}$ chosen for y and y' satisfy $y_{k'} \ge y_k$, so also $y_{k'+1} \ge y_{k+1}$. The argumentation that in such case $f(x,y_{k'})[i] \ge f(x,y_k)[i]$ is identical as for the change of x.

To calculate the fixpoint expression, first see that $\nu x.f(x,y_k) = y_{k+1}$. It follows immediately from the definition: $f(y_{k+1},y_k) = y_{k+1}$ and for any $x > y_{k+1}$ we get $f(x,y_k) \ne x$, because $f(x,y_k)$ differs from x on the last position i where $x[i] > y_{k+1}[i]$, we get there $y_{k+1}[i]$ instead of $x[i]$. The main fixpoint satisfies $\mu y.\nu x.f(x,y) = y_n = z$, because $y_{k+1} > y_k$ for all $k < n$ and $y_{n+1} = y_n$. □

4 The Proof

Now we will show that at least $\frac{n(n+1)}{2}$ queries are needed to calculate $\mu y.\nu x.f(x,y)$, even if we allow as f only functions from our family. The problem can be considered as a game between two players, we call them an algorithm and an oracle. In each round the algorithm player asks a query to the function, after what the oracle player chooses an answer (which is consistent with the previous answers). The algorithm player wins if after $\frac{n(n+1)}{2} - 1$ steps each function consistent with the answers has the same

value of $\mu y.\nu x.f(x,y)$. Otherwise the oracle player wins. We have to show a winning strategy for the oracle player.

First see informally what may happen. Consider first a standard algorithm evaluating fixpoint expressions. It starts from $y = y_0 = 0\ldots0$ and $x = 1\ldots1$. Then it repeats $x := f(x,y)$ until x stops changing, in which case $x = \nu x.f(x,y)$. For our functions it means that in each step the last 1 in x is replaced by the corresponding letter of y_1. The loop ends after n steps with $x = y_1$. Then the algorithm does $y := x,\ x := 1\ldots1$, and repeats the above until y stops changing. For any $y = y_k$ the situation is very similar: in each step the last 1 in x is replaced by the corresponding letter of y_{k+1} (we may say that this letter is uncovered).

In fact, by choosing appropriate x the algorithm may decide which letter of y_{k+1} he wants to uncover, but always at most one. For the algorithm only the letter on which y_{k+1} differs from y_k is important, as he already knows all letters of y_k. However the difference may be on any position on which y_k has 0 (it depends on σ). The oracle player may choose this position in the most malicious way: whenever the algorithm player uncovers some letter, the oracle decides that this is not the letter on which y_k and y_{k+1} differs. So the algorithm has to try all possibilities (all positions on which y_k has 0), which takes $\frac{n(n+1)}{2}$ steps. He may also ask for some other y. It can give him any profit only if he accidentally guesses some letters of z. However the oracle may always decide that the guess of the algorithm is incorrect (that the value of z is different).

Now come to a more formal proof. We show a strategy for the oracle player. During the game we (the oracle player) keep a variable cur ($0 \leq cur < n$), which is equal to 0 at the beginning and is increased during the game. Intuitively it means how many letters of z are already known to the algorithm player. By s we denote the number of queries already asked (it increases by 1 after each query) and by s_{lok} the number of queries asked for this value of cur (it increases by 1 after each query and is reset to 0 when cur changes).

At every moment we keep a set F of functions consistent with all the answers till now (there may be more consistent functions, but each function in our set has to be consistent). The set will be described by a set of permutations Π and by sets of allowed values $A_i \subseteq \Gamma_n$, one for each coordinate $1 \leq i \leq n$. The sets should satisfy the following conditions:

1. for each $i \leq cur$ there is only one value of $\sigma(i)$ for $\sigma \in \Pi$;
2. in Π there are permutations σ with at least $n - cur - s_{lok}$ different values of $\sigma(cur+1)$;
3. for each permutation $\sigma \in \Pi$ when we take any other permutation σ' which agrees with σ on the first $cur+1$ arguments ($\sigma(i) = \sigma'(i)$ for each $1 \leq i \leq cur+1$), we have $\sigma' \in \Pi$ as well;
4. for each $\sigma \in \Pi$ and $i \leq cur$ there is only one value in $A_{\sigma(i)}$ (note that thanks to condition 1, the value $\sigma(i)$ does not depend on the choice of σ);
5. for each $\sigma \in \Pi$ and $i > cur$ there are at least $\frac{n(n+1)}{2} + 1 - s$ values in the set $A_{\sigma(i)}$ (note that the set $\{\sigma(i) : i > cur\}$ does not depend on the choice of σ, as $\sigma(i)$ for $i \leq cur$ are fixed).

In the set F there are all functions $f_{z,\sigma}$ for which $\sigma \in \Pi$ and $z[i] \in A_i$ for each i. We see that in particular at the beginning all functions are in the set F. Note, that at each moment the value of y_{cur} is fixed, i.e. is the same for all functions in F (because $\sigma(i)$ and $z[\sigma(i)]$ are fixed for $i \leq cur$).

Now we specify how the answers are done for a query x,y. Whenever $y \leq y_i$ for some $i < cur$, we answer according to all the functions in our set F. The answer of each function is the same, as it depends only on the value of y_{i+1} (for the smallest i such that $y \leq y_i$), which is already the same for all functions. Such question does not give any new knowledge to the algorithm player.

Whenever $y \not\leq y_{cur}$, we remove the value $y[i]$ from the set A_i (only if it was there, in particular only if $y[i] \in \Gamma_n$) for each i such that $\sigma^{-1}(i) > cur$ for any $\sigma \in \Pi$ (note that once again this condition is satisfied for exactly the same i for every permutation in Π). All the conditions of F are still satisfied, as we removed only one value from the sets A_i after one additional query was done. In other words we remove all functions $f_{z,\sigma}$, in which $z[\sigma(i)] = y[\sigma(i)]$ for some $i > cur$. Then for each function from F we have $y \not\leq z$ (if $y \leq z$ then $y[i] = 0$ for each i with $\sigma^{-1}(i) > cur$, which means that $y \leq y_{cur}$). So we reply to the query by a sequence of ones, which is the case for all the functions in F. Intuitively this case talks about a situation when someone tries to guess z (or its part) instead of gently asking for $y = y_{cur}$. We prefer to answer that his guess was incorrect and to eliminate all functions with z similar to the y about which he asked.

Consider now the case when $y \leq y_{cur}$ but $y \not\leq y_{cur-1}$. Let ask be the greatest number such that $x[ask] \not\leq y_{cur}[ask]$ (if there is no such number we take $ask = 0$). Intuitively the algorithm player asks whether $\sigma(cur+1) = ask$; we prefer to answer NO, so he will have to try all the possibilities until he will discover the value of $\sigma(cur+1)$. The first case is when in Π there are permutations with $\sigma(cur+1) \neq ask$. Note that this is true at least $n - cur - 1$ times for this cur due to condition 2. In such case we remove from Π all the permutations with $\sigma(cur+1) = ask$ and we answer according to all the functions left in F. We have to argue that for each of them the answer is the same. On positions $i < ask$ there is always case 3, because $x[ask] \not\leq y_{cur}[ask] = y_{cur+1}[ask]$ (the equality is true, because $\sigma(cur+1) \neq ask$). On the positions $i \geq ask$ there is $x[j] \leq y_{cur}[j] \leq y_{cur+1}[j]$ for $j > i$, so we fall into the first two cases. For $i = ask$ the result depends only on $y_{cur+1}[ask]$ which for all functions is equal to $y_{cur}[ask]$. Consider positions $i > ask$. When $x[i] = 0$ the answer is 0 in both cases 1 and 2 (we may get case 2 only when $y_{cur+1}[i] = 0$). When $x[i] > 0$ it has to be $y_{cur+1}[i] = x[i]$, as $x[i] \leq y_{cur}[i] \leq y_{cur+1}[i] \neq 1$, we get case 2 and we answer $y_{cur+1}[i] = x[i]$.

The last case is when all the permutations $\sigma \in \Pi$ have $\sigma(cur+1) = ask$. Then we choose any letter from A_{ask} and we remove all other letters from A_{ask}. In other words y_{cur+1} becomes fixed, so answers for all the functions left in F are the same. We increase cur (when cur becomes equal to n, we fail). It is easy to see, that all the conditions on the set F are still satisfied.

As already mentioned, before the last case holds there has to be $n - cur - 1$ earlier queries for this cur (we increase cur after at least $n - cur$ queries), so before $\frac{n(n+1)}{2}$ queries there is no danger that cur becomes equal to n. Moreover we are sure that in F there are functions with two different value of z (which is the result of the fixpoint expression): it is enough to take any σ from Π and then in $A_{\sigma(n)}$ there are at least two values ($z[\sigma(n)]$ may be equal to both of them).

References

[1] A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theor. Comput. Sci.*, 178(1–2):237–255, 1997.

[2] A. Dawar and S. Kreutzer. Generalising automaticity to modal properties of finite structures. *Theor. Comput. Sci.*, 379(1–2):266–285, 2007.

[3] E. A. Emerson. Model checking and the mu-calculus. In N. Immerman and P. G. Kolaitis, eds., *Proc. of DIMACS Wksh. on Descriptive Complexity and Finite Models (Princeton Univ., Jan. 1996)*, v. 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 185–214. AMS, 1997.

[4] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In C. Courcoubetis, ed., *Proc. of 5th Int. Conf. on Computer-Aided Verification, CAV '93 (Elounda, June/July 1993)*, v. 697 of *Lect. Notes in Comput. Sci.*, pp. 385–396. Springer, 1993.

[5] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proc. of 1st Ann. IEEE Symp. on Logic in Computer Science, LICS '86 (Cambridge, MA, June 1986)*, pp. 267–278. IEEE CS Press, 1986.

[6] E. Grädel, W. Thomas, and T. Wilke, eds. *Automata, Logics, and Infinite Games: A Guide to Current Research*, v. 2500 of *Lect. Notes in Comput. Sci.* Springer, 2002.

[7] M. Jurdzinski, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM J. on Comput.*, 38(4):1519–1532, 2008.

[8] D. E. Long, A. Browne, E. M. Clarke, S. Jha, and W. R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In D. L. Dill, ed., *Proc. of 6th Int. Conf. on Computer-Aided Verification, CAV '94 (Stanford, CA, June 1994)*, v. 818 of *Lect. Notes in Comput. Sci.*, pp. 338–350. Springer, 1994.

[9] S. Schewe. Solving parity games in big steps. In V. Arvind and S. Prasad, eds., *Proc. of 27th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2007 (New Delhi, Dec. 2007)*, v. 4855 of *Lect. Notes in Comput. Sci.*, pp. 449–460. Springer, 2007.