Algorithms in the W-Hierarchy

Jonathan F. Buss and Tarique Islam*
School of Computer Science, University of Waterloo
{jfbuss,tmislam}@uwaterloo.ca

October 31, 2005

Abstract

We study variations on the "WRAM" model of programs that characterize the classes in the W-hierarchy of fixed-parameter complexity. An expanded model removes many of the constraints imposed on the use of registers in the checking phase, while a restricted model adds an additional constraint; both, however, are equivalent to the original model. We use the expanded model to show new upper bounds on several problems in the W-hierarchy, including SUBSETSUM, MAXIMUM IRREDUNDANT SET, and various DFA intersection problems. We use the restricted W-RAM model to show that ODD SET is $W[1]$-hard.

1 Introduction

The notion of fixed-parameter tractability has been introduced to deal with computational problems that are difficult (not known to be in P) in the classical context. A large number of NP-complete or NP-hard problems have been shown to be fixed-parameter tractable. However, there are parameterized problems for which fixed-parameter algorithms could not be developed and these problems are considered as (apparently) fixed-parameter intractable. A theory of fixed-parameter intractability has been developed to classify the problems according to their parametric hardness. The fundamental classes of the parametric intractability are the $W[t]$ classes forming the W-hierarchy. The classes $W[SAT]$ and $W[P]$ contain the entire W-hierarchy. These classes were originally characterized in terms of WEIGHTED CIRCUIT SATISFIABILITY problem on different circuit families [6, 7, 9]. The fundamental completeness proofs were complicated and the proof techniques followed a pattern different from the classical approach to obtain NP-completeness.

Recently, Flum, et al. [4, 5, 10] gave a new characterization of these classes using nondeterministic programs (herein termed $W[t]$-programs, originally termed AW-programs) running on a modified random-access machines called W-RAMs. They based their results on various old and new model-checking problems.

The programs running on a W-RAM must satisfy some restrictions. The most severe restriction is that the programs cannot use any nondeterministically guessed value in standard operations (such as arithmetic operations). Nonetheless, the machine characterization allows one to

*Both authors supported in part by a grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

analyze the classes of the W-hierarchy from a traditional computational point of view. In an earlier work, we presented direct, self-contained proofs of the fundamental completeness results in the parameterized complexity theory [1]. The proof techniques mimick the classical approach used for fundamental NP-completeness results. However, the W-RAM programs remained in their restricted form.

In this paper, we show that many of the restrictions on $W[t]$-programs can be relaxed without changing the complexity class. We show that one can allow $W[t]$-programs to directly access the values guessed in second or subsequent nondeterministic blocks. In addition, the final checking phase may use time that depends on the input size n, provided that the number of accesses to nondeterministically set registers remains restricted.

The new enhanced W-RAMs permit easier design of algorithms for these machines. We identify some subsets of the classes $W[P]$ and $L[2]$ that are contained in $W[2]$. Most of these results generalize to higher levels of the W-hierarchy. We also present new (mostly optimal) membership results for the parametric problems SubsetSum, MaximalIrredundantSet, Bounded DFA Intersection, i/o-deterministic Finite State Transducer Intersection and i/o-deterministic Finite-State Trans

In addition, we make use of our earlier reduction [1] to obtain new proofs of $W[t]$-completeness. Our proofs use a highly-restricted form of W-RAM that is sufficient to accept the complete problem Weighted t-Normalized SAT.

2 Nondeterministic Random-Access Machines and Parametric Classes

Flum, et al. [4, 5, 10] considered various model-checking problems in the W-hierarchy and from them developed a new characterization of its classes. Their underlying machine model, called a W-RAM, is a nondeterministic version of the standard random-access machine. As one might expect, some of the details of the machine model (or of the programs that run on it) can be changed without affecting the classes of problems defined. We shall consider two variations, the original "basic" model suitable for proving hardness results and an "extended" model that eases the task of designing algorithms to demonstrate class membership.

A W-RAM has two sets of registers: standard registers (labelled $r_0, r_1, \ldots$) and guess registers (labelled $g_0, g_1, \ldots$). It has the usual operations of a RAM and also nondeterministic assignment of registers, but some restrictions will apply to these operations, as explained below. For the purposes of studying the W-hierarchy, we view the computation of a W-RAM as occurring in three phases, a preprocessing (or preparation) phase, a guessing phase, and a checking phase, in that order.[1] The phases differ in their use of registers.

A basic $W[t]$-program is a program running on a W-RAM that meets the following restrictions: there are functions f and h, a polynomial p and a constant c such that on an input of length n and parameter k,

- both the total number of steps and the maximum value stored in any register are bounded above by $O\left(f(k)p(n)\right)$,
- the preprocessing phase uses only deterministic operations and only the standard registers,

[1] We expect that higher classes such as W[SAT] and W[P] have similar characterizations, but we leave that for future work.

- the guessing phase may have existential assignments to the guess registers, with a total of at most $h(k)$ operations, and
- the checking phase may have at most c existential and universal assignments to the guess registers giving $t-1$ alternating blocks, with a total of at most $h(k)$ operations.

In addition, operations on the guess registers are limited to the following.

EXISTS $\uparrow j$: Existentially assign a value in the range $[0 \ldots r_0]$ to guess register g_j.

FORALL $\uparrow j$: Universally assign a value in the range $[0 \ldots r_0]$ to guess register g_j.

JEQUAL $i\ j\ t$: If $g_{r_i} = g_{r_j}$ then jump to instruction t.

JZERO $i\ j\ t$: If $r_{\langle g_{r_i}, g_{r_j} \rangle} = 0$ then jump to instruction t. Here $\langle \cdot, \cdot \rangle$ is any reasonable pairing function.

Theorem 2.1 (Chen, Flum[4]) *A parametric problem Q can be decided by a basic $W[t]$-program if and only if Q is in $W[t]$. The value of t is one less than the maximum number of alternations used by the program.*

Following the initial proof [4] of this result, the current authors gave a direct proof [1]. One directi of our proof was an algorithm for a basic $W[t]$-program to accept the $W[t]$-complete language WEIGHTED t-NORMALIZED SAT.

For the other direction, we gave a generic reduction from any problem Q decided by a basic $W[t]$-program R to the language WEIGHTED t-NORMALIZED SAT. The reduction has three parts: (1) construction of a sub-formula mimicking the preprocessing phase of R, (2) selection of variables to code the existential assignments made during the guessing phase, and (3) a sub-formula representing whether a particular assignment to the variables of part (2) results in acceptance during the checking phase.

Part (1) of the proof uses a standard construction in the style of Cook's Theorem (modified to obtain a needed special form [2]). Part (2) merely selects names of Boolean variables, one for each possible value of each existentially assigned register. For part (3), the reduction considers all possible computations for the checking phase—that is, all possible universal and existential assignments made during this phase. To each assignment of values $v_1, \ldots, v_c$ to these registers there corresponds a formula $\beta(v_1, \ldots, v_c)$ that evaluates to true if the outcomes of the JEQUAL and JZERO tests on registers of part (2) cause R to accept. Here β has polynomial size and can be taken in either CNF or DNF form, as desired. The overall formula has the form

$$\varphi \wedge \bigwedge_{v_1=0}^{fp} \bigvee_{v_2=0}^{fp} \cdots \beta(v_1, \ldots, v_c) \ .$$

With appropriate choice of form for the β formulas, this formula is t-normalized. It has a satisfying assignment of weight k if and only if program R accepts.

Examining this proof, one sees the importance of the restrictions incorporated into the definition of a basic $W[t]$-program. The checking phase uses at most $h(k)$ existential registers from the checking phase, each of which has $O(f(k)p(n))$ values—thus the reduction can produce a variable for each combination. The number of different β formulas is $(fp)^c$, and each has a tree of test outcomes with $2^{h(k)}$ branches. Thus the whole formula remains polynomial-size.

The proof does not require any restrictions on the use of the registers that are nondeterministically assigned in the checking phase—all possible values are considered for the assignments, and arbitrary deterministic operations can simply be computed during the reduction. Likewise, the number of steps in each branch of the checking phase need not be bounded independently of n; only the number of tests of the registers set in the guessing phase needs to be bounded, and a larger bound of $O\big(\log\big(f(k)p(n)\big)bigr) = O(h(k) + \log n)$ suffices for this number. Additionally, the feature required of the JZERO and JEQUAL operations is that they have only two possible outcomes and change no register values. Any operations meeting this restriction may be allowed in their place.

We thus define an extended $W[t]$-program to meet the following relaxed restrictions: there are functions f and h, a polynomial p and a constant c such that on an input of length n and parameter k,

- both the total number of steps and the maximum value stored in any register are bounded above by $O\left(f(k)p(n)\right)$,
- the guessing phase consists of at most $h(k)$ existential assignments to the guess registers, and
- in the checking phase,
  - up to c nondeterministic assignments may be made to *standard* registers, but *none* to guess registers,
  - at most $h(k) + O(\log n)$ operations may access the guess registers assigned in the guessing phase, and
  - every operation that depends on the value of a guess register must be a pure conditional branch—that is, the operation must affect only the next instruction to be executed and none of the register values.

An extended $W[t]$-program has the same decision power as a basic $W[t]$-program.

Theorem 2.2 *A parametric problem Q can be decided by an extended $W[t]$-program if and only if Q is in $W[t]$. The value of t is one less than the maximum number of alternations used by the program.*

PROOF. The second proof of the previous theorem proof applies to the extended $W[t]$-program as well as a basic $W[t]$-program. ∎

Chen, Flum et al. introduced a variation of the W-RAM model called the ARAM [4, 5]. The distinguishing feature of an ARAM is that programs running on an ARAM can transfer a guessed value from a guess register to a standard register in a single computation step. Thus an ARAM is like our extended $W[t]$-program except that arbitrary operations may be used on the guess registers. They defined an additional hierarchy, known as the L-hierarchy, based on the new computation model. If one further relaxes the constraints to allow $h(k)$ universal assignment operations (retaining constant alternation), one obtains the classes $A[t]$ that form the A-hierarchy. Nontrivial containment or non-containment properties among these three hierarchies remain unknown.

3 3-Normalized Circuits and $W[2]$

In this section, we analyze the 3-NORMALIZED WCS in an attempt to identify instances decidable by a $W[2]$-program. We will make use of the extended $W[t]$-programs presented above. Let $\langle C, k\rangle$ be an input instance of 3-NORMALIZED WCS. A $W[2]$-program can perform an universal branching that corresponds to the output *and*-gate of C. The challenge is to decide the satisfiability of DNF sub-circuits in each universal branch within the constraints of the final (deterministic) checking phase. Rather than aiming for the apparently difficult general case, we start by identifying some easy cases. Most of the results in this section generalize to higher levels of the W-hierarchy.

3.1 Number of input variables in the sub-circuits at input level

Let $G_\vee$ be the output gate of the DNF sub-circuit $C_\vee$ of our interest and $G_\wedge = \{G_{\wedge 1}, \ldots G_{\wedge fp}\}$ are the *and*-gates providing inputs to $G_\vee$. Also, let R be the $W[2]$-program we are going to construct and let register r_u store the (universally selected) index of the sub-circuit $C_\vee$. Let $n_v = \max(n^{c/k}, n^{h(k)/t(n)})$, where $c > 0$ is a constant, h is any arbitrary function, and t is any unbounded nondecreasing function.

Lemma 3.1 *If $C_\vee$ is a DNF circuit with at most $f'(k)n_v$ input variables for some arbitrary function f', then a $W[2]$-program can decide whether the $\exists_1$-assignment satisfies $C_\vee$.*

PROOF. The $W[2]$-program R existentially guesses k true variables and then universally guesses a DNF sub-circuit $C_\vee$ as described above. Let L be a sorted list of $f'(k)n_v$ input variables of $C_\vee$. R performs k binary searches to determine which of the k existentially guessed input variables appear in L. If a variable $v \in L$ matches the value in an $\exists_1$-register, then v is set to true, otherwise, v is set to false. Finally, R deterministically evaluates the circuit $C_\vee$ on the truth assignment computed for the variables in L.

A naive implementation of the above will access existential registers too many times, in determining whether a variable that occurs in the circuit is one of the variables guessed true. We make use of the pre-processing phase to meet the restriction.

Let $L[i]$ be the i-th variable (in sorted order) in L. In the pre-processing phase, R constructs a lookup table A with $f'(k)n_v$ rows (one row for each input variable in L) and $f(k)p(n)$ columns (one column for each possible value in an $\exists_1$-register) for performing binary search on L. The contents of the table for circuit $C_\vee$ are set to be

$$A[i,j] = \begin{cases} 1 & j < L[i], \\ 0 & \text{otherwise.} \end{cases}$$

In the checking phase, R searches for the value in $\exists_1$-register r_i, $1 \leq i \leq k$, in the list L as follows. R uses a register r_u (referring to the sub-circuit $C_\vee$) to index a row in A and the $\exists_1$-register r_i to index a column in A. Initially, r_u is set to refer to the middle row of A. The search continues in the lower or upper half of A depending on whether $A[r_u, r_i]$ is 1 or 0 respectively and r_u is updated accordingly. Each binary search requires $\log(f'(k)n_v)$ $\exists_1$-comparisons resulting in a total of $k \log f'(k) + k \log n_v$ $\exists_1$-comparisons, for all k binary searches. Note that

$$\begin{aligned} k \log n_v &= \max\left(k \tfrac{c}{k} \log n, \tfrac{kh(k)}{t(n)} \log n\right) \\ &= \max\left(c \log n, o(h'(k) \log n)\right) \qquad \text{(where } h'(k) = kh(k)\text{)} \\ &= \max\left(c \log n, h''(k) + \log n\right) \\ &\qquad\qquad \text{, where } h''(k) = \tfrac{h'(k)}{t(n_k)} \log n_k, n_k = t^{-1}(h'(k)). \end{aligned}$$

Therefore, the total number of $\exists_1$-comparisons performed by R, is within the required bound. ∎

Note that the algorithm described in the proof of Lemma 3.1 can identify up to k existentially chosen true variables out of the $f'(k)n_v$ variables of interest. Once the assignment has been determined, the $W[2]$-program evaluates the corresponding sub-circuit $C_\vee$ on the computed assignment. The evaluation of the circuit does not require any additional tests of guess registers. Since a $W[2]$-program can have a parametric polynomial number of steps in the final checking phase, the algorithm can evaluate any polynomial size circuit having $f'(k)n_v$ input variables within the time bound. Therefore, we have the following result.

Corollary 3.2 *Consider a family of circuits $\mathcal{F}_1[2]$ of the form $(C_1 \wedge C_2 \wedge \ldots C_{fp})$, where $C_i, 1 \leq i \leq fp$, is any polynomial sized circuit having at most $f'(k)n_v$ input variables. The parametric problem* WCS *on $\mathcal{F}$ is in $W[2]$.*

The $W[2]$-program described in the proof of Theorem 3.1 does not use any property that is specific to $W[2]$ The result can be generalized for the higher classes in the W-hierarchy.

Corollary 3.3 *For $t \geq 2$, let the circuit family $\mathcal{F}_1[t]$ consist of the circuits of the form $\bigwedge_{i_1} \bigvee_{i_2} \ldots Q_{i_t}\ C(i_1, i_2, \ldots, i_t)$, where each $C(i_1, i_2, \ldots, i_t)$ is a polynomial size circuit with at most $f'(k)n_v$ input variables, Q is an* and-*gate for even t and Q is an* or-*gate for odd t. A $W[t]$-program can decide the* WCS *on $\mathcal{F}_t$.*

3.2 Minimum number of positive literals in each clause

In this subsection, we consider the WCS on a sub-family of 3-normalized circuits where each *and*-gate at the input level contains a minimum number (specified below) of positive literals and an unbounded number of negative literals. Let $C_\vee$ be a DNF circuit such that each product in $C_\vee$ has at least r positive literals. We say that $C_\vee$ is a *r-monotone* DNF circuit. Let C be a 3-normalized circuit of the form $C_{1\vee} \wedge C_{2\vee} \wedge \ldots \wedge C_{fp\vee}$, where each $C_{i\vee}$ is a r-monotone DNF circuit. We show that if $r \geq k - c$ for some constant c, then a $W[2]$-program can decide WCS on C.

Theorem 3.4 WCS *on the circuit family $\mathcal{F}_2[2]$ of the form $\bigwedge \bigvee (l_1 \wedge \ldots l_r \wedge \neg l_{r+1} \wedge \ldots \wedge \neg l_{fp})$, $r \geq (k-c)$ for some constant c, is in $W[2]$.*

PROOF. Let C be a circuit in the family $\mathcal{F}_2[2]$. Let $C_\wedge = (l_1 \wedge \ldots l_r \wedge \neg l_{r+1} \wedge \ldots \wedge \neg l_{fp})$ be a product appearing at the input level of C. We say that $(l_1 \wedge \ldots \wedge l_r)$, consisting of all positive literals in $C_\wedge$, is the *monotone group* of the product $C_\wedge$. We construct a $W[2]$-program R to decide whether C has a weight-k satisfying assignment, as follows.

1. Construct an enumeration M of all monotone groups appearing in the input circuit C. This eumeration has size bounded by a parametric polynomial.

2. Existentially guess k true variables. Let V_p be the set of these k variables.
3. Existentially guess the number of monotone groups $l \leq k^k$ that are satisfied by the variables in V_p.
4. Existentially guess $l \leq k^k$ monotone groups from M, that are satisfied by the assignment. Let $M' \subseteq M$ be the set of these monotone group(s).
5. Universally select a monotone group $x \in M$.
6. Deterministically check that either x is in M' or x is inconsistent with the variables in V_p (this check ensures that M' is maximal).
7. Universally select an input branch C_{DNF} of the output *and*-gate of C.
8. Deterministically check whether any of the guessed monotone groups in M' appears in C_{DNF}, using k^k $\exists_1$-tests. If no such monotone group is found, then reject.
9. Let x be a monotone group in M' that appears in C_{DNF}. Retrieve the value in x, one bit at a time, using at most $(\log f(k) + d \log n)$ many $\exists_1$-tests, where d is a constant. The value in x implies the values in at least $(k - c)$ variables in V_p.
10. Retrieve the values of the remaining c variables of V_p using at most $c(\log f(k) + d \log n)$ $\exists_1$-tests.
11. Deterministically evaluate the circuit C_{DNF} on the guessed assignment in parametric polynomial time and accept or reject accordingly (R now knows the values in all variables in V_p).

∎

The result can be generalized to higher levels of the W-hierarchy. It is easy to see that the negated literals in each product at the input level can be replaced by circuits of parametric polynomial size.

Corollary 3.5 *Consider the circuit family $\mathcal{F}_3[2]$ of the form $\bigwedge \bigvee (l_1 \wedge \ldots l_r \wedge C_{r+1} \wedge \ldots \wedge C_{fp})$, where $r \geq (k - c)$ for some constant c and $C_{r+1}, \ldots, C_{fp}$ are circuits of parametric polynomial size.* WCS *on $\mathcal{F}_3[2]$ is in $W[2]$.*

3.3 Number of monotone groups and number of variables in each monotone group

Theorem 3.6 *Let $\mathcal{F}_4[2]$ be a family of 3-normalized circuits of the form $(C_1 \wedge \ldots \wedge C_{fp})$ such that each DNF circuit C_i, $1 \leq i \leq fp$, satisfies the following conditions.*

1. *The number of monotone groups (of any size) appearing in C_i is bounded by $n^{o(h(k))}$.*
2. *For each monotone group x, there are at most $n^{o(h(k))}$ variables appearing in all products in C_i with monotone group x.*

The parameterized problem WCS *on $\mathcal{F}_4[2]$ is in $W[2]$.*

PROOF. Let C be a circuit in $\mathcal{F}_4[2]$. We construct a $W[2]$-program to decide the WCS on C, as follows.

1. Construct an enumeration M of all monotone groups appearing in the input circuit C. Note that $|M|$ is bounded by a parametric polynomial.

2. Existentially guess k *true* variables. Let V_p be the set of these k variables.

3. Existentially guess the number $l \leq k^k$ of monotone groups that are satisfied by the variables in V_p.

4. Existentially guess $l \leq k^k$ monotone groups from M, that are satisfied by the assignment. Let $M' \subseteq M$ be the set of these monotone groups.

5. Universally select a positive group $x \in M$.

6. Deterministically check that either x is in M' or x is inconsistent with the variables in V_p (i.e., ensure that L' is maximal).

7. Universally select an input branch C_i of the output *and*-gate of C.

8. Let $M_i \subseteq M$ be the list of monotone groups that appear in C_i. Perform l binary searches (as was done in the proof of Theorem 3.1) on M_i to identify the satisfied monotone groups in M_i. Since $|M_i|$ is $n^{o(h(k))}$, this step requires $o(lh(k) \log n)$ many $\exists_1$-tests. Let S_i be the set of satisfied monotone groups in M_i ($S_i = M_i \cap M'$).

9. For each positive group y in S_i, let V_{iy} be the set of input variables that appear in any product of C_i with y as the monotone group.

   (a) Perform k binary searches to determine which of the k variables in V_p are members of V_{iy}. Since $|V_{iy}|$ is $n^{o(h(k))}$, this step requires $o(kh(k) \log n)$ $\exists_1$-tests for each satisfied monotone group y.

   (b) Evaluate the products with monotone group y on the (partially) computed assignment. Accept if any such product is satisfied.

∎

3.4 Range of the functions in parameterized model-checking problems

Theorem 3.7 *Let τ be a vocabulary with binary relations and binary functions such that the range of each function is a set of size at most $n^{o(h(k))}$. For all $t \geq 1$, the parameterized model checking problem on vocabulary τ and the class of all $\Sigma_{t,u}$ formulas is in $W[t]$.*

PROOF. We prove the theorem for $t = 2$. The proof can be easily generalized for higher levels. Let τ be a vocabulary that satisfies the conditions in the theorem, and let φ be a formula in $\Sigma_{t,u}$. By definition, the length of φ is bounded by some function $h'(k)$. We construct a $W[2]$-program R to decide the corresponding model checking problem, as follows.

1. Perform existential guesses to simulate the existential quantifiers in φ.

2. Perform universal guesses to simulate the universal quantifiers in φ. Note that there can be at most u such quantifiers in φ, for a constant u.

3. In each nondeterministic branch, deterministically check that the relations in φ are satisfied by the guessed values. To check a single binary relation, use the JEQUAL or JZERO instruction (with appropriate lookup tables) in constant time.

4. Compute the value of each binary function, one bit at a time, using appropriate lookup tables.

5. Check whether the matrix of φ is satisfied by the guesses and accept or reject accordingly.

R makes a constant number (at most u) of universal guesses in the second nondeterministic block. The matrix of φ includes at most $h'(k)$ binary relations and/or binary functions. Checking each relation takes constant time. Computing each function value requires $o(h(k)\log n)$ many $\exists_1$-tests, as the range is a set of size at most $n^{o(h(k))}$. Thus computing all the required function values can be done using at most $o(h'(k)h(k)\log n)$ $\exists_1$-tests and in parametric polynomial number of steps. Therefore, R satisfies all the constraints of a $W[2]$-program. ∎

4 Upper Bounds for Some Parametric Problems

As in the previous section, we shall use the extended $W[t]$-program model in our computations of this section.

4.1 SUBSETSUM

The parametric problem SUBSETSUM is defined as follows.

SUBSET SUM

Input: A set of integers $X = \{x_1, \ldots, x_m\}$, an integer s. These values, which have n total bits, may be coded into input registers in any reasonable way.

Parameter: A positive integer k.

Question: Is there a set $X' \subseteq X$ such that $|X'| = k$ and the sum of all the integers in X' is exactly s?

Previous results [7, 11] have shown that SUBSETSUM is hard for $W[1]$ and lies in $W[P]$. We show that it lies in $W[3]$.

Theorem 4.1 SUBSET SUM *is in* $W[3]$.

PROOF. We construct a $W[3]$-program R to decide SUBSETSUM. We assume w.l.o.g. that all numbers in X are positive. Let $r = 2^{\lceil \log k \rceil}$. All numbers will be represented in radix r, with each digit in a separate register.

Conceptually, R works in two major steps. In the first step, R considers the k existentially selected integers as radix-r numbers. R adds them, without carries from one digit to the next, to produce two integers s_{sum} and s_{carry}—the first containing the sum digits and the second containing the carry digits (a single column sum must be less than r^2). In the second step, R adds the two numbers s_{sum} and s_{carry}, in binary, to produce s_{result} and checks that s_{result} and s are the same. The details are as follows.

1. Existentially select a subset X' of k integers from X.
2. Universally select a bit position b of the target sum s.
3. Extract all radix-r digits from the k selected numbers at positions b and $b-1$ and compute the radix-r digits of s_{sum} and s_{carry} at position b.
4. Add the computed digits of s_{sum} and s_{carry} and compare them with the corresponding digit of the target sum s. If the difference is neither 0 nor 1, reject; a larger carry-in cannot occur in the sum of two numbers.
5. If the required carry-in at position b is 0, then do the following.
   (a) Universally select a position $0 \leq b' < b$ that generates a carry.
   (b) Existentially select a position i with $b' < i < b$ such that carry is not propagated through position i.

   Otherwise, the required carry-in at position b is 1; do the following.
   (a) Universally select a bit position $-1 \leq b' < b$.
   (b) If $b' = -1$ or the digits at position b' produce no carry, existentially select a position g with $b' < i < b$ such that a carry is generated at position g.

   Computing whether a position generates or propagates a carry requires an additional computation as in step 3.

In the algorithm described above, the number of alternations between quantifiers is at most three. Most steps are trivial and require constant time. Step 3 requires $O(k \log r)$ steps to extract the digits bit-by-bit (given appropriate lookup tables constructed in the preprocessing phase) and add them together. Thus program R satisfies all the constraints of a $W[3]$-program. ∎

4.2 BOUNDED DFA INTERSECTION

The problem BOUNDED DFA INTERSECTION takes the descriptions of a number of DFAs as input and asks for a string accepted by all the DFAs (if there exists any). The problem has different versions depending on the choice of the parameter. The version we are interested in is the following.

BOUNDED DFA INTERSECTION (BDFAI)

Input: An input alphabet Σ, a set A of k DFAs $A = \{A_1, \ldots, A_k\}$ on Σ, a positive integer m. Let Q_i be the state set of DFA A_i, $1 \leq i \leq k$. Also, let $q = \max\{\,|Q_i| \mid 1 \leq i \leq k\,\}$.

Parameter: m, q.

Question: Is there a string $x \in \Sigma^m$ such that x is accepted by all DFAs $A_1, \ldots, A_k$?

BDFAI is known to be $W[2]$-hard and in $W[P]$ [3, 12]. We show that BDFAI is in $W[2]$ and hence is $W[2]$-complete.

Theorem 4.2 BDFAI *is* $W[2]$*-complete.*

PROOF. We construct a $W[2]$-program R to decide BDFAI. The algorithm is essentially the same as the one described by Cesati [3]. In the preprocessing step, R sets aside $O(q \log q\,|\Sigma|)$ space for the transition table of a single DFA in A. R then nondeterministically guesses m symbols from Σ (representing a string of length m) and stores them in guess registers $g_1, \ldots, g_m$. R then universally guesses a DFA A_i from the set A. In the final checking phase, R simulates A_i on the existentially guessed string of length m.

Before starting the simulation, R constructs the transition table T_i for A_i in the space reserved during the preprocessing phase. Note that all universal branches will use the same space to create the transition table of the corresponding DFAs. The content of T_i is defined as follows.

$$T_i[\langle u, b\rangle, v] = \begin{cases} 0 & \text{if } A_i \text{ at state } u \in Q_i \text{ and on symbol } v \in \Sigma \text{ makes a transition to state } u' \in Q_i \\ & \text{and } b\text{-th bit of the coding of state } u' \text{ is } 0,\ b \leq \log q, \\ 1 & \text{if } A_i \text{ at state } u \in Q_i \text{ and on symbol } v \in \Sigma \text{ makes a transition to state } u' \in Q_i \\ & \text{and } b\text{-th bit of the coding of state } u' \text{ is } 1,\ b \leq \log q, \end{cases}$$

The construction of T_i can be done in parametric polynomial time. Let us assume that the guess register g_{m+1} contains the universally guessed value that refers to DFA A_i. To start the simulation, R assigns the coding of the start state of A_i to g_{m+1}. In general, let u be the present state stored in g_{m+1} and v be the next input symbol stored in some $\exists_1$-register g_j, $1 \leq j \leq m$. R executes the JZERO instruction $\log q$ times to retrieve the $\log q$-bits representing the next state. To retrieve bit b, R modifies the value in g_{m+1} so that it represents $\langle u, b\rangle$, where u is the current state. R constructs the next state u' from these bits and assigns u' to g_{m+1}. This process is repeated at most m times. R accepts in this nondeterministic branch if and only if A_i accepts the existentially guessed string. The final checking phase requires parametric polynomial time and at most $m \log q$ tests involving $\exists_1$-registers (where both m and q are parameters). ∎

4.3 I/O-DETERMINISTIC FST INTERSECTION

Wareham [12] has shown that the following two problems are respectively $W[1]$-hard and $W[2]$-hard.

FST-I-1

Input: A set of finite-state transducers $A = \{A_1, \ldots, A_k\}$, having common input and output alphabets Σ_i and Σ_o, and a string $s_{in} \in \Sigma_i^+$.

Parameter: k, $|s_{in}|$, $|\Sigma_i|$.

Question: Is there a string $s_{out} \in \Sigma_o^{|s_{in}|}$ such that each FST $A_i \in A$ accepts s_{in}/s_{out}?

FST-I-2

As FST-I-1, except the parameters are $|s_{in}|$, $|\Sigma_i|$, and $Q = \max(|Q_i|, 1 \leq i \leq k)$, where Q_i is the state set of FST A_i.

We show that FST-I-1 is in $W[1]$ and FST-I-2 is in $W[2]$; thus they are complete for these classes.

Theorem 4.3 FST-I-1 *is in* $W[1]$ *and hence is* $W[1]$*-complete.*

PROOF. We construct an extended $W[1]$-program R that decides FST-I-1. In particular, R can access the existential guess registers directly and can have at most $h(k, |s_{in}|)$ steps in the final checking phase, for some function h. Program R essentially guesses the $|s_{in}|$ symbols of s_{out} and then simulates the k FSTs in A on s_{in}/s_{out}. Therefore, R needs to simulate a total of $k\,|s_{in}|$ steps in the final checking phase. Note that $k\,|s_{in}|$ is a function of the parameters only and is independent of n.

In the preprocessing step, R constructs a lookup table T to store the transition functions of all the FSTs in A such that each lookup operation can be performed in constant time. R then existentially guesses the $|s_{in}|$ symbols of s_{out} from the output alphabet Σ_o. In the final checking phase, R sequentially simulates all the FSTs in A, one after another, on the input/output pair s_{in}/s_{out}. R accepts if all the FSTs accept, R rejects otherwise. Let the standard registers $r_1, \ldots, r_k$ store the starting states of the FSTs $A_1, \ldots, A_k$ respectively. In order to start the simulation of FST A_i on s_{in}/s_{out}, R retrieves the start state in r_i and the first character of s_{in} and records them as the current state and current input symbol of A_i. In addition, R records the first existentially guessed value as the current output symbol. R computes the unique next state of A_i from the current state, current input symbol and current output symbol in constant time by consulting the lookup table T. R repeats this process at most $|s_{in}|$ times for each FST A_i, $1 \leq i \leq k$. Thus, R can simulate all the FSTs in the final checking phase within the time bound. ∎

We now show that the parameterized problem FST-I-2 is in $W[2]$. The proof idea is very similar to that of Theorem 4.3. The crucial difference is that the number of FSTs in A is now unbounded and hence it is unlikely that a $W[1]$-program can simulate all FSTs within the time bound of the final checking phase. Also, a $W[2]$-program cannot access the existentially guessed values directly. Therefore, the lookup operation described in the proof of Theorem 4.3 can no longer be implemented in this case. We present the proof in detail below.

Theorem 4.4 FST-I-2 *is in* $W[2]$ *and hence is* $W[2]$*-complete.*

PROOF. We construct a $W[2]$ program R that decides FST-I-2. R existentially guesses the target output string s_{out} of length $|s_{in}|$, universally selects an FST A_i and then simulates the computation of A_i on s_{in}/s_{out} in the final checking phase.

While simulating the computation of FST A_i, R has to compute the next state based on the current state, current input symbol (from s_{in}) and current output symbol (from the corresponding guess register). Since R is not allowed to access the $\exists_1$-registers directly, this computation cannot be performed in the way it was done in the proof of Theorem 4.3. Instead, R computes each of the $\log Q$ bits of the next state, one at a time, using $\log Q$ $\exists_1$-tests on the current output symbol (as was done in the proof of Theorem 4.2). The required number of $\exists_1$-tests is $|s_{in}| \log Q$, which is a function of the parameters only and independent of n. ∎

As shown above, the $W[2]$-program R can perform $\max(c \log n, o(h \log n))$ $\exists_1$-tests in the final checking phase, where $c > 0$ is a constant and h is any function dependent on the parameters only. Since, each nondeterministic computation branch simulates $|s_{in}|$ computation steps of the corresponding FST, R can perform $\max(\frac{c \log n}{|s_{in}|}, \frac{o(h \log n)}{|s_{in}|})$ $\exists_1$-tests per simulated steps, allowing Q to be as large as $\max(n^{\frac{c}{|s_{in}|}}, n^{o(h)})$. Let us define the following stronger version of FST-I-2.

I/O-DETERMINISTIC FST INTERSECTION (FST-I-2A)

Input: A set of FST $A = \{A_1, \ldots, A_k\}$ such that all of them have common input and output alphabets Σ_i and Σ_o respectively, a string $s_{in} \in \Sigma_i^+$. Let Q_i be the state set of FST A_i and $Q = \max(Q_i)$.

Parameter: $|s_{in}|$, m such that $Q \leq \max(n^{\frac{c}{m}}, n^{o(h(|s_{in}|,m))})$ where $c > 0$ is a constant and h is any function.

Question: Is there a string $s_{out} \in \Sigma_o^{|s_{in}|}$ such that each FST $A_i \in A$ accepts s_{in}/s_{out}?

Corollary 4.5 FST-I-2A *is* $W[2]$*-complete.*

Next, we show that the problem I/O-DETERMINISTIC FST COMPOSITION, parameterized by the number of FSTs and the length of each target string, is $W[1]$-complete.

I/O-DETERMINISTIC FST COMPOSITION (FST-C)

Input: A set of FST $A = A_1, \ldots, A_k$ such that all of them have common input alphabets and output alphabets Σ_i and Σ_o respectively, a string $s_0 \in \Sigma_i^+$.

Parameter: k, $|s_0|$.

Question: Do there exist k strings $s_1, \ldots, s_k$, each of length $|s_0|$ such that FST A_i accepts $s_{(i-1)}/s_i$?

Theorem 4.6 FST-C *is* $W[1]$*-complete.*

PROOF SKETCH. The $W[1]$-hardness of FST-C was proved by Wareham [12]. The proof of Theorem 4.3 can be modified in the obvious way to prove the membership result.
∎

4.4 MAXIMAL IRREDUNDANT SET

Let $G = (V, E)$ be a graph, V' a subset of the vertices, and $u \in V'$. A *private neighbour* of u is a vertex u' that is adjacent to u and to no other vertex in V'.

MAXIMAL IRREDUNDANT SET

Input: A graph $G = (V, E)$, a positive integer k.

Parameter: k.

Question: Is there a set $V' \subseteq V$ such that (1) Each vertex $u \in V'$ has a private neighbour, and (2) V' is not a proper subset of any other set $V'' \subseteq V$ that also has this property?

MAXIMAL IRREDUNDANT SET is known to be $W[2]$-hard and in $W[P]$ [3]. In this section, we show that MAXIMAL IRREDUNDANT SET is in $W[4]$ by constructing a $W[4]$-program to decide the problem. The algorithm is essentially the same as Cesati's algorithm [3]. We only show that the algorithm can be implemented by a $W[4]$ program.

Theorem 4.7 MAXIMAL IRREDUNDANT SET *is in* $W[4]$.

PROOF. We construct a $W[4]$-program R to decide MAXIMAL IRREDUNDANT SET, as follows. In the first nondeterministic block, R existentially guesses k vertices for V' and their k private neighbours. R then checks in time $O(k^2)$ and with $O(k^2)$ $\exists_1$-tests that the k guessed vertices indeed form an irredundant set in the input graph $G = (V, E)$. To ensure maximality, R universally guesses another vertex w and, in each universal branch, R verifies that $V' \cup \{w\}$ is not an irredundant set. R existentially guesses an index (between 1 and $k+1$) of a vertex v_i in V''. R then universally guesses a neighbour w of v_i and checks that w is adjacent to some vertex in $V'' \setminus v_i$ (i.e., w is not a private neighbour of v_i). Overall the algorithm is equivalent to the following formula.

$$\exists v_1, \ldots, v_k\, x_1, \ldots, x_k\, \forall v_{k+1} \in V - \{v_1, \ldots, v_k\}\ \ \exists i \in \{1, \ldots, k+1\}\ \ \forall w \in V - \{v_1, \ldots, v_{k+1}\}$$

$$\left[\prod_{1 \le r \le k} \left[[(v_r, x_r) \in E \vee v_r = x_r] \wedge \prod_{1 \le s \le k, r \ne s} (v_s, x_r) \notin E\right]\right]$$

$$\wedge \left[(w \notin N[v_i]) \vee \sum_{1 \le j \le k+1, j \ne i} (w \in N[v_j])\right]$$

Clearly the algorithm can be implemented by a $W[4]$-program, and hence MAXIMAL IRREDUNDANT SET is in $W[4]$. ∎

Corollary 4.8 MAXIMAL IRREDUNDANT SET *is in* $W^*[3]$.

PROOF. The range of values for the third existential quantifier is $[1, k+1]$. ∎

5 Hardness Results for Some Parametric Problems

In this section we show the $W[1]$-hardness of the parametric problems EXACT ODD SET, ODD SET, and EXACT EVEN SET. Our proofs will use generic reductions from $W[t]$-programs. To make the proofs as simple as possible, we shall return to the basic W-RAM model and even make one further restriction.

Definition 5.1 *A restricted $W[1]$-program is a basic $W[1]$-program that has only one accepting path in the checking phase, determined by the outcomes of tests of guess registers (at most $h(k)$ tests).*

Theorem 5.1 *A parametric problem Q is in $W[1]$ if and only if there exists a restricted $W[1]$-program that accepts Q.*

PROOF. Our previous $W[1]$-program to decide c-CNFSAT [1] tested the conjunction of a set of $h(k)$ consistency tests on the guessed registers. Only JZERO and JEQUAL tests are required these registers. Thus it was a restricted $W[1]$-program.

In general, given a parametric problem Q in $W[1]$, a restricted $W[1]$-program for Q can be constructed by combining a deterministic FPT reduction with the above program. ∎

In the rest of this section, all of our $W[1]$-programs will be restricted. We start with the EXACT ODD SET problem and then extend the proof for EXACT EVEN SET.

Theorem 5.2 EXACT ODD SET *is hard for* $W[1]$.

PROOF. Let Q be a parametric problem in $W[1]$. By Theorem 5.1, there exists a $W[1]$-program R_Q that decides Q. We construct a fixed-parameter reduction $\mathcal{A}$ that takes $\langle x, k\rangle$ as input and constructs an instance of $\langle x', k'\rangle$ of EXACT ODD SET such that $\langle x', k'\rangle$ is in EXACT ODD SET if and only if R_Q accepts $\langle x, k\rangle$.

$\mathcal{A}$ deterministically simulates the computation of R_Q on $\langle x, k\rangle$ in the preprocessing phase until R_Q is about the make the first existential guess. $\mathcal{A}$ now constructs a red/blue graph $G = (R, B, E)$ to represent the rest of the computation of R_Q on $\langle x, k\rangle$, where R and B are the set of red vertices and blue vertices respectively and E is the set of edges. Let us assume that R_Q makes $h(k)$ existential guesses and each guess is an integer in the range $[0 \ldots (fp-1)]$. The red set R consists of two types of red vertices. The red vertices of the first type are $R_1 = \{\, iv_i \mid 1 \le i \le h(k), 0 \le v_i \le (fp-1)\,\}$. A red vertex $iv_i \in R_1$ represents the fact that the i-th existential guess in R_Q's computation is v_i. The red vertices of the second type are of the form

$$R_2 = \{\, \langle iv_i, jv_j\rangle \mid 1 \le i < j \le h(k), 0 \le v_i, v_j \le (fp-1)\,\} \quad .$$

A red vertex $\langle iv_i, jv_j\rangle \in R_2$ represents the fact that the i-th existential guess is v_i and the j-th existential guess is v_j.[2] The set of red vertices in G is $R = R_1 \cup R_2$. Given a red vertex $\langle iv_i, jv_j\rangle$, we refer to iv_i as the *first component*, jv_j as the *second component*, i as the *index* of the first component and j as the *index* of the second component. The instance of EXACT ODD SET asks to select $h(k) + \binom{h(k)}{2}$ red vertices. Our goal is to have a one-to-one mapping between the set of red vertices in the odd set of G and the set consisting of the $h(k)$ existential guesses of R_Q and $\binom{h(k)}{2}$ pairs resulting from them. $\mathcal{A}$ adds the blue vertices to add constraints so that we have the desired bijection. The blue vertices are added as follows.

1. The first set of blue vertices B_1 ensures that the odd set includes at least one red vertex from R_1 for each i, $1 \le i \le h(k)$ and at least one red vertex from R_2 for each pair of i and j, $1 \le i < j \le h(k)$. B_1 contains $h(k) + \binom{h(k)}{2}$ vertices for this purpose.

$$B_1 = \left\{ b_{1u} \mid 1 \le u \le h(k) + \binom{h(k)}{2} \right\}$$

The first $h(k)$ blue vertices in B_1 add constraints for red vertices in R_1, while the remaining blue vertices in B_1 are for the red vertices in R_2. The edges connected to each blue vertex $b_{1u} \in B_1$ are as follows.

$$E_{b_{1u}} = \begin{cases} \{\, \langle b_{1u}, uv_u\rangle \mid 0 \le v_u \le (fp-1)\,\} & , \quad 1 \le u \le h(k) \\ \{\, \langle b_{1u}, \langle i_u v_{i_u}, j_u v_{j_u}\rangle\rangle \mid 0 \le v_{i_u}, v_{j_u} \le (fp-1)\,\} & , \quad h(k)+1 \le u \le \binom{h(k)}{2} \\ & \quad\ \text{and } u \text{ corresponds to index} \\ & \quad\ \text{pair } \langle i_u, j_u\rangle \end{cases}$$

$$E_1 = \bigcup_{b_{1u} \in B_1} E_{b_{1u}}$$

[2]Since the tests in the final checking phase of R_Q are binary, we are interested in the pair of guessed values.

Note that the neighbourhood of the vertices in B_1 are pairwise disjoint and hence they partition the set R into $(h(k)+\binom{h(k)}{2})$ disjoint groups. By construction, any odd set of G must include at least one vertex from each group in the partition. On the other hand, the instance of the EXACT ODD SET that $\mathcal{A}$ constructs, asks for an odd set S of size $(h(k)+\binom{h(k)}{2})$. These two conditions together imply that S includes exactly one vertex from R_1 for each i, $1 \le i \le h(k)$ and exactly one vertex from R_2 for each index pair $\langle i,j\rangle$, $1 \le i < j \le h(k)$.

2. The next set of blue vertices B_2 ensures that if the odd set S includes two vertices iv_i and jv_j from R_1 then S also includes the vertex $\langle iv_i, jv_j\rangle$ and vice versa. For each vertex iv_i in R_1 and for each index j, $1 \le j \le h(k), i \ne j$, $\mathcal{A}$ adds one blue vertex $b^2_{iv_i,j}$ to B_2. The edges connected to $b^2_{iv_i,j}$ are as follows.

$$E^2_{iv_i,j} = \begin{cases} \left\{\left\langle b^2_{iv_i,j}, iv_i\right\rangle\right\} \cup \left\{\left\langle b^2_{iv_i,j}, \langle jv_j, iv_i'\rangle\right\rangle \,|\, 0 \le v_i', v_j \le (fp-1), v_i' \ne v_i\right\} & ,i > j \\ \left\{\left\langle b^2_{iv_i,j}, iv_i\right\rangle\right\} \cup \left\{\left\langle b^2_{iv_i,j}, \langle iv_i', jv_j\rangle\right\rangle \,|\, 0 \le v_i', v_j \le (fp-1), v_i' \ne v_i\right\} & ,i < j \end{cases}$$

A blue vertex $b^2_{iv_i,j} \in B_2$ is connected to the red vertex $iv_i \in R_1$ and to all red vertices in R_2 that pair up indices i and j, except those that have iv_i in the first (or second) component. By condition 1, the solution can contain exactly one red vertex r_{ij} from R_2 such that r_{ij} has i and j as the indices (in any order) of its components. Hence, for each $b^2_{iv_i,j}$, any odd set for G includes either (a) the only neighbour in R_1 or (b) exactly one neighbour in R_2, but not both.[3]

Lemma 5.3 *A red vertex $\langle iv_i, jv_j\rangle$ from R_2 is in the odd set if and only if both iv_i and jv_j from R_1 are in the odd set.*

PROOF. Let us assume that the odd set S contains the red vertex $\langle iv_i, jv_j\rangle$ from R_2. Each blue vertex $b^2_{iv_i',j}$, $v_i' \ne v_i$, is a neighbour of $\langle iv_i, jv_j\rangle$. The red vertex iv_i' is the neighbour of $b^2_{iv_i',j}$ from R_1, $v_i' \ne v_i$. Hence, none of the vertices iv_i', $v_i' \ne v_i$ can be included in the odd set S (case (b)). By condition 1, S must include one vertex with index i from R_1 and that must be iv_i. The same argument holds for jv_j.

For the other direction, let us assume that S contains both iv_i and jv_j, $i < j$, for some v_i and v_j. Consider the neighbourhood of the blue vertex $b^2_{iv_i,j}$. The odd set S cannot contain any red vertex of the form $\left\langle iv_i', jv_j'\right\rangle$, $v_i' \ne v_i$ (case (a)). Since, S must contain at least one red vertex with indices i and j from R_2 (by condition 1), there must be a vertex $\left\langle iv_i, jv_j''\right\rangle$, for some v_j'', in S. By a similar argument, S must have a vertex $\langle iv_i'', jv_j\rangle$, for some v_i''. Therefore, the only red vertex in S with indices i and j must be $\langle iv_i, jv_j\rangle$. ∎

The blue vertex set B_2 and the edge set E_2 for the second set of constraints are as follows.

$$B_2 = \left\{ b^2_{iv_i,j} \,|\, 0 \le v_i \le (fp-1), 1 \le i,j \le h(k), j \ne i \right\}$$

[3] If both the neighbour in R_1 and exactly one neighbour in R_2 are included in S, then $b^2_{iv_i,j}$ will have two neighbours in the solution.

$$E_2 = \bigcup_{\substack{1 \le i,j \le h(k), j \ne i \\ 0 \le v_i \le (fp-1)}} E^2_{iv_i,j}$$

3. The last set of constraints relates to the test instructions appearing in the unique accepting branch of R_Q. Let $T_R = \{T_1, T_2, \ldots, T_{h'(k)}\}$ be the binary tests of interest. $\mathcal{A}$ adds one blue vertex $b^3_{T_i}$ to B for each test T_i in T_R, $1 \le i \le h'(k)$. Let $Succ_i$ be the set of pair of existentially guessed values for which the outcome of test T_i is *true* (or 1). $\mathcal{A}$ connects $b^3_{T_i}$ to each red vertex $\langle iv_i, jv_j \rangle$ in $Succ_i$. Formally, the set of edges $E^3_{T_i}$ connected to the blue vertex $b^3_{T_i}$ is as follows.

$$E^3_{T_i} = \left\{ \left\langle b^3_{T_i}, \langle iv_i, jv_j \rangle \right\rangle \mid \langle iv_i, jv_j \rangle \in Succ_i \right\}$$

Therefore, the blue vertices and corresponding edges enforcing these constraints are as follows.

$$B_3 = \left\{ b^3_{T_i} \mid 1 \le i \le h'(k) \right\}$$

$$E_3 = \bigcup_{T_i \in T_R} E^3_{T_i}$$

The set of blue vertices in the red/blue graph G is $B = B_1 \cup B_2 \cup B_3$ and the edge set is $E = E_1 \cup E_2 \cup E_3$. A solution to the EXACT ODD SET problem must select a consistent set S of $(h(k) + \binom{h(k)}{2})$ red vertices by constraints 1 and 2. In addition, the solution must include at least one red vertex in $Succ_i$, $1 \le i \le h'(k)$. Altogether this gives $h(k)$ values for the $h(k)$ existential guesses in the computation of R_Q such that all test outcomes along the unique accepting branch of R_Q are *true*. Thus, R_Q accepts if and only if $G = (R, B, E)$ has an exact odd set of size $(h(k) + \binom{h(k)}{2})$. ∎

Corollary 5.4 ODD SET *is hard for* $W[1]$.

PROOF. The generic reduction for the ODD SET is the same as the reduction $\mathcal{A}$ described in the previous proof. By constraint 1, any solution to the instance of the ODD SET problem must select at least $\binom{h(k)+h(k)}{2}$ red vertices. ∎

Next we present a generic reduction to show the $W[1]$-hardness of EXACT EVEN SET. It is a modified version of the reduction in Theorem 5.2, and it implements techniques similar to those used by Downey, et al. [8].

Theorem 5.5 EXACT EVEN SET *is hard for* $W[1]$.

PROOF. We make the following changes to the red/blue graph $G = (R, B, E)$ constructed in the proof of theorem 5.2. Let the modified red/blue graph be $G_e = (R_e, B_e, E_e)$.

1. R_e consists of two copies of the set R and a special red vertex r_{parity}. We denote the copies of R by R' and R''.

2. B_e consists of the set B and another set of blue vertices B_{double} such that $|B_{double}| = |R|$.

3. E_e consists of the set E and two additional sets, E_{parity} and E_{double}, which are defined below.

$$E_{parity} = \{ \langle r_{parity}, b \rangle \mid b \in B \}$$

Let r'_i, r''_i and b_i be the i-th vertices of R', R'' and B_{double} respectively, $1 \leq i \leq |R|$. Let

$$E^i_{double} = \left\{ \langle b^i_{double}, r'_i \rangle, \langle b^i_{double}, r''_i \rangle \right\}, \quad 1 \leq i \leq |R|$$

and

$$E_{double} = \bigcup_{1 \leq i \leq |R|} E^i_{double}$$

The instance of EXACT EVEN SET asks for an even set of size exactly $2(h(k) + \binom{h(k)}{2}) + 1$. If the even set S contains the i-th vertex r'_i from R', then S must include the corresponding vertex r''_i from R''. Otherwise, b^i_{double} will have only one neighbour in S. Thus the even set S includes an even number of red vertices from the set $R_e \setminus \{r_{parity}\}$, which implies that the vertex r_{parity} must also be in S. Note that r_{parity} is a neighbour of each blue vertex in $B_e \setminus B_{double}$. Therefore, inclusion of r_{parity} into S transforms the EXACT EVEN SET problem on G_e into the EXACT ODD SET problem on the subgraph G contained in G_e. The argument described in the proof of Theorem 5.2 now applies directly. ∎

References

[1] J. Buss and T. Islam, "Simplifying the Weft Hierarchy," *Theoretical Computer Science*, in press, 2005.

[2] L. Cai, J. Chen, R.G. Downey, M.R. Fellows, "On the Complexity of Short Computation and Factorization," *Archiv Math. Logic*.

[3] M. Cesati, "The Turing Way to Parameterized Complexity," *Journal of Computer and System Sciences* 67, 4 (2003) 654–685.

[4] Y. Chen and J. Flum, "Machine Characterization of the Classes of the W-hierarchy," in *Computer Science Logic: CSL 2003*, Lecture Notes in Computer Science, Springer, 2003, pp. 114–127.

[5] Y. Chen, J. Flum and M. Grohe, "Bounded Nondeterminism and Alternation in Parameterized Complexity Theory," in *18th Ann. IEEE Conf. Computational Complexity*, 2003, pp. 18–29.

[6] R.G. Downey and M.R. Fellows,"Fixed-Parameter Tractability and Completeness I: Basic Results," *SIAM Journal on Computing* 24, 4 (1995) 873 - 921.

[7] R.G. Downey and M.R. Fellows,"Fixed-Parameter Tractability and Completeness II: On Completeness for $W[1]$," *Theoretical Computer Science* 141, 1-2 (1995) 109 - 131.

[8] R.G. Downey, M.R. Fellows, A. Vardy and G. Whittle,"The parameterized complexity of some fundamental problems in coding theory," *SIAM Journal on Computing* 29, 2 (1999) 545 - 570.

[9] R.G. Downey and M.R. Fellows, *Parameterized Complexity*, Springer, New York, 1999.

[10] J. Flum and M. Grohe, "Fixed-Parameter Tractability, Definability, and Model-Checking," *SIAM J. Computing* 31,1 (2001) 113–145.

[11] M.R. Fellows, and N. Koblitz, "Fixed-Parameter Complexity and Cryptography," *Proceedings of the Tenth International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC'93)*, Lecture Notes in Computer Science 673, Springer-Verlag (1993) 121–131.

[12] H.T. Wareham, "The Parameterized Complexity of Intersection and Composition Operations on Sets of Finite-State Automata," *International Conference on Implementation and Application of Automata*, Lecture Notes in Computer Science 2088, Springer (2000) 302–310.