Journal of Computer and System Sciences 76 (2010) 267–277

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

An approximation trichotomy for Boolean #CSP ☆

Martin Dyer [a], Leslie Ann Goldberg [b], Mark Jerrum [c,*]

[a] School of Computing, University of Leeds, Leeds LS2 9JT, UK
[b] Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK
[c] School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London E1 4NS, UK

ARTICLE INFO

Article history:
Received 13 February 2008
Received in revised form 3 June 2009
Available online 22 August 2009

Keywords:
Approximation algorithms
Combinatorial enumeration
Computational complexity
Constraint satisfaction problems (CSPs)

ABSTRACT

We give a trichotomy theorem for the complexity of approximately counting the number of satisfying assignments of a Boolean CSP instance. Such problems are parameterised by a constraint language specifying the relations that may be used in constraints. If every relation in the constraint language is affine then the number of satisfying assignments can be exactly counted in polynomial time. Otherwise, if every relation in the constraint language is in the co-clone IM_2 from Post's lattice, then the problem of counting satisfying assignments is complete with respect to approximation-preserving reductions for the complexity class $\#\mathrm{RH}\Pi_1$. This means that the problem of approximately counting satisfying assignments of such a CSP instance is equivalent in complexity to several other known counting problems, including the problem of approximately counting the number of independent sets in a bipartite graph. For every other fixed constraint language, the problem is complete for #P with respect to approximation-preserving reductions, meaning that there is no *fully polynomial randomised approximation scheme* for counting satisfying assignments unless NP = RP.

1. Introduction

This paper gives a trichotomy theorem for the complexity of approximately counting the number of satisfying assignments of a Boolean CSP instance. Such problems are parameterised by a constraint language Γ which specifies relations that may be used in constraints. In the Boolean case, the relations are on a domain which has two elements. Then $\#\mathrm{CSP}(\Gamma)$ will denote the problem of determining the number of (distinct) satisfying assignments of a CSP instance with constraint language Γ. Further details are given in Section 1.1 below.

Creignou and Hermann [6] have given a dichotomy theorem for the *exact* counting problem. They have shown that if every relation in Γ is affine, then $\#\mathrm{CSP}(\Gamma)$ is in FP. Otherwise, it is #P-complete. The complexity classes FP and #P are the analogues of P and NP for counting problems. FP, is the class of functions computable in deterministic polynomial time. #P is the class of integer functions that can be expressed as the number of accepting computations of a polynomial-time non-deterministic Turing machine.

In this paper we build on previous work on the complexity of approximate counting to identify a trichotomy in the complexity of approximate counting for Boolean #CSP.

Together with Greenhill [9], we have previously studied approximation-preserving reductions (AP-reductions) between counting problems. We will give details of AP-reductions in Section 1.2. For now it suffices to note that if an AP-reduction

☆ This work was partially supported by the EPSRC grant "The Complexity of Counting in Constraint Satisfaction Problems".
* Corresponding author.
E-mail address: M.Jerrum@qmul.ac.uk (M. Jerrum).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2009.08.003

exists from a counting problem f to a counting problem g and g has a *Fully Polynomial Randomised Approximation Scheme* (FPRAS) then f also has an FPRAS.

If an AP-reduction from f to g exists we write $f \leqslant_{\mathrm{AP}} g$, and say that f *is AP-reducible to* g. If $f \leqslant_{\mathrm{AP}} g$ and $g \leqslant_{\mathrm{AP}} f$ then we say that f *and* g *are AP-interreducible*, and write $f =_{\mathrm{AP}} g$.

We previously identified [9] three natural classes of counting problems that are interreducible under AP-reductions. These are (i) those problems that have an FPRAS, (ii) those problems that are complete for #P with respect to AP-reducibility, and a third class of intermediate complexity. Two counting problems played a special role in [9].

Name. #SAT.
Instance. A Boolean formula φ in conjunctive normal form.
Output. The number of satisfying assignments of φ.

Name. #BIS.
Instance. A bipartite graph B.
Output. The number of independent sets in B.

All problems in #P are AP-reducible to #SAT (see [9, Section 3]). Thus #SAT is complete for #P with respect to AP-reducibility. This means that #SAT cannot have an FPRAS unless NP = RP. The same is true of any problem in #P to which #SAT is AP-reducible.

We showed in [9, Sections 4, 5] that #BIS is AP-interreducible with many other natural counting problems such as counting downsets in a partial order. Moreover, #BIS is complete for $\#\mathrm{RH}\Pi_1$, a logically-defined subclass of #P, with respect to AP-reductions.

The main theorem of our current paper (Theorem 3) shows that every problem $\#\mathsf{CSP}(\Gamma)$ falls neatly into one of the three classes from [9]: If every relation in Γ is affine, then trivially $\#\mathsf{CSP}(\Gamma)$ has an FPRAS since it is in FP. Otherwise, if every relation in Γ is in a certain set IM_2, then $\#\mathsf{CSP}(\Gamma) =_{\mathrm{AP}} \#\mathsf{BIS}$. Otherwise $\#\mathsf{CSP}(\Gamma) =_{\mathrm{AP}} \#\mathsf{SAT}$. A formal definition of IM_2 appears in Section 1.4 — it is the set of relations which can be expressed as conjunctions involving only binary implication and unary relations.

It is worth pointing out that, while every problem $\#\mathsf{CSP}(\Gamma)$ falls into one of the three approximation classes from [9], the three classes may well not provide a partition of all approximate counting problems in #P. For example, the problem of approximately counting 3-colourings of a bipartite graph is a problem that may well lie between #BIS and #SAT in approximability (see [9]).

1.1. Constraint satisfaction

Constraint Satisfaction, which originated in Artificial Intelligence, provides a general framework for modelling decision problems, and has many practical applications. (See, for example [18].) Decisions are modelled by *variables*, which are subject to *constraints*, modelling logical and resource restrictions. The paradigm is sufficiently broad that many interesting problems can be modelled, from satisfiability problems to scheduling problems and graph-theory problems. Understanding the complexity of constraint satisfaction problems has become a major and active area within computational complexity [7,11].

A Constraint Satisfaction Problem (CSP) typically has a finite *domain*, which we denote by $\{0, \ldots, q-1\}$ for a positive integer q. In this paper we are interested in the *Boolean* case $q = 2$. A *constraint language* Γ with domain $\{0, \ldots, q-1\}$ is a set of relations on $\{0, \ldots, q-1\}$. For example, take $q = 2$. The relation $R = \{(0,0,1), (0,1,0), (1,0,0), (1,1,1)\}$ is a 3-ary relation on the domain $\{0,1\}$, with four tuples.

Once we have fixed a constraint language Γ, an *instance* of the CSP is a set of *variables* $V = \{v_1, \ldots, v_n\}$ and a set of *constraints*. Each constraint has a *scope,* which is a tuple of variables (for example, (v_4, v_5, v_1)) and a relation from Γ of the same arity, which constrains the variables in the scope. An *assignment* σ is a function from V to $\{0, \ldots, q-1\}$. The assignment σ is *satisfying* if the scope of every constraint is mapped to a tuple that is in the corresponding relation. In our example above, an assignment σ satisfies the constraint with scope (v_4, v_5, v_1) and relation R, written $R(v_4, v_5, v_1)$, if and only if it maps an odd number of the variables in $\{v_1, v_4, v_5\}$ to the value 1. Given an instance I of a CSP with constraint language Γ, the *decision problem* $\mathsf{CSP}(\Gamma)$ asks us to determine whether any assignment satisfies I. The *counting problem* $\#\mathsf{CSP}(\Gamma)$ asks us to determine the *number* of (distinct) satisfying assignments of I, which we will denote by $\#\mathsf{csp}(I)$.

Varying the constraint language Γ defines the classes CSP and #CSP of decision and counting problems. These contain problems of different computational complexities. For example, consider the binary relations defined by $\mathsf{OR} = \{(0,1), (1,0), (1,1)\}$, $\mathsf{Implies} = \{(0,0), (0,1), (1,1)\}$, and $\mathsf{NAND} = \{(0,0), (0,1), (1,0)\}$. If $\Gamma = \{\mathsf{OR}, \mathsf{Implies}, \mathsf{NAND}\}$ then $\mathsf{CSP}(\Gamma)$ is the classical 2-Satisfiability problem, which is in P. On the other hand, there is a similar constraint language Γ' with four relations of arity 3 such that 3-Satisfiability (which is NP-complete) can be represented in $\mathsf{CSP}(\Gamma')$. It may happen, as here, that the counting problem is harder than the decision problem: $\#\mathsf{CSP}(\Gamma)$ contains the problem of counting independent sets in graph, and is thus #P-complete.

Any decision problem $\mathsf{CSP}(\Gamma)$ is in NP, but not every problem in NP can be represented as a CSP. For example, the question "Is G Hamiltonian?" cannot be expressed as a CSP, because the property of being Hamiltonian cannot be captured

by relations of bounded size. This limitation of the class CSP has an important advantage. If P ≠ NP, then there are problems which are neither in P nor NP-complete [15]. But, for well-behaved smaller classes of decision problems, the situation can be simpler. We may have a *dichotomy theorem*, partitioning all problems in the class into those which are in P and those which are NP-complete. There are no "leftover" problems of intermediate complexity. It has been conjectured that there is a dichotomy theorem for CSP. The conjecture is that $\mathsf{CSP}(\Gamma)$ is in P for some constraint languages Γ, and $\mathsf{CSP}(\Gamma)$ is NP-complete for all other constraint languages Γ. This conjecture appeared in a seminal paper of Feder and Vardi [13], but has not yet been proved. A similar dichotomy, between FP and #P-complete, is conjectured for #CSP [4]. Recently, Bulatov [3] has announced a positive resolution of this conjecture.

There have been many important results for subclasses of CSP and #CSP. We mention the most relevant to our paper here. The first decision dichotomy was that of Schaefer [19], for the Boolean domain $\{0,1\}$. Schaefer's result is as follows.

Theorem 1. *(Schaefer [19].) Let Γ be a constraint language with domain $\{0,1\}$. The problem $\mathsf{CSP}(\Gamma)$ is in P if Γ satisfies one of the conditions below. Otherwise, $\mathsf{CSP}(\Gamma)$ is NP-complete.*

(i) *Γ is 0-valid or 1-valid.*
(ii) *Γ is weakly positive or weakly negative.*
(iii) *Γ is affine.*
(iv) *Γ is bijunctive.*

We will not give detailed definitions of the conditions in Theorem 1, but the interested reader is referred to the paper [19] or to Theorem 6.2 of the textbook [7]. An interesting feature is that the conditions in [7, Theorem 6.2] are all checkable. That is, there is an algorithm to determine whether $\mathsf{CSP}(\Gamma)$ is in P or NP-complete, given a constraint language Γ with domain $\{0,1\}$. We say in this case that the dichotomy is *effective*.

A Boolean relation R is said to be *affine* if the set of tuples $x \in R$ is the set of solutions to a system of linear equations over GF(2). Creignou and Hermann [6] adapted Schaefer's decision dichotomy to obtain a counting dichotomy for the Boolean domain. Their result is as follows.

Theorem 2. *(Creignou and Hermann [6].) Let Γ be a constraint language with domain $\{0,1\}$. The problem $\#\mathsf{CSP}(\Gamma)$ is in FP if every relation in Γ is affine. Otherwise, $\#\mathsf{CSP}(\Gamma)$ is #P-complete.*

Creignou and Hermann's result is an important starting point for our work, and we will discuss it further below. Note that there is an algorithm for determining whether a relation is affine, so the dichotomy is effective.

We have recently [10] extended Creignou and Hermann's dichotomy to the domain of *weighted* Boolean #CSP giving an effective dichotomy between FP and $\mathsf{FP}^{\#\mathsf{P}}$ for the problem of computing the partition function of a weighted Boolean CSP instance.

1.2. The complexity of approximate counting

We now recall the necessary background from [9]. A *randomised approximation scheme* is an algorithm for approximately computing the value of a function $f : \Sigma^* \to \mathbb{N}$. The approximation scheme has a parameter $\varepsilon > 0$ which specifies the error tolerance. A *randomised approximation scheme* for f is a randomised algorithm that takes as input an instance $x \in \Sigma^*$ (e.g., an encoding of a CSP instance) and an error tolerance $\varepsilon > 0$, and outputs an integer z (a random variable on the "coin tosses" made by the algorithm) such that, for every instance x,

$$\Pr\bigl[e^{-\varepsilon} f(x) \leqslant z \leqslant e^{\varepsilon} f(x)\bigr] \geqslant \frac{3}{4}. \tag{1}$$

The randomised approximation scheme is said to be a *fully polynomial randomised approximation scheme*, or *FPRAS*, if it runs in time bounded by a polynomial in $|x|$ and ε^{-1}. (See Mitzenmacher and Upfal [16, Definition 10.2].) Note that the quantity 3/4 in Eq. (1) could be changed to any value in the open interval $(\frac{1}{2}, 1)$ without changing the set of problems that have randomised approximation schemes [14, Lemma 6.1].

Suppose that f and g are functions from Σ^* to $\mathbb{N}$. An "approximation-preserving reduction" (AP-reduction) from f to g gives a way to turn an FPRAS for g into an FPRAS for f. An AP-reduction from f to g is a randomised algorithm $\mathcal{A}$ for computing f using an oracle for g.[1] The algorithm $\mathcal{A}$ takes as input a pair $(x, \varepsilon) \in \Sigma^* \times (0,1)$, and satisfies the following three conditions: (i) every oracle call made by $\mathcal{A}$ is of the form (w, δ), where $w \in \Sigma^*$ is an instance of g, and $0 < \delta < 1$ is an error bound satisfying $\delta^{-1} \leqslant \mathrm{poly}(|x|, \varepsilon^{-1})$; (ii) the algorithm $\mathcal{A}$ meets the specification for being a randomised approximation scheme for f (as described above) whenever the oracle meets the specification for being a randomised approximation scheme for g; and (iii) the run-time of $\mathcal{A}$ is polynomial in $|x|$ and ε^{-1}. In formulating a definition of

[1] The reader who is not familiar with oracle Turing machines can just think of this as an imaginary (unwritten) subroutine for computing g.

approximation-preserving reduction, a number of choices must be faced. The key requirement is that the class of functions computable by an FPRAS should be closed under AP-reducibility. Informally, we have gone for the most liberal notion of reduction meeting this requirement.

1.3. Notation for relations

Define the unary relations $\delta_0 = \{(0)\}$ and $\delta_1 = \{(1)\}$. Recall the binary relation $\mathsf{Implies} = \{(0,0),(0,1),(1,1)\}$.

For convenience, according to context, we view a k-ary relation R either as a set of k-tuples or as a k-ary predicate. Thus the notations $R(x_1,\ldots,x_k) = 1$ (or just $R(x_1,\ldots,x_k)$) and $(x_1,\ldots,x_k) \in R$ are equivalent. For example, $\delta_0(x) = \bar{x}$, $\delta_1(x) = x$ and $\mathsf{Implies}(x,y) = \bar{x} \vee y$.

1.4. The set of relations IM_2

An n-ary relation R is in IM_2 if and only if $R(x_1,\ldots,x_n)$ is logically equivalent to a conjunction of predicates of the form $\delta_0(x_i)$, $\delta_1(x_i)$ and $\mathsf{Implies}(x_i,x_j)$.

As we will discuss below, Creignou, Kolaitis, and Zanuttini [8] have shown that IM_2 is a co-clone in Post's lattice (see [2]).

1.5. Our result

We can now state our main theorem.

Theorem 3. *Let Γ be a constraint language with domain $\{0,1\}$. If every relation in Γ is affine then $\#\mathsf{CSP}(\Gamma)$ is in FP. Otherwise if every relation in Γ is in IM_2 then $\#\mathsf{CSP}(\Gamma) =_{\mathrm{AP}} \#\mathsf{BIS}$. Otherwise $\#\mathsf{CSP}(\Gamma) =_{\mathrm{AP}} \#\mathsf{SAT}$.*

The main ingredients in the proof are: (1) the AP-reduction technology of [9], which allows us to effectively "pin" certain CSP variables in hardness proofs (see Section 2.3); (2) the "implementations" of Creignou, Khanna and Sudan [7], which show how to construct the key relations OR, Implies, and NAND from a non-affine relation and δ_0 or δ_1 (see Section 2.5); (3) the complexity class $\#\mathrm{RH}\Pi_1$ from [9], consisting of those problems which are AP-interreducible with #BIS; and (4) the co-clone IM_2 in Post's lattice (see Section 2.8), since the complexity of $\#\mathsf{CSP}(\Gamma)$ for $\Gamma \subseteq IM_2$ turns out to be closely connected to the complexity of #BIS.

2. The pieces of the proof

2.1. Types of relations

A relation R is *0-valid* if the all-zero tuple is in R. Similarly, R is *1-valid* if the all-ones tuple is in R. Following [7], we say that a k-ary relation R is *complement-closed* (C-closed in [7]) if

$$(x_1,\ldots,x_k) \in R \quad \Leftrightarrow \quad (x_1 \oplus 1,\ldots,x_k \oplus 1) \in R,$$

where $\oplus$ is the exclusive or operator.

We say that Γ is 0-valid if every $R \in \Gamma$ is 0-valid and we define what it means for Γ to be 1-valid or complement-closed similarly.

2.2. Some preliminary complexity results

We start by observing that every problem $\#\mathsf{CSP}(\Gamma)$ is AP-reducible to #SAT.

Observation 4. Let Γ be a constraint language with domain $\{0,1\}$. Then $\#\mathsf{CSP}(\Gamma) \leqslant_{\mathrm{AP}} \#\mathsf{SAT}$.

Observation 4 follows from the fact that all problems in #P are AP-reducible to #SAT [9]. Another, very simple, but useful, observation is the following.

Observation 5. Let Γ be a constraint language with domain $\{0,1\}$. Suppose $\Gamma' \subseteq \Gamma$. Then $\#\mathsf{CSP}(\Gamma') \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$.

Observation 5 is true for the simple reason that every instance of $\#\mathsf{CSP}(\Gamma')$ *is* an instance of $\#\mathsf{CSP}(\Gamma)$.

Recall the relations $\mathsf{OR} = \{(0,1),(1,0),(1,1)\}$ and $\mathsf{NAND} = \{(0,0),(0,1),(1,0)\}$. These relations are particularly fundamental for us, and we start with complexity results about these.

Lemma 6. $\#\mathsf{SAT} \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{\mathsf{NAND}\})$.

Proof. It was shown in [9] that the following problem is AP-interreducible with #SAT.

Name. #IS.
Instance. A graph G.
Output. The number of independent sets in G.

We show that #IS $\leqslant_{\mathrm{AP}}$ #CSP({NAND}). Let $G = (V, E)$ be an instance of #IS. Construct an instance I of #CSP({NAND}) with variable set V. For every edge $(u, v) \in E$, add constraint NAND(u, v). There is now a bijection between independent sets of G and satisfying assignments σ of I: variables v with $\sigma(v) = 1$ correspond to vertices in the independent set. □

Lemma 7. #SAT $\leqslant_{\mathrm{AP}}$ #CSP({OR}).

Proof. The proof that #IS $\leqslant_{\mathrm{AP}}$ #CSP({OR}) is similar (just associate variables v with $\sigma(v) = 1$ with vertices that are *out* of the independent set). □

Finally, we will need a couple of complexity results involving #BIS.

Lemma 8. #BIS $\leqslant_{\mathrm{AP}}$ #CSP({Implies}).

Proof. Let G be an instance of #BIS with vertex sets U and V and edge set E. Construct an instance I of #CSP({Implies}) with variable set $U \cup V$. For every edge $(u, v) \in E$ with $u \in U$ add constraint Implies(u, v). There is now a bijection between independent sets of G and satisfying assignments σ of I: a variable $u \in U$ with $\sigma(u) = 1$ is in the independent set and a variable $v \in V$ with $\sigma(v) = 0$ is in the independent set. □

Lemma 9. *Suppose* $\Gamma \subseteq IM_2$. *Then* #CSP$(\Gamma) \leqslant_{\mathrm{AP}}$ #BIS.

Proof. It is straightforward to show that #CSP(Γ) is in the complexity class #RHΠ_1 which has #BIS as a complete problem [9].

However, to avoid giving a definition of #RHΠ_1, which requires some notation, we will instead show #CSP$(\Gamma) \leqslant_{\mathrm{AP}}$ #Downsets, where #Downsets is the following counting problem which was shown in [9] to be AP-interreducible with #BIS.

Name. #Downsets.
Instance. A partially ordered set $(X, \preccurlyeq)$.
Output. The number of downsets[2] in $(X, \preccurlyeq)$.

Consider an instance I of #CSP(Γ) with variables $v_1, \ldots, v_n$. The set of constraints can be viewed as an equivalent set of constraints of the form $\delta_0(v_i)$, $\delta_1(v_i)$ or Implies(v_i, v_j). Denote by Implies* the transitive closure of the Implies relation on $\{v_1, \ldots, v_n\}$: thus Implies$^*(v_i, v_j)$ if there is a sequence of variables, starting with v_i and ending with v_j, such that every adjacent pair in the sequence is constrained by Implies.

Let $N_0(I)$ be the set of variables v_i for which either (i) a constraint $\delta_0(v_i)$ occurs in I, or (ii) there exists a variable v_j such that Implies$^*(v_i, v_j)$ and a constraint $\delta_0(v_j)$ occurs in I. These are the variables that are forced to be 0 in any satisfying assignment of I. Define $N_1(I)$ analogously to be the set of variables that are forced to be 1 in any satisfying assignment. We can assume without loss of generality that $N_0(I)$ and $N_1(I)$ are disjoint. Otherwise the instance I has no satisfying assignments, and we can determine this without even using the downsets oracle.

Now remove all the variables in $N_0(I)$ and $N_1(I)$ from the instance I: this does not affect the number of satisfying assignments, since these variables do not constrain any of the others. Also identify all pairs of variables v_i, v_j such that Implies$^*(v_i, v_j)$ and Implies$^*(v_j, v_i)$: again, this does not affect the number of satisfying assignments.

The remaining variables and relations define a partial order $(X, \preccurlyeq)$ since our construction forces antisymmetry. The satisfying assignments of I correspond 1–1 with the downsets of $(X, \preccurlyeq)$. □

2.3. A useful tool: Pinning

Pinning is the ability to tie certain CSP variables to specific values in hardness proofs. This idea was used by Creignou and Hermann in their dichotomy theorem [6]. Similar ideas have been used in many other hardness proofs and dichotomy theorems [4,5,10,12]. As we show in this section, AP-reductions facilitate a particularly useful form of pinning.

[2] A *downset* in $(X, \preccurlyeq)$ is a subset $D \subseteq X$ that is closed under $\preccurlyeq$; i.e., $x \preccurlyeq y$ and $y \in D$ implies $x \in D$.

Lemma 10. *Let Γ be a constraint language with domain $\{0,1\}$. Suppose there is a relation $R \in \Gamma$ for which, for some position j, R has more tuples t with $t_j = 0$ than with $t_j = 1$. Then $\#\mathsf{CSP}(\Gamma \cup \{\delta_0\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$. Similarly, if there is a relation $R \in \Gamma$ for which, for some position j, R has more tuples t with $t_j = 1$ than with $t_j = 0$ then $\#\mathsf{CSP}(\Gamma \cup \{\delta_1\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$.*

Proof. Consider an instance I of $\#\mathsf{CSP}(\Gamma \cup \{\delta_0\})$ with n variables. Suppose there is an arity-k relation $R \in \Gamma$ for which, for position j, R has w tuples t with $t_j = 0$ and $w' < w$ tuples t with $t_j = 1$.

As in the proof of Lemma 9, let $N_0(I)$ be the set of variables x to which one or more constraints $\delta_0(x)$ occurs in I and let $N_1(I)$ be the set of variables y to which one or more constraints $\delta_1(y)$ occurs. Let $n_0 = |N_0(I)|$. Let $m = \lceil (n+2)/\lg(w/w') \rceil$. Construct an instance I' of $\#\mathsf{CSP}(\Gamma)$. Include all constraints in I other than those involving δ_0. For each variable $x \in N_0(I)$, and every $a \in \{1,\ldots,m\}$, introduce $k-1$ new variables $x'_{a,b}$ for $b \in \{1,\ldots,k\} - \{j\}$. Introduce a new constraint in I' with relation R and variable x in the jth position, and $x'_{a,b}$ in the bth position, for all b.

Now a satisfying assignment for I can be extended in w^{mn_0} ways to satisfying assignments of I'. An assignment for I that violates one of the $\delta_0(x)$ constraints can be extended in at most $w^{m(n_0-1)}w'^m$ ways to satisfying assignments of I'. Thus,

$$\#\mathsf{csp}(I)w^{mn_0} \leqslant \#\mathsf{csp}(I') \leqslant \#\mathsf{csp}(I)w^{mn_0} + 2^n w^{m(n_0-1)}w'^m,$$

i.e.,

$$\#\mathsf{csp}(I) \leqslant \frac{\#\mathsf{csp}(I')}{w^{mn_0}} \leqslant \#\mathsf{csp}(I) + 2^n(w'/w)^m.$$

So, by definition of m,

$$\#\mathsf{csp}(I) \leqslant \frac{\#\mathsf{csp}(I')}{w^{mn_0}} \leqslant \#\mathsf{csp}(I) + \frac{1}{4}.$$

Thus we have constructed a reduction from $\#\mathsf{CSP}(\Gamma \cup \{\delta_0\})$ to $\#\mathsf{CSP}(\Gamma)$: Given an instance I of $\#\mathsf{CSP}(\Gamma \cup \{\delta_0\})$, use an oracle for $\#\mathsf{CSP}(\Gamma)$ to approximate $\#\mathsf{csp}(I')$, divide by w^{mn_0}, and round to the nearest integer (always down). Note that the reduction makes only one oracle call (and uses no randomisation).

To show that the reduction is indeed an AP-reduction, we add some technical details concerning the choice of the accuracy parameter δ in the oracle call (see the definition of AP-reduction in Section 1.2). These details are here to make the proof complete, but they are not essential for understanding the rest of the paper.

If we had

$$\#\mathsf{csp}(I) = \frac{\#\mathsf{csp}(I')}{w^{mn_0}},$$

we could simply set $\delta = \varepsilon$, since division by a constant preserves relative error. Instead we have

$$\#\mathsf{csp}(I) = \left\lfloor \frac{\#\mathsf{csp}(I')}{w^{mn_0}} \right\rfloor.$$

The discontinuous floor function could spoil the approximation when its argument is small.

The situation here is that the true answer $N = \#\mathsf{csp}(I)$ is obtained by rounding the fraction $Q = \frac{\#\mathsf{csp}(I')}{w^{mn_0}}$ where we have $|Q - N| \leqslant 1/4$.

Suppose that the oracle provides an approximation $\widehat{Q}$ to Q satisfying $Qe^{-\delta} \leqslant \widehat{Q} \leqslant Qe^{\delta}$ (as it is required to do with probability at least $3/4$). Set $\delta = \varepsilon/21$, where ε is the accuracy parameter governing the final result. There are two cases. If $N \leqslant 2/\varepsilon$, then a short calculation yields $|\widehat{Q} - Q| < 1/4$ implying that the result returned by the algorithm is exact. If $N > 2/\varepsilon$, then the result returned is in the range $[(N-1/4)e^{-\delta} - 1/2, (N+1/4)e^{\delta} + 1/2]$ which, for the chosen δ, is contained in $[Ne^{-\varepsilon}, Ne^{\varepsilon}]$.

Thus, we have an AP-reduction from $\#\mathsf{CSP}(\Gamma \cup \{\delta_0\})$ to $\#\mathsf{CSP}(\Gamma)$. The reduction showing $\#\mathsf{CSP}(\Gamma \cup \{\delta_1\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$ is similar. □

2.4. Affine relations

We use the following well-known facts about affine relations.

Lemma 11.

(i) *A k-ary Boolean relation R is affine if and only if $a, b, c \in R$ implies $d = a \oplus b \oplus c \in R$, where the $\oplus$ operator is applied componentwise.*
(ii) *If R is not affine, then for any fixed $a \in R$ there are $b, c \in R$ such that $a \oplus b \oplus c \notin R$.*
(iii) *If R is not affine, then there are a, b in R such that $a \oplus b \notin R$.*

Proof. For Part (i) see, for example, Lemma 4.10 of [7]. Part (ii) is proved in the same place, but since it is a little less well known, we provide the proof: Suppose the contrary that R is not affine, but for all $b, c \in R$, $a \oplus b \oplus c \in R$. Choose $s_0, s_1, s_2 \in R$ such that $s_0 \oplus s_1 \oplus s_2 \notin R$. From $b = s_0$, $c = s_1$, $d = a \oplus s_0 \oplus s_1$ we have $d \in R$. From $b = s_2$, $c = d$ we have $a \oplus s_2 \oplus d = s_0 \oplus s_1 \oplus s_2 \in R$, a contradiction.

To see Part (iii), note that the condition "$\forall a, b$: $a, b \in R$ implies $a \oplus b \in R$" implies that R is affine, so, if R is not affine then the condition is false. □

2.5. Implementation

Let Γ be a constraint language with domain $\{0, 1\}$. Γ is said to *implement*[3] a k-ary relation R if, for some $k' \geqslant k$ there is a CSP instance I with variables $x_1, \ldots, x_{k'}$ and constraints in Γ such that, for every tuple $(s_1, \ldots, s_k) \in R$, there is exactly one satisfying assignment σ of I with $\sigma(x_1) = s_1, \ldots, \sigma(x_k) = s_k$ and for every tuple $(s_1, \ldots, s_k) \notin R$, there are no satisfying assignments σ of I with $\sigma(x_1) = s_1, \ldots, \sigma(x_k) = s_k$. Note the following straightforward observation, which is essentially a parsimonious reduction [17, p. 441].

Observation 12. If Γ implements R then $\#\mathsf{CSP}(\Gamma \cup \{R\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$.

We will use several implementations of Creignou, Khanna and Sudan. Proofs are provided in Appendix A in order to make the paper self-contained.

Lemma 13. *(See Creignou, Khanna and Sudan [7, Lemmas 5.24 and 5.25].) Let Γ be a constraint language with domain* $\{0, 1\}$.

(i) *If Γ contains a relation R that is* 0*-valid,* 1*-valid and not complement-closed then Γ implements the relation $R' = \{(0, 0), (1, 1), (1, 0)\}$.*
(ii) *If Γ contains a relation R that is not* 0*-valid, not* 1*-valid and not complement-closed then Γ implements δ_0 and δ_1.*
(iii) *If Γ contains a relation R that is* 0*-valid and not* 1*-valid then Γ implements δ_0.*
(iv) *If Γ contains a relation R that is* 1*-valid and not* 0*-valid then Γ implements δ_1.*

Lemma 14. *(See Creignou, Khanna and Sudan [7, Claim 5.31].) Let R be a ternary relation containing* $(0, 0, 0)$, $(0, 1, 1)$ *and* $(1, 0, 1)$ *but not* $(1, 1, 0)$. *Then $\{R, \delta_0\}$ implements one of* Implies *and* NAND.

Lemma 15. *(See Creignou, Khanna and Sudan [7, Lemma 5.30].) If R is a relation over* $\{0, 1\}$ *that is not affine then $\{R, \delta_0\}$ implements one of* OR, Implies, *and* NAND *and so does $\{R, \delta_1\}$.*

2.6. Pinning revisited

Combining the useful pinning that we get from AP-reductions (Lemma 10) with the implementations of OR, Implies and NAND in Section 2.5, we obtain a useful lemma which says that we can *always* do some pinning.

Lemma 16. *Let Γ be a constraint language with domain* $\{0, 1\}$. *Then either* $\#\mathsf{CSP}(\Gamma \cup \{\delta_0\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$ *or* $\#\mathsf{CSP}(\Gamma \cup \{\delta_1\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$ *(or both).*

Proof. First, suppose that Γ is not complement-closed. If Γ contains a relation R that is not 0-valid, not 1-valid and not complement-closed then we finish by Observation 12 and Part (ii) of Lemma 13. If Γ contains a relation R that is 0-valid, 1-valid and not complement-closed then it implements the relation R' from Part (i) of Lemma 13 so by Observation 12, $\#\mathsf{CSP}(\Gamma \cup \{R'\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$. But Lemma 10 shows both $\#\mathsf{CSP}(\Gamma \cup \{R', \delta_0\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma \cup \{R'\})$ and $\#\mathsf{CSP}(\Gamma \cup \{R', \delta_1\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma \cup \{R'\})$. Otherwise Γ contains a relation R that is 0-valid and not 1-valid (or vice versa) and we finish by Part (iii) (or Part (iv)) of Lemma 13, and Observation 12.

Second (and finally), suppose that Γ is complement-closed. Here is a simple AP-reduction from $\#\mathsf{CSP}(\Gamma \cup \{\delta_0\})$ to $\#\mathsf{CSP}(\Gamma)$. Let I be an instance of $\#\mathsf{CSP}(\Gamma \cup \{\delta_0\})$. Construct an instance I' of $\#\mathsf{CSP}(\Gamma)$ by adding a new variable z_0. For all $x \in N_0(I)$ (all variables x to which one or more constraints $\delta_0(x)$ in I apply), replace all occurrences of variable x with z_0 in I'. Now note that $2\#\mathsf{csp}(I) = \#\mathsf{csp}(I')$ since there is a one-to-two map from satisfying assignments of I and satisfying assignments of I'. In particular, if s is an assignment to all variables of I other than those in $N_0(I)$ and s is satisfying, provided the rest of the variables are assigned value 0, then s is mapped to $s; z_0 = 0$ and $\bar{s}; z_0 = 1$, where $\bar{s}$ is the tuple obtained from s by complementing the assignment of every variable. Both satisfy I' since Γ is complement-closed. It is clear that all satisfying assignments of I' arise in this way. □

[3] There are many variants of "implement" defined in the literature. See [7, Chapter 5], where the kind of implementation we define here is called "faithful" and "perfect".

2.7. Notation for Boolean functions

The following definitions are from [1,2]. An m-ary Boolean function f is *monotonic* if and only if $(a_1, \ldots, a_m) \leqslant (b_1, \ldots, b_m)$ componentwise implies $f(a_1, \ldots, a_m) \leqslant f(b_1, \ldots, b_m)$. Let M_2 be the set of all monotone Boolean functions f satisfying $f(0, \ldots, 0) = 0$ and $f(1, \ldots, 1) = 1$. Given a set B of Boolean functions, the closure $[B]$ consists of all functions that can be defined by propositional formulas with connectives from B (see [1]).

An m-ary Boolean function f is said to be a *polymorphism* of an n-ary relation $R(x_1, \ldots, x_n)$ if applying f componentwise to m tuples in R results in a tuple that is also in R.

2.8. Polymorphisms and IM_2

In the terminology of universal algebra, Creignou, Kolaitis, and Zanuttini [8] have shown that IM_2 is precisely the co-clone corresponding to M_2, which is a clone in Post's lattice (see [2]). The direction of this result that we will use is the following.

Lemma 17. *(See Creignou, Kolaitis, Zanuttini [8].) If the relation R is not in IM_2 then there is an $f \in M_2$ that is not a polymorphism of R.*

Corollary 18. *If the n-ary relation R is not in IM_2 then there are Boolean tuples $(a_1, \ldots, a_n) \in R$ and $(b_1, \ldots, b_n) \in R$ such that either $(a_1 \wedge b_1, \ldots, a_n \wedge b_n) \notin R$ or $(a_1 \vee b_1, \ldots, a_n \vee b_n) \notin R$ (or both).*

Proof. We will use the fact (see [1]) that $M_2 = [\{\vee, \wedge\}]$ where $x \vee y$ is the OR of the Boolean values x and y and $x \wedge y$ is the AND of x and y. Thus, every function $f \in M_2$ can be defined by a propositional formula using the 2-ary connectives $\vee$ and $\wedge$.

The proof is by induction on the number of connectives used in the propositional formula used to represent the function f from Lemma 17.

The case $f(x) = x$ (in which f has no connectives) cannot arise since the identity function is a polymorphism of every relation. The cases $f(x, y) = x \vee y$ and $f(x, y) = x \wedge y$ (in which f has one connective) immediately give the corollary.

For the inductive step, we assume either $f(x_1, \ldots, x_m) = f'(x_1, \ldots, x_m) \vee f''(x_1, \ldots, x_m)$ or $f(x_1, \ldots, x_m) = f'(x_1, \ldots, x_m) \wedge f''(x_1, \ldots, x_m)$ where f' and f'' have fewer connectives than f. Note that f' and f'' may not actually use all of the variables in $x_1, \ldots, x_m$.

These two cases are similar, so suppose we are in the first of them. That is, suppose

$$f(x_1, \ldots, x_m) = f'(x_1, \ldots, x_m) \vee f''(x_1, \ldots, x_m).$$

Suppose also that f' and f'' are polymorphisms of R (otherwise we will apply the inductive hypothesis to one of these functions which has fewer connectives). Let $t^1, \ldots, t^m$ be m n-tuples in R, such that the tuple obtained by applying f componentwise to $t^1, \ldots, t^m$ is not in R. Let t' be the n-tuple obtained by applying f' componentwise to $t^1, \ldots, t^m$ and let t'' be the n-tuple obtained by applying f'' componentwise to $t^1, \ldots, t^m$. Since f' and f'' are polymorphisms of R, we know that t' and t'' are in R. However, since f is not a polymorphism of R, the tuple $t' \vee t''$ is not in R, proving the corollary. □

3. Putting it all together: The proof of Theorem 3

We start with a lemma establishing a reduction from #SAT.

Lemma 19. *Let R_1 and R_2 be relations on $\{0, 1\}$. If R_1 is not affine and R_2 is not in IM_2 then* $\#\mathsf{SAT} \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{R_1, R_2\})$.

Proof. Apply Lemma 16 with $\Gamma = \{R_1, R_2\}$. Then either $\#\mathsf{CSP}(\{R_1, R_2, \delta_0\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{R_1, R_2\})$ or $\#\mathsf{CSP}(\{R_1, R_2, \delta_1\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{R_1, R_2\})$. Assume the former (the latter case is symmetric).

Now use Lemma 15 together with Observation 12. Since R_1 is not affine this shows one of the following.

- $\#\mathsf{CSP}(\{R_1, R_2, \delta_0, \mathsf{OR}\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{R_1, R_2, \delta_0\})$, or
- $\#\mathsf{CSP}(\{R_1, R_2, \delta_0, \mathsf{NAND}\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{R_1, R_2, \delta_0\})$, or
- $\#\mathsf{CSP}(\{R_1, R_2, \delta_0, \mathsf{Implies}\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{R_1, R_2, \delta_0\})$.

In the first two of these cases, we are finished by Observation 5 and Lemmas 6 and 7, so assume the final case. Using Lemma 10 with the second position of Implies, we get $\#\mathsf{CSP}(\{R_1, R_2, \delta_0, \mathsf{Implies}, \delta_1\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{R_1, R_2, \delta_0\})$.

Simplifying the chain of reductions and using Observation 5 to drop R_1 from the left-hand side, we get $\#\mathsf{CSP}(\{\mathsf{Implies}, R_2, \delta_0, \delta_1\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{R_1, R_2\})$. We will now finish by showing $\#\mathsf{SAT} \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\{\mathsf{Implies}, R_2, \delta_0, \delta_1\})$.

Case 1. Using Corollary 18, suppose that t and t' are tuples in R_2 but the tuple $t \wedge t'$ (in which the operator $\wedge$ is applied componentwise) is not in R_2. We will show that $\{\mathsf{Implies}, R_2, \delta_0, \delta_1\}$ implements one of OR and $\mathsf{XOR} = \{(0,1),(1,0)\}$. Let k be the arity of R_2. As in the implementations of Creignou et al. [7], define r_i to be u if $t_i = t'_i = 0$ or x if $t_i = 0$, $t'_i = 1$ or y if $t_i = 1$, $t'_i = 0$, or v if $t_i = t'_i = 1$. Let R' be the relation implemented by $R'(x,y) = R_2(r_1,\ldots,r_k) \wedge \delta_0(u) \wedge \delta_1(v)$. Note that both x and y appear as arguments of R' since $t \neq t \wedge t'$ and $t' \neq t \wedge t'$. If $t \vee t'$ is in R_2 then $R'(x,y)$ implements $\mathsf{OR}(x,y)$, so we are finished. Otherwise $R' = \mathsf{XOR}$ (which we now assume).

Using Observations 12 and 5, we have

$$\#\mathsf{CSP}\big(\{\mathsf{Implies}, \mathsf{XOR}\}\big) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}\big(\{R_1, R_2\}\big).$$

We will finish by showing that $\{\mathsf{Implies}, \mathsf{XOR}\}$ implements NAND. (The result then follows by Lemma 6 and Observation 12.)

The implementation is given by $\mathsf{NAND}(x,z) = \mathsf{Implies}(x,y) \wedge \mathsf{XOR}(y,z)$.

Case 2. Otherwise, by Corollary 18, there are t and t' in R_2 such that $t \vee t'$ is not in R_2. This case is dual to Case 1. □

We can now prove the main theorem.

Theorem 3. *Let Γ be a constraint language with domain $\{0,1\}$. If every relation in Γ is affine then $\#\mathsf{CSP}(\Gamma)$ is in FP. Otherwise if every relation in Γ is in IM_2 then $\#\mathsf{CSP}(\Gamma) =_{\mathrm{AP}} \#\mathsf{BIS}$. Otherwise $\#\mathsf{CSP}(\Gamma) =_{\mathrm{AP}} \#\mathsf{SAT}$.*

Proof. First, suppose that every relation in Γ is affine. In this case, the number of satisfying assignments of an instance I of $\#\mathsf{CSP}(\Gamma)$ is the number of solutions to a system of linear equations over GF(2). This can be computed exactly, by Gaussian elimination, in polynomial time, as Creignou and Hermann have noted [6].

Next, suppose that Γ contains a relation R that is not affine, but every relation in Γ is in IM_2. By Lemma 9, $\#\mathsf{CSP}(\Gamma) \leqslant_{\mathrm{AP}} \#\mathsf{BIS}$.

To see that $\#\mathsf{BIS} \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$, apply Lemma 16. Then we know that either $\#\mathsf{CSP}(\Gamma \cup \{\delta_0\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$ or $\#\mathsf{CSP}(\Gamma \cup \{\delta_1\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$ (or both). We will show

$$\#\mathsf{BIS} \leqslant_{\mathrm{AP}} \#\mathsf{CSP}\big(\Gamma \cup \{\delta_0\}\big) \tag{2}$$

and

$$\#\mathsf{BIS} \leqslant_{\mathrm{AP}} \#\mathsf{CSP}\big(\Gamma \cup \{\delta_1\}\big) \tag{3}$$

and then we will be able to conclude $\#\mathsf{BIS} \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$. The proofs of Eqs. (2) and (3) are similar, so we just prove (2). By Lemma 15, $\Gamma \cup \{\delta_0\}$ implements one of OR, Implies, and NAND. So by Observation 12 we have (at least) one of the following.

(i) $\#\mathsf{CSP}(\Gamma \cup \{\delta_0, \mathsf{OR}\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma \cup \{\delta_0\})$;
(ii) $\#\mathsf{CSP}(\Gamma \cup \{\delta_0, \mathsf{Implies}\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma \cup \{\delta_0\})$;
(iii) $\#\mathsf{CSP}(\Gamma \cup \{\delta_0, \mathsf{NAND}\}) \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma \cup \{\delta_0\})$.

Eq. (2) follows from the combination of Lemma 8 and (ii) using Observation 5. Also, since $\#\mathsf{BIS} \leqslant_{\mathrm{AP}} \#\mathsf{SAT}$ (see [9]), Eq. (2) follows from the combination of Lemma 7 and (i) using Observation 5. Similarly, it follows from the combination of Lemma 6 and (iii) using Observation 5.

Finally, suppose that Γ contains a relation R_1 that is not affine and a relation R_2 that is not in IM_2. (R_1 and R_2 might possibly be the same relation.) The fact that $\#\mathsf{CSP}(\Gamma) \leqslant_{\mathrm{AP}} \#\mathsf{SAT}$ follows from Observation 4 and the fact that $\#\mathsf{SAT} \leqslant_{\mathrm{AP}} \#\mathsf{CSP}(\Gamma)$ follows from Lemma 19 and Observation 5. □

Appendix A. The implementations of Creignou, Khanna and Sudan

In order to make our paper self-contained, we give the details of the implementations of Creignou, Khanna and Sudan that we use. In particular, we provide the proofs for Lemmas 13, 14 and 15. (These proofs can be found in [7].)

We start with the construction for Lemma 13. Suppose $R \in \Gamma$ is not complement-closed. Choose $(s_1,\ldots,s_k)$ in R such that $(s_1 \oplus 1,\ldots,s_k \oplus 1)$ is not in R. Now consider the relation R' implemented by $R'(x,y) = R(r_1,\ldots,r_k)$ where $r_i = x$ if $s_i = 1$ and $r_i = y$ otherwise. In the first case, R' is the relation $\{(0,0),(1,1),(1,0)\}$. In the second case, $R' = \{(1,0)\}$ so R' gives an implementation of both δ_1 and δ_0. The construction for the third and fourth cases are the trivial implementations $\delta_0(x) = R(x,\ldots,x)$ and $\delta_1(x) = R(x,\ldots,x)$.

We now give the construction for Lemma 14. If R excludes exactly one of $(0,1,0)$ and $(1,1,1)$ then $R(x,y,x)$ implements $\mathsf{Implies}(y,x)$ or $\mathsf{NAND}(x,y)$ (depending on which is excluded). Similarly, if R excludes exactly one of $(1,0,0)$ and $(1,1,1)$ then $R(x,y,y)$ implements $\mathsf{Implies}(x,y)$ or $\mathsf{NAND}(x,y)$. If both $(0,1,0)$ and $(1,0,0)$ are in R then $f_R(x,y,z) \wedge \delta_0(z)$ implements $f_{\mathsf{NAND}}(x,y)$. If $(0,1,0)$, $(1,1,1)$ and $(1,0,0)$ are excluded from R and so is $(0,0,1)$ then $R(x,y,z)$ implements

$\mathsf{NAND}(x, y)$. Finally, if $(0,1,0)$, $(1,1,1)$ and $(1,0,0)$ are excluded but $(0,0,1)$ is in R then $R(x,y,z) \wedge \delta_0(x)$ implements $\mathsf{Implies}(y,z)$.

Finally, we give the construction for Lemma 15. We will show that $\{R, \delta_0\}$ implements one of the named relations. A similar argument shows that $\{R, \delta_1\}$ does. Let k be the arity of R.

First, suppose that R is 0-valid. Using Part (iii) of Lemma 11, let s and s' be tuples in R such that $s \oplus s'$ is not in R. Let $r_i = w$ if $s_i = s'_i = 0$. Let $r_i = x$ if $s_i = 0, s'_i = 1$. Let $r_i = y$ if $s_i = 1, s'_i = 0$. Let $r_i = z$ if $s_i = s'_i = 1$. Now we know that at least one of x and y occurs as an r_i, since $s \neq s'$. Let R' be the relation implemented by $R(r_1, \ldots, r_k) \wedge \delta_0(w)$. There are a few cases to consider. If x occurs as an argument to R but y does not then z occurs since $s \neq 0$. Thus, the relation $R'(x,z)$ is $\mathsf{Implies}$. (Technically, this is a ternary relation in variables x, y and z, but it can be viewed as a binary relation since y does not appear.) The situation is similar if y occurs as an argument to R but x does not. If both x and y occur as arguments but z does not then the relation $R'(x,y)$ is NAND. Otherwise, x, y and z all occur as arguments. Furthermore, since R is 0-valid, lemma 14 applies to the relation given by $R'(x,y,z)$.

Second (and finally), suppose that R is not 0-valid. Note that $\{R, \delta_0\}$ can implement δ_1. To see this, let s be a tuple in R. Let $r_i = x$ if $s_i = 1$ and let $r_i = y$ otherwise. Then $\delta_1(x)$ is implemented by $R(r_1, \ldots, r_k) \wedge \delta_0(y)$. Now consider two sub-cases.

For the first sub-case, suppose that for any two tuples, t and t', in R, the tuple $t \wedge t'$, where $\wedge$ is applied componentwise, is also in R. Let s be the intersection of all tuples in R. Then $s \in R$. By Part (ii) of Lemma 11, there are two tuples s' and s'' in R such that $s \oplus s' \oplus s''$ is not in R. Let $r_i = u$ if $s_i = s'_i = s''_i = 0$. Let $r_i = x$ if $s_i = 0, s'_i = 0, s''_i = 1$. Let $r_i = y$ if $s_i = 0, s'_i = 1, s''_i = 0$. Let $r_i = z$ if $s_i = 0, s'_i = 1, s''_i = 1$. Let $r_i = v$ if $s_i = s'_i = s''_i = 1$. Let R' be the relation implemented by $R(r_1, \ldots, r_k) \wedge \delta_0(u) \wedge \delta_1(v)$. If y does not occur as an argument of R' then $R'(x,z)$ implements $\mathsf{Implies}$. Similarly, if x does not occur as an argument of R' then $R'(y,z)$ implements $\mathsf{Implies}$. If z does not occur as an argument of R' then $R'(x,y)$ implements NAND. So we assume that x, y and z occur as arguments. Then apply Lemma 14 to $R'(x,y,z)$.

For the final sub-case, suppose that there are tuples t and t' in R such that $t \wedge t'$ is not in R. Define r_i to be u if $t_i = t'_i = 0$ or x if $t_i = 0, t'_i = 1$ or y if $t_i = 1, t'_i = 0$, or v if $t_i = t'_i = 1$. Let R' be the relation implemented by $R'(x,y) = R(r_1, \ldots, r_k) \wedge \delta_0(u) \wedge \delta_1(v)$. If $t \vee t'$ is in R then $R'(x,y)$ implements $\mathsf{OR}(x,y)$, so we are finished. Otherwise $R' = \{(0,1),(1,0)\}$ (which we now assume).

Now using Part (i) of Lemma 11, let s, s' and s'' be tuples in R so that $s \oplus s' \oplus s''$ is not in R. Define r_i as follows.

s_i	s'_i	s''_i	r_i
0	0	0	u
0	0	1	x
0	1	0	y
0	1	1	z
1	0	0	z'
1	0	1	y'
1	1	0	x'
1	1	1	u'

Let R'' be the relation implemented by

$$R(r_1, \ldots, r_k) \wedge \delta_0(u) \wedge R'(u,u') \wedge R'(x,x') \wedge R'(y,y') \wedge R'(z,z').$$

By writing $x' = \bar{x}$, $y' = \bar{y}$ and $z' = \bar{z}$, we can think of R'' as a function of x, y and z. If x does not occur as an argument then $R''(y,z)$ implements $\mathsf{Implies}(y,z)$. Similarly, we can assume that y and z occur as arguments. Now consider the relation $R''(x,y,z)$. We know that $(0,0,0),(0,1,1),(1,0,1) \in R''$, since $s, s', s'' \in R$. Also $(1,1,0) \notin R''$ since $s \oplus s' \oplus s'' \notin R$. Then apply Lemma 14 to R''.

References

[1] E. Böhler, N. Creignou, S. Reith, H. Vollmer, Playing with Boolean blocks, Part I: Post's lattice with applications to complexity theory, ACM SIGACT Newsletter 34 (2003) 38–52.

[2] E. Böhler, S. Reith, H. Schnoor, H. Vollmer, Bases for Boolean co-clones, Inform. Process. Lett. 96 (2005) 59–66.

[3] A. Bulatov, The complexity of the counting constraint satisfaction problem, in: Proc. 35th International Colloquium for Automata, Languages and Programming, in: Lecture Notes in Comput. Sci., vol. 5125, Springer-Verlag, 2008, pp. 646–661.

[4] A. Bulatov, V. Dalmau, Towards a dichotomy theorem for the counting constraint satisfaction problem, in: Proc. 44th Annual IEEE Symposium on Foundations of Computer Science, 2003, pp. 562–573.

[5] A. Bulatov, M. Grohe, The complexity of partition functions, Theoret. Comput. Sci. 348 (2005) 148–186.

[6] N. Creignou, M. Hermann, Complexity of generalized satisfiability counting problems, Inform. and Comput. 125 (1996) 1–12.

[7] N. Creignou, S. Khanna, M. Sudan, Complexity Classifications of Boolean Constraint Satisfaction Problems, SIAM Press, 2001.

[8] N. Creignou, P. Kolaitis, B. Zanuttini, Preferred representations of Boolean relations, Electronic Colloquium on Computational Complexity, Report No. 119, 2005.

[9] M. Dyer, L.A. Goldberg, C. Greenhill, M. Jerrum, The relative complexity of approximate counting problems, Algorithmica 38 (2004) 471–500.

[10] M. Dyer, L.A. Goldberg, M. Jerrum, The complexity of weighted Boolean #CSP, SIAM J. Comput. 38 (2009) 1970–1986.

[11] P. Hell, J. Nešetřil, Graphs and Homomorphisms, Oxford University Press, Oxford, 2004.

[12] M. Dyer, C. Greenhill, The complexity of counting graph homomorphisms, Random Structures Algorithms 17 (2000) 260–289.

[13] T. Feder, M. Vardi, The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory, SIAM J. Comput. 28 (1999) 57–104.
[14] M. Jerrum, L. Valiant, V. Vazirani, Random generation of combinatorial structures from a uniform distribution, Theoret. Comput. Sci. 43 (1986) 169–188.
[15] R. Ladner, On the structure of polynomial time reducibility, Journal of the Association for Computing Machinery 22 (1975) 155–171.
[16] M. Mitzenmacher, E. Upfal, Probability and Computing, Cambridge University Press, Cambridge, 2005.
[17] C.H. Papadimitriou, Computational Complexity, Addison–Wesley, 1994.
[18] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006.
[19] T. Schaefer, The complexity of satisfiability problems, in: Proc. 10th Annual ACM Symposium on Theory of Computing, ACM Press, 1978, pp. 216–226.