The Classes PPA-k:
Existence from Arguments Modulo k

Alexandros Hollender[*]

Department of Computer Science, University of Oxford
alexandros.hollender@cs.ox.ac.uk

Abstract

The complexity classes PPA-k, $k \geq 2$, have recently emerged as the main candidates for capturing the complexity of important problems in fair division, in particular Alon's Necklace-Splitting problem with k thieves. Indeed, the problem with two thieves has been shown complete for PPA = PPA-2. In this work, we present structural results which provide a solid foundation for the further study of these classes. Namely, we investigate the classes PPA-k in terms of (i) equivalent definitions, (ii) inner structure, (iii) relationship to each other and to other TFNP classes, and (iv) closure under Turing reductions.

1 Introduction

The complexity class TFNP is the class of all search problems such that every instance has a least one solution and any solution can be checked in polynomial time. It has attracted a lot of interest, because, in some sense, it lies between P and NP. Moreover, TFNP contains many natural problems for which no polynomial-time algorithm is known, such as Factoring (given a integer, find a prime factor) or Nash (given a bimatrix game, find a Nash equilibrium). However, no problem in TFNP can be NP-hard, unless NP = co-NP [29]. Furthermore, it is believed that no TFNP-complete problem exists [32, 34]. Thus, the challenge is to find some way to provide evidence that these TFNP problems are indeed hard.

Papadimitriou [32] proposed the following idea: define *subclasses* of TFNP and classify the natural problems of interest with respect to these classes. Proving that many natural problems are complete for such a class, shows that they are "equally" hard. Then, investigating how these classes relate to each other, yields a relative classification of all these problems. In other words, it provides a unified framework that gives a better understanding of how these problems relate to each other. TFNP subclasses are based on various non-constructive existence results. Some of these classes and their corresponding existence principle are:

- PPAD : given a directed graph and an unbalanced vertex (i.e., out-degree $\neq$ in-degree), there must exist another unbalanced vertex.

[*] A preliminary version of this paper appeared in the Proceedings of the 15th Conference on Web and Internet Economics (WINE 2019).

- PPA : given an undirected graph and vertex with odd degree, there must exist another vertex with odd degree (Handshaking Lemma).
- PPP : given a function mapping a finite set to a smaller set, there must exist a collision (Pigeonhole Principle).

Other TFNP subclasses are PPADS, PLS [26], CLS [10], PTFNP [21], EOPL and UEOPL [15]. It is known that PPAD $\subseteq$ PPADS $\subseteq$ PPP, PPAD $\subseteq$ PPA and UEOPL $\subseteq$ EOPL $\subseteq$ CLS $\subseteq$ PPAD$\cap$PLS. Very recently it was shown that in fact CLS = PPAD$\cap$PLS [16]. Any separation between TFNP subclasses would imply P $\neq$ NP, but various oracle separations exist [3, 31, 4, 5] (see Section 2 for more details).

TFNP subclasses have been very successful in capturing the complexity of natural problems. The most famous result is that the problem Nash is PPAD-complete [11, 7], but various other natural problems have also been shown PPAD-complete [9, 6, 8, 27]. Many local optimisation problems have been proved PLS-complete [26, 33, 28, 14, 13]. Recently, the first natural complete problems were found for PPA [17, 18] and PPP [35]. The famous Factoring problem has been partially related to PPA and PPP [23].

Necklace-Splitting. The natural problem recently shown PPA-complete is a problem in fair division, called the 2-Necklace-Splitting problem [18]. For $k \geq 2$, the premise of the k-Necklace-Splitting problem is as follows. Imagine that k thieves have stolen a necklace that has beads of different colours. Since the thieves are unsure of the value of the different beads, they want to divide the necklace into k parts such that each part contains the same number of beads of each colour. However, the string of the necklace is made of precious metal, so the thieves don't want to use too many cuts. Alon's famous result [1] says that this can always be achieved with a limited number of cuts.

The corresponding computational problem can be described as follows. We are given an open necklace (i.e., a segment) with n beads of c different colours, i.e., there are a_i beads of colour i and $\sum_{i=1}^{c} a_i = n$. Furthermore, assume that for each i, a_i is divisible by k (the number of thieves). The goal is to cut the necklace in (at most) $c(k-1)$ places and allocate the pieces to the k thieves, such that every thief gets exactly a_i/k beads of colour i, for each colour i. By Alon's result [1], a solution always exists, and thus the problem lies in TFNP.

The complexity of this problem has been an open problem for almost 30 years [32]. While the 2-thieves version is now resolved, the complexity of the problem with k thieves ($k \geq 3$) remains open. The main motivation of the present paper is to investigate the classes PPA-k, which are believed to be the most likely candidates to capture the complexity of k-Necklace-Splitting. Indeed, in the conclusion of the paper where they prove that 2-Necklace-Splitting is PPA-complete, Filos-Ratsikas and Goldberg [18, arXiv version] mention:

> "What is the computational complexity of k-thief Necklace-splitting, for k not a power of 2? As discussed in [30, 12], the proof that it is a total search problem, does *not* seem to boil down to the PPA principle. Right now, we do not even know if it belongs to PTFNP [21].
>
> Interestingly, Papadimitriou in [32] (implicitly) also defined a number of computational complexity classes related to PPA, namely PPA-p, for a parameter $p \geq 2$. [...] Given the discussion above, it could possibly be the case that the principle associated with Necklace-Splitting for k-thieves is the PPA-k principle instead."

PPA-p. The TFNP subclasses PPA-p were defined by Papadimitriou almost 30 years ago in his seminal paper [32]. Recall that the existence of a solution to a PPA problem is guaranteed by a parity argument, i.e., an argument modulo 2. The classes PPA-p are a generalisation of this. For every prime p, the existence of a solution to a PPA-p problem is guaranteed by an argument modulo p. In particular, PPA-2 = PPA. Surprisingly, these classes have received very little attention. As far as we know, they have only been studied in the following:

- Papadimitriou [32] defined the classes PPA-p and proved that a problem called Chevalley-mod-p lies in PPA-p and a problem called Cubic-Subgraph lies in PPA-3.
- In an online thread on Stack Exchange [24], Jeřábek provided two other equivalent ways to define PPA-3. The problems and proofs can be generalised to any prime p.
- In his thesis [25], Johnson defined the classes PMOD^k for any $k \geq 2$, which were intended to capture the complexity of counting arguments modulo k. He proved various oracle separation results involving his classes and other TFNP classes. While the PPA-p classes are not mentioned by Johnson, using Jeřábek's results [24] it is easy to show that PMOD^p = PPA-p for any prime p. In Section 6, we characterise PMOD^k in terms of the classes PPA-p when k is not prime. In particular, we show that PMOD^k only partially captures existence arguments modulo k.

Our contribution. In this paper, we use the natural generalisation of Papadimitriou's definition of the classes PPA-p to define PPA-k for any $k \geq 2$. We then provide a characterisation of PPA-k in terms of the classes PPA-p. In particular, we show that PPA-k is completely determined by the set of prime factors of k. In order to gain a better understanding of the inner structure of the class PPA-k, we also define new subclasses that we denote PPA-$k[\#\ell]$ and investigate how they relate to the other classes. We show that PPA-$k[\#\ell]$ is completely determined by the set of prime factors of $k/\gcd(k,\ell)$.

Furthermore, we provide various equivalent complete problems that can be used to define PPA-k and PPA-$k[\#\ell]$ (Section 4). While these problems are not "natural", we believe that they provide additional tools that can be very useful when proving that natural problems are complete for these classes. In Section 7, we provide an additional tool for showing that problems lie in these classes: we prove that PPA-p^r (p prime, $r \geq 1$) and PPA-$k[\#\ell]$ ($k \geq 2$) are closed under Turing reductions. On the other hand, we provide evidence that PPA-k might not be closed under Turing reductions when k is not a prime power.

Finally, in Section 6 we investigate the classes PMOD^k defined by Johnson [25] and provide a full characterisation in terms of the classes PPA-k. In particular, we show that PMOD^k = PPA-k if k is a prime power. However, when k is not a prime power, we provide evidence that PMOD^k does not capture the full strength of existence arguments modulo k, unlike PPA-k. This characterisation of PMOD^k in terms of PPA-k leads to some oracle separation results involving PPA-k and other TFNP classes (using Johnson's oracle separation results).

We note that a significant fraction of our results were also obtained by Göös, Kamath, Sotiraki and Zampetakis in concurrent and independent work [22]. In their work, they have also provided the first "natural" complete problem for the classes PPA-p (a variant of Chevalley-mod-p), namely the first complete problem that does not involve circuits or other computational devices in its description. The present work, and in particular the equivalent characterisations of the classes PPA-k, have been pivotal in subsequent work [19] showing that the k-Necklace-Splitting problem lies in PPA-k under Turing reductions. However, the question of whether k-Necklace-Splitting is also PPA-k-hard remains open.

2 Preliminaries

TFNP. Let $\{0,1\}^*$ denote the set of all finite length bit-strings and for $w \in \{0,1\}^*$ let $|w|$ be its length. A computational search problem is given by a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$. The problem is: given an instance $I \in \{0,1\}^*$, find an $s \in \{0,1\}^*$ such that $(I, s) \in R$, or return that no such s exists. The search problem R is in FNP (*Functions in NP*), if R is polynomial-time computable (i.e., $(I, s) \in R$ can be decided in polynomial time in $|I| + |s|$) and there exists some polynomial p such that $(I, s) \in R \implies |s| \leq p(|I|)$. Thus, FNP is the search problem version of NP (and FNP-complete problems are equivalent to NP-complete problems under Turing reductions).

The class TFNP (*Total Functions in NP* [29]) contains all FNP search problems R that are *total*: for every $I \in \{0,1\}^*$ there exists $s \in \{0,1\}^*$ such that $(I, s) \in R$. With a slight abuse of notation, we can say that P lies in TFNP. Indeed, if a decision problem is solvable in polynomial time, then both the "yes" and "no" answers can be verified in polynomial time. In this sense, TFNP lies between P and NP.

Note that TFNP problems are not promise problems, i.e., we are not allowed to restrict the instance space $\{0,1\}^*$. This means that for any instance in $\{0,1\}^*$, there must always exist at least one solution. Nevertheless, TFNP can indirectly capture various settings where the instance space is restricted. For example, if a problem R in FNP is total only on a subset L of the instances and $L \in P$, then we can transform it into an equivalent TFNP problem by adding $(I, 0)$ to R for all instances $I \notin L$.

Reductions. Let R and S be total search problems in TFNP. We say that R (many-one) reduces to S, denoted $R \leq S$, if there exist polynomial-time computable functions f, g such that

$$(f(I), s) \in S \implies (I, g(I, s)) \in R.$$

Note that if S is polynomial-time solvable, then so is R. We say that two problems R and S are (polynomial-time) equivalent, if $R \leq S$ and $S \leq R$.

There is also a more general type of reduction. A Turing reduction from R to S is a polynomial-time oracle Turing machine that solves problem R with the help of queries to an oracle for S. Note that a Turing reduction that only makes a single oracle query immediately yields a many-one reduction.

Encoding of Sets. Many of the computational problems we consider in this paper involve Boolean circuits whose output is interpreted as a set. For example, it will often be the case that a circuit C takes as input a bit-string in $\{0,1\}^n$ and outputs a set of at most m bit-strings in $\{0,1\}^n$. We will denote this by $C : \{0,1\}^n \to \mathsf{Set}_{\leq m}(\{0,1\}^n)$. Of course, a Boolean circuit has a fixed number of output bits and so the circuit will in fact be of the form $C : \{0,1\}^n \to \{0,1\}^t$, for some t that is sufficiently large so that there are enough bits to encode any set of size at most m. It is easy to see that taking $t = mn + 1$ is enough. Indeed, we can for example use the following encoding: the set $\{x_1, \dots, x_\ell\} \subseteq \{0,1\}^n$, $\ell \leq m$, is represented by the bit-string $x_1 \cdots x_\ell 1 0^{(m-\ell)n} \in \{0,1\}^{mn+1}$. Clearly, we can efficiently check whether a bit-string in $\{0,1\}^{mn+1}$ is a valid representation of a set, and if not, we can just interpret it as the empty set $\emptyset \subset \{0,1\}^n$.

PPA. The class PPA (Polynomial Parity Argument) [32] is defined as the set of all TFNP problems that many-one reduce to the problem Leaf [32, 3]: given an undirected graph with maximum degree 2 and a leaf (i.e., a vertex of degree 1), find another leaf. The important thing to note is that the graph is not given explicitly (in which case the problem would be very easy), but it is provided implicitly through a succinct representation.

The vertex set is $\{0,1\}^n$ and the undirected graph is represented by a Boolean circuit $C : \{0,1\}^n \to \mathsf{Set}_{\leq 2}(\{0,1\}^n)$. By this we mean that for any $x \in \{0,1\}^n$, we interpret $C(x)$ as the set of potential neighbours of x, where we syntactically enforce that $x \notin C(x)$. We say that there is an edge between x and y if $x \in C(y)$ and $y \in C(x)$. Thus, every vertex has at most two neighbours. Note that the size of the graph can be exponential with respect to its description size.

The full formal definition of the problem Leaf is: given a Boolean circuit $C : \{0,1\}^n \to \mathsf{Set}_{\leq 2}(\{0,1\}^n)$ representing an undirected graph on the vertex set $\{0,1\}^n$ such that $|C(0^n)| = 1$ (i.e., 0^n is a leaf), find

- $x \neq 0^n$ such that $|C(x)| = 1$ (another leaf)
- or x, y such that $x \in C(y)$ but $y \notin C(x)$ (an inconsistent edge)

Type 2 Problems and Oracle Separations. We work in the standard Turing machine model, but TFNP subclasses have also been studied in the black-box model. In this model, one considers the type 2 versions of the problems, namely, the circuits in the input are replaced by black-boxes. In that case, it is possible to prove unconditional separations between type 2 TFNP subclasses (in the standard model this would imply P ≠ NP). The interesting point here is that separations between type 2 classes yield separations of the corresponding classes in the standard model with respect to any generic oracle (see [3] for more details on this). This technique has been used to prove various oracle separations between TFNP subclasses [3, 31, 4, 5]. In Section 6 we provide some oracle separations involving PPA-k and other TFNP subclasses.

On the other hand, any reduction that works in the type 2 setting, also works in the standard setting. Indeed, it suffices to replace the calls to the black boxes by the corresponding circuits that compute them. In this paper, our reductions are stated in the standard model, but they also work in the type 2 setting, because they don't examine the inner workings of the circuits.

3 Definition of the Classes

3.1 PPA-k : Polynomial Argument modulo k

For any prime p, Papadimitriou [32] defined the class PPA-p as the set of all TFNP problems that many-one reduce to the following problem, that we call Bipartite-mod-p: We are given an undirected bipartite graph (implicitly represented by a circuit) and a vertex with degree $\neq 0 \mod p$ (which we call the *trivial solution*). The goal is to find another such vertex. This problem lies in TFNP: if all other vertices had degree $= 0$ mod p, then the sum of the degrees of all vertices on each side would have a different value modulo p, which is impossible.

The problem remains well-defined and total if p is not a prime, and so we will instead define it for any $k \geq 2$. Let us now provide a formal definition of the problem. A vertex of the bipartite graph is represented as a bit-string in $\{0,1\} \times \{0,1\}^n$, where the first bit indicates whether the vertex lies on the "left" or "right" side of the bipartite graph. The graph will be represented by a Boolean circuit that outputs a set of potential neighbours, just as we did for Leaf. Instead of at most two neighbours, here we allow at most k neighbours (see Remark 1 for why this is enough). Note that we can syntactically enforce that the graph is bipartite, i.e., that a vertex $0x$ can only have neighbours of the type $1y$ and vice versa.

Definition 1 (Bipartite-mod-k [32])**.** Let $k \geq 2$. The problem Bipartite-mod-k is defined as: given a Boolean circuit $C : \{0,1\} \times \{0,1\}^n \to \mathsf{Set}_{\leq k}(\{0,1\} \times \{0,1\}^n)$ representing a bipartite graph on the vertex set $(\{0\} \times \{0,1\}^n, \{1\} \times \{0,1\}^n)$ with $|C(00^n)| \in \{1,\dots,k-1\}$, find

- $x \neq 00^n$ such that $|C(x)| \notin \{0,k\}$
- or x, y such that $y \in C(x)$ but $x \notin C(y)$.

Here the trivial solution is the vertex 00^n. The first type of solution corresponds to a vertex with degree $\neq 0 \mod k$. The second type of solution corresponds to an edge that is not well-defined. We can always ensure that all edges are well-defined by doing some pre-processing. Indeed, in polynomial time we can construct a circuit C' such that all solutions are of the first type and yield a solution for C. On input $0x$ the circuit C' first computes $C(0x) = \{1y_1, \dots, 1y_m\}$ and then for each i removes $1y_i$ from this set, if $0x \notin C(1y_i)$.

Remark 1. Note that in this problem statement we require that all degrees lie in $\{0,1,\dots,k\}$. This is easily seen to be equivalent to the more general formulation where vertices can have more than k neighbours. Indeed, any vertex that has more than k edges can be split into multiple copies such that all the copies have 0 or k edges, except for one copy which is allowed to have any number of edges in $\{0,1,\dots,k\}$. A solution of the original instance is then easily recovered from a solution of this modified instance. Note that since the set of neighbours is given as the output of a circuit, it will have length bounded by some polynomial in the input size and so this argument can indeed be applied.

Definition 2 (PPA-k [32])**.** For any $k \geq 2$, the class PPA-k is defined as the set of all TFNP problems that many-one reduce to Bipartite-mod-k.

As a warm-up let us show the following:

Proposition 1 ([32])**.** PPA-2 = PPA

Proof. Recall that PPA can be defined using the canonical complete problem Leaf [32, 3]: given an undirected graph where every vertex has degree at most 2, and a leaf (i.e., degree = 1), find another leaf. This immediately yields PPA-2 $\subseteq$ PPA, since Bipartite-mod-2 is just a special case of Leaf where the graph is bipartite.

Given an instance of Leaf with graph $G = (\{0,1\}^n, E)$ we construct an instance of Bipartite-mod-2 on the vertex set $\{0,1\} \times \{0,1\}^{2n}$ as follows. For any $u \in \{0,1\}^n$ we have a vertex $x_u := 0u0^n$ on the left side of the bipartite graph. For any edge $\{u,v\} \in E$ (u, v ordered lexicographically) we have a vertex $y_{uv} := 1uv$ on the right side of the bipartite graph and we create the edges $\{x_u, y_{uv}\}$ and $\{x_v, y_{uv}\}$. All other vertices in $\{0,1\} \times \{0,1\}^{2n}$ are isolated. In polynomial time we can construct a circuit that computes the neighbours of any vertex. Furthermore, $w \in \{0,1\}^n$ is a leaf, if and only if x_w has degree 1. Finally, all vertices on the right-hand side have degree 0 or 2. □

3.2 PPA-$k[\#\ell]$: Fixing the degree of the trivial solution

In the definition of the PPA-k-complete problem Bipartite-mod-k (Definition 1) the degree of the trivial solution 00^n can be any number in $\{1,\dots,k-1\}$. In this section we define more refined classes where the degree of the trivial solution is fixed. In Section 5, these classes will be very useful to describe how the PPA-k classes relate to each other. These definitions are inspired by the corresponding "counting principles" studied in Beame et al. [2] that were also defined in a refined form in order to describe how they relate to each other. We believe that these refined classes will also be useful to capture the complexity of natural problems. Note that for $k=2$, the degree of the trivial solution will always be 1 and thus the question does not even appear in the study of PPA.

Definition 3. Let $k \geq 2$ and $1 \leq \ell \leq k-1$. The problem Bipartite-mod-$k[\#\ell]$ is defined as Bipartite-mod-k (Definition 1) but with the additional condition $|C(00^n)| = \ell$.

Note that this problem remains in TFNP, since the condition can be checked efficiently.

Definition 4 (PPA-$k[\#\ell]$)**.** Let $k \geq 2$ and $1 \leq \ell \leq k-1$. The class PPA-$k[\#\ell]$ is defined as the set of all TFNP problems that many-one reduce to Bipartite-mod-$k[\#\ell]$.

If k is some prime p, then these classes are not interesting. Indeed, it holds that PPA-$p[\#\ell]$ = PPA-p for all $1 \leq \ell \leq p-1$. This can be shown using the following technique: take multiple copies of the instance and "glue" the trivial solutions together. If p is prime, then any other degree of the glued trivial solution can be obtained (by taking the right number of copies). In fact this technique yields the stronger result:

Lemma 1. *If* $\gcd(k,\ell_1)$ *divides* ℓ_2*, then* PPA-$k[\#\ell_1] \subseteq$ PPA-$k[\#\ell_2]$.

Proof. Since $\gcd(k,\ell_1)$ divides ℓ_2, there exists $m < k$ such that $m \times \ell_1 = \ell_2 \mod k$. Given an instance of Bipartite-mod-$k[\#\ell_1]$, take the union of m copies of the instance, i.e., $m2^n$ vertices on each side (and any additional isolated vertices needed to reach a power of 2). Then, merge the m different copies of the trivial solution into one (by redirecting edges to a single one). This vertex will have degree $m\ell_1 = \ell_2 \mod k$. Finally, apply the usual trick to ensure all degrees are in $\{0,1,\dots,k\}$ (Remark 1). □

In particular, we also get the nice result PPA-$k[\#\ell]$ = PPA-$k[\#\gcd(k,\ell)]$. Applying the result to the case $k=6$, we get that PPA-6[#1] = PPA-6[#5], PPA-6[#2] = PPA-6[#4], as well as PPA-6[#1] $\subseteq$ PPA-6[#2] and PPA-6[#1] $\subseteq$ PPA-6[#2]. Thus, we have three "equivalence classes" $\{1,5\}$, $\{2,4\}$ and $\{3\}$ and the relationships $\{1,5\} \leq \{2,4\}$ and $\{1,5\} \leq \{3\}$. In Section 5, we will show that $\{2,4\}$ corresponds to PPA-3, $\{3\}$ to PPA-2 and $\{1,5\}$ to PPA-2 $\cap$ PPA-3.

Now let us introduce some notation that will allow us to precisely describe the relationship between PPA-k and the PPA-$k[\#\ell]$.

Definition 5 (& operation [5])**.** Let R_0 and R_1 be two TFNP problems. Then the problem $R_0 \,\&\, R_1$ is defined as: given an instance I_0 of R_0, an instance I_1 of R_1 and a bit $b \in \{0,1\}$, find a solution to I_b.

This operation is commutative and associative (up to many-one equivalence). Indeed, $R_0 \,\&\, R_1$ is many-one equivalent to $R_1 \,\&\, R_0$, and $(R_0 \,\&\, R_1) \,\&\, R_2$ is many-one equivalent to $R_0 \,\&(R_1 \,\&\, R_2)$. Since the & operation is associative, the problem $\&_{\ell=1}^{k} R_\ell$ is well-defined up to many-one equivalence. It is also equivalent to the following problem: given instances $I_1,\dots,I_k$ of $R_1,\dots,R_k$ and an integer $j \in \{1,\dots,k\}$, find a solution to I_j.

We extend the & operation to TFNP subclasses in the natural way. Let C_0 and C_1 be TFNP subclasses with complete problems R_0 and R_1 respectively. Then $C_0 \,\&\, C_1$ is the class of all TFNP problems that many-one reduce to $R_0 \,\&\, R_1$. Note that the choice of complete problems does not matter. Intuitively, this class contains all problems that can be solved in polynomial time by a Turing machine with a single oracle query to either C_0 or C_1.

Using this definition we obtain:

Lemma 2. *For all $k \geq 2$ we have* $\text{PPA-}k = \overset{k-1}{\underset{\ell=1}{\&}} \text{PPA-}k[\#\ell]$.

Proof. One containment immediately follows from the fact that $\text{PPA-}k[\#\ell] \subseteq \text{PPA-}k$ for all $\ell \in \{1, \dots, k-1\}$. For the other containment, note that for any instance I of Bipartite-mod-k we can easily compute $\ell \in \{1, \dots, k-1\}$ such that I is also an instance of Bipartite-mod-$k[\#\ell]$. □

Together with Lemma 1, Lemma 2 yields, e.g., PPA-6 = PPA-6[#2] & PPA-6[#3].

4 Equivalent Definitions

In this section we show that PPA-k can be defined by using other problems instead of Bipartite-mod-k. The totality of these problems is again based on arguments modulo k. By showing that these problems are indeed PPA-k-complete, we provide additional support for the claim that PPA-k captures the complexity of "polynomial arguments modulo k". While these problems are not "natural" and thus not interesting in their own right, they provide equivalent ways of defining PPA-k, which can be very useful when working with these classes. In particular, we make extensive use of these equivalences in this work.

The TFNP problems we consider are the following:

- Imbalance-mod-k : given a directed graph and a vertex that is *unbalanced-mod-k*, i.e., out-degree − in-degree $\neq 0 \mod k$, find another such vertex.

- Hypergraph-mod-k : given a hypergraph and a vertex that has degree $\neq 0 \mod k$, find another such vertex or a hyperedge that has size $\neq k$.

- Partition-mod-k : given a set of size $\neq 0 \mod k$ and a partition into subsets, find a subset that has size $\neq k$.

As usual, the size of the graph (respectively hypergraph, set) can be exponential in the input size, and the edges (resp. hyperedges, subsets) can be computed efficiently locally. We also define the corresponding problems Imbalance-mod-$k[\#\ell]$, Hypergraph-mod-$k[\#\ell]$ and Partition-mod-$k[\#\ell]$ analogously. The formal definitions of all these problems are provided in Section 4.1.

Theorem 1. *Let $k \geq 2$ and $1 \leq \ell \leq k-1$.*

- *Imbalance-mod-$k[\#\ell]$, Hypergraph-mod-$k[\#\ell]$, Partition-mod-$k[\#\ell]$ are PPA-$k[\#\ell]$-complete,*

- *Imbalance-mod-k, Hypergraph-mod-k, Partition-mod-k are PPA-k-complete.*

In his online post [24], Jeřábek proves that Bipartite-mod-3, Imbalance-mod-3 and Partition-mod-3 are equivalent and (correctly) claims that the proof generalises to any other prime. Thus, our contribution is the definition of the problems for any $k \geq 2$ (and the ℓ-parameter versions) and the generalisation of the result to any $k \geq 2$ (not only primes) and to the ℓ-parameter versions of the problems, as well as to the new problem Hypergraph-mod-k. The proof of Theorem 1 can be found in Section 4.2.

The problem Imbalance-mod-k is a generalisation of the PPAD-complete problem Imbalance [3, 20] : given a directed graph and a vertex that is unbalanced (i.e., out-degree $-$ in-degree $\neq 0$), find another unbalanced vertex. It is known [20] that in Imbalance we can assume without loss of generality that the given vertex has imbalance exactly 1. As a result, Imbalance trivially reduces to Imbalance-mod-k, and thus Theorem 1 also yields[1]:

Corollary 1. *For all $k \geq 2$, we have* PPAD $\subseteq$ PPA-k.

Furthermore, if we use the convention that $a = b \mod 0$ if and only if $a = b$, then Imbalance-mod-0 actually corresponds to Imbalance. Thus, in a certain sense we could define PPA-0 = PPAD. On the other hand, Imbalance-mod-1 is a trivial problem.

4.1 Formal definitions

Imbalance. A directed graph on the vertex set $\{0,1\}^n$ is represented by Boolean circuits $S, P : \{0,1\}^n \to \mathsf{Set}_{\leq k}(\{0,1\}^n)$ that output the successor and predecessor set of a given vertex, respectively. As before, it is enough to consider the case where the in- and out-degree of any vertex is at most k, since the general case reduces to this (analogously to Remark 1). We syntactically enforce that $x \notin S(x) \cup P(x)$ and we interpret $S(x)$ (respectively, $P(x)$) as the set of potential successors (respectively, predecessors) of x. There is a directed edge from x to y if $y \in S(x)$ and $x \in P(y)$. The following problem was defined by Jeřábek [24], but only for prime k and without the ℓ-parameter version.

Definition 6. Let $k \geq 2$. The problem Imbalance-mod-k is defined as: given Boolean circuits $S, P : \{0,1\}^n \to \mathsf{Set}_{\leq k}(\{0,1\}^n)$ representing a directed graph on the vertex set $\{0,1\}^n$ with $|S(0^n)| - |P(0^n)| \notin \{-k, 0, k\}$, find

- $x \neq 0^n$ such that $|S(x)| - |P(x)| \notin \{-k, 0, k\}$
- or x, y such that $y \in S(x)$ but $x \notin P(y)$, or $y \in P(x)$ but $x \notin S(y)$.

For $1 \leq \ell \leq k-1$, Imbalance-mod-$k[\#\ell]$ is defined with the additional condition $|S(0^n)| - |P(0^n)| = \ell$.

Hypergraph. A hypergraph on the vertex set $\{0,1\}^n$ is represented as follows. For every vertex $x \in \{0,1\}^n$, a circuit $C : \{0,1\}^n \to \mathsf{Set}_{\leq k}(\mathsf{Set}_{\leq k}(\{0,1\}^n))$ outputs the set $C(x)$ of all hyperedges containing x, where each hyperedge is a set of vertices in $\{0,1\}^n$. As usual, we only need to consider the case where every vertex is contained in at most k hyperedges and every hyperedge has size at most k. A hyperedge $\{x_1, \dots, x_m\}$ exists in the hypergraph, if all the vertices involved indeed agree that it is present, i.e., if $\{x_1, \dots, x_m\} \in C(x_i)$ for all $i \in \{1, \dots, m\}$.

Definition 7. Let $k \geq 2$. The problem Hypergraph-mod-k is defined as: given a Boolean circuit $C : \{0,1\}^n \to \mathsf{Set}_{\leq k}(\mathsf{Set}_{\leq k}(\{0,1\}^n))$ representing a hypergraph on the vertex set $\{0,1\}^n$ with $|C(0^n)| \notin \{0, k\}$, find

[1] This observation was also made by Jeřábek for the classes PPA-p (p prime).

- $x \neq 0^n$ such that $|C(x)| \notin \{0, k\}$
- or x such that $C(x)$ contains a hyperedge of size $\neq k$
- or x, y such that $C(x)$ and $C(y)$ are not consistent with one another.

For $1 \leq \ell \leq k-1$, HYPERGRAPH-MOD-$k[\#\ell]$ is defined with the additional condition $|C(0^n)| = \ell$.

Note that for $k = 2$ this problem essentially corresponds to the PPA-complete problem LEAF and its (equivalent) generalisation ODD [32, 3]: given an undirected graph and a vertex with odd degree, find another one.

Partition. A partition of $\{0,1\}^n$ is represented by a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^n$ as follows: $x \in \{0,1\}^n$ lies in the subset given by the orbit of x with respect to C, i.e., $\{C^i(x) : i \geq 0\}$, where $C^i(x) = C(C(\dots C(x))\dots)$ (i times). The problem we define below is based on the simple observation that a base set of size $\neq 0 \mod k$ cannot be partitioned into sets of size k. The base set consists of all elements in $\{0,1\}^n$ except for m elements that have been removed, for some $m < 2^n$ such that $2^n - m \neq 0 \mod k$. Here it is convenient to identify $\{0,1\}^n$ with $\{0, 1, \dots, 2^n - 1\}$ in the natural way. Thus, we can think of the base set as simply being $\{m, m+1, \dots, 2^n - 1\}$.

Definition 8 (PARTITION-MOD-k)**.** Let $k \geq 2$. The problem PARTITION-MOD-k is defined as: given $m < 2^n$ with $2^n - m \neq 0 \mod k$ and a Boolean circuit $C : \{0,1\}^n \to \{0,1\}^n$, such that $C(x) = x$ for all $x < m$, find

- $x \geq m$ and $d \in \mathbb{N}$ such that $C^d(x) = x$ and $d|k$, $d \neq k$
- or $x \in \{0,1\}^n$ such that $C^k(x) \neq x$

where $d|k$ means that d divides k.

For $1 \leq \ell \leq k-1$, PARTITION-MOD-$k[\#\ell]$ is defined with the additional condition $2^n - m = \ell \mod k$.

The condition "$C(x) = x$ for all $x < m$" corresponds to excluding elements that do not lie in the base set and it can be enforced syntactically. The first solution type corresponds to finding a set in the partition such that its size divides k (but is $\neq k$), while the second solution type corresponds to finding a set such that its size does not divide k. Note that a solution is guaranteed to exist since $2^n - m \neq 0 \mod k$.

The definition of this problem is inspired by the MOD^k problems defined by Buss and Johnson [5] (for prime $k > 2$) and by Johnson [25] (for any $k \geq 2$). In Section 6 we argue that, unlike the problem defined above, the MOD^k problems only partially capture the complexity of arguments modulo k (when k is not a prime power). The problem was also defined by Jeřábek [24], but only for k prime and without the ℓ-parameter version. Finally, note that PARTITION-MOD-2 essentially corresponds to the PPA-complete problem LONELY [3].

The definition of this problem can be modified in various ways without changing its complexity. For instance, the first solution type can be changed to simply ask for $x \geq m$ such that $C^d(x) = x$ for some $d < k$. We have defined the problem in a slightly more complicated way to make the connection with the MOD^k problems more immediate (see Section 6). Yet another equivalent way of defining the problem would be to consider a Boolean circuit $C : \{0,1\}^n \to \mathsf{Set}_{\leq k}(\{0,1\}^n)$ where $C(x) \subseteq \{0,1\}^n$ is interpreted as the set containing x in the partition. A solution would then be any $x \geq m$ with $|C(x)| < k$ or any x, y witnessing an inconsistency in the partition given by C.

4.2 Proof of Theorem 1

We omit some details that are easy to fill in. For example, when given an instance of Imbalance-mod-$k[\#\ell]$, we assume that all the edges are well-defined, i.e., solutions of the second type never occur. Indeed, given a generic instance of the problem, it can be reduced to an instance where this holds by modifying the circuits so that they check and correct the successor/predecessor list before outputting it. Note that in the new instance only solutions of the first type can occur, but they can yield a solution of the second type of the original problem. The same observation also holds for Bipartite-mod-$k[\#\ell]$ (edges well-defined), Hypergraph-mod-$k[\#\ell]$ (hyperedges well-defined) and Partition-mod-$k[\#\ell]$ (size of any subset divides k).

Bipartite-mod-$k[\#\ell]$ $\leq$ Hypergraph-mod-$k[\#\ell]$: We construct a hypergraph on the vertex set $\{0,1\}^n$. We identify every vertex u of the hypergraph with the vertex $0u$ on the left-hand side of the bipartite graph. The hyperedges are given by the vertices on the right-hand side of the bipartite graph. More precisely, if for every right-hand side vertex $1v$ we let $N(1v)$ be the set of neighbours (on the left-hand side), then the set of hyperedges is exactly $\{N(1v) : v \in \{0,1\}^n\}$. Note that given $u \in \{0,1\}^n$, we can find all the hyperedges containing u in polynomial time. Furthermore, since the vertex 00^n has degree ℓ in the bipartite graph, the corresponding vertex 0^n in the hypergraph will also have degree ℓ. It is easy to check that any solution of the Hypergraph-mod-$k[\#\ell]$ instance (in particular also any hyperedge that does not have size k) yields a solution to the Bipartite-mod-$k[\#\ell]$ instance.

Hypergraph-mod-$k[\#\ell]$ $\leq$ Imbalance-mod-$k[\#\ell]$: We construct a directed graph on the vertex set $\{0,1\}^{kn+2}$. For each vertex u of the hypergraph there is a vertex v_u in the directed graph (e.g., $v_u = 0u0^{(k-1)n+1}$) and for each hyperedge e of the hypergraph there is a vertex v_e in the directed graph (e.g., $v_e = 1u_1 \dots u_m 10^{(k-m)n}$ where $e = \{u_1, \dots, u_m\}$ is ordered lexicographically). We put a directed edge from v_u to v_e iff $u \in e$ (i.e., iff e appears in the hyperedge list of u). All other vertices are isolated. Note that $0^{kn+2} = v_{0^n}$ has imbalance ℓ and for any vertex in $\{0,1\}^{kn+2}$ we can compute the predecessors and successors in polynomial time. If there is an imbalance modulo k in a vertex v_e, then the corresponding hyperedge e does not have size k. If there is an imbalance modulo k in a vertex v_u, then u has degree $\neq 0 \mod k$ in the hypergraph.

Imbalance-mod-$k[\#\ell]$ $\leq$ Partition-mod-$k[\#\ell]$: Consider an instance of the problem Imbalance-mod-$k[\#\ell]$. Split every vertex v into two vertices v_{in} and v_{out}, such that v_{in} gets all the incoming edges and v_{out} gets all the outgoing edges. If v was balanced, i.e., in-deg(v) = out-deg$(v) = d$, then we add $k-d$ edges from v_{out} to v_{in}. We can assume that out-deg$(0^n) = \ell$ and in-deg$(0^n) = 0$ (just create a copy of 0^n that takes in-deg(0^n) incoming and outgoing edges), and thus out-deg$(0^n_{out}) = \ell$ and in-deg$(0^n_{in}) = 0$. Note that we are using multi-edges, which are not allowed in the definition of the problem. However, this is not an issue, since this is just an intermediate step of the reduction. This new instance has the property that no vertex has both incoming and outgoing edges. Furthermore, any solution (i.e., a vertex with in- or out-degree not in $\{0,k\}$, except 0^n_{out}) yields a solution of the original instance.

Thus, we can assume wlog that the Imbalance-mod-$k[\#\ell]$ instance (with multi-edges) is such that no vertex has both incoming and outgoing edges. We construct an instance of Partition-mod-$k[\#\ell]$ on the set $\{0,1\}^{k+n}$. Every vertex u of the directed graph has

k corresponding elements in the set $\{0,1\}^{k+n}$, namely $u_1 = 10^{k-1}u$, …, $u_k = 0^{k-1}1u$. If u does not have any outgoing edges, then $u_1, \ldots, u_k$ form a subset of the partition, i.e., $C(u_i) = u_{i+1 \bmod k}$. If u has outgoing edges to $v^{(1)}, v^{(2)}, \ldots, v^{(j)}$ ($j \leq k$, ordered lexicographically), then for every $i = 1, \ldots, j$ we put u_i in a subset that we denote $S_{v^{(i)}}$. $u_{j+1}, \ldots, u_k$ are put into isolated subsets, i.e., $C(u_i) = u_i$ for all $i = j+1, \ldots, m$. Note that if v has k incoming edges, then S_v will contain k elements. Given any element u_i, we can compute all the elements in its subset in polynomial time (and thus efficiently construct C that cycles through them in lexicographic order). Furthermore, since out-deg(0^n) $= \ell$, the vertices $0^n_{\ell+1}, \ldots, 0^n_k$ will be in singleton sets. Consider the subset of $\{0,1\}^{k+n}$ $X = \{u_i : u \in \{0,1\}^n, i \in \{1, \ldots, k\}\} \setminus \{0^n_{\ell+1}, \ldots, 0^n_k\}$. Then $|X| = k2^n - (k - \ell) = \ell \mod k$. It is easy to check that any element in X that is not contained in a subset of size k (according to C), must yield a solution to the Imbalance-mod-$k[\#\ell]$ instance. Finally, the last step is to construct an efficient bijection between X and the set of all integers $\{j : 2^{n+k} - |X| \leq j < 2^{n+k}\}$, which is easy to do. Thus, we have reduced the original instance to an instance of Partition-mod-$k[\#\ell]$ with inputs $m = 2^{n+k} - |X|$ and C (modified according to the bijection).

Partition-mod-$k[\#\ell]$ $\leq$ Bipartite-mod-$k[\#\ell]$: Let us consider any instance (C, m) of Partition-mod-$k[\#\ell]$ with parameter n. In particular, it holds that $m < 2^n$ and $2^n - m = \ell \mod k$. We construct a bipartite graph as follows. The vertex sets on the left and right side are $A = \{0,1\}^n$ and $B = \{0,1\}^n$ respectively. We can define a canonical partition of the numbers $m, m+1, \ldots, 2^n - 1$ into sets of size k (and one set of size ℓ). For example, $\{m, m+1, \ldots, m+\ell-1\}$, $\{m+\ell, m+\ell+1, \ldots, m+\ell+k-1\}$, etc. Each set of the canonical partition corresponds to a vertex in A as follows: a set containing k numbers $x_1 < x_2 < \cdots < x_k$ is represented by the vertex $x_1 \in A$. For the set of size ℓ in the canonical partition, we introduce a special case: it is represented by $0^n \in A$. Note that many vertices in A will not correspond to any set of the canonical partition. For a number $x \geq m$, we let $L(x) \in A$ denote the vertex in A representing the set containing x in the canonical partition.

Another partition of the numbers $m, m+1, \ldots, 2^n - 1$ into sets of size at most k is given by the instance (C, m) of Partition-mod-$k[\#\ell]$. Similarly to what we did above, we can associate each set in the partition given by C with a vertex in B. We let $R(x) \in B$ denote the vertex of B representing the set containing x in the partition given by C. Note that for any vertex in A or B we can efficiently determine whether it represents a set of one of the two partitions and if so, which set exactly it represents.

For every $x \geq m$ we add an edge between $L(x) \in A$ and $R(x) \in B$, i.e., between the sets that contain x in the two different partitions. This construction might introduce multi-edges (if some x and y lie in the same set in both partitions) but this can easily be resolved by using the *Mitosis gadgets* described below. It is easy to see that for any vertex of the bipartite graph we can efficiently compute the set of all its neighbours. Finally, note that the vertex $0^n \in A$ has degree ℓ, and any other vertex with degree $\neq 0 \mod k$ must necessarily lie in B and correspond to a set in the partition given by C that contains strictly less than k elements. Thus any such vertex immediately yields a solution to the original Partition-mod-$k[\#\ell]$ instance.

Mitosis gadgets. Let $k \geq 2$. We now show how to construct a small bipartite graph such that exactly one vertex on each side has degree 1 and all other vertices have degree k (or 0). This "gadget" can then be used to increase the degree of two vertices (one on each side of the bipartite graph) without adding any solutions, i.e., vertices with degree $\neq 0$

mod k.

The gadget is a bipartite graph with $k+1$ vertices on each side: $a_1, \dots, a_{k+1}$ and $b_1, \dots, b_{k+1}$. It contains all the edges $\{a_i, b_j\}$ for $i, j \leq k$, except the edge $\{a_k, b_k\}$. It also contains the edges $\{a_k, b_{k+1}\}$ and $\{a_{k+1}, b_k\}$. Thus, all vertices have degree k, except for a_{k+1} and b_{k+1} which have degree 1.

We call this the "Mitosis" gadget, because it allows us to duplicate edges that already exist. Let u and v be two vertices in a bipartite graph, one on each side. Furthermore, consider the case where there is an edge $\{u, v\}$. We would like to increase the degree of u and v by 1, but without introducing any new solutions, in particular without introducing any vertex with degree $\neq 0 \mod k$. Using the Mitosis gadget, we can just add new vertices $a_1, \dots, a_k$ and $b_1, \dots, b_k$, and identify a_{k+1} with u and b_{k+1} with v. Adding the corresponding vertices of the gadget yields a bipartite graph where the degree of u and v has increased by 1, but no new solutions have been introduced. Note that this gadget can, in particular, be used to turn a bipartite graph with multi-edges into one without them, without changing the degree of existing vertices and without adding any new solutions.

5 Relationship Between the Classes

In this section, we present some results that provide deeper insights into how the classes relate to each other. For any $k \geq 2$, $\mathsf{PF}(k)$ denotes the set of all prime factors of k. The main conceptual result is that PPA-k is entirely determined by the set of prime factors of k:

Theorem 2. *For any $k \geq 2$ we have* PPA-$k = \underset{p \in \mathsf{PF}(k)}{\&}$ PPA-p.

This equation can be understood as saying the following:

- Given a single query to an oracle for PPA-k, we can solve any problem in PPA-p for any $p \in \mathsf{PF}(k)$
- Given a single query to an oracle that solves any PPA-p problem for any $p \in \mathsf{PF}(k)$, we can solve any problem in PPA-k.

Corollary 2. *In particular, we have:*

- *For $k_1, k_2 \geq 2$, if $\mathsf{PF}(k_1) \subseteq \mathsf{PF}(k_2)$, then* PPA-$k_1 \subseteq$ PPA-k_2.
- *For all $k_1, k_2 \geq 2$,* PPA-$k_1 k_2$ = PPA-k_1 & PPA-k_2.
- *For all $k \geq 2$ and all $r \geq 1$ we have* PPA-k^r = PPA-k.

Using the PPA-$k[\#\ell]$ classes, we can formulate an even stronger and more detailed result. For any $k \geq 2$, $1 \leq \ell \leq k-1$, we define $\mathsf{PF}(k, \ell) = \mathsf{PF}(k/\gcd(k, \ell))$. In this case the conceptual result says that PPA-$k[\#\ell]$ is entirely determined by the set of prime factors of $k/\gcd(k, \ell)$.

Theorem 3. *For any integer constants k and ℓ with $k \geq 2$ and $0 < \ell < k$, it holds that*

$$\text{PPA-}k[\#\ell] = \text{PPA-}\left(\prod_{p \in \mathsf{PF}(k,\ell)} p\right)[\#1] = \bigcap_{p \in \mathsf{PF}(k,\ell)} \text{PPA-}p.$$

The proof of Theorem 3 can be found in the next section. Before we move on to that, let us briefly show that Theorem 2 follows from Theorem 3.

Proof of Theorem 2. Using Lemma 2 and Theorem 3 we can write

$$\text{PPA-}k = \mathop{\&}_{\ell=1}^{k-1} \text{PPA-}k[\#\ell] = \mathop{\&}_{\ell=1}^{k-1} \left(\bigcap_{p \in \mathsf{PF}(k,\ell)} \text{PPA-}p \right) = \mathop{\&}_{p \in \mathsf{PF}(k)} \text{PPA-}p$$

where the last equality follows by noting that $\mathsf{PF}(k,\ell) \subseteq \mathsf{PF}(k)$ for all ℓ, and $\mathsf{PF}(k,k/p) = \{p\}$ for all $p \in \mathsf{PF}(k)$. □

5.1 Proof overview

Proof of Theorem 3. All containment results follow from Theorem 4 below, except

$$\text{PPA-}\left(\prod_{p \in \mathsf{PF}(k,\ell)} p \right) [\#1] \supseteq \bigcap_{p \in \mathsf{PF}(k,\ell)} \text{PPA-}p.$$

Let $\mathsf{PF}(k,\ell) = \{p_1, \dots, p_d\}$. We will show how to combine a set of instances (C_1, m_1), $\dots$, (C_d, m_d), where (C_i, m_i) is an instance of Partition-mod-$p_i[\#1]$, into a single instance of Partition-mod-$s[\#1]$, where $s = p_1 p_2 \cdots p_d$, such that any solution to this instance yields a solution to one of the (C_i, m_i) instances. Without loss of generality, we can assume that the parameter n is the same for all (C_i, m_i) instances. Without loss of generality, we can assume that $2^n - m_i = 1 \mod s$ for all i, because we can add at most $\prod_{j \neq i} p_i$ sets of size p_i to achieve this (see the proof of Lemma 5). Note that we then have $(2^n - m_1)(2^n - m_2) \cdots (2^n - m_d) = 1 \mod s$. Furthermore, for $(x_1, \dots, x_d)$ and $(y_1, \dots, y_d)$ with $x_i, y_i \geq m_i$ we can define $x \equiv y$ if and only if $x_i \equiv_i y_i$ for all i, where $x_i \equiv_i y_i$ means that x_i and y_i lie in the same set in instance (C_i, m_i). If for all i, x_i lies in a set of size p_i in (C_i, m_i), then x will lie in a set of size s. Thus any solution yields a solution to one of the original instances. The details to fully formalise this are very similar to the proof of Lemma 6. □

In [2] Beame et al. investigated the relative proof complexity of so-called "counting principles". These counting principles are formulas that represent the fact that a set of size $\neq 0 \mod k$ cannot be partitioned into sets of size k. They investigated the relationship between these principles in terms of whether one can be proved from the other by using a constant-depth, polynomial-size Frege proof. Their main result is a full characterisation of when this is possible or not. As noted by Johnson [25], these counting formulas do not yield NP search problems, but they can be related to corresponding NP search problems (TFNP, in fact). Indeed, Johnson uses this connection to obtain some separation results between his PMOD^k classes (see Section 6) from Beame et al.'s negative results. Our contribution is using Beame et al.'s positive results in order to prove inclusion results about the PPA-$k[\#\ell]$ classes. More precisely, we modify their proofs to obtain polynomial-time reductions between our Partition-mod-$k[\#\ell]$ problems. Thus, we obtain the following analogous result:

Theorem 4. *Let $k_1, k_2 \geq 2$ and $0 < \ell_i < k_i$ for $i = 1, 2$. If $\mathsf{PF}(k_2, \ell_2) \subseteq \mathsf{PF}(k_1, \ell_1)$, then* PPA-$k_1[\#\ell_1] \subseteq$ PPA-$k_2[\#\ell_2]$.

Proof. From Lemma 1 we know that PPA-$k_i[\#\ell_i]$ = PPA-$k_i[\# \gcd(k_i, \ell_i)]$ for $i = 1, 2$. The result then follows from a few technical lemmas proved in Appendix A:

$$\text{PPA-}k_1[\#\gcd(k_1,\ell_1)] \underset{\text{L 5}}{\subseteq} \text{PPA-}\frac{k_1}{\gcd(k_1,\ell_1)}[\#1] \underset{\text{L 6}}{\subseteq} \text{PPA-}\frac{k_2}{\gcd(k_2,\ell_2)}[\#1]$$
$$\underset{\text{L 4}}{\subseteq} \text{PPA-}k_2[\#\gcd(k_2,\ell_2)]$$

□

6 Johnson's PMOD^k Classes and Oracle Separations

Inspired by the definition of the PPA-complete problem Lonely [3], Buss and Johnson [5] defined TFNP problems called MOD^p to represent arguments modulo some prime p. Their main motivation was to use these problems to show separations (in the type 2 setting) between Turing reductions with m oracle queries and Turing reductions with $m+1$ oracle queries. In his thesis [25], Johnson generalised the definition of MOD^k to any $k \geq 2$ and defined corresponding classes PMOD^k. He also proved some separations between these classes and other TFNP classes in the type 2 setting (which yield oracle separations in the standard setting). It seems that Johnson was not aware of Papadimitriou's [32] PPA-p classes.

In this section, we study the classes PMOD^k and prove a characterisation in terms of the classes PPA-p. In particular, we show that PMOD^k does not capture the full strength of arguments modulo k, when k is not a prime power. This characterisation also allows us to use Johnson's separations to obtain some oracle separations involving PPA-k and other TFNP classes.

Informally, the problem MOD^k can be defined as follows. We are given a partition of $\{0,1\}^n$ into subsets and the goal is to find one of these subsets that has size $\neq k$. If k is not a power of 2, then such a subset must exist. If k is a power of 2, then we instead consider $\{0,1\}^n \setminus \{0^n\}$ and the problem remains total.

Definition 9 (MOD^k [5, 25])**.** Let $k \geq 2$. The problem MOD^k is defined as: given a Boolean circuit C with n inputs and outputs,

- If k is not a power of 2: Find
  - $x \in \{0,1\}^n$ and $d \in \mathbb{N}$ such that $C^d(x) = x$ and $d|k$, $d \neq k$
  - or $x \in \{0,1\}^n$ such that $C^k(x) \neq x$
- If k is a power of 2: Let additionally $C(0^n) = 0^n$ and find
  - $x \in \{0,1\}^n \setminus \{0^n\}$ and $d \in \mathbb{N}$ such that $C^d(x) = x$ and $d|k$, $d \neq k$
  - or $x \in \{0,1\}^n$ such that $C^k(x) \neq x$

where $C^\ell(x) = C(C(\dots C(x))\dots)$ (ℓ times) and $d|k$ means that d divides k.

Definition 10 (PMOD^k [25])**.** For any $k \geq 2$, the class PMOD^k is defined as the set of all TFNP problems that many-one reduce to MOD^k.

Note that the problem MOD^k is a special case of our problem Partition-mod-k (which was indeed inspired by this definition). As a result, we immediately get that PMOD$^k \subseteq$ PPA-k. Unless k is a prime power, we don't expect this to hold with equality. The intuition is that restricting the size of the base set to always be a power 2 has the effect of only achieving a subset of the possible ℓ-parameter values of PPA-$k[\#\ell]$. Namely, only $\ell \in \{2^n \mod k : n \in \mathbb{N}\}$ are achieved (for k not a power of 2).

Johnson proves a lemma [25, Lemma 7.4.5] that gives some idea of how the PMOD^k classes relate to each other. It can be stated as follows: if $k = p_1 p_2 \dots p_r$, where the p_i are distinct primes, then PMOD$^k = \cap_i$PMOD^{p_i}. He proves this if all $p_i \neq 2$ and claims that the proof also works if some $p_i = 2$. However, if some $p_i = 2$ then the proof does not work. This is easy to see, since our results below prove that PMOD^6 = PMOD^3 which is not equal to PMOD$^2 \cap$ PMOD^3, unless PMOD$^2 \subseteq$ PMOD^3. However, Johnson proves that PMOD$^2 \not\subseteq$ PMOD^3 in the type 2 setting.

The following result provides a full characterisation of PMOD^k in terms of the classes PPA-p.

Theorem 5. *Let $k \geq 2$.*

- *If k is not a power of 2, then* PMOD^k = PPA-$\widetilde{k}[\#1] = \cap_{p \in \mathsf{PF}(\widetilde{k})}$PPA-$p$ *where $\widetilde{k}$ is the largest odd divisor of k.*
- *If k is a power of 2, then* PMOD^k = PPA-2.

The proof of Theorem 5 is given below in Section 6.1.

Corollary 3. *In particular, we have:*

- *for all primes p and $r \geq 1$,* PMOD$^{p^r}$ = PPA-p^r = PPA-p
- *for all $k \geq 2$,* PMOD^{2k} = PMOD^k
- *for all odd $k \geq 3$,* PMOD^k = PPA-$k[\#1] = \cap_{p \in \mathsf{PF}(k)}$PPA-$p$

If k is a prime power, then PMOD^k is the same as PPA-k. However, for other values of k, we argue that PMOD^k fails to capture the full strength of arguments modulo k. For example, PMOD^{15} = PPA-15[#1] = PPA-3 $\cap$ PPA-5, whereas PPA-15 = PPA-3 & PPA-5. This means that PPA-15 can solve any problem that lies in PPA-3 or PPA-5, while PMOD^{15} can only solve problems that lie *both* in PPA-3 and PPA-5. In particular, if PMOD^{15} = PPA-15, then it would follow that PPA-3 = PPA-5, which is not believed to hold (see oracle separations below). Even worse perhaps, is the fact that PMOD^{2k} = PMOD^k for any $k \geq 2$. In particular, this means that PMOD^6 = PMOD^3, which indicates that PMOD^6 does not really capture arguments modulo 6.

Nevertheless, Johnson's oracle separation results (obtained from the corresponding type 2 separations as in [3]) also yield corresponding results for the PPA-k classes (using Theorem 5). We briefly mention a few of the results obtained this way. See Johnson [25, Chapter 8] for additional results. Relative to any generic oracle (see [3]):

- PPA-$p \not\subseteq$ PPA-q for any distinct primes p, q
- PPA-$k \not\subseteq$ PPP, PPA-$k \not\subseteq$ PLS, PPA-$k \not\subseteq$ PPADS for any $k \geq 2$
- PPP $\not\subseteq$ PPA-p, PLS $\not\subseteq$ PPA-p for any prime p

6.1 Proof of Theorem 5

For $k = 2$, MOD^2 corresponds to the PPA-complete problem Lonely [3], and thus PMOD^2 = PPA = PPA-2. Let $r \geq 2$. Consider an instance (C, m) of Partition-mod-$2^r[\#(2^r - 1)]$ on the set $\{0,1\}^n$. Without loss of generality, assume $n \geq r$. Then $2^n = 0$

mod 2^r and thus $m = 2^n - (2^n - m) = -(2^r - 1) \mod 2^r = 1 \mod 2^r$. This means that we can (efficiently) partition $\{0, 1, \ldots, m-1\}$ into subsets of size 2^r, leaving only $0 = 0^n$ out. Thus, we have reduced PARTITION-MOD-$2^r[\#(2^r - 1)]$ to MOD^{2^r}. Since PPA-$2^r[\#(2^r - 1)]$ = PPA-2 (Theorem 3), we obtain PPA-2 $\subseteq$ PMOD^{2^r}. On the other hand we also have $\text{PMOD}^{2^r} \subseteq$ PPA-2^r = PPA-2 by Corollary 2.

Consider some $k \geq 3$ that is not a power of 2. First, let us show that $\text{PMOD}^{2k} = \text{PMOD}^k$. MOD^{2k} reduces to MOD^k by splitting every subset into two subsets of size k (or less, if the subset has size $< 2k$). Conversely, consider an instance of MOD^k on the set $\{0, 1\}^n$. Make a copy of the instance, thus obtaining an instance on the set $\{0, 1\}^{n+1}$. For every subset of the original instance, take the union with its copy. If the subset had size k, the new subset has size $2k$. Thus, we have reduced to MOD^{2k}.

Let $k \geq 3$ be coprime with 2. We will show PMOD^k = PPA-$k[\#1]$. Consider an instance of MOD^k on the set $\{0, 1\}^n$. Since k and 2 are coprime, there exists $i \in \{0, \ldots, k-1\}$ such that $2^{n+i} = 1 \mod k$ (e.g., by using Euler's theorem). Thus, we take 2^i copies of the instance and obtain an instance on the set $\{0, 1\}^{n+i}$, which is an instance of PARTITION-MOD-$k[\#1]$ (with $m = 0$), since $2^{n+i} = 1 \mod k$. Conversely, consider an instance (C, m) of PARTITION-MOD-$k[\#1]$ on the set $\{0, 1\}^n$. As before, there exists $i \in \{0, \ldots, k-1\}$ such that $2^{n+i} = 1 \mod k$. We construct an instance C' of MOD^k on $\{0, 1\}^{n+i}$ as follows. The element $x \in \{0, 1\}^n$ of the original instance corresponds to the element $1^i x \in \{0, 1\}^n$ of the new instance. If $x \geq m$, set $C'(1^i x) = 1^i C(x)$. The number of elements that have not yet been assigned to a subset is $m + (2^i - 1)2^n = (m - 2^n) + 2^{n+i} = 0 \mod k$. Thus, we can efficiently partition them into subsets of size k without introducing any solution. We have obtained an instance of MOD^k.

7 Many-one vs Turing Reductions

Theorem 6. *For any prime $p \geq 2$,* PPA-p *is closed under Turing reductions.*

In particular, PPA-p^r = PPA-p is also closed under Turing reductions. The proof of Theorem 6 can be found in Section 7.1. Furthermore, we also obtain:

Corollary 4. *For all $k \geq 2$ and $0 < \ell < k$,* PPA-$k[\#\ell]$ *is closed under Turing reductions.*

Proof of Corollary 4. Using Theorem 3, we have PPA-$k[\#\ell] = \bigcap_{p=1}^{d}$ PPA-p_i, where we let $\{p_1, \ldots, p_d\} = \mathsf{PF}(k, \ell)$. Consider a Turing reduction from some problem to PPA-$k[\#\ell]$. Since PPA-$k[\#\ell] \subseteq$ PPA-p_i, this yields a Turing reduction to PPA-p_i, in particular. By Theorem 6, it follows that there exists a many-one reduction to PPA-p_i, i.e., the problem lies in PPA-p_i. Since this holds for all p_i, the result follows. □

If k is not a prime power, then it is not known whether PPA-k is closed under Turing reductions. Using our results from Section 6, we can actually provide an oracle separation between PPA-k and the Turing-closure of PPA-k, i.e., an oracle under which PPA-k is not closed under Turing reductions. Let $R_1, \ldots, R_k$ be TFNP problems. Following Johnson [25] we define $\bigotimes_{j=1}^{k} R_j$ as the problem: given instances $(I_1, \ldots, I_k)$, where I_j is an instance of R_j, solve I_j for all j. As we did with the & operation, with a slight abuse of notation, we can also use the operation $\otimes$ with the PPA-k classes. In [25, Theorem 7.6.1], Johnson proved that for $m \geq 2$ and distinct primes $p_1, \ldots, p_m$, $\bigotimes_{i=1}^{m} \text{MOD}^{p_i}$ does not many-one reduce to $\&_{i=1}^{m} \text{MOD}^{p_i}$ in the type 2 setting. Together with our Theorems 2 and 5 this yields:

Theorem 7. *Let $k \geq 2$ not a power of a prime. Relative to any generic oracle, it holds that $\bigotimes_{p \in \mathsf{PF}(k)}$ PPA-$p \not\subseteq$ PPA-k. In particular, relative to any generic oracle,* PPA-k *is not closed under Turing reductions.*

$S = \bigotimes_{p \in \mathsf{PF}(k)}$ PPA-p corresponds to solving PPA-p for all prime factors p of k *simultaneously.* In particular, this can be done by using $|\mathsf{PF}(k)|$ queries to PPA-k, i.e., a Turing reduction to PPA-k. Thus, S lies in the Turing closure of PPA-k, but not in PPA-k (relative to any generic oracle).

7.1 Proof of Theorem 6

We essentially apply the same technique that was used by Buss and Johnson [5] to show that PPA, PPAD, PPADS and PLS are closed under Turing reductions.

Let Π be a problem that Turing-reduces to some problem in PPA-p. This means that there exists a Turing machine M with access to a PPA-p-oracle that solves Π in polynomial time. Since Imbalance-mod-p is PPA-p-complete (Theorem 1), we assume that the oracle provides solutions to Imbalance-mod-p instances. Our goal is to show that all the oracle queries can be combined into a single one. Indeed, a Turing reduction that always uses a single oracle query immediately yields a many-one reduction. Thus, by the definition of PPA-p, this would yield $\Pi \in$ PPA-p.

We begin by showing that any Imbalance-mod-p-instance can be efficiently transformed into an instance that has a particular form, namely: the starting node has imbalance $+1$ (in-degree 0 and out-degree 1), and any solution has imbalance -1 (in-degree 1 and out-degree 0). This can be achieved by the following steps:

1. Ensure that all vertices have in- and out-degree at most p (by splitting vertices into multiple copies).

2. Ensure that any unbalanced vertex has in- or out-degree 0 (by creating a copy that will take all the edges that yield the imbalance).

3. Since p is prime, we can ensure that the starting vertex has imbalance $+1$.

4. Ensure that all vertices that have imbalance $\neq 0 \mod p$, actually have imbalance $+1$ or -1 (by splitting every such vertex into p vertices, each getting at most one edge).

5. Transform every solution that has imbalance $+1$ into $p-1$ solutions with imbalance -1 instead (by pointing to $p-1$ new vertices).

From now on we assume that all Imbalance-mod-p-instances have this form. Given an instance I of problem Π, let (G_1^I, s_1^I) denote the first oracle query made by M on input I, where G_1^I is the Imbalance-mod-p graph (represented implicitly by circuits) and s_1^I is the starting vertex. From now on we omit the superscript I for better readability. For any solution t_1 to (G_1, s_1), let $(G_2(t_1), s_2(t_1))$ be the second oracle query made by M, if the first query returned t_1. We construct a big graph G that contains a copy of G_1 and a copy of $G_2(t_1)$ for each solution t_1 of (G_1, s_1). A vertex u in $G_2(t_1)$ is represented as (t_1, u) in G. For each such t_1, we add an edge from t_1 to $(t_1, s_2(t_1))$. Note that these two vertices are now balanced. Thus, the instance (G, s_1) has the following property: all solutions are of the form (t_1, t_2), where t_1 is a solution to (G, s_1), and t_2 is a solution to $(G_2(t_1), s_2(t_1))$. The straightforward generalisation of this construction for a polynomial number of queries (instead of 2), yields a graph G such that any solution yields consistent query answers for a complete run of M on input I. Thus, we obtain a Turing reduction that only needs to make one oracle query and then simulates M with these query answers.

It remains to show that this graph G can be constructed in polynomial time from I, i.e., we can efficiently construct circuits that compute the edges incident on any given node. This is easy to see, because any node contains enough information to simulate a run of M up to the point that is needed to determine the neighbours in G. We omit the full details, since the formal arguments are analogous to the ones in the corresponding proofs in [5, 25].

Acknowledgements

I would like to thank Aris Filos-Ratsikas and Paul Goldberg for helpful discussions, as well as an anonymous reviewer for suggestions that helped improve the presentation of the paper. This work was supported by an EPSRC doctoral studentship (Reference 1892947).

References

[1] N. Alon. Splitting necklaces. *Advances in Mathematics*, 63(3):247–253, 1987. doi:10.1016/0001-8708(87)90055-7.

[2] P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, and P. Pudlák. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. *Proceedings of the London Mathematical Society*, s3-73(1):1–26, 1996. doi:10.1112/plms/s3-73.1.1.

[3] P. Beame, S. Cook, J. Edmonds, R. Impagliazzo, and T. Pitassi. The relative complexity of NP search problems. *Journal of Computer and System Sciences*, 57(1): 3–19, 1998. doi:10.1006/jcss.1998.1575.

[4] J. Buresh-Oppenheim and T. Morioka. Relativized NP search problems and propositional proof systems. In *Proceedings of the 19th IEEE Conference on Computational Complexity (CCC)*, pages 54–67, 2004. doi:10.1109/CCC.2004.1313795.

[5] S. R. Buss and A. S. Johnson. Propositional proofs and reductions between NP search problems. *Annals of Pure and Applied Logic*, 163(9):1163–1182, 2012. doi:10.1016/j.apal.2012.01.015.

[6] X. Chen, D. Dai, Y. Du, and S.-H. Teng. Settling the complexity of Arrow-Debreu equilibria in markets with additively separable utilities. In *Proceedings of the 50th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 273–282, 2009. doi:10.1109/FOCS.2009.29.

[7] X. Chen, X. Deng, and S.-H. Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3):14:1–14:57, 2009. doi:10.1145/1516512.1516516.

[8] X. Chen, D. Paparas, and M. Yannakakis. The complexity of non-monotone markets. *Journal of the ACM*, 64(3):20:1–20:56, 2017. doi:10.1145/3064810.

[9] B. Codenotti, A. Saberi, K. Varadarajan, and Y. Ye. The complexity of equilibria: Hardness results for economies via a correspondence with games. *Theoretical Computer Science*, 408(2-3):188–198, 2008. doi:10.1016/j.tcs.2008.08.007.

[10] C. Daskalakis and C. Papadimitriou. Continuous local search. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 790–804, 2011. doi:10.1137/1.9781611973082.62.

[11] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, 39(1):195–259, 2009. doi:10.1137/070699652.

[12] M. de Longueville and R. T. Živaljević. The Borsuk-Ulam-property, Tucker-property and constructive proofs in combinatorics. *Journal of Combinatorial Theory, Series A*, 113(5):839–850, 2006. doi:10.1016/j.jcta.2005.08.002.

[13] D. Dumrauf, B. Monien, and K. Tiemann. Multiprocessor scheduling is PLS-complete. In *Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS)*, pages 1–10, 2009. doi:10.1109/HICSS.2009.317.

[14] A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure Nash equilibria. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, pages 604–612, 2004. doi:10.1145/1007352.1007445.

[15] J. Fearnley, S. Gordon, R. Mehta, and R. Savani. Unique End of Potential Line. *Journal of Computer and System Sciences*, 114:1–35, 2020. doi:10.1016/j.jcss.2020.05.007.

[16] J. Fearnley, P. W. Goldberg, A. Hollender, and R. Savani. The complexity of gradient descent: CLS = PPAD ∩ PLS. In *Proceedings of the 53rd ACM Symposium on Theory of Computing (STOC)*, pages 46–59, 2021. doi:10.1145/3406325.3451052.

[17] A. Filos-Ratsikas and P. W. Goldberg. Consensus halving is PPA-complete. In *Proceedings of the 50th ACM Symposium on Theory of Computing (STOC)*, pages 51–64, 2018. doi:10.1145/3188745.3188880.

[18] A. Filos-Ratsikas and P. W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proceedings of the 51st ACM Symposium on Theory of Computing (STOC)*, pages 638–649, 2019. doi:10.1145/3313276.3316334.

[19] A. Filos-Ratsikas, A. Hollender, K. Sotiraki, and M. Zampetakis. A topological characterization of modulo-p arguments and implications for necklace splitting. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2615–2634, 2021. doi:10.1137/1.9781611976465.155.

[20] P. W. Goldberg and A. Hollender. The Hairy Ball problem is PPAD-complete. *Journal of Computer and System Sciences*, 122:34–62, 2021. doi:10.1016/j.jcss.2021.05.004.

[21] P. W. Goldberg and C. H. Papadimitriou. Towards a unified complexity theory of total functions. *Journal of Computer and System Sciences*, 94:167–192, 2018. doi:10.1016/j.jcss.2017.12.003.

[22] M. Göös, P. Kamath, K. Sotiraki, and M. Zampetakis. On the complexity of modulo-q arguments and the Chevalley-Warning theorem. In *Proceedings of the 35th Computational Complexity Conference (CCC)*, pages 19:1–19:42, 2020. doi:10.4230/LIPIcs.CCC.2020.19.

[23] E. Jeřábek. Integer factoring and modular square roots. *Journal of Computer and System Sciences*, 82(2):380–394, 2016. doi:10.1016/j.jcss.2015.08.001.

[24] E. Jeřábek. Theoretical Computer Science Stack Exchange, 2017. `https://cstheory.stackexchange.com/q/37794` (version: 2017-03-20).

[25] A. S. Johnson. *Reductions and propositional proofs for total NP search problems*. PhD thesis, UC San Diego, 2011. URL https://escholarship.org/uc/item/89r774x7.

[26] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

[27] S. Kintali, L. J. Poplawski, R. Rajaraman, R. Sundaram, and S. Teng. Reducibility among fractional stability problems. *SIAM Journal on Computing*, 42(6):2063–2113, 2013. doi:10.1137/120874655.

[28] M. W. Krentel. Structure in locally optimal solutions. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 216–221, 1989. doi:10.1109/SFCS.1989.63481.

[29] N. Megiddo and C. H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. doi:10.1016/0304-3975(91)90200-L.

[30] F. Meunier. Simplotopal maps and necklace splitting. *Discrete Mathematics*, 323: 14–26, 2014. doi:10.1016/j.disc.2014.01.008.

[31] T. Morioka. Classification of search problems and their definability in bounded arithmetic. Master's thesis, University of Toronto, 2001. URL https://www.collectionscanada.ca/obj/s4/f2/dsk3/ftp04/MQ58775.pdf.

[32] C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994. doi:10.1016/S0022-0000(05)80063-7.

[33] C. H. Papadimitriou, A. A. Schäffer, and M. Yannakakis. On the complexity of local search. In *Proceedings of the 22nd ACM Symposium on Theory of Computing (STOC)*, pages 438–445, 1990. doi:10.1145/100216.100274.

[34] P. Pudlák. On the complexity of finding falsifying assignments for Herbrand disjunctions. *Archive for Mathematical Logic*, 54(7-8):769–783, 2015. doi:10.1007/s00153-015-0439-6.

[35] K. Sotiraki, M. Zampetakis, and G. Zirdelis. PPP-completeness with connections to cryptography. In *Proceedings of the 59th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 148–158, 2018. doi:10.1109/FOCS.2018.00023.

A Technical Lemmas for Theorem 4

The proof ideas from [2] are used to construct some of these reductions.

Lemma 3. *Let $k \geq 2$ and $r \geq 1$. For all $0 < \ell < kr$ with $\ell \neq 0 \mod k$, it holds that* $\mathrm{PPA}\text{-}kr[\#\ell] \subseteq \mathrm{PPA}\text{-}k[\#(\ell \mod k)]$.

Proof. Consider any instance of Bipartite-mod-$kr[\#\ell]$. Split every vertex v into r versions $v_1, \ldots, v_r$ and assign every incident edge to exactly one of these versions, e.g., by ordering the neighbours in increasing lexicographic order. Then exactly one version of the known starting vertex will have degree $\ell \mod k$ and all its other versions will have degree

0 or k. In the new graph, any vertex that has degree not in $\{0, k\}$ (apart from this one version of the starting vertex) will immediately yield a solution of the original instance. Thus, we have reduced to an instance of BIPARTITE-MOD-$k[\#(\ell \mod k)]$. □

Lemma 4. *Let $k \geq 2$ and $r \geq 1$. For any $0 < \ell < k$ it holds that* PPA-$k[\#\ell] \subseteq$ PPA-$kr[\#\ell r]$.

Proof. Consider any instance of BIPARTITE-MOD-$k[\#\ell]$. Assign weight r to every edge. Clearly, the starting vertex now has degree ℓr and any other vertex with degree not in $\{0, kr\}$ yields a solution to the original instance. Finally, note that we can remove the weights without changing the degrees and adding any new solutions by using the "mitosis" gadgets. □

Lemma 5. *Let $k \geq 2$ and $\ell \geq 1$. Then* PPA-$k\ell[\#\ell] \subseteq$ PPA-$k[\#1]$.

Proof. We reduce PARTITION-MOD-$k\ell[\#\ell]$ to PARTITION-MOD-$k[\#1]$ by adapting the proof in [2, Lemma 2.3] to obtain a reduction. Consider an instance of PARTITION-MOD-$k\ell[\#\ell]$, i.e., a partition of the set of integers $[m, 2^n - 1]$ given by a circuit C for some m such that $2^n - m = \ell \mod k\ell$. This means that there exists some integer $\alpha \in [0, 2^{n-1}]$ such that $2^n - m = \ell + \alpha k\ell$. Clearly, there exists some integer $\beta \in [0, (\ell-1)! - 1]$ such that $\alpha + \beta = 0 \mod (\ell-1)!$, which implies $2^n - m + \beta k\ell = \ell \mod (k \cdot \ell!)$ (if $\ell \leq 2$, then this holds with $\beta = 0$). Thus, if we add β sets of size $k\ell$, the size of the ground set will be $= \ell \mod (k \cdot \ell!)$. Assuming that n is large enough such that $2^n \geq k \cdot \ell! \geq \beta k\ell$, we can achieve this by letting $n' = n + 1$, $m' = m + 2^n - \beta k\ell$ and extending C to also include the additional β sets of size $k\ell$. It is easy to see that this can be done efficiently and will yield a partition of $[m', 2^{n+1} - 1]$ such that any mistake immediately yields a mistake in the original partition. Thus, we can assume without loss of generality that $2^n - m = \ell \mod (k \cdot \ell!)$.

Let S denote the set of all subsets of $\{m, m+1, \ldots, 2^n - 1\}$ of size exactly ℓ. Note that $|S| = \binom{2^n - m}{\ell} = \prod_{i=0}^{\ell-1} \frac{2^n - m - i}{\ell - i} = 1 \mod k$, since $2^n - m - i = \ell - i \mod k(\ell - i)$. We will now describe how to construct a partition of S into sets of size k such that any mistake yields a solution of the original instance.

Recall that we can assume that $C^{k\ell}(x) = x$ for all x. Thus, every x yields an orbit $O(x) = \{x, C(x), C^2(x), \ldots, C^{k\ell-1}(x)\}$ of size at most $k\ell$. In particular, we can pick the lexicographically smallest element of every orbit to be its representative. Denote by $R(x)$ the representative of the orbit containing x. We then have that $R(x) = R(y)$, if and only if x and y lie in the same orbit, i.e., in the same set in the original partition. For $a, b \in S$ we write $a \equiv b$ if a and b contain exactly the same number of elements from each set of the original partition. This can be checked efficiently by computing the representative of each element in a, ordering these lexicographically (with repetitions), doing the same for b and checking if the two lists are identical. This is an equivalence relation and we denote the equivalence class of $a \in S$ under $\equiv$ by $[a]$.

For any $a \in S$ there exist distinct representatives $x_1, \ldots, x_s$, $s \leq \ell$, and $\alpha_1, \ldots, \alpha_s \geq 1$ with $\sum_{i=1}^{s} \alpha_i = \ell$ such that a contains exactly α_i elements from the orbit represented by x_i, for all i. Thus, the size of $[a]$ is exactly $\prod_{i=1}^{s} \binom{k\ell}{\alpha_i}$, assuming that the orbits of $x_1, \ldots, x_s$ all have size $k\ell$, i.e., do not yield a solution to the original problem. It was shown in the proof of [2, Lemma 2.3] that this quantity is a multiple of k. Thus, the equivalence class of a can be perfectly partitioned into sets of size k. We now describe a way to do this explicitly and efficiently. Assume that the representatives $x_1, \ldots, x_s$ are in increasing lexicographic order. Find the smallest index i such that k divides $\binom{k\ell}{\alpha_i}$. Let F denote an arbitrary fixed efficient bijection between $\{0,\dots,\binom{k\ell}{\alpha_i}-1\}$ and $\binom{O(x_i)}{\alpha_i}$, where this denotes the set of all subsets of $O(x_i)$ of size exactly α_i.

The circuit C' determines the image of $a \in S$ by first computing $x_1,\dots,x_s$ and $\alpha_1,\dots,\alpha_s$ as described above, and determining the smallest index i as explained above. Let $a_i = a \cap O(x_i)$. The circuit outputs

$$(a \setminus a_i) \cup F(\lfloor F^{-1}(a_i)/k \rfloor \cdot k + (F^{-1}(a_i) + 1 \mod k)).$$

It is easy to check that as long as $|O(x_j)| = k\ell$ for all j, this procedure partitions $[a]$ into sets of size k. The last step is to set $m' = 2^{n\ell} - |S|$ and construct an efficient bijection between S and $\{2^{n\ell} - |S|,\dots,2^{n\ell}-1\}$, which is easy to do. □

Lemma 6. *Let $k_1, k_2 \geq 2$. If all prime factors of k_2 also divide k_1, then* PPA-k_1[#1] $\subseteq$ PPA-k_2[#1].

Proof. Similarly to our proof of Lemma 5, we adapt the proof of the corresponding statement for the counting formulas from Beame et al. [2, Lemma 2.5] in order to obtain a polynomial-time reduction.

Since all prime factors of k_2 divide k_1, there exists some ℓ (bounded by a constant, e.g., k_2) such that k_2 divides k_1^ℓ. From Lemma 3 we know that PPA-k_1^ℓ[#1] $\subseteq$ PPA-k_2[#1]. Thus, it suffices to show that PPA-k_1[#1] $\subseteq$ PPA-k_1^ℓ[#1]. We write $k = k_1$ from now on.

Consider an instance (C, m) of PARTITION-MOD-k[#1]. Since $2^n - m = 1 \mod k$, it follows that there exists some r such that $(2^n - m)^r = 1 \mod k^\ell$ by Euler's totient theorem. Pick such an $r \geq \ell$ – any large enough multiple of the totient $\phi(k^\ell)$ will do – and note that r is bounded by some constant, since both ℓ and $\phi(k^\ell)$ are.

Assume without loss of generality that $C^k(x) = x$ for all x. We construct a partition of $\{m,\dots,2^n-1\}^r$ explicitly as follows. For any $(x_1,\dots,x_r) \in \{m,\dots,2^n-1\}^r$, let $i \in \{1,\dots\ell\}$ denote the largest index such that x_j is the lexicographically smallest element in the orbit of x_j under C (i.e., x_j is the representative of $O(x_j)$), for all $j < i$. If there is no such x_j, set $i = 1$. If $i > \ell$, set $i = \ell$. The circuit C' computes $C'(x_1,\dots,x_r) = (C(x_1), C(x_2),\dots,C(x_i), x_{i+1},\dots x_r)$. It is easy to see that if the orbits $O(x_j)$ under C all have size k, then this yields an orbit of size k^ℓ. Thus, any orbit in the new instance that has size different from k^ℓ immediately yields some orbit of the original instance that does not have size k.

The final step is to set $m' = 2^{nr} - (2^n - m)^r$ and construct an efficient bijection between $\{m',\dots,2^{nr}-1\}$ and $\{m,\dots,2^n-1\}^r$, which is easy to do. □