ON THE CONNECTION BETWEEN INTERVAL SIZE FUNCTIONS AND PATH COUNTING

Evangelos Bampas, Andreas-Nikolas Göbel, Aris Pagourtzis, and Aris Tentes

Abstract. We investigate the complexity of hard (#P-complete) counting problems that have easy decision version. By "easy decision," we mean that deciding whether the result of counting is non-zero is in P. This property is shared by several well-known problems, such as counting the number of perfect matchings in a given graph or counting the number of satisfying assignments of a given DNF formula. We focus on classes of such hard-to-count easy-to-decide problems which emerged through two seemingly disparate approaches: one taken by Hemaspaandra *et al.* (2007, SIAM J. Comput. 36(5), 1264–1300), who defined classes of functions that count the size of intervals of ordered strings, and one followed by Kiayias *et al.* (2001, Lect. Notes Comput. Sc. 2563, 453–463), who defined the class TotP, consisting of functions that count the total number of paths of NP computations. We provide inclusion and separation relations between TotP and interval size counting classes, by means of new classes that we define in this work. Our results imply that many known #P-complete problems with easy decision are contained in the classes defined by Hemaspaandra et al, but are unlikely to be complete for these classes under reductions under which these classes are downward closed, e.g. parsimonious reductions. This, applied to the #MonSat problem, partially answers an open question of Hemaspaandra et al. We also define a new class of interval size functions which strictly contains FP and is strictly contained in TotP under reasonable complexity-theoretic assumptions. We show that this new class contains hard counting problems.

The final publication is available at Springer via the following URL: `http://dx.doi.org/10.1007/s00037-016-0137-8`. *Theorem numbering in this accepted manuscript is different from that of the final publication.*

Keywords. Counting functions, interval size functions, path counting, feasibility constraints, easy decision.

Subject classification. 03D15, 06A05, 68Q05, 68Q10, 68Q15, 68Q17.

1. Introduction

Valiant's pioneering work on counting problems associated with NP computations (Valiant 1979a) revealed the existence of functions that are quite hard to compute exactly (#P-complete), despite the fact that deciding whether the function value is nonzero is easy (in P). This category contains the problem of evaluating the permanent of a 0-1 matrix (PERMANENT), which is equivalent to counting perfect matchings in bipartite graphs, the problem of counting satisfying assignments to monotone Boolean formulae in 2-CNF form (#MON2SAT), and many more (Valiant 1979b). A common feature of all these problems is that their #P completeness property holds under the Cook (polynomial-time Turing) reduction, which blurs structural differences between complexity classes, since counting classes are not necessarily downward closed under the Cook reduction. For example, PERMANENT is complete under the Cook reduction not only for #P, but also for the whole counting version of the Polynomial Hierarchy (Toda 1991; Toda & Watanabe 1992) and for subclasses of #P (Kiayias *et al.* 2001). Hence, #P is not considered to be the most appropriate class to describe the complexity of these problems.

During the last three decades, there has been increasing interest in the complexity of counting problems with easy decision version, due to their relevance in various disciplines, and to the fact that many of them can be well approximated (Jerrum *et al.* 2004; Karp *et al.* 1989; Liu *et al.* 2014; Weitz 2006). A path taken by some researchers is to identify subclasses of #P containing such problems (Àlvarez & Jenner 1993; Dyer *et al.* 2003; Hemaspaandra *et al.* 2007; Kiayias *et al.* 2001; Pagourtzis 2001; Pagourtzis & Zachos 2006; Saluja *et al.* 1995) and study their properties and relations among them.

In this work, we investigate the relations among subclasses of #P defined and studied through two independent lines of research: (a) classes $\mathsf{IF}_{\mathsf{p}}^{\prec}$ and $\mathsf{IF}_{\mathsf{t}}^{\prec}$ (Hemaspaandra *et al.* 2007) that consist of functions that count the size of intervals of strings under polynomial-time decidable partial or total (respectively) orders that admit *efficient adjacency checks*, and (b) the class TotP (Kiayias *et al.* 2001) that consists of functions that count the total number of paths of Nondeterministic Polynomial-time Turing Machines (NPTMs), and the class #PE (Pagourtzis 2001) that contains all functions of #P, for which deciding whether the function value is nonzero is easy (in P). Since it is clear from properties of $\mathsf{IF}_{\mathsf{p}}^{\prec}$ shown in (Hemaspaandra *et al.* 2007) that $\mathsf{IF}_{\mathsf{p}}^{\prec} = \#\mathsf{PE}$, we turn our focus to the relation between $\mathsf{IF}_{\mathsf{t}}^{\prec}$ and TotP, which are subclasses of $\mathsf{IF}_{\mathsf{p}}^{\prec}$. To this end, we define new interval size function classes by replacing efficient adjacency checks with other suitable feasibility constraints. In summary, we prove the following:

- TotP is equal to $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$, i.e., the class of interval size functions defined on total *p-orders* with efficiently computable lexicographically nearest function. This new characterization of TotP functions supplies formal evidence for the fact that, among several structural properties of TotP computations, being able to find a computation path whose encoding is lexicographically closest to a given string actually characterizes the class. Moreover, this characterization resolves in a strong sense the question of comparing TotP to the interval size function framework of Hemaspaandra *et al.* (2007).

- $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$, hence also TotP, is contained in $\mathsf{IF}_{\mathsf{t}}^{\prec}$. The inclusion is strict unless $\mathsf{P} = \mathsf{UP} \cap \mathsf{coUP}$. This, among others, implies that several problems that lie in TotP are unlikely to be $\mathsf{IF}_{\mathsf{t}}^{\prec}$-complete via reductions under which TotP is downward closed (for example, under Karp reductions); in particular, the class of problems that reduce to #MonSat by such reductions is strictly contained in $\mathsf{IF}_{\mathsf{t}}^{\prec}$ unless $\mathsf{P} = \mathsf{UP} \cap \mathsf{coUP}$. This partially answers an open question posed in Hemaspaandra *et al.* (2007), concerning the downward closure, under various reductions, of #MonSat, of $\mathsf{IF}_{\mathsf{t}}^{\prec}$, and of $\mathsf{IF}_{\mathsf{p}}^{\prec}$ (see Section 6 for a more detailed discussion).

- We define a new class, namely $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$, that lies between FP and $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}} = \mathsf{TotP}$. We prove that the inclusions are strict, assuming $\mathsf{FP} \neq \#\mathsf{P}$ and $\mathsf{P} \neq \mathsf{NP}$ respectively. We also show that $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$ contains hard counting problems: we define $\#\mathrm{SAT}_{+2^n}$, a problem in $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$ which is $\#\mathsf{P}$-complete under Cook reductions. We also show that any $\#\mathsf{P}$ function can be obtained by subtracting a function in FP from a function in $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$. Therefore, $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$ is Cook-interreducible with TotP, $\mathsf{IF}_{\mathsf{t}}^{\prec}$, $\mathsf{IF}_{\mathsf{p}}^{\prec} = \#\mathsf{PE}$, and $\#\mathsf{P}$.

In order to obtain inclusions between complexity classes, we describe nondeterministic computations that employ feasibility constraints on string orders as oracles. We also define appropriate string orders based on structural properties of NPTMs. For our separation results, we make use of operators that act on function classes yielding classes of decision problems.

2. Definitions and preliminaries

We assume a fixed alphabet Σ, conventionally $\Sigma = \{0, 1\}$. For $a \in \Sigma$ and k a non-negative integer, we use the standard notation a^k for the string consisting of k repetitions of a (a^0 is the empty string ε). The symbol $\Sigma^\star$ denotes the set of all finite strings over the alphabet Σ. The length of a string $x \in \Sigma^\star$ is denoted by $|x|$. If S is a set, $\|S\|$ denotes the cardinality of S.

2.1. Interval size functions. A binary relation over $\Sigma^\star$ is a *partial order* if it is reflexive, antisymmetric, and transitive. A partial order A is a *total order* if for any $x, y \in \Sigma^\star$, it holds that $(x, y) \in A$ or $(y, x) \in A$. An order A is called a *p-order* if there exists a bounding polynomial p such that for all $(x, y) \in A$ it holds that $|x| \leq p(|y|)$.

Definition 2.1. *For any order A we will use the following notation:*

(i) $x \leq_A y$ *is equivalent to* $(x, y) \in A$.

(ii) $x <_A y$ *is equivalent to* $(x \leq_A y \wedge x \neq y)$.

(iii) $x \prec_A y$ *is equivalent to* $(x <_A y \wedge \neg\exists z \in \Sigma^\star(x <_A z <_A y))$. *We say that* x *is a* predecessor *of* y, *or* y *is a* successor *of* x.

(iv) $A_\prec \stackrel{\text{def}}{=} \{(x, y) : x \prec_A y\}$. *A is said to have* efficient adjacency checks *if* $A_\prec \in \mathsf{P}$.

(v) $(x, y)_A \stackrel{\text{def}}{=} \{z \in \Sigma^\star : x <_A z \wedge z <_A y\}$. $(x, y)_A$ *will be called an* interval, *even if A is a partial order. We will also use* $[x, y]_A$, $[x, y)_A$, *and* $(x, y]_A$ *for the* closed, right-open, *and* left-open intervals *respectively.*

We will use lex *to denote the standard lexicographic order of the strings in* $\Sigma^\star$.

Remark 2.2. *For any p-order A with bounding polynomial p and any* $y \in \Sigma^\star$, $\|\{x : x \leq_A y\}\| \leq 2^{p(|y|)+1} - 1$. *As a corollary, every p-order has a bottom element.*

Definition 2.3. *For any total order A we will use the following notation:*

(i) $\mathrm{succ}_A : \Sigma^\star \to \Sigma^\star$ *is the* successor function *for A.*

(ii) $\mathrm{pred}_A : \Sigma^\star \to \Sigma^\star$ *is the* predecessor function *for A. If A contains a bottom element,* pred_A *is undefined for that element.*

(iii) $\mathrm{med}_A : \Sigma^\star \times \Sigma^\star \to \Sigma^\star$ *is the* median function *for A, defined recursively as follows:*

- *if* $y <_A x$ *then* $\mathrm{med}_A(x, y)$ *is undefined,*
- *otherwise if* $x = y$ *or* $x \prec_A y$ *then* $\mathrm{med}_A(x, y) = y$,
- *otherwise* $\mathrm{med}_A(x, y) = \mathrm{med}_A(\mathrm{succ}_A(x), \mathrm{pred}_A(y))$.

(iv) $\mathrm{LN}_A : \Sigma^\star \times \Sigma^\star \times \Sigma^\star \to \Sigma^\star$ *is the* lexicographically nearest *function for A:* $\mathrm{LN}_A(x, y, z)$ *is the string* $w \in [x, y]_A$ *such that w is closest to z in the lexicographic order, breaking ties arbitrarily. If* $[x, y]_A$ *is empty, then* $\mathrm{LN}_A(x, y, z)$ *is undefined for any z.*

(v) *For* $c \in \left(0, \frac{1}{2}\right]$, $f : \Sigma^\star \times \Sigma^\star \to \Sigma^\star$ *is a* c-relaxed median *function for* A, *if it satisfies the following properties:*

- *if* $y <_A x$, *then* $f(x, y)$ *is undefined,*
- *otherwise, if* $x = y$ *or* $x \prec_A y$, *then* $f(x, y) = y$,
- *otherwise,* $f(x, y)$ *is a string* $z \in (x, y)_A$ *such that*

$$\|[x, z)_A\| \geq \lfloor c \cdot \|[x, y]_A\| \rfloor$$
$$\text{and } \|[z, y]_A\| \geq \lceil c \cdot \|[x, y]_A\| \rceil \quad .$$

For a total order A, *we define* rmed_A *to be the family of* c*-relaxed median functions for all* $c \in \left(0, \frac{1}{2}\right]$. *We will slightly abuse notation and say that* $\mathrm{rmed}_A \in \mathsf{FP}$ *(instead of* $\mathrm{rmed}_A \cap \mathsf{FP} \neq \emptyset$*) if there is* some c*-relaxed median function in* FP, *for some* $c \in \left(0, \frac{1}{2}\right]$.

Remark 2.4. *Observe that* med_A *satisfies the properties of a* $\frac{1}{2}$*-relaxed median function, therefore if* $\mathrm{med}_A \in \mathsf{FP}$ *then also* $\mathrm{rmed}_A \in \mathsf{FP}$.

We say that a function $f : \Sigma^\star \to \mathbb{N}$ is an *interval size function defined on an order* A if there exist *boundary functions* $b, t : \Sigma^\star \to \Sigma^\star$ such that for all $x \in \Sigma^\star$, $f(x) = \|(b(x), t(x))_A\|$. In the following, we will primarily be concerned with interval size functions defined on P-decidable p-orders via polynomial-time computable boundary functions.

Definition 2.5 (Hemaspaandra *et al.* 2007). $\mathsf{IF}_{\mathsf{p}}^{\prec}$ *(resp.* $\mathsf{IF}_{\mathsf{t}}^{\prec}$*) is the class of interval size functions defined on* P*-decidable partial (resp. total)* p*-orders with efficient adjacency checks via polynomial-time computable boundary functions.*

Remark 2.6. *Note that in Hemaspaandra* et al. *(2007),* $\mathsf{IF}_{\mathsf{p}}^{\prec}$ *and* $\mathsf{IF}_{\mathsf{t}}^{\prec}$ *were called* IF_{p} *and* IF_{t}, *respectively. We will use the superscript in order to highlight the fact that the functions contained in these classes are defined on orders that possess efficient adjacency checks.*

Furthermore, we will be interested in interval size functions defined on P-decidable p-orders with various other feasibility constraints, apart from $A_\prec \in \mathsf{P}$. We define the following classes:

DEFINITION 2.7. $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}}$ *(resp.* $\mathsf{IF}_{\mathsf{t}}^{\mathsf{pred}}$, $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$, $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$, $\mathsf{IF}_{\mathsf{t}}^{\mathsf{med}}$*) is the class of interval size functions each of which is defined on some* P*-decidable total p-order A via polynomial-time computable boundary functions, where in addition* $\mathrm{succ}_A \in \mathsf{FP}$ *(resp.* pred_A, LN_A, rmed_A, $\mathrm{med}_A \in \mathsf{FP}$*).*

2.2. Counting with Turing Machines. The computational model we are going to use is the Nondeterministic Polynomial-time Turing Machine (NPTM). For an NPTM M, we denote by $M(x)$ the computation of M on input x. For our purposes, $M(x)$ is the computation tree which results from considering all possible nondeterministic choices of M at each step of the computation with input x. We consider only NPTMs with at most two nondeterministic choices at each step (this does not affect the definitions of classes used in this paper). A computation path is encoded by a binary string representing the sequence of nondeterministic choices in this path; a deterministic step is represented by a '0' choice. The length of the encoding is at most $p(|x|)$, where p is the polynomial that bounds the running time of M. Without loss of generality, p can be assumed to be strictly increasing in $|x|$, e.g. of the form $|x|^k + k$ for some constant k.

We say that an NPTM M is in *standard form* if for any input x, each path of $M(x)$ is encoded by a string of length exactly $p(|x|)$. We say that M is in *normal form* if it is in standard form and, in addition, for any input x there are exactly $2^{p(|x|)}$ computation paths in $M(x)$.

We use the notation $\mathrm{acc}_M(x)$ for the number of accepting computation paths of $M(x)$ and the notation $\mathrm{tot}_M(x)$ for the total number of computation paths of $M(x)$. In the descriptions of our algorithms, we will occasionally use statements of the form "$\sigma \leftarrow$ choose from $\{0,1\}$", which implement nondeterministic branchings. Upon execution of such a statement, the current computation path is split into two computation paths, in each of which the variable σ assumes a different value in $\{0,1\}$.

DEFINITION 2.8 (Valiant 1979b). #P *is the class of all total functions f for which there exists an NPTM M such that for all $x \in \Sigma^\star$, $f(x) = \mathrm{acc}_M(x)$.*

Definition 2.9 (Pagourtzis 2001). #PE *(#P-easy) is the class of* #P *functions* f *whose* corresponding language $L_f = \{x \in \Sigma^\star : f(x) > 0\}$ *is in* P.

Definition 2.10 (Kiayias *et al.* 2001). TotP *is the class of all total functions* f *for which there exists an NPTM* M *such that for all* $x \in \Sigma^\star$, $f(x) = \mathrm{tot}_M(x) - 1$.

Note that the classes #P and #PE can be defined using only NPTMs in normal form, whereas TotP can be defined using only NPTMs in standard form.

In Pagourtzis & Zachos (2006), it is proved that TotP is exactly the closure under Karp (parsimonious) reduction of the set of self-reducible functions of #PE. The results can be summarized by the following chain of inclusions:

$$\mathsf{FP} \subseteq \mathsf{TotP} \subseteq \#\mathsf{PE} \subseteq \#\mathsf{P} \ ,$$

where all the inclusions are proper unless $\mathsf{P} = \mathsf{NP}$.

Hemaspaandra *et al.* (2007) show implicitly that $\#\mathsf{PE} = \mathsf{IF}_{\mathsf{p}}^{\prec}$, and furthermore that:

$$\mathsf{FP} \subseteq \mathsf{IF}_{\mathsf{t}}^{\prec} \subseteq \mathsf{IF}_{\mathsf{p}}^{\prec} \subseteq \#\mathsf{P} \ .$$

Again, the inclusions are proper unless unlikely complexity class collapses occur.

Definition 2.11. *Polynomial-time reductions between functions:*

- *Cook (polynomial-time Turing):* $f \leq_T^p g$ *if and only if* f *can be computed in polynomial time using an oracle for* g*:* $f \in \mathsf{FP}^g$.
- *Karp (parsimonious):* $f \leq_m^p g$ *if and only if there exists* $h \in \mathsf{FP}$ *such that for all* $x \in \Sigma^\star$, $f(x) = g(h(x))$.

For a function f and a reduction $\leq_x^p$, $x \in \{m, T\}$, we define $\mathrm{R}_x^p(f) = \{g : g \leq_x^p f\}$, following notation from Hemaspaandra & Ogihara (2002, p. 307). For a function class $\mathcal{F}$ and a reduction $\leq_x^p$, $x \in \{m, T\}$, we use the notation $\mathrm{R}_x^p(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} \mathrm{R}_x^p(f)$ for the *downward closure* of $\mathcal{F}$ under $\leq_x^p$.

PROPOSITION 2.12. *For every interval size function class $\mathcal{F}$ that contains functions defined via polynomial-time boundary functions,* $\mathrm{R}^p_m(\mathcal{F}) \subseteq \mathcal{F}$.

PROOF. Consider $f \in \mathcal{F}$ via an arbitrary order A and boundary functions $b, t \in \mathsf{FP}$. That is, for every x, $f(x) = \|(b(x), t(x))_A\|$. Assume also that $g \leq^p_m f$, that is $\exists h \in \mathsf{FP}$ such that $\forall x, g(x) = f(h(x))$. This implies that $g(x) = f(h(x)) = \|(b(h(x)), t(h(x)))_A\|$, therefore $g \in \mathcal{F}$ via the same order A and boundary functions $b' = b \circ h \in \mathsf{FP}$ and $t' = t \circ h \in \mathsf{FP}$. □

It is not hard to verify the following:

FACT 2.13. $\mathrm{R}^p_m(\mathcal{F}) \subseteq \mathcal{F}$ *holds for* $\mathcal{F} \in \{\#\mathsf{P}, \#\mathsf{PE}, \mathsf{TotP}\}$.

The above fact can either be shown by direct construction of appropriate NPTMs or by means of Proposition 2.12; the latter has to be combined with the (explicit or implicit) characterizations of $\#\mathsf{P}$ and $\#\mathsf{PE}$ via interval size functions (Hemaspaandra *et al.* 2007) and the analogous characterization of TotP that we provide later in this paper (Theorem 4.1).

3. The *status quo* between TotP and $\mathsf{IF}^{\prec}_{\mathsf{t}}$

As we already mentioned in the previous section, both TotP and $\mathsf{IF}^{\prec}_{\mathsf{t}}$ are contained in $\#\mathsf{PE} = \mathsf{IF}^{\prec}_{\mathsf{p}}$. In this section, we investigate the relationship between these two classes. Specifically, we show that $\mathsf{TotP} \subseteq \mathsf{IF}^{\prec}_{\mathsf{t}}$ and that the inclusion is, in fact, proper unless $\mathsf{P} = \mathsf{UP} \cap \mathsf{coUP}$. The first step of the proof consists in establishing that $\mathsf{TotP} \subseteq \mathsf{IF}^{\mathsf{succ}}_{\mathsf{t}}$. We actually prove in Theorem 3.2 a slightly stronger result, namely that any TotP function is also an interval size function defined on some P-decidable total order with FP-computable boundary functions, in which both the successor and predecessor functions are also FP-computable.

DEFINITION 3.1. $\mathsf{IF}^{\mathsf{succ,pred}}_{\mathsf{t}}$ *is the class of interval size functions defined on* P*-decidable total p-orders via polynomial-time computable boundary functions, in which additionally* $\mathrm{succ}_A \in \mathsf{FP}$ *and* $\mathrm{pred}_A \in \mathsf{FP}$.

THEOREM 3.2. $\mathsf{TotP} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{succ,pred}}$.

PROOF. Let f be a TotP function and M be the corresponding NPTM such that for all $x \in \Sigma^\star$, $f(x) = \mathrm{tot}_M(x) - 1$. Let p be the polynomial that bounds the running time of M. Without loss of generality, we assume that M is in standard form, hence there always exists a computation path encoded by the string $0^{p(|x|)}$, which we will refer to as the *leftmost* path. Similarly, the *rightmost* path is the computation path whose encoding is lexicographically larger than the encoding of any other path in $M(x)$.

We define a total order A on $\Sigma^\star$ as follows: A coincides with the lexicographic order except that, for every $x \in \Sigma^\star$, the interval $[x0^{p(|x|)+2}, x10^{p(|x|)+1}]_{\mathrm{lex}}$ is ordered in the following way:

- First come the elements of the set

$$\{x0y0 : |y| = p(|x|) \wedge y \text{ encodes a path of } M(x)\} \ ,$$

  in lexicographic order (note that $x0^{p(|x|)+2}$ is always an element of this set),

- next comes $x10^{p(|x|)+1}$,

- and last come the elements of the set

$$\{x0y0 : |y| = p(|x|) \wedge y \text{ does not encode a path of } M(x)\}$$
$$\cup\{x0y1 : |y| = p(|x|)\} \ ,$$

  in lexicographic order.

It is not hard to see that A is a p-order, since $\forall w, z \in \Sigma^\star, w \leq_A z \rightarrow |w| \leq |z|$. Moreover, $A \in \mathsf{P}$ since, given two strings $w, z \in \Sigma^\star$, we can check in polynomial time whether $w \leq_A z$ as follows[1]:

- If $w = x0ya$ and $z = x0y'b$, where $y \leq_{\mathrm{lex}} y'$, $|y| = |y'| = p(|x|)$, and $a, b \in \Sigma$, then $z \leq_A w$ if and only if $b = 0 \wedge y'$ encodes a computation path of $M(x) \wedge y$ does not encode a computation path of $M(x)$.

[1]Note that, assuming p is a strictly increasing polynomial, we can efficiently decide for any given string w whether it can be decomposed into the form $w = xayb$, where $a, b \in \Sigma$ and $|y| = p(|x|)$, and find such a decomposition if it exists.

- If $w = x0ya$ and $z = x10^{p(|x|)+1}$, where $|y| = p(|x|)$ and $a \in \Sigma$, then $w \leq_A z$ if and only if $a = 0 \wedge y$ encodes a computation path of $M(x)$.

- Otherwise, $w \leq_A z$ if and only if $w \leq_{\text{lex}} z$.

We define the boundary functions $b, t \in \mathsf{FP}$ as follows, for any $x \in \Sigma^\star$:

$$b(x) = x0^{p(|x|)+2}, \quad t(x) = x10^{p(|x|)+1} \ .$$

These functions satisfy $\|(b(x), t(x))_A\| = \text{tot}_M(x) - 1 = f(x)$, for any $x \in \Sigma^\star$, since by definition the size of $(b(x), t(x))_A$ is exactly equal to the number of computation paths of $M(x)$ minus one (the leftmost path is excluded from the interval).

It remains to be shown that both $\text{succ}_A \in \mathsf{FP}$ and $\text{pred}_A \in \mathsf{FP}$. Algorithm 3.3 computes the successor of a given string w in polynomial time. The predecessor function for A can also be computed in polynomial time in a similar manner.

Algorithm 3.3. The *successor* for an order associated with a TotP function.

Input: a string $w \in \Sigma^\star$
Goal: compute the successor of w in the order A defined in the proof of Theorem 3.2

1. If w is not of the form $x10^{p(|x|)+1}$ or $x0ya$, where $|y| = p(|x|)$ and $a \in \Sigma$
   then
2. Return $\text{succ}_{\text{lex}}(w)$
3. Else if $w = x10^{p(|x|)+1}$ then
4. Return $x0^{p(|x|)+1}1$
5. Else if $w = x0y0$, where y encodes a path of $M(x)$ then
6. simulate the NPTM M on input x, following the nondeterministic choices encoded in y
7. If y is the rightmost path in the computation then
8. Return $x10^{p(|x|)+1}$
9. Else
10. backtrack until the last point where path y follows a choice of '0', while a choice of '1' is also available

11. at that point choose '1' and continue the simulation of M, always choosing '0' instead of '1' whenever there is a choice
12. Return $x0y'0$, where y' is the encoding of the new computation path

13. Else if $w = x0y0$, where y does not encode a path of $M(x)$ then
14. Return $x0y1$
15. Else if $w = x0y1$ then
16. If $y = 1^{p(|x|)}$ then
17. Return $\mathrm{succ}_{\mathrm{lex}}(x10^{p(|x|)+1})$
18. Else
19. Return $x0y'b$, where $y' = \mathrm{succ}_{\mathrm{lex}}(y)$ and $b = 1$ iff y' encodes a path of $M(x)$

□

It is immediate from the definitions that $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ,pred}} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}}$ and $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ,pred}} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{pred}}$. Moreover, if A is a total order, then $x \prec_A y \iff y = \mathrm{succ}_A(x) \iff x = \mathrm{pred}_A(y)$. Thus, if $\mathrm{succ}_A \in \mathsf{FP}$ or $\mathrm{pred}_A \in \mathsf{FP}$, then the adjacency relationship $\prec_A$ can be decided in polynomial time. This implies that $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}} \subseteq \mathsf{IF}_{\mathsf{t}}^{\prec}$ and $\mathsf{IF}_{\mathsf{t}}^{\mathsf{pred}} \subseteq \mathsf{IF}_{\mathsf{t}}^{\prec}$. In fact, we prove in Theorem 3.4 the equivalence of the successor and predecessor functions as polynomial-time feasibility constraints on P-decidable total p-orders with FP-computable boundary functions.

THEOREM 3.4. $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}} = \mathsf{IF}_{\mathsf{t}}^{\mathsf{pred}}$.

PROOF. We only prove that $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{pred}}$. The other inclusion can be derived by similar arguments. Let $f \in \mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}}$, so there is some total p-order A with bounding polynomial p and $\mathrm{succ}_A \in \mathsf{FP}$, and boundary functions $b, t \in \mathsf{FP}$ such that for all $x \in \Sigma^\star$, $f(x) = \|(b(x), t(x))_A\|$. Moreover, let $\mathrm{pad} : \Sigma^\star \times \Sigma^\star \to \Sigma^\star$ be a padding function defined as follows: for any $x \in \Sigma^\star$ and $z \in \Sigma^\star$ with $|z| \leq p(|t(x)|)$, $\mathrm{pad}(x, z)$ is a string of length exactly $2p(|t(x)|)$ that is produced by replacing every occurrence of '0' in z by '01', every occurrence of '1' by '10', and padding left with '0's. Clearly,

both pad and pad^{-1}, the inverse of pad with respect to the first argument, are FP-computable. (Definition of pad^{-1}: for $x, y \in \Sigma^\star$, $\text{pad}^{-1}(x, y)$ is the unique z such that $y = \text{pad}(x, z)$, or it is undefined if no such z exists.)

We will prove that $f \in \mathsf{IF}_{\mathsf{t}}^{\mathsf{pred}}$. Consider a total order B which coincides with the lexicographic order on $\Sigma^\star$ except that, for every $x \in \Sigma^\star$, the interval $[x0^{2p(|t(x)|)+1}, x1^{2p(|t(x)|)+1}]_{\text{lex}}$ is ordered as follows:

- First come the elements of $\{xy0 : \text{pad}^{-1}(x, y) \in [b(x), t(x)]_A\}$, ordered so that $xy0 <_B xy'0$ if and only if $\text{pad}^{-1}(x, y) >_A \text{pad}^{-1}(x, y')$,
- and then come the rest of the strings of the form xya, where $|y| = 2p(|t(x)|)$ and $a \in \Sigma$, in lexicographic order.

Now, B is clearly a p-order since $\forall w, z \in \Sigma^\star, w \leq_B z \rightarrow |w| \leq |z|$. Moreover, given two strings $w, z \in \Sigma^\star$, we can check in polynomial time whether $w \leq_B z$ as follows:

- If $w = xya$ and $z = xy'b$, where $y \leq_{\text{lex}} y'$, $|y| = |y'| = 2p(|t(x)|)$, and $a, b \in \Sigma$, then $w \leq_B z$ if and only if one of the following holds:
  - $\big(\text{pad}^{-1}(x, y') \notin [b(x), t(x)]_A$ or it is undefined$\big)$ and $(a = 0 \vee b = 1 \vee y \neq y')$.
  - $\big(\text{pad}^{-1}(x, y) \notin [b(x), t(x)]_A$ or it is undefined$\big)$ and $b = 1$ and $\text{pad}^{-1}(x, y') \in [b(x), t(x)]_A$.
  - $\{\text{pad}^{-1}(x, y), \text{pad}^{-1}(x, y')\} \subseteq [b(x), t(x)]_A$ and $(b = 1 \vee (a = 0 \wedge b = 0 \wedge \text{pad}^{-1}(x, y) \geq_A \text{pad}^{-1}(x, y')))$.
- Otherwise, $w \leq_B z$ if and only if $w \leq_{\text{lex}} z$.

Therefore, $B \in \mathsf{P}$. If we define the boundary functions as $b'(x) = xy0$, where $y = \text{pad}(x, t(x))$, and $t'(x) = xz0$, where $z = \text{pad}(x, b(x))$, then for any $x \in \Sigma^\star$ it holds that $\|(b'(x), t'(x))_B\| = \|(b(x), t(x))_A\| = f(x)$.

Finally, using the polynomial-time-computable successor function for A, it is possible to compute the predecessor function for B also in polynomial time. Algorithm 3.5 contains the details.

ALGORITHM 3.5. The *predecessor* for an order associated with an $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}}$ function.

Input: a string $w \in \Sigma^\star$
Goal: compute the predecessor of w in the order B defined in the proof of Theorem 3.4

1. If w is not of the form xy, where $|y| = 2p(|t(x)|) + 1$ then
2. Return $\mathrm{pred}_{\mathrm{lex}}(w)$
3. Else if $w = xy0$, where $\mathrm{pad}^{-1}(x, y) \in [b(x), t(x)]_A$ then
4. If $\mathrm{pad}^{-1}(x, y) = t(x)$ then
5. Return $\mathrm{pred}_{\mathrm{lex}}(x0^{2p(|t(x)|)+1})$
6. Else
7. Return $xy'0$, where $y' = \mathrm{pad}(x, \mathrm{succ}_A(\mathrm{pad}^{-1}(x, y)))$
8. Else if $w = xy1$, where $\mathrm{pad}^{-1}(x, y) \in [b(x), t(x)]_A$ then
9. If $y = 0^{2p(|t(x)|)}$ then
10. Return $xy'0$, where $y' = \mathrm{pad}(x, b(x))$
11. Else
12. Return $xy'1$, where $y' = \mathrm{pred}_{\mathrm{lex}}(y)$
13. Else if $w = x0^{2p(|t(x)|)+1}$ then
14. Return $xy'0$, where $y' = \mathrm{pad}(x, b(x))$
15. Else
16. Return $\mathrm{pred}_{\mathrm{lex}}(w)$

□

Note that, even though $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}} = \mathsf{IF}_{\mathsf{t}}^{\mathsf{pred}}$, this does not imply that, for a given total order A, $\mathrm{succ}_A \in \mathsf{FP}$ is equivalent to $\mathrm{pred}_A \in \mathsf{FP}$. Indeed, in the proof of Theorem 3.4, we had to define a different order B in which pred_B was polynomial-time computable. Therefore, $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ,pred}}$ is not necessarily equal to $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}}$ and $\mathsf{IF}_{\mathsf{t}}^{\mathsf{pred}}$. We summarize the results obtained so far in this section in the following theorem:

THEOREM 3.6. $\mathsf{TotP} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{succ,pred}} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}} = \mathsf{IF}_{\mathsf{t}}^{\mathsf{pred}} \subseteq \mathsf{IF}_{\mathsf{t}}^{\prec}$.

Finally, we show that $\mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}}$, and therefore also TotP in view of Theorem 3.6, is strictly contained in $\mathsf{IF}_{\mathsf{t}}^{\prec}$ under the assumption that $\mathsf{P} \neq \mathsf{UP} \cap \mathsf{coUP}$. We begin by defining a generalization of the $\exists\cdot$ and $\mathrm{Sig}\cdot$ operators used by Hemaspaandra *et al.* (2007) and Hempel & Wechsung (2000), respectively.

DEFINITION 3.7. *For any integer constant $k \geq 0$, we define the operator $\mathcal{C}_{>k}$. If $\mathcal{F}$ is any function class, then $\mathcal{C}_{>k} \cdot \mathcal{F}$ is the following class of languages:*

$$\mathcal{C}_{>k} \cdot \mathcal{F} = \{L : \exists f \in \mathcal{F},\ \forall x \in \Sigma^\star\ (x \in L \iff f(x) > k)\}\ .$$

Observe that if $\mathcal{F} \subseteq \mathcal{G}$, then $\mathcal{C}_{>k} \cdot \mathcal{F} \subseteq \mathcal{C}_{>k} \cdot \mathcal{G}$.

Next, we make use of the existence of a unique decisive path in any $\mathsf{UP} \cap \mathsf{coUP}$ computation, in order to show that each such computation can be mapped to an interval size function where one or two strings fall into an appropriate interval, depending on whether the decisive path is rejecting or accepting, respectively. Therefore, applying the $\mathcal{C}_{>1} \cdot$ operator would reveal which is the case. This leads to the following lemma:

LEMMA 3.8. $\mathsf{UP} \cap \mathsf{coUP} \subseteq \mathcal{C}_{>1} \cdot \mathsf{IF}_{\mathsf{t}}^{\prec}$.

PROOF. Let $L \in \mathsf{UP} \cap \mathsf{coUP}$, so there is an NPTM M that decides L with the property that, for any input x, $M(x)$ has exactly one decisive path (either accepting or rejecting) and all the other paths terminate in an indecisive state "?". We assume that M is in normal form and let p be the polynomial that bounds the running time of M.

We construct an order A that coincides with the lexicographic order of $\Sigma^\star$, except that, for every $x \in \Sigma^\star$, the corresponding interval $[x0^{p(|x|)+2}, x1^{p(|x|)+2}]_{\text{lex}}$ is ordered in the following way:

- First comes $x0^{p(|x|)+2}$,
- if $x \notin L$, then next comes $x01z$, where z encodes the unique rejecting path of M on input x, while if $x \in L$, then next come $x01z$ and $x10z$, where z encodes the unique accepting path of M on input x,
- next comes $x110^{p(|x|)}$,
- and last come the rest of the strings of the form xw, where $|w| = p(|x|) + 2$, in lexicographic order.

Table 3.1: Different cases in deciding the order A defined in the proof of Lemma 3.8. Specifically, for two strings w, u of the form $w = xabz$ and $u = xa'b'z'$, where $a, b, a', b' \in \Sigma$ such that $ab \leq_{\text{lex}} a'b'$ and $|z| = |z'| = p(|x|)$, the table provides necessary and sufficient conditions for $w \leq_A u$. Each row contains the necessary and sufficient condition for the corresponding combination of values of a, b, a', b'.

ab	$a'b'$	$w \leq_A u$ if and only if
00	00	$z \leq_{\text{lex}} z'$
00	01	z' indecisive $\vee$ $z = 0^{p(\|x\|)}$
00	10	z' indecisive $\vee$ $z = 0^{p(\|x\|)}$ $\vee$ z' rejects
00	11	$z = 0^{p(\|x\|)}$ $\vee$ $z' \neq 0^{p(\|x\|)}$
01	01	z accepts $\vee$ z rejects $\vee$ (z' indecisive $\wedge$ $z \leq_{\text{lex}} z'$)
01	10	$\neg$ (z indecisive $\wedge$ z' accepts)
01	11	z accepts $\vee$ z rejects $\vee$ $z' \neq 0^{p(\|x\|)}$
10	10	z accepts $\vee$ (z rejects $\wedge$ (z' rejects $\vee$ $z \leq_{\text{lex}} z'$)) $\vee$ (z' does not accept $\wedge$ $z \leq_{\text{lex}} z'$)
10	11	z accepts $\vee$ $z' \neq 0^{p(\|x\|)}$
11	11	$z \leq_{\text{lex}} z'$

The order A is clearly a p-order since lex is a p-order. Moreover, given any string one can efficiently decide whether it belongs to some interval of the form $[x0^{p(|x|)+2}, x1^{p(|x|)+2}]_{\text{lex}}$, and if so, whether it corresponds to an accepting, rejecting or indecisive computation path of $M(x)$; this information clearly suffices in order to decide its order relative to any other string. In more detail, if $w = xabz$ and $u = xa'b'z'$, where $a, b, a', b' \in \Sigma$ with $ab \leq_{\text{lex}} a'b'$ and $|z| = |z'| = p(|x|)$, then we can decide in deterministic polynomial time whether $w \leq_A u$ by consulting Table 3.1. In case that w and u are not of that form, then $w \leq_A u$ if and only if $w \leq_{\text{lex}} u$. Therefore, $A \in \mathsf{P}$. By similar arguments, the adjacency relation $\prec_A$ is also deterministically decidable in polynomial time.

We now define the boundary functions $b, t \in \mathsf{FP}$: for any $x \in \Sigma^\star$, $b(x) = x0^{p(|x|)+2}$ and $t(x) = x110^{p(|x|)}$. It holds that, for any

$x \in \Sigma^\star$, $\|(b(x), t(x))_A\| > 1$ if and only if $x \in L$. Therefore, $L \in \mathcal{C}_{>1} \cdot \mathsf{IF}_{\mathsf{t}}^{\prec}$. □

LEMMA 3.9. $\mathcal{C}_{>1} \cdot \mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}} \subseteq \mathsf{P}$.

PROOF. For any $f \in \mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}}$, we can decide in polynomial time the set $\{x : f(x) > 1\}$ as follows: For a given x, compute $\mathrm{succ}_A(b(x))$ and $\mathrm{succ}_A(\mathrm{succ}_A(b(x)))$, where A is the underlying total p-order with $\mathrm{succ}_A \in \mathsf{FP}$ and $b, t \in \mathsf{FP}$ are the boundary functions for f. If either of the computed strings is equal to $t(x)$ then reject, else accept. □

By Lemmas 3.8 and 3.9, we obtain immediately the following:

THEOREM 3.10. *If* $\mathsf{IF}_{\mathsf{t}}^{\prec} = \mathsf{IF}_{\mathsf{t}}^{\mathsf{succ}}$, *then* $\mathsf{P} = \mathsf{UP} \cap \mathsf{coUP}$.

4. TotP as an interval size function class

This section is devoted to the characterization of TotP as the class of interval size functions defined on polynomial-time-decidable p-orders with FP-computable boundary and *lexicographically nearest* functions:

THEOREM 4.1. $\mathsf{TotP} = \mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$.

We prove the inclusion $\mathsf{TotP} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$ in Lemma 4.2 below. The crux of the argument is that, given an NPTM M in standard form, an input x, and a string y which has the right length to possibly encode a computation path of $M(x)$, we can compute in polynomial time a string which actually encodes a computation path of $M(x)$ and is lexicographically closest to the given string y.

LEMMA 4.2. $\mathsf{TotP} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$.

PROOF. Let f be a TotP function and M be the corresponding NPTM in standard form such that for all $x \in \Sigma^\star$, $f(x) = \mathrm{tot}_M(x) - 1$. Let $p(n) = n^k + k$ be the polynomial that bounds the running time of M.

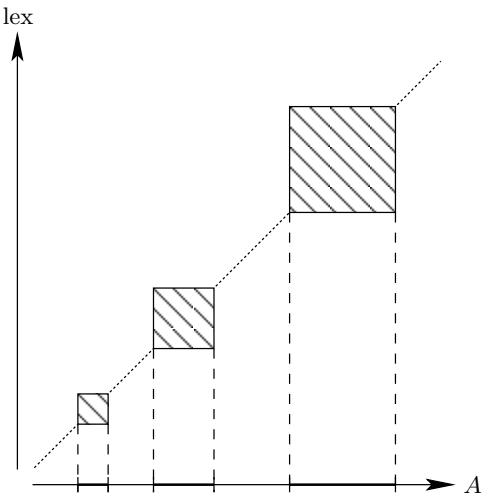

Figure 4.1: A high-level illustration of the order A used in the proofs of Theorem 3.2 and Lemma 4.2. The horizontal axis contains the binary strings ordered under A and the vertical axis represents the rank of the strings in the lexicographic order. The ranks in the two orders (A and lex) are identical (thus yielding in this lex-rank plot the dotted line going from bottom left to top right), except for those strings belonging in the intervals $\mathcal{I}(x)$ (for $x \in \Sigma^\star$), which locally deviate from the lexicographic order (depicted as shaded boxes in the figure and as thick intervals on the A-axis).

We will use the order A and the boundary functions b and t that we defined in the proof of Theorem 3.2. Recall that the order A is essentially the same as the lex order, except that, for each x, the interval which we will henceforth denote by $\mathcal{I}(x) = [x0^{p(|x|)+2}, x10^{p(|x|)+1}]_{\text{lex}}$ (and which is equal, as a set, to the interval $[x0^{p(|x|)+2}, x01^{p(|x|)+1}]_A$) is ordered differently. Figure 4.1 illustrates the order A against lex. Figure 4.2 illustrates an example of an interval $\mathcal{I}(x)$. Recall also that $b(x) = x0^{p(|x|)+2}$ and $t(x) = x10^{p(|x|)+1}$.

It suffices to show that the order A admits a polynomial-time computable lexicographically-nearest function. We can compute $\text{LN}_A(u, v, z)$ as follows: If $[u, v]_A$ is empty, then $\text{LN}_A(u, v, z)$ is undefined. If $z \in [u, v]_A$, then clearly $\text{LN}_A(u, v, z) = z$. In the

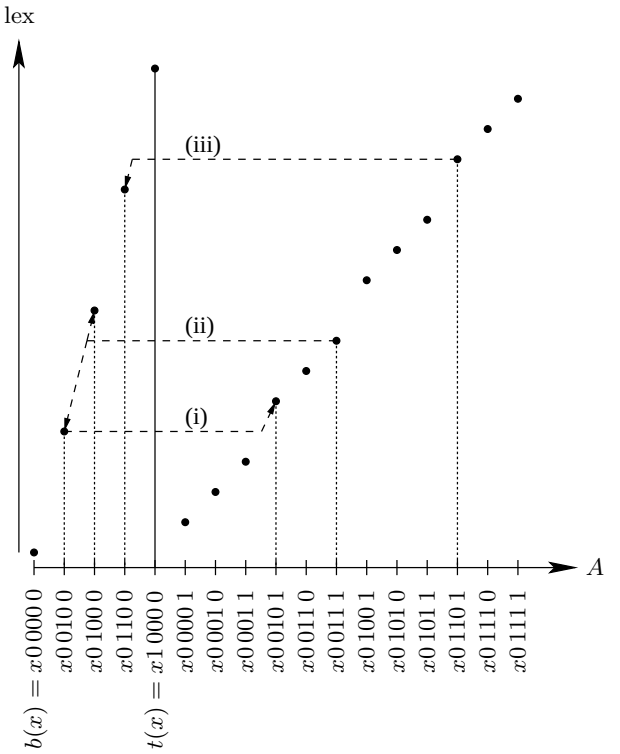

Figure 4.2: Example of an interval $\mathcal{I}(x)$ in the proof of Lemma 4.2. The horizontal axis contains the strings of $\mathcal{I}(x)$ ordered under A and the vertical axis represents the rank of the strings in the lexicographic order. For this particular x, all computation paths of $M(x)$ have length 3. The only paths that appear in $M(x)$ are $\{000, 010, 100, 110\}$. The following calculations of $\tilde{z}$ or of $\tilde{z}^-$ and $\tilde{z}^+$ are illustrated: (i) for $z = x0\,010\,0$, $\tilde{z} = x0\,010\,1$ (Case 1 of the proof of Lemma 4.2), (ii) for $z = x0\,011\,1$, $\tilde{z}^- = x0\,010\,0$ and $\tilde{z}^+ = x0\,100\,0$ (Case 2 of the proof of Lemma 4.2 when $y = 011$ does not correspond to a computation path), and (iii) for $z = x0\,110\,1$, $\tilde{z}^- = \tilde{z}^+ = x0\,110\,0$ (Case 2 of the proof of Lemma 4.2 when $y = 110$ corresponds to a computation path).

following, we assume that $u \leq_A v$ and $z \notin [u, v]_A$.

Note that, for any given $z \in \Sigma^\star$, we can find in polynomial time an x such that $z \in \mathcal{I}(x)$, or decide that no such x exists. If, for all x, $z \notin \mathcal{I}(x)$, then by construction of the order A, z is lexicographically larger than all of its preceding elements under A and it is lexicographically smaller than all of its subsequent elements under A (see Figure 4.1). Therefore, since $z \notin [u, v]_A$, we know that $\mathrm{LN}_A(u, v, z)$ is either the lexicographically maximum string in $[u, v]_A$ (if $v <_A z$), or the lexicographically minimum string in $[u, v]_A$ (if $z <_A u$). In view of the structure of the intervals $\mathcal{I}(w)$ (see Figure 4.2), for any two strings u, v with $u \leq_A v$, we can compute the lexicographically maximum and minimum strings in $[u, v]_A$ in polynomial time as follows:

$$\text{lex-max}(u, v) = \begin{cases} t(w), & \text{if } v \in \mathcal{I}(w) \text{ and } u \leq_A t(w) \leq_A v \\ v, & \text{otherwise} \end{cases}$$

$$\text{lex-min}(u, v) = \begin{cases} w0^{p(|w|)+1}1, & \text{if } u \in \mathcal{I}(w) \text{ and } b(w) <_A u \leq_A w0^{p(|w|)+1}1 \leq_A v \\ u, & \text{otherwise} \end{cases}$$

Finally, assume that $z \in \mathcal{I}(x)$, for some $x \in \Sigma^\star$. We distinguish the following two cases:

Case 1: $z \leq_A t(x)$. We define a string $\tilde{z} \in \mathcal{I}(x)$ as follows: If $z = x0y0$, where y encodes a computation path of $M(x)$, then $\tilde{z} = x0y1$, whereas if $z = t(x)$, then $\tilde{z} = x01^{p(|x|)+1}$. Figure 4.2 illustrates an example of computing $\tilde{z}$ in this case (item (i) in the figure). Note that it always holds that $t(x) <_A \tilde{z} \leq_A x01^{p(|x|)+1}$, and thus in particular $z <_A \tilde{z}$. We have the following subcases for computing $\mathrm{LN}_A(u, v, z)$:

- $[u, v]_A$ contains $\tilde{z}$: Since $z \notin [u, v]_A$ and $\tilde{z}$ is, by construction, adjacent to z in the lexicographic order, we can safely return $\tilde{z}$.

- $[u, v]_A$ is located before z in the order A: Because z is lexicographically larger than all of its preceding elements under A, and thus in particular it is lexicographically larger than all of the elements of $[u, v]_A$, we can safely return lex-max(u, v) in this case.

- $[u, v]_A$ is located after $\tilde{z}$ in the order A: Because z is lexicographically smaller than $\tilde{z}$ and $\tilde{z}$ is lexicographically smaller than all of its subsequent elements under A, and thus in particular it is lexicographically smaller than all of the elements of $[u, v]_A$, we can safely return lex-min(u, v) in this case.

- $[u, v]_A$ is located between z and $\tilde{z}$ in A: As can be verified by inspection of Figure 4.2, in this case we have either that *(a):* $u \leq_{\text{lex}} v$, in which case the whole of $[u, v]_A$ is lexicographically between u and v, whereas $z \notin [u, v]_{\text{lex}}$, or *(b):* $v <_{\text{lex}} u$, in which case the whole of $[u, v]_A$ is lexicographically outside of $(v, u)_{\text{lex}}$, whereas $z \in (v, u)_{\text{lex}}$. In both cases, we can safely return either u or v, whichever is lexicographically closer to z.

To summarize Case 1, if $z \in \mathcal{I}(x)$, $z \leq_A t(x)$, and $z \notin [u, v]_A$, then we compute $\tilde{z}$ as explained above and $\text{LN}_A(u, v, z)$ is given by:

$$\text{LN}_A(u, v, z) = \begin{cases} \tilde{z}, & \text{if } u \leq_A \tilde{z} \leq_A v \\ \text{lex-max}(u, v), & \text{if } u \leq_A v <_A z \\ \text{lex-closest to } z \text{ among } \{u, v\}, & \text{if } z <_A u \leq_A v <_A \tilde{z} \\ \text{lex-min}(u, v), & \text{if } \tilde{z} <_A u \leq_A v \end{cases}$$

Case 2: $z >_A t(x)$. In this case, z is of the form $z = x0ya$, where $|y| = p(|x|)$ and $a \in \{0, 1\}$. We simulate $M(x)$ following the nondeterministic choices encoded in y, until the computation ends or we encounter a choice of '1' that is not available (recall that, by definition of the encoding, a choice of '0' is always available). If we encounter an unavailable nondeterministic choice, then y is clearly not a computation path of $M(x)$. In this case, we produce two strings $\tilde{z}^-$ and $\tilde{z}^+$ as follows:

> $\tilde{z}^- = x0y^-0$, where y^- is the string that encodes the computation path obtained by following the available choice of '0' at the point where the simulation of y stopped, and then following a choice of '1' wherever it is available.

$\tilde{z}^+ = x0y^+0$, where y^+ is the string that encodes the computation path obtained by rolling back the simulation of y to the last point where a nondeterministic choice of '1' was available, but y dictated to follow the choice of '0'. At that point, follow the choice of '1' and after that always choose '0'. If, there is no point at which a nondeterministic choice of '1' is available, then set $\tilde{z}^+ = t(x)$.

On the other hand, if all nondeterministic choices encoded in y are available, then y is actually a computation path of $M(x)$ and we set $\tilde{z}^- = \tilde{z}^+ = x0y0$. Figure 4.2 illustrates the two different scenarios for computing $\tilde{z}^-$ and $\tilde{z}^+$ in this case (items (ii) and (iii) in the figure). Note that in all cases we have $\tilde{z}^-, \tilde{z}^+ \in [b(x), t(x)]_A$ and $\tilde{z}^-, \tilde{z}^+$ are either equal or adjacent in A. We have the following subcases for computing $\mathrm{LN}_A(u, v, z)$:

- $[u, v]_A$ contains both $\tilde{z}^-$ and $\tilde{z}^+$: By inspection of Figure 4.2, in this case the lexicographically nearest to z is clearly one of $\{\tilde{z}^-, \tilde{z}^+, v\}$.

- $[u, v]_A$ is located before $\tilde{z}^-$ in A (possibly with $v = \tilde{z}^-$): Because z is lexicographically larger than $\tilde{z}^-$ and $\tilde{z}^-$ is lexicographically larger than all of its preceding elements under A, we return lex-max(u, v) in this case.

- $[u, v]_A$ is located between $\tilde{z}^+$ and z (possibly with $u = \tilde{z}^+$): Similarly to the subcase $z <_A u \leq_A v <_A \tilde{z}$ in Case 1 above, we return either u or v, whichever is lexicographically closer to z.

- $[u, v]_A$ is located after z in A: Because z is lexicographically smaller than all of its subsequent elements under A, we return lex-min(u, v) in this case.

To summarize Case 2, if $z \in \mathcal{I}(x)$, $z >_A t(x)$, and $z \notin [u, v]_A$, then we compute $\tilde{z}^-$ and $\tilde{z}^+$ as explained above and $\mathrm{LN}_A(u, v, z)$

is given by:

$$\mathrm{LN}_A(u,v,z) = \begin{cases} \text{lex-closest to } z \text{ among } \{\tilde{z}^-, \tilde{z}^+, v\}, & \text{if } u \leq_A \tilde{z}^- \leq_A \tilde{z}^+ \leq_A v \\ \text{lex-max}(u,v), & \text{if } u \leq_A v \leq_A \tilde{z}^- \\ \text{lex-closest to } z \text{ among } \{u,v\}, & \text{if } \tilde{z}^+ \leq_A u \leq_A v <_A z \\ \text{lex-min}(u,v), & \text{if } z <_A u \leq_A v \end{cases}$$

□

For the other inclusion, $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}} \subseteq \mathsf{TotP}$, we need a preliminary result concerning the functions LN_A^- and LN_A^+ defined as follows:

Definition 4.3. *For a total order A we define the following partial functions:*

(i) $\mathrm{LN}_A^-(u,v,x)$ *is the lexicographically largest* $y \in [u,v]_A$ *such that* $y \leq_{\mathrm{lex}} x$.

(ii) $\mathrm{LN}_A^+(u,v,x)$ *is the lexicographically smallest* $y \in [u,v]_A$ *such that* $x \leq_{\mathrm{lex}} y$.

It is immediate that if $\mathrm{LN}_A^- \in \mathsf{FP}$ and $\mathrm{LN}_A^+ \in \mathsf{FP}$, then also $\mathrm{LN}_A \in \mathsf{FP}$. We now prove the converse:

Lemma 4.4. *For a total p-order A, if* $\mathrm{LN}_A \in \mathsf{FP}$ *then* $\mathrm{LN}_A^- \in \mathsf{FP}$ *and* $\mathrm{LN}_A^+ \in \mathsf{FP}$.

Proof. We prove the claim only for LN_A^+, since the proof for LN_A^- is completely symmetric. Let p be the bounding polynomial of A. For given $u,v,x \in \Sigma^\star$, we compute $\mathrm{LN}_A^+(u,v,x)$ as follows: Let $y = \mathrm{LN}_A(u,v,x)$. If $x \leq_{\mathrm{lex}} y$, then by definition of the function LN_A we have $(x,y)_{\mathrm{lex}} \cap [u,v]_A = \emptyset$. Thus, $\mathrm{LN}_A^+(u,v,x) = y$. For the rest of the proof we assume that $y <_{\mathrm{lex}} x$ and let $\delta = \|[y,x)_{\mathrm{lex}}\|$.

We compute a sequence of strings $x = x_0 <_{\mathrm{lex}} x_1 <_{\mathrm{lex}} \cdots <_{\mathrm{lex}} x_k$ where for all i in the range $0 \leq i \leq k-1$:

$$\|[y,x_i)_{\mathrm{lex}}\| = \|[x_i,x_{i+1})_{\mathrm{lex}}\| = 2^i \cdot \delta \ ,$$

and $k \geq 1$ is the smallest index such that $\mathrm{LN}_A(u,v,x_k) = y' \neq y$.

We prove that $\mathrm{LN}^+_A(u,v,x) = y'$. We have $\mathrm{LN}_A(u,v,x_{k-1}) = y <_{\mathrm{lex}} x_{k-1}$, but $\mathrm{LN}_A(u,v,x_k) = y' \neq y$ and $\|[y,x_{k-1})_{\mathrm{lex}}\| = \|[x_{k-1},x_k)_{\mathrm{lex}}\|$. Therefore, we must have $y' \geq_{\mathrm{lex}} x_k >_{\mathrm{lex}} x$ (otherwise, if we had $y <_{\mathrm{lex}} x_k$, then y' would be lexicographically closer to x_{k-1} than y is, and thus we would have $\mathrm{LN}_A(u,v,x_{k-1}) = y'$) and also $(x_k,y')_{\mathrm{lex}} \cap [u,v]_A = \emptyset$ (if there was a $y'' \in [u,v]_A$ such that $x_k <_{\mathrm{lex}} < y'' <_{\mathrm{lex}} y'$, then we would have $\mathrm{LN}_A(u,v,x_k) = y''$). In fact, it is clear that either $y' = x_k$ or $x_k \notin [u,v]_A$ (this follows from the observation that, if $x_k \in [u,v]_A$, then certainly $y' = x_k$), so we can rewrite the equation $(x_k,y')_{\mathrm{lex}} \cap [u,v]_A = \emptyset$ in a slightly stronger form:

$$[x_k,y')_{\mathrm{lex}} \cap [u,v]_A = \emptyset \ . \tag{4.5}$$

Furthermore, for all i in the range $0 \leq i \leq k-1$, we know that $\mathrm{LN}_A(u,v,x_i) = y$. Thus, from the definition of LN_A, we have $(x_i,x_{i+1})_{\mathrm{lex}} \cap [u,v]_A = \emptyset$. Moreover, it is clear that $x_i \notin [u,v]_A$, so we actually have $[x_i,x_{i+1})_{\mathrm{lex}} \cap [u,v]_A = \emptyset$. Taking the union for all i, we deduce:

$$\bigcup_{i=0}^{k-1} ([x_i,x_{i+1})_{\mathrm{lex}} \cap [u,v]_A) = [x_0,x_k)_{\mathrm{lex}} \cap [u,v]_A = \emptyset \ . \tag{4.6}$$

Equations (4.5) and (4.6) yield $[x,y')_{\mathrm{lex}} \cap [u,v]_A = \emptyset$, therefore $\mathrm{LN}^+_A(u,v,x) = y'$.

If, during the process of computing the sequence of x_i's, we stumble upon some x_i with length $|x_i| > p(|v|)$, then for all $w \geq_{\mathrm{lex}} x_i$ we have $|w| \geq |x_i| > p(|v|)$, which implies that $w \notin [u,v]_A$. Since it is also clear that $[x_0,x_i)_{\mathrm{lex}} \cap [u,v]_A = \emptyset$, we can safely conclude that $[u,v]_A$ does not contain any strings lexicographically larger than x, and we can halt the computation leaving $\mathrm{LN}^+_A(u,v,x)$ undefined. Note that the size of $[y,x_i)_{\mathrm{lex}}$ is doubled after each iteration, so in any case no more than $\mathcal{O}(p(|v|))$ elements of the sequence will need to be computed. □

In order to show that $\mathsf{IF}^{\mathsf{LN}}_{\mathsf{t}} \subseteq \mathsf{TotP}$, we essentially reduce the problem of counting the strings in $(b(x),t(x))_A$ to the problem of counting the strings in $(w_{\min},w_{\max})_{\mathrm{lex}}$, where $w_{\min}$ and $w_{\max}$ are, respectively, the lexicographically smallest and largest strings in

$(b(x), t(x))_A$. At the same time, we use the functions LN_A^+ and LN_A^- in such a way as to avoid creating computation paths for those strings of $(w_{\min}, w_{\max})_{\mathrm{lex}}$ that are not actually inside the interval $(b(x), t(x))_A$.

LEMMA 4.7. $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}} \subseteq \mathsf{TotP}$.

PROOF. Let f be an $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$ function, via a total p-order $A \in \mathsf{P}$ with bounding polynomial p and boundary functions $b, t \in \mathsf{FP}$. By definition, $\mathrm{LN}_A \in \mathsf{FP}$, therefore by Lemma 4.4 we also have $\mathrm{LN}_A^+ \in \mathsf{FP}$ and $\mathrm{LN}_A^- \in \mathsf{FP}$.

We claim that the NPTM N that executes Algorithm 4.8 below performs a computation with exactly $\|(b(x), t(x))_A\| + 1$ computation paths, given input x. First, note that if the procedure MAKETREE is correct, then Algorithm 4.8 is indeed correct: In Steps 1–2, u and v are assigned the lexicographically smallest and the lexicographically largest strings in $[b(x), t(x)]_A$, respectively (assuming the interval is non-empty). Steps 3–4 catch all the cases in which $\|(b(x), t(x))_A\| = 0$: This happens exactly when $t(x) <_A b(x)$, or $b(x) = t(x)$ (which is equivalent to $u = v$), or $b(x) \prec_A t(x)$ (which is equivalent to $u = \mathrm{LN}_A^-(b(x), t(x), \mathrm{pred}_{\mathrm{lex}}(v))$). Finally, if MAKETREE is correct, Steps 5–8 produce a computation tree with exactly $\|(b(x), t(x))_A\| + 1$ paths, since u, v, x clearly satisfy the input assertion.

ALGORITHM 4.8. Computing an $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$ function in the TotP sense.

Input: a string $x \in \Sigma^\star$

Goal: produce a computation tree with $\|(b(x), t(x))_A\| + 1$ computation paths

1. $u \leftarrow \mathrm{LN}_A(b(x), t(x), \varepsilon)$
2. $v \leftarrow \mathrm{LN}_A(b(x), t(x), 0^{p(|t(x)|)+1})$
3. If $t(x) <_A b(x) \vee u \in \{v, \mathrm{LN}_A^-(b(x), t(x), \mathrm{pred}_{\mathrm{lex}}(v))\}$ then
4. halt
5. $\sigma \leftarrow$ choose from $\{0, 1\}$
6. If $\sigma = 1$ then
7. MAKETREE(u, v, x)
8. halt

9. procedure MAKETREE(u, v, x)

Input: strings u, v, x such that $u, v \in [b(x), t(x)]_A$, $u <_{\text{lex}} v$, and $\|[u, v]_{\text{lex}} \cap (b(x), t(x))_A\| > 0$

Goal: produce a computation tree with $\|[u, v]_{\text{lex}} \cap (b(x), t(x))_A\|$ computation paths

10. $z \leftarrow \text{med}_{\text{lex}}(u, v)$
11. $z^- \leftarrow \text{LN}_A^-(b(x), t(x), z)$
12. $z^+ \leftarrow \text{LN}_A^+(b(x), t(x), z)$
13. If $z^- = z^+$ then
14. $\quad z^- \leftarrow \text{LN}_A^-(b(x), t(x), \text{pred}_{\text{lex}}(z^-))$
15. branch_left $\leftarrow \quad u \notin \{b(x), t(x)\}$
$\qquad \vee\, u \notin \{z^-, \text{LN}_A^-(b(x), t(x), \text{pred}_{\text{lex}}(z^-))\}$
$\qquad \vee\, z^- \notin \{b(x), t(x)\}$
16. branch_right $\leftarrow \quad v \notin \{b(x), t(x)\}$
$\qquad \vee\, v \notin \{z^+, \text{LN}_A^+(b(x), t(x), \text{succ}_{\text{lex}}(z^+))\}$
$\qquad \vee\, z^+ \notin \{b(x), t(x)\}$
17. If branch_left $\wedge$ branch_right then
18. $\quad \sigma \leftarrow$ choose from $\{0, 1\}$
19. Else if branch_left then
20. $\quad \sigma \leftarrow 0$
21. Else if branch_right then
22. $\quad \sigma \leftarrow 1$
23. If $\sigma = 0 \wedge u \neq z^-$ then
24. $\quad$ MAKETREE(u, z^-, x)
25. If $\sigma = 1 \wedge z^+ \neq v$ then
26. $\quad$ MAKETREE(z^+, v, x)
27. end procedure

We now show the correctness of procedure MAKETREE. By the input assertion, we obtain immediately Claim 4.9:

CLAIM 4.9. *After the execution of Steps 10–14 in Algorithm 4.8, we have the following:*

(i) $u, v, z^-, z^+ \in [b(x), t(x)]_A$,

(ii) $u \leq_{\text{lex}} z^- <_{\text{lex}} z^+ \leq_{\text{lex}} v$, *and*

(iii) $(z^-, z^+)_{\text{lex}} \cap (b(x), t(x))_A = \emptyset$.

CLAIM 4.10. *After the execution of Steps 15–16 in Algorithm 4.8,* branch_left *is true if and only if* $[u, z^-]_{\text{lex}} \cap (b(x), t(x))_A \neq \emptyset$ *and* branch_right *is true if and only if* $[z^+, v]_{\text{lex}} \cap (b(x), t(x))_A \neq \emptyset$.

PROOF. We prove the claim for branch_left only. The proof for branch_right is similar. By Step 15 in Algorithm 4.8, if branch_left is true, then we must have one of the following:

1. $u \notin \{b(x), t(x)\} \vee z^- \notin \{b(x), t(x)\}$, or
2. $u \notin \{z^-, \text{LN}_A^-(b(x), t(x), \text{pred}_{\text{lex}}(z^-))\}$.

In the first case, by Claim 4.9(i), $[u, z^-]_{\text{lex}} \cap (b(x), t(x))_A \neq \emptyset$. In the second case, assuming $\{u, z^-\} = \{b(x), t(x)\}$, we have that $u \neq \text{LN}_A^-(b(x), t(x), \text{pred}_{\text{lex}}(z^-))$, hence there is at least one element of $(b(x), t(x))_A$ in $[u, z^-]_{\text{lex}}$.

For the other direction, assume that branch_left is false. Then, we must have $u, z^- \in \{b(x), t(x)\}$ and also

$$u \in \{z^-, \text{LN}_A^-(b(x), t(x), \text{pred}_{\text{lex}}(z^-))\} \ .$$

If $u = z^-$, then clearly $[u, z^-]_{\text{lex}} \cap (b(x), t(x))_A = \emptyset$. On the other hand, if $u \neq z^-$ and $u = \text{LN}_A^-(b(x), t(x), \text{pred}_{\text{lex}}(z^-))$, then we must have $\{u, z^-\} = \{b(x), t(x)\}$ and there cannot be any string of $(b(x), t(x))_A$ in $[u, z^-]_{\text{lex}}$, thus $[u, z^-]_{\text{lex}} \cap (b(x), t(x))_A = \emptyset$. □

CLAIM 4.11. *After the execution of Steps 15–16 in Algorithm 4.8,* branch_left *and* branch_right *are not both false.*

PROOF. By Claim 4.10, if both flags are false, then

$$[u, z^-]_{\text{lex}} \cap (b(x), t(x))_A = [z^+, v]_{\text{lex}} \cap (b(x), t(x))_A = \emptyset \ .$$

In view of Claim 4.9(iii), this implies that $[u, v]_{\text{lex}} \cap (b(x), t(x))_A = \emptyset$. However, this contradicts the input assertion. □

By Claim 4.11, exactly one of the assignments to the variable σ in Steps 17–22 in Algorithm 4.8 is executed. In any case, Claims 4.10, 4.9(i), and 4.9(ii) guarantee that, if a recursive call

in Steps 24 or 26 in Algorithm 4.8 is executed, then its input satisfies the input assertion and, in fact, $\|[u, z^-]_{\text{lex}}\| < \|[u, v]_{\text{lex}}\|$ and $\|[z^+, v]_{\text{lex}}\| < \|[u, v]_{\text{lex}}\|$. The correctness of MAKETREE follows by a straightforward induction on $\|[u, v]_{\text{lex}}\|$.

Finally, note that the maximum recursion depth is at most polynomial in $|x|$: By construction (Steps 10–14 in Algorithm 4.8), we have $\|[u, z^-]_{\text{lex}}\| \leq \left\lceil \frac{\|[u,v]_{\text{lex}}\|}{2} \right\rceil$ and $\|[z^+, v]_{\text{lex}}\| \leq \left\lceil \frac{\|[u,v]_{\text{lex}}\|}{2} \right\rceil$. However, the maximum possible lexicographic distance between the two arguments u and v of MAKETREE in Step 7 in Algorithm 4.8 is $2^{p(|t(x)|)+1}$. It follows that the recursion depth is at most $\mathcal{O}(p(|t(x)|))$, which is polynomial because $t \in \mathsf{FP}$. □

REMARK 4.12. *Note that in Pagourtzis & Zachos (2006) it was shown that* #MON2SAT *is in* TotP *but the proof can be easily adapted to show that* #MONSAT *is in* TotP *as well. In fact, by slightly extending a property shown in Hemaspaandra* et al. *(2007), namely that it is easy to find the least satisfying assignment that is lexicographically greater than a given assignment, it is possible to show directly that* #MONSAT *is in* $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$.

5. Inside TotP

In this section, we give a characterization of FP as an interval size function class, and show that $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$ is a class that contains FP and is contained in TotP. We provide strong evidence that these inclusions are proper.

THEOREM 5.1. $\mathsf{FP} = \mathsf{IF}_{\mathsf{t}}^{\mathsf{med}} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}} \subseteq \mathsf{TotP}$.

PROOF. For the inclusion $\mathsf{FP} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{med}}$, just note that any FP function can be defined on the lexicographic order by appropriately setting the boundary functions, and that the median function for the lexicographic order is trivially in FP.

For the opposite direction, let $f \in \mathsf{IF}_{\mathsf{t}}^{\mathsf{med}}$ via a total p-order $A \in \mathsf{P}$ with bounding polynomial p and boundary functions $b, t \in \mathsf{FP}$. We claim that $\text{succ}_A \in \mathsf{FP}$. To compute the successor of a given string x, first compute $w = 0^{p(|x|)+1}$. Since A is a total p-order with

bounding polynomial p, it must be that $x <_A w$. Then, compute a sequence of strings $w_0, \ldots, w_k$, where $k \geq 1$, such that $w_0 = w$, for each i, $w_{i+1} = \mathrm{med}_A(x, w_i)$, and k is the smallest index such that $w_k = w_{k-1}$. It then follows from the definition of the median function that $w_k = \mathrm{succ}_A(x)$. Moreover, the computation is concluded in polynomial time, since the size of $[x, w_0]_A$ is $\mathcal{O}(2^{p(p(|x|)+1)})$, and after each iteration the size of $[x, w_i]_A$ is halved. Thus the claim is proved.

Now, given any two strings $x, y \in \Sigma^\star$ with $x <_A y$, we can decide in polynomial time if $\|[x, y)_A\|$ is odd or even: $\|[x, y)_A\|$ is odd if and only if $\mathrm{med}_A(x, y) = \mathrm{med}_A(\mathrm{succ}_A(x), y)$. This implies that we can compute in polynomial time the size of $[x, y)_A$: if $y = \mathrm{succ}_A(x)$ then $\|[x, y)_A\| = 1$, otherwise the size of $[x, y)_A$ is computed recursively as follows:

- if $\|[x, y)_A\|$ is odd then $\|[x, y)_A\| = 2\|[x, \mathrm{med}_A(x, y))_A\| - 1$,
- else $\|[x, y)_A\| = 2\|[x, \mathrm{med}_A(x, y))_A\|$.

Therefore, we can compute $f(x)$ in polynomial time as follows: $f(x) = \|[b(x), t(x))_A\| - 1$.

The inclusion $\mathsf{IF}_{\mathsf{t}}^{\mathsf{med}} \subseteq \mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$ follows from Remark 2.4.

For the last inclusion, $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}} \subseteq \mathsf{TotP}$, recall that if $f \in \mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$, then there exists a total p-order $A \in \mathsf{P}$, a c-relaxed median function $\mathrm{rmed}_A^c \in \mathsf{FP}$, for some $c \in \left(0, \frac{1}{2}\right]$, and $b, t \in \mathsf{FP}$ such that $f(x) = \|(b(x), t(x))_A\|$. Observe that for two strings $u, v \in \Sigma^\star$, $u \prec_A v \iff \mathrm{rmed}_A^c(u, v) = v$. The latter gives us an efficient adjacency check. In order to prove that $f \in \mathsf{TotP}$, we construct an NPTM N which, given a string x, performs a computation with exactly $\|[b(x), t(x))_A\| = \|(b(x), t(x))_A\| + 1$ computation paths. A straightforward induction on $\|[b, t)_A\|$ shows that Algorithm 5.2 below produces the correct number of computation paths. On the other hand, by the properties of the c-relaxed median function, the size of the interval $[b, t)_A$ in each iteration is at most $(1 - c)$ times the size of the interval in the previous iteration. Moreover, $\|[b(x), t(x)]_A\| \leq 2^{\mathcal{O}(p(|t(x)|)}$, therefore each computation path halts in polynomial time.

ALGORITHM 5.2. Computing an $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$ function in the TotP sense.

Input: a string $x \in \Sigma^\star$
Goal: produce a computation tree with $\|(b(x), t(x))_A\| + 1$ computation paths

1. $b \leftarrow b(x)$, $t \leftarrow t(x)$
2. While $b \not\prec_A t$ do 3–8
3. $\quad z \leftarrow \mathrm{rmed}^c_A(b, t)$
4. $\quad \sigma \leftarrow$ choose from $\{0, 1\}$
5. $\quad$ If $\sigma = 0$ then
6. $\quad\quad t \leftarrow z$
7. $\quad$ If $\sigma = 1$ then
8. $\quad\quad b \leftarrow z$

□

We now proceed to define the exponential gap operator, which will bring out the differences between the function classes TotP and $\mathsf{IF}^{\mathsf{rmed}}_{\mathsf{t}}$.

Definition 5.3. *We define the* exponential gap *operator* $\mathcal{C}_{\mathrm{eg}}\cdot$ *as follows: if* $\mathcal{F}$ *is a function class, then* $\mathcal{C}_{\mathrm{eg}} \cdot \mathcal{F}$ *contains exactly the languages* L *for which there exist an* $f \in \mathcal{F}$*, an asymptotically positive polynomial* q*, and a* $\gamma \in \omega(1)$ *such that for all* x*: if* $x \notin L$ *then* $f(x) \leq 2^{q(|x|)}$*, while if* $x \in L$ *then* $f(x) \geq 2^{q(|x|)\cdot\gamma(|x|)}$*.*

It turns out that for any language in NP, there exists a TotP function which has an exponential gap between yes and no instances of the language. On the other hand, we can approximate in polynomial time the value of any $\mathsf{IF}^{\mathsf{rmed}}_{\mathsf{t}}$ function well enough to be able to discern between yes and no instances of any language defined by means of an exponential gap on that function. These claims are proved in Lemmas 5.4 and 5.5 below.

Lemma 5.4. $\mathsf{NP} \subseteq \mathcal{C}_{\mathrm{eg}} \cdot \mathsf{TotP}$.

Proof. Let $L \in \mathsf{NP}$, let N be the NPTM that decides L, and let p be the polynomial that bounds the running time of N. We construct an NPTM N' as follows: on input x, first simulate $N(x)$. If the simulation reaches a rejecting state of N, halt. If an accepting state of N is reached, pick nondeterministically a string of length

$p(|x|) \cdot \log p(|x|)$ and then halt. Now, it is not hard to see that if $x \notin L$ then $\mathrm{tot}_{N'}(x) \leq 2^{p(|x|)}$, while if $x \in L$ then $\mathrm{tot}_{N'}(x) \geq 2^{p(|x|)\cdot \log p(|x|)}$. This proves that $L \in \mathcal{C}_{\mathrm{eg}} \cdot \mathsf{TotP}$. □

LEMMA 5.5. $\mathcal{C}_{\mathrm{eg}} \cdot \mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}} \subseteq \mathsf{P}$.

PROOF. Let $L \in \mathcal{C}_{\mathrm{eg}} \cdot \mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$, so there exist an $f \in \mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$, an asymptotically positive polynomial q, and a $\gamma \in \omega(1)$ as per Definition 5.3. Moreover, let $A \in \mathsf{P}$ be the p-order with bounding polynomial p and $b, t \in \mathsf{FP}$ that are implied by the fact that $f \in \mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$. Let c be the constant of the associated c-relaxed median function $\mathrm{rmed}_A \in \mathsf{FP}$. Finally, let θ be a constant such that for all $n \geq \theta$, the following hold:

$$q(n) \geq \log \frac{1}{c} \geq 1 \text{ and } \gamma(n) \geq 4 \cdot \frac{\log \frac{1}{c}}{\log \frac{1}{1-c}} \tag{5.6}$$

Such a θ must exist in view of the fact that q is asymptotically positive and $\gamma \in \omega(1)$.

In order to decide membership of an input string x in L, we proceed as follows: First, if $|x| \leq \theta$, then membership is decided by using a look-up table of constant size $2^{\theta+1} - 1$. If $|x| > \theta$, then we compute a sequence of strings $t(x) = z_0 >_A z_1 >_A \cdots >_A z_k$, where for all i in the range $1 \leq i \leq k$:

$$z_i = \mathrm{rmed}_A(b(x), z_{i-1})$$

and k is the smallest index such that $b(x) \prec_A z_k$. Finally, we accept if and only if $\frac{1}{(1-c)^k} - \frac{1}{c} > 2^{q(|x|)}$. Note that we can check whether $b(x) \prec_A z_i$ in polynomial time, since it is equivalent to $b(x) \neq z_i \wedge \mathrm{rmed}_A(b(x), z_i) = z_i$.

The correctness of the algorithm is based on the following claim:

CLAIM 5.7. $\frac{1}{(1-c)^k} - \frac{1}{c} \leq f(x) < \frac{2}{c^k}$.

PROOF. We prove first the upper bound. For $i \geq 0$, we denote $\mathcal{S}_i = \|[b(x), z_i)_A\|$. By definition of the relaxed median, we have for $i \geq 1$:

$$\mathcal{S}_i \geq \lfloor c \cdot \|[b(x), z_{i-1}]_A\| \rfloor = \lfloor c \cdot \mathcal{S}_{i-1} + c \rfloor > c \cdot \mathcal{S}_{i-1} + c - 1 \ .$$

From this recurrence, we obtain for all i:

$$\mathcal{S}_i \geq c^i \cdot \mathcal{S}_0 + c^i - 1 \ . \tag{5.8}$$

We set $i = k$ in (5.8) and, since $\mathcal{S}_0 = f(x) + 1$ and $\mathcal{S}_k = 1$, we obtain $1 \geq c^k \cdot (f(x) + 1) + c^k - 1$ and, therefore, $f(x) \leq \frac{2}{c^k} - 2 < \frac{2}{c^k}$.

We now prove the lower bound. For $i \geq 1$, we denote $\mathcal{T}_i = \|[z_i, z_{i-1}]_A\|$. Observe that $\mathcal{S}_i + \mathcal{T}_i = \mathcal{S}_{i-1} + 1$. Moreover, by definition of the relaxed median, we have:

$$\mathcal{T}_i \geq \lceil c \cdot \|[b(x), z_{i-1}]_A\| \rceil = \lceil c \cdot \mathcal{S}_{i-1} + c \rceil \geq c \cdot \mathcal{S}_{i-1} + c \ .$$

It follows, then, that

$$\mathcal{S}_i = \mathcal{S}_{i-1} - \mathcal{T}_i + 1 \leq (1 - c) \cdot \mathcal{S}_{i-1} + 1 - c \ .$$

From this recurrence, we obtain for $i \geq 1$:

$$\mathcal{S}_i \leq (1 - c)^i \cdot \mathcal{S}_0 + \frac{1 - c}{c} \cdot \left(1 - (1 - c)^i\right) \ . \tag{5.9}$$

Equation (5.9) also clearly holds for $i = 0$. It follows that we can always set $i = k$ (even if $k = 0$) and we obtain, after rearranging, $f(x) \geq \frac{1}{(1-c)^k} - \frac{1}{c}$. □

If the algorithm accepts, then we have $\frac{1}{(1-c)^k} - \frac{1}{c} > 2^{q(|x|)}$. In view of Claim 5.7, this implies $f(x) > 2^{q(|x|)}$ and, therefore, $x \in L$ by Definition 5.3.

On the other hand, if the algorithm rejects, then it must be the case that $\frac{1}{(1-c)^k} - \frac{1}{c} \leq 2^{q(|x|)}$, which gives $k \leq \log_{\frac{1}{1-c}} \left(\frac{1}{c} + 2^{q(|x|)}\right)$. By the first part of (5.6), we have $\frac{1}{c} + 2^{q(|x|)} \leq 2^{q(|x|)+1} \leq 2^{2q(|x|)}$. Therefore,

$$k \leq q(|x|) \cdot \log_{\frac{1}{1-c}} 4 \ . \tag{5.10}$$

Additionally, by the second part of (5.6), we have $\log_{\frac{1}{1-c}} 4 \leq \gamma(|x|) \cdot \log_{\frac{1}{c}} \sqrt{2}$. Combining this with (5.10), we obtain $k \leq q(|x|) \cdot \gamma(|x|) \cdot \log_{\frac{1}{c}} \sqrt{2} = \frac{q(|x|) \cdot \gamma(|x|)}{2} \cdot \log_{\frac{1}{c}} 2$. From (5.6), we can easily derive that $q(|x|) \cdot \gamma(|x|) > 2$ for all $c \in \left(0, \frac{1}{2}\right]$. Therefore, $\frac{q(|x|) \cdot \gamma(|x|)}{2} \leq q(|x|) \cdot$

$\gamma(|x|) - 1$ and we finally have that $k \leq (q(|x|) \cdot \gamma(|x|) - 1) \cdot \log_{\frac{1}{c}} 2$. By rearranging, we obtain $\frac{2}{c^k} \leq 2^{q(|x|)\cdot\gamma(|x|)}$. In view of Claim 5.7, this implies $f(x) < 2^{q(|x|)\cdot\gamma(|x|)}$ and, therefore, $x \notin L$ by Definition 5.3.

It remains to show that the algorithm runs in polynomial time. It suffices to show that k is bounded by a polynomial. By the lower bound of Claim 5.7 for $f(x)$, we have $k \leq \log_{\frac{1}{1-c}} \left(f(x) + \frac{1}{c}\right)$. Since A is a p-order with bounding polynomial p, from Remark 2.2 we have $f(x) \leq 2^{p(|t(x)|)+1} - 1$, therefore $k = \mathcal{O}\left(p(|t(x)|)\right)$, which is polynomial because $t \in \mathsf{FP}$. □

Lemmas 5.4 and 5.5 directly imply the following.

THEOREM 5.11. *If* $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}} = \mathsf{TotP}$, *then* $\mathsf{P} = \mathsf{NP}$.

We now proceed to separate $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$ from FP. Consider the following problem:

PROBLEM 5.12 ($\#\mathrm{SAT}_{+2^n}$). *Given a Boolean formula φ with n variables, count the number of satisfying assignments of the formula $\varphi \vee x_{n+1}$, where x_{n+1} is a fresh variable not appearing in φ.*

It is immediate that $\#\mathrm{SAT}_{+2^n} \in \#\mathsf{P}$ and $\#\mathrm{SAT}_{+2^n}(\varphi) = \#\mathrm{SAT}(\varphi) + 2^n$, where n is the number of variables in φ and $\#\mathrm{SAT}(\varphi)$ is the $\#\mathsf{P}$-complete function giving the number of satisfying assignments of φ.[2] Therefore, $\#\mathrm{SAT}_{+2^n}$ is $\#\mathsf{P}$-complete under the Cook-1 reduction. We show that $\#\mathrm{SAT}_{+2^n}$ is actually in $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$. Consequently, $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}} \neq \mathsf{FP}$, unless $\#\mathsf{P}$ also collapses to FP.

THEOREM 5.13. $\#\mathrm{SAT}_{+2^n} \in \mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$.

PROOF. We assume a reasonable encoding scheme of boolean formulas as binary strings. Without loss of generality, we also assume that, under this scheme, there is no pair of distinct formulas φ, φ' such that the encoding of φ is a prefix of the encoding of φ'. This

[2] The appearance of n in the right-hand side of $\#\mathrm{SAT}_{+2^n}(\varphi) = \#\mathrm{SAT}(\varphi) + 2^n$ is not to be confused with the appearance of n in the left-hand side, where it is simply part of the name of the function $\#\mathrm{SAT}_{+2^n}$.

can be ensured, for example, by simultaneously replacing, in the encoding of a formula, all occurrences of '0' by '01', all occurrences of '1' by '10', and terminating with '00'. In particular, this implies that, for any given string w, there is at most one prefix of w that encodes a boolean formula, and it can be found in polynomial time.

If x encodes a boolean formula $\varphi(x)$ with $n(x) \geq 0$ variables denoted by $v_1, \ldots, v_{n(x)}$, then a truth assignment for $\varphi(x)$ is represented by a binary string ξ with $|\xi| = n(x)$, such that the i-th bit of ξ corresponds to the truth value of v_i. Let $\mathrm{Sat}(x)$ denote the set of strings ξ that represent satisfying truth assignments for $\varphi(x)$.

We will show that the function $f(x) = \|\mathrm{Sat}(x)\| + 2^{n(x)}$ is in $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}}$. We can assume that $f(x) = 0$ if x is not a valid encoding of a boolean formula. We construct an order A that coincides with the lexicographic order of $\Sigma^\star$, except that for every $x \in \Sigma^\star$ that is a valid encoding of a boolean formula $\varphi(x)$, the interval of size $4 \cdot 2^{n(x)}$ which we will henceforth denote by $\mathcal{I}(x) = [x0^{n(x)+2}, x1^{n(x)+2}]_{\mathrm{lex}}$ is ordered differently as follows:

- First come the elements of the set

$$\begin{aligned} &\{x\xi 00 : |\xi| = n(x)\} \\ \cup&\{x\xi 10 : |\xi| = n(x) \wedge \xi \in \mathrm{Sat}(x)\} \ , \end{aligned}$$

  in lexicographic order,

- and last come the elements of the set

$$\begin{aligned} &\{x\xi 01 : |\xi| = n(x)\} \\ \cup&\{x\xi 10 : |\xi| = n(x) \wedge \xi \notin \mathrm{Sat}(x)\} \\ \cup&\{x\xi 11 : |\xi| = n(x)\} \ , \end{aligned}$$

  in lexicographic order.

Note that the properties of our encoding ensure that, for any pair of distinct boolean formula encodings x, x', we have $\mathcal{I}(x) \cap \mathcal{I}(x') = \emptyset$. Moreover, for any given string w, we can find in polynomial time an x such that $w \in \mathcal{I}(x)$, or decide that no such x exists.

REMARK 5.14. *It may be helpful to spend a moment to visualize the ordering of the strings of an interval $\mathcal{I}(x)$. Note that the first part of $\mathcal{I}(x)$ contains, in lexicographic order, for* each *truth assignment ξ of length $n(x)$, either the two strings $\{x\xi 00, x\xi 10\}$ (if $\xi \in \mathrm{Sat}(x)$) or the single string $\{x\xi 00\}$ (if $\xi \notin \mathrm{Sat}(x)$). The total number of strings in the first part of $\mathcal{I}(x)$ is, therefore, exactly equal to $f(x) = \|\mathrm{Sat}(x)\| + 2^{n(x)}$. Similarly, the second part of $\mathcal{I}(x)$ contains, in lexicographic order, for* each *truth assignment ξ of length $n(x)$, either the two strings $\{x\xi 01, x\xi 11\}$ (if $\xi \in \mathrm{Sat}(x)$) or the three strings $\{x\xi 01, x\xi 10, x\xi 11\}$ (if $\xi \notin \mathrm{Sat}(x)$). Therefore, the second part of $\mathcal{I}(x)$ contains the remaining $3 \cdot 2^{n(x)} - \|\mathrm{Sat}(x)\|$ strings. It is easy to decide in which part of the interval lies a given string $x\xi ab$, where $a, b \in \Sigma$: it is in the first part if and only if $ab = 00$, or $ab = 10$ and ξ satisfies $\varphi(x)$.*

In view of the definition of the order A and Remark 5.14, it is not hard to verify that A is a P-decidable p-order with $\mathrm{succ}_A \in \mathsf{FP}$ (therefore, it also has efficient adjacency checks). Indeed, to decide whether $w_1 <_A w_2$, we first identify the intervals to which the two strings belong: If it is not the case that both are in the same interval, then $w_1 <_A w_2$ if and only if $w_1 <_{\mathrm{lex}} w_2$, whereas if both are in the same interval, then it suffices to identify which part of the interval each string lies in and, if they are in the same part, to compare them lexicographically. In order to compute succ_A in polynomial time, we first check whether the input string belongs to some interval $\mathcal{I}(x)$: If this is not the case, then the successor is simply the lexicographic successor. On the other hand, if the input string is in some interval $\mathcal{I}(x)$, then the successor is computed in polynomial time according to Table 5.1.

By choosing the boundary functions $b(x) = \mathrm{pred}_{\mathrm{lex}}(x0^{n(x)+2}) = \mathrm{pred}_A(x0^{n(x)+2})$, i.e., the string immediately before the first element of $\mathcal{I}(x)$ under A, and $t(x) = x0^{n(x)+1}1$, i.e., the string immediately after the $\|\mathrm{Sat}(x)\| + 2^{n(x)}$ elements contained in the first part of the interval, we have $\|(b(x), t(x))_A\| = \|\mathrm{Sat}(x)\| + 2^{n(x)} = f(x)$ (if x does not encode a boolean formula, we can choose $b(x) = t(x) = \varepsilon$). In the following, we will show that, for some fixed constant c, a c-relaxed median between any two given elements $w_1 <_A w_2$ of A can be computed in polynomial time.

Table 5.1: Computation of the successor function for the order A defined in the proof of Theorem 5.13. Let $w = x\xi ab \in \mathcal{I}(x)$, where $a, b \in \Sigma$, and let $\xi' = \mathrm{succ}_{\mathrm{lex}}(\xi)$. Each row of the table contains one of the different cases for computing $w' = \mathrm{succ}_A(w)$, based on the values of a and b, whether $\xi = 1^{n(x)}$, and whether $\xi \in \mathrm{Sat}(x)$.

Condition	Successor
$ab = 00 \wedge \xi \in \mathrm{Sat}(x)$	$w' = x\xi 10$
$ab = 00 \wedge \xi \notin \mathrm{Sat}(x) \wedge \xi \neq 1^{n(x)}$	$w' = x\xi' 00$
$ab = 00 \wedge \xi \notin \mathrm{Sat}(x) \wedge \xi = 1^{n(x)}$	$w' = x0^{n(x)+1}1$
$ab = 01 \wedge \xi \in \mathrm{Sat}(x)$	$w' = x\xi 11$
$ab = 01 \wedge \xi \notin \mathrm{Sat}(x)$	$w' = x\xi 10$
$ab = 10 \wedge \xi \in \mathrm{Sat}(x) \wedge \xi \neq 1^{n(x)}$	$w' = x\xi' 00$
$ab = 10 \wedge \xi \in \mathrm{Sat}(x) \wedge \xi = 1^{n(x)}$	$w' = x0^{n(x)+1}1$
$ab = 10 \wedge \xi \notin \mathrm{Sat}(x)$	$w' = x\xi 11$
$ab = 11 \wedge \xi \neq 1^{n(x)}$	$w' = x\xi' 01$
$ab = 11 \wedge \xi = 1^{n(x)}$	$w' = \mathrm{succ}_{\mathrm{lex}}(w)$

We start with the observation that, for any fixed constant s, we can check in polynomial time whether $\|[w_1, w_2]_A\| \leq s$: we apply s times the function $\mathrm{succ}_A \in \mathsf{FP}$ starting from w_1, and check whether we reached w_2. If $\|[w_1, w_2]_A\| \leq s$, then we can clearly also compute a $\frac{1}{2}$-relaxed median for w_1 and w_2 in polynomial time, by applying succ_A at most $\frac{s}{2}$ times. Therefore, in the rest of the proof, we will assume that $\|[w_1, w_2]_A\| > s$, where s is a constant whose value is to be determined.

Case 1: We first handle the case where both $w_1, w_2 \in \mathcal{I}(x)$, for some x. That is, for $i \in \{1, 2\}$, we have $w_i = x\xi_i a_i b_i$, where $a_i, b_i \in \Sigma$ and $|\xi_i| = n(x)$. We distinguish the following subcases:

Subcase 1.(i): If $w_1 <_A w_2 <_A t(x)$, then, by demanding $s \geq 2$, we can assume that $\xi_1 <_{\mathrm{lex}} \xi_2$. We return $w = x\xi 00$ as a relaxed median of w_1 and w_2, where $\xi = \mathrm{med}_{\mathrm{lex}}(\xi_1, \xi_2)$. Indeed, we know by construction of the order A and the definition of the median function that $\|[w_1, w)_A\| \geq \|[\xi_1, \xi)_{\mathrm{lex}}\| \geq \left\lfloor \frac{1}{2}\|[\xi_1, \xi_2]_{\mathrm{lex}}\| \right\rfloor$ and $\|[w, w_2]_A\| \geq \|[\xi, \xi_2]_{\mathrm{lex}}\| \geq \left\lceil \frac{1}{2}\|[\xi_1, \xi_2]_{\mathrm{lex}}\| \right\rceil$. On the other hand,

$\|[w_1, w_2]_A\| \leq 2\|[\xi_1, \xi_2]_{\text{lex}}\|$. Therefore, we obtain $\|[w_1, w)_A\| \geq \left\lfloor \frac{1}{4}\|[w_1, w_2]_A\| \right\rfloor$ and $\|[w, w_2]_A\| \geq \left\lceil \frac{1}{4}\|[w_1, w_2]_A\| \right\rceil$.

Subcase 1.(ii): If $t(x) \leq_A w_1 <_A w_2$, then, by demanding $s \geq 3$, we can assume that $\xi_1 <_{\text{lex}} \xi_2$. We return $w = x\xi 01$, where $\xi = \text{med}_{\text{lex}}(\xi_1, \xi_2)$. We have, as in Subcase 1.(i), that $\|[w_1, w)_A\| \geq \left\lfloor \frac{1}{2}\|[\xi_1, \xi_2]_{\text{lex}}\| \right\rfloor$ and $\|[w, w_2]_A\| \geq \left\lceil \frac{1}{2}\|[\xi_1, \xi_2]_{\text{lex}}\| \right\rceil$, whereas we have $\|[w_1, w_2]_A\| \leq 3\|[\xi_1, \xi_2]_{\text{lex}}\|$. Therefore, $\|[w_1, w)_A\| \geq \left\lfloor \frac{1}{6}\|[w_1, w_2]_A\| \right\rfloor$ and $\|[w, w_2]_A\| \geq \left\lceil \frac{1}{6}\|[w_1, w_2]_A\| \right\rceil$.

Subcase 1.(iii): If $w_1 <_A t(x) \leq_A w_2$, let $\delta_1 = \|[\xi_1, 1^{n(x)}]_{\text{lex}}\|$ and $\delta_2 = \|[0^{n(x)}, \xi_2]_{\text{lex}}\|$. We claim that the relaxed median of w_1 and w_2 can be computed as follows: if $\delta_1 > \delta_2$, then the relaxed median is $w = x0\xi 00$, where ξ is a string with $\|[\xi_1, \xi)_{\text{lex}}\| = \left\lfloor \frac{\delta_1 + \delta_2}{2} \right\rfloor$, otherwise the relaxed median is $w = x0\xi 01$, where ξ is a string with $\|[\xi, \xi_2]_{\text{lex}}\| = \left\lceil \frac{\delta_1 + \delta_2}{2} \right\rceil$.

Indeed, if $\delta_1 > \delta_2$, then we have $\|[w_1, w)_A\| \geq \|[\xi_1, \xi)_{\text{lex}}\| = \left\lfloor \frac{\delta_1 + \delta_2}{2} \right\rfloor$ and $\|[w, w_2]_A\| \geq \|[\xi, 1^{n(x)}]_{\text{lex}}\| + \|[0^{n(x)}, \xi_2]_{\text{lex}}\| = \delta_1 - \left\lfloor \frac{\delta_1 + \delta_2}{2} \right\rfloor + \delta_2 = \left\lceil \frac{\delta_1 + \delta_2}{2} \right\rceil$. At the same time, $\|[w_1, w_2]_A\| \leq 3(\delta_1 + \delta_2)$, from which we obtain $\|[w_1, w)_A\| \geq \left\lfloor \frac{1}{6}\|[w_1, w_2]_A\| \right\rfloor$ and also $\|[w, w_2]_A\| \geq \left\lceil \frac{1}{6}\|[w_1, w_2]_A\| \right\rceil$. The case $\delta_1 \leq \delta_2$ is handled by a completely symmetric argument.

Case 2: We now assume that there is no x, such that both $w_1, w_2 \in \mathcal{I}(x)$. Let $w = \text{med}_{\text{lex}}(w_1, w_2)$. We will call a string *free* if its rank under A is equal to its rank under lex. This is the case for strings that do not belong in $\mathcal{I}(x)$, for any $x \in \Sigma^\star$, and for the first and last elements of each interval $\mathcal{I}(x)$. Note that if z_1, z_2 are free, then $[z_1, z_2]_A = [z_1, z_2]_{\text{lex}}$. We have the following subcases:

Subcase 2.(i): If w_1 and w_2 are free, then we have two further subcases for w. If w is also free, then we can safely return w as the relaxed median. On the other hand, if $w \in \mathcal{I}(x)$ for some x, then we return $w' = t(x)$ as the relaxed median. Let $u = x1^{n(x)+2}$, i.e., u is the last element of $\mathcal{I}(x)$ both lexicographically and under A, and define δ_1 and δ_2 as:

$$\delta_1 = \|[w_1, b(x)]_A\| = \|[w_1, b(x)]_{\text{lex}}\| \ ,$$
$$\delta_2 = \|(u, w_2]_A\| = \|(u, w_2]_{\text{lex}}\| \ .$$

Now, it is easy to verify that we must have $|\delta_1 - \delta_2| \leq \|\mathcal{I}(x)\|$, otherwise the lexicographic median w would not fall in $\mathcal{I}(x)$. We

rewrite this as $\max\{\delta_1, \delta_2\} - \min\{\delta_1, \delta_2\} \leq \|\mathcal{I}(x)\|$. Furthermore, it holds by construction of the interval $\mathcal{I}(x)$ that $\|(b(x), t(x))_A\| \geq \frac{1}{4}\|\mathcal{I}(x)\|$ and $\|[t(x), u]_A\| \geq \frac{1}{2}\|\mathcal{I}(x)\|$. From these observations we obtain:

$$\begin{aligned} \|[w_1, w_2]_A\| &= \delta_1 + \|\mathcal{I}(x)\| + \delta_2 \\ &= \max\{\delta_1, \delta_2\} + \min\{\delta_1, \delta_2\} + \|\mathcal{I}(x)\| \\ &\leq 2\min\{\delta_1, \delta_2\} + 2\|\mathcal{I}(x)\| \end{aligned} \tag{5.15}$$

On the other hand, for the candidate relaxed median $w' = t(x)$ we have:

$$\begin{aligned} \|[w_1, w')_A\| &= \|[w_1, b(x)]_A\| + \|(b(x), t(x))_A\| \\ &\geq \min\{\delta_1, \delta_2\} + \frac{1}{4}\|\mathcal{I}(x)\| \end{aligned} \tag{5.16}$$

and

$$\begin{aligned} \|[w', w_2]_A\| &= \|[t(x), u]_A\| + \|(u, w_2]_A\| \\ &\geq \min\{\delta_1, \delta_2\} + \frac{1}{2}\|\mathcal{I}(x)\| \end{aligned} \tag{5.17}$$

From (5.15), (5.16), and (5.17), we deduce that $\|[w_1, w')_A\| \geq \frac{1}{8}\|[w_1, w_2]_A\| \geq \left\lfloor \frac{1}{8}\|[w_1, w_2]_A\| \right\rfloor$ and $\|[w', w_2]_A\| \geq \frac{1}{4}\|[w_1, w_2]_A\|$, thus $\|[w', w_2]_A\| \geq \left\lceil \frac{1}{4}\|[w_1, w_2]_A\| \right\rceil$, since $\|[w', w_2]_A\|$ is an integer.

Subcase 2.(ii): If $w_1 \in \mathcal{I}(x)$ and w_2 is free, then we have four further subcases, which we present below as Claims 5.18–5.21. Let $u = x1^{n(x)+2}$ and $u' = \mathrm{succ}_{\mathrm{lex}}(u) = \mathrm{succ}_A(u)$. We have $\|[u', w_2]_A\| = \|[u', w_2]_{\mathrm{lex}}\|$ and, by construction of the interval $\mathcal{I}(x)$, if $w_1 \geq_A t(x)$, then $\frac{1}{2}\|[w_1, u]_{\mathrm{lex}}\| \leq \|[w_1, u]_A\| \leq \|[w_1, u]_{\mathrm{lex}}\|$. Let $\delta = \|[u', w_2]_A\|$.

Claim 5.18. *If $\delta < 4 \cdot 2^{n(x)}$ and $w_1 <_A t(x)$, then $w' = x10^{n(x)}1$ is a relaxed median for w_1 and w_2.*

Proof. From the assumption about δ, we have $\|[w_1, w_2]_A\| \leq 8 \cdot 2^{n(x)}$. On the other hand, $\|[w_1, w')_A\| \geq \|[t(x), w')_A\| \geq 2^{n(x)}$ and $\|[w', w_2]_A\| \geq \|[w', u]_A\| \geq 2^{n(x)}$. The claim follows. □

CLAIM 5.19. *If $w_1 \geq_A t(x)$ and $w \in \mathcal{I}(x)$, then a relaxed median for w_1 and w_2 is given by w if $w \geq_A t(x)$, or by $w' = \mathrm{succ}_{\mathrm{lex}}(w)$ if $w <_A t(x)$.*

PROOF. We first assume that $w \geq_A t(x)$. By demanding that $s \geq 3$, we can also assume that $\|[w_1, w_2]_A\| \geq 4$, which implies $\|[w_1, w)_{\mathrm{lex}}\| \geq 2$. Therefore, we have $\|[w_1, w)_A\| = \|[w_1, w]_A\| - 1 \geq \frac{1}{2}\|[w_1, w]_{\mathrm{lex}}\| - 1 = \frac{1}{2}(\|[w_1, w)_{\mathrm{lex}}\| - 1) \geq \frac{1}{4}\|[w_1, w)_{\mathrm{lex}}\|$ and $\|[w, w_2]_A\| = \|[w, u]_A\| + \|[u', w_2]_A\| \geq \frac{1}{2}\|[w, u]_{\mathrm{lex}}\| + \|[u', w_2]_{\mathrm{lex}}\| \geq \frac{1}{2}\|[w, w_2]_{\mathrm{lex}}\|$. On the other hand, it holds that $\|[w_1, w_2]_A\| = \|[w_1, u]_A\| + \|[u', w_2]_A\| \leq \|[w_1, u]_{\mathrm{lex}}\| + \|[u', w_2]_{\mathrm{lex}}\| \leq \|[w_1, w_2]_{\mathrm{lex}}\|$. We may, then, conclude that $\|[w_1, w)_A\| \geq \frac{1}{4}\left\lfloor \frac{1}{2}\|[w_1, w_2]_{\mathrm{lex}}\| \right\rfloor \geq \frac{1}{4}\left\lfloor \frac{1}{2}\|[w_1, w_2]_A\| \right\rfloor \geq \frac{1}{10}\|[w_1, w_2]_A\|$ (taking also into account that $\|[w_1, w_2]_A\| \geq 4$) and $\|[w, w_2]_A\| \geq \frac{1}{4}\|[w_1, w_2]_{\mathrm{lex}}\| \geq \frac{1}{4}\|[w_1, w_2]_A\|$.

If $w <_A t(x)$, note that, by construction of $\mathcal{I}(x)$, we have $\mathrm{succ}_{\mathrm{lex}}(w) \in \mathcal{I}(x)$ and, in fact, $\mathrm{succ}_{\mathrm{lex}}(w) \geq_A w_1 \geq_A t(x)$. As before, we have $\|[w', w_2]_A\| = \|[w', u]_A\| + \|[u', w_2]_A\| \geq \frac{1}{2}\|[w', u]_{\mathrm{lex}}\| + \|[u', w_2]_{\mathrm{lex}}\| = \frac{1}{2}\left(\|[w, u]_{\mathrm{lex}}\| - 1\right) + \|[u', w_2]_{\mathrm{lex}}\| \geq \frac{1}{4}\|[w, w_2]_{\mathrm{lex}}\|$ (in view of the fact that $w <_A t(x)$, which implies $\|[w, u]_{\mathrm{lex}}\| \geq 2$ and therefore $\|[w, u]_{\mathrm{lex}}\| - 1 \geq \frac{1}{2}\|[w, u]_{\mathrm{lex}}\|$), and the other bounds remain $\|[w_1, w')_A\| \geq \frac{1}{4}\|[w_1, w)_{\mathrm{lex}}\|$ and $\|[w_1, w_2]_A\| \leq \|[w_1, w_2]_{\mathrm{lex}}\|$. We conclude that $\|[w_1, w')_A\| \geq \frac{1}{10}\|[w_1, w_2]_A\|$ and $\|[w', w_2]_A\| \geq \frac{1}{8}\|[w_1, w_2]_A\|$. □

CLAIM 5.20. *If $\delta \geq 4 \cdot 2^{n(x)}$, then $w' = \mathrm{rmed}_A(u', w_2)$, as computed according to Subcase 2.(i), is a relaxed median for w_1 and w_2.*

PROOF. Note that u' and w_2 are both free, therefore w' can be computed according to Subcase 2.(i) and we have $\|[u', w')_A\| \geq \frac{1}{8}\|[u', w_2]_A\| = \frac{1}{8}\delta$ and $\|[w', w_2]_A\| \geq \frac{1}{8}\|[u', w_2]_A\| = \frac{1}{8}\delta$. Moreover, $\|[w_1, w_2]_A\| = \|[w_1, u]_A\| + \|[u', w_2]_A\| \leq 4 \cdot 2^{n(x)} + \delta \leq 2\delta$ and $\|[w_1, w')_A\| \geq \|[u', w')_A\| \geq \frac{1}{8}\delta$. The claim follows. □

CLAIM 5.21. *If $\delta < 4 \cdot 2^{n(x)}$, $w_1 \geq_A t(x)$, and $w \notin \mathcal{I}(x)$, then $w' = \mathrm{rmed}_A(u', w_2)$, as computed according to Subcase 2.(i), is a relaxed median for w_1 and w_2.*

PROOF. The strings u' and w_2 are both free, thus Case 2.(i) applies. The string w' is such that $\|[u', w')_A\| \geq \frac{1}{8}\|[u', w_2]_A\| = \frac{1}{8}\delta$ and $\|[w', w_2]_A\| \geq \frac{1}{8}\|[u', w_2]_A\| = \frac{1}{8}\delta$. Moreover, since the lexicographic median w falls outside of $\mathcal{I}(x)$, it follows that $\delta \geq \|[w_1, u]_{\text{lex}}\| \geq \|[w_1, u]_A\|$. We have, then, that $\|[w_1, w_2]_A\| = \|[w_1, u]_A\| + \delta$, $\|[w_1, w')_A\| = \|[w_1, u]_A\| + \|[u', w')_A\| \geq \|[w_1, u]_A\| + \frac{1}{8}\delta$, and, finally, $\|[w', w_2]_A\| \geq \frac{1}{8}\delta = \frac{1}{16}\delta + \frac{1}{16}\delta \geq \frac{1}{16}\delta + \frac{1}{16}\|[w_1, u]_A\|$. The claim follows. □

Subcase 2.(iii): If w_1 is free and w_2 is not free, then the relaxed median is obtained by operations and arguments that are completely symmetric to those of Subcase 2.(ii).

Subcase 2.(iv): Finally, we handle the case where $w_1 \in \mathcal{I}(x)$ and $w_2 \in \mathcal{I}(y)$ with $x \neq y$. Let $u = x1^{n(x)+2}$ and $v = y0^{n(y)+2}$. Also, let $w_1 = x\xi_1 a_1 b_1$ and $w_2 = y\xi_2 a_2 b_2$, where $|\xi_1| = n(x)$, $|\xi_2| = n(y)$, and $a_i, b_i \in \Sigma$. We define the following quantities:

$$\tilde{R} = \begin{cases} 2\|(\xi_1, 1^{n(x)}]_{\text{lex}}\| + 1 + 2 \cdot 2^{n(x)}, & \text{if } w_1 <_A t(x) \\ 2\|(\xi_1, 1^{n(x)}]_{\text{lex}}\| + 1, & \text{if } w_1 \geq_A t(x) \end{cases} \text{ and}$$

$$\tilde{L} = \begin{cases} \|[0^{n(y)}, \xi_2]_{\text{lex}}\|, & \text{if } w_2 <_A t(y) \\ 3\|[0^{n(y)}, \xi_2]_{\text{lex}}\| + 1 + 2^{n(y)}, & \text{if } w_2 \geq_A t(y) \end{cases},$$

which are computable in polynomial time, given w_1 and w_2, respectively. We use $\tilde{R}$ and $\tilde{L}$ to approximate $\|[w_1, u]_A\|$ and $\|[v, w_2]_A\|$, respectively (Claims 5.22 and 5.23). We only prove Claim 5.22, as Claim 5.23 can be proved by similar arguments, the difference in the constant being due to the lopsided distribution of strings on the left and on the right of $t(x)$, for any given interval $\mathcal{I}(x)$.

CLAIM 5.22. $\tilde{R} \leq \|[w_1, u]_A\| \leq 3\tilde{R}$.

PROOF. For the first inequality: If $w_1 \geq_A t(x)$, then, by construction of the order A, on the right of w_1 in $\mathcal{I}(x)$, there are at least the two strings $x\xi01$ and $x\xi11$ for each assignment $\xi >_{\text{lex}} \xi_1$, plus w_1 itself. If $w_1 <_A t(x)$, then on the right of w there are four strings for each assignment $\xi >_{\text{lex}} \xi_1$, at least two strings for each assignment $\xi <_{\text{lex}} \xi_1$, and at least three strings for the assignment $\xi = \xi_1$.

Therefore, there are at least $4\|(\xi_1, 1^{n(x)}]_{\text{lex}}\| + 2\|[0^{n(x)}, \xi_1)_{\text{lex}}\| + 3 = 2 \cdot 2^{n(x)} + 1 + 2\|(\xi_1, 1^{n(x)}]_{\text{lex}}\|$ strings.

For the second inequality: If $w_1 \geq_A t(x)$, then there are at most three strings on the right of w_1 for each $\xi \geq_{\text{lex}} \xi_1$. If $w_1 <_A t(x)$, then there are at most three strings for each $\xi <_{\text{lex}} \xi_1$ and at most four for each $\xi \geq_{\text{lex}} \xi_1$, for a total of at most $3 \cdot 2^{n(x)} + 1 + \|(\xi_1, 1^{n(x)}]_{\text{lex}}\|$ strings. □

CLAIM 5.23. $\tilde{L} \leq \|[v, w_2]_A\| \leq 2\tilde{L}$.

We now distinguish four further subcases for Subcase 2.(iv), which we present below as Claims 5.24–5.26 and 5.28. Let $u' = \text{succ}_{\text{lex}}(u) = \text{succ}_A(u)$, and $\delta = \|[u', b(y)]_{\text{lex}}\| = \|[u', b(y)]_A\|$.

CLAIM 5.24. *If $\delta \geq \max\{\tilde{R}, \tilde{L}\}$, then $w' = \text{rmed}_A(u', b(y))$, as computed according to Subcase 2.(i), is a relaxed median for w_1 and w_2.*

PROOF. Subcase 2.(i) is applicable, since u' and $b(y)$ are both free. By assumption and by Claims 5.22 and 5.23, we have that $\|[w_1, w_2]_A\| = \|[w_1, u]_A\| + \delta + \|[v, w_2]_A\| \leq 3\tilde{R} + \delta + 2\tilde{L} \leq 6\delta$. Moreover, by the properties of the relaxed median w', for some positive $c \leq \frac{1}{2}$, we have $\|[w_1, w')_A\| \geq \|[u', w')_A\| \geq c\delta$ and $\|[w', w_2]_A\| \geq \|[w', b(y)]_A\| \geq c\delta$. □

CLAIM 5.25. *If $\tilde{L} \leq \delta < \tilde{R}$, then $w' = \text{rmed}_A(w_1, b(y))$, as computed according to Subcase 2.(ii), is a relaxed median for w_1 and w_2.*

PROOF. Case 2.(ii) is applicable, since $b(y)$ is free. By assumption and by Claims 5.22 and 5.23, $\|[w_1, w_2]_A\| \leq 3\tilde{R} + 3\delta$. Furthermore, for some positive $c \leq \frac{1}{2}$, $\|[w_1, w')_A\| \geq c\|[w_1, b(y)]_A\| = c\|[w_1, u]_A\| + c\delta \geq c\tilde{R} + c\delta$ and $\|[w', w_2]_A\| \geq \|[w', b(y)]_A\| \geq c\|[w_1, b(y)]_A\| \geq c\tilde{R} + c\delta$. □

Similarly to Claim 5.25, we can prove the following:

CLAIM 5.26. *If $\tilde{R} \leq \delta < \tilde{L}$, then $w' = \mathrm{rmed}_A(u', w_2)$, as computed according to Subcase 2.(iii), is a relaxed median for w_1 and w_2.*

For the remaining subcase $\delta < \min\{\tilde{R}, \tilde{L}\}$, we assume that $\delta < \tilde{R} \leq \tilde{L}$. The case $\delta < \tilde{L} < \tilde{R}$ can be handled symmetrically. First, we choose $w' \in \mathcal{I}(y)$ as follows:

- If $w_2 <_A t(y)$, then $w' = y\xi'00$, where ξ' satisfies $\|[\xi', \xi_2]_{\mathrm{lex}}\| = \left\lceil \frac{\tilde{R}+\tilde{L}}{6} \right\rceil$.
- If $w_2 \geq_A t(y)$ and $\left\lceil \frac{\tilde{R}+\tilde{L}}{6} \right\rceil < \|[0^{n(y)}, \xi_2]_{\mathrm{lex}}\|$, then $w' = y\xi'01$, where ξ' is such that $\|[\xi', \xi_2]_{\mathrm{lex}}\| = \left\lceil \frac{\tilde{R}+\tilde{L}}{6} \right\rceil$.
- If $w_2 \geq_A t(y)$ and $\left\lceil \frac{\tilde{R}+\tilde{L}}{6} \right\rceil \geq \|[0^{n(y)}, \xi_2]_{\mathrm{lex}}\|$, then $w' = y\xi'00$, where ξ' is such that $\|[\xi', 1^{n(y)}]_{\mathrm{lex}}\| = \left\lceil \frac{\tilde{R}+\tilde{L}}{6} \right\rceil - \|[0^{n(y)}, \xi_2]_{\mathrm{lex}}\|$.

In the two first cases, it is easy to verify that we can always choose a suitable ξ'. For the last case, it suffices to have $\left\lceil \frac{\tilde{R}+\tilde{L}}{6} \right\rceil - \|[0^{n(y)}, \xi_2]_{\mathrm{lex}}\| \leq 2^{n(y)}$, therefore, since $\tilde{R} \leq \tilde{L}$, it suffices to have $\left\lceil \frac{\tilde{L}}{3} \right\rceil \leq \|[0^{n(y)}, \xi_2]_{\mathrm{lex}}\| + 2^{n(y)}$. By the definition of $\tilde{L}$ for $w_2 \geq_A t(y)$, it can be seen that this holds whenever $n(y) \geq 1$. On the other hand, if $n(y) = 0$, then we have $\|[v, w_2]_A\| \leq \|\mathcal{I}(y)\| = 4$, $\delta < \tilde{L} \leq \|[v, w_2]_A\| \leq 4$, and $\|[w_1, u]_A\| \leq 3\tilde{R} \leq 3\tilde{L} \leq 12$, which imply $\|[w_1, w_2]_A\| \leq 20$. Therefore, in order to cover the case $n(y) = 0$, it suffices to demand that $s \geq 20$ (recall that s is the constant threshold for $\|[w_1, w_2]_A\|$, up to which we compute the relaxed median by repeatedly applying the successor function for A).

CLAIM 5.27. $\left\lceil \frac{\tilde{R}+\tilde{L}}{6} \right\rceil \leq \|[w', w_2]_A\| \leq 3 \left\lceil \frac{\tilde{R}+\tilde{L}}{6} \right\rceil$.

PROOF. It suffices to observe that, for every ξ in $[\xi', \xi_2]_{\mathrm{lex}}$ or in $[\xi', 1^{n(y)}]_{\mathrm{lex}} \cup [0^{n(y)}, \xi_2]_{\mathrm{lex}}$ (depending on the case according to which w' was computed), there exist at least one and at most three strings starting with $y\xi$ in $[w', w_2]_A$. □

CLAIM 5.28. *If $\delta < \tilde{R} \leq \tilde{L}$, then w', as defined above, is a relaxed median for w_1 and w_2.*

PROOF. By assumption and by Claims 5.22 and 5.23, we have $\|[w_1, w_2]_A\| \leq 3\tilde{R} + 3\tilde{L}$. Moreover, by Claim 5.27, $\|[w', w_2]_A\| \geq \frac{\tilde{R}+\tilde{L}}{6}$. Finally, $\|[w_1, w')_A\| \geq \|[w_1, u]_A\| + \|[v, w')_A\| = \|[w_1, u]_A\| + \|[v, w_2]_A\| - \|[w', w_2]_A\|$, which, by Claims 5.22, 5.23, and 5.27, yields $\|[w_1, w')_A\| \geq \tilde{R} + \tilde{L} - 3\left\lceil \frac{\tilde{R}+\tilde{L}}{6} \right\rceil$. It can be easily verified that the latter inequality yields $\|[w_1, w')_A\| \geq \frac{\tilde{R}+\tilde{L}}{7}$ whenever $\tilde{R}+\tilde{L} \geq 4$, which concludes the proof for this case.

On the other hand, in order to handle the case $\tilde{R} + \tilde{L} \leq 3$, we observe that this gives $\|[w_1, w_2]_A\| \leq 3\tilde{R} + 3\tilde{L} \leq 9$, therefore it suffices to demand $s \geq 9$. □

This concludes Subcase 2.(iv) and the proof of Theorem 5.13. □

COROLLARY 5.29. *If $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}} = \mathsf{FP}$, then $\#\mathsf{P} = \mathsf{FP}$.*

PROOF. In view of Theorem 5.13, $\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}} = \mathsf{FP}$ implies that $\#\mathrm{SAT}_{+2^n} \in \mathsf{FP}$, therefore $\#\mathsf{P} \subseteq \mathsf{FP}^{\#\mathrm{SAT}_{+2^n}} \subseteq \mathsf{FP}$. □

6. Concluding remarks

In Figure 6.1 we present some known inclusions ($\mathsf{IF}_{\mathsf{t}}^{\prec} \subseteq \mathsf{IF}_{\mathsf{p}}^{\prec} \subseteq \#\mathsf{P}$), as well as the ones proven here.

The following question was posed in Hemaspaandra *et al.* (2007): "What can one say about the downward closure, under various reductions, of #MONSAT, of $\mathsf{IF}_{\mathsf{t}}^{\prec}$, and of $\mathsf{IF}_{\mathsf{p}}^{\prec}$?" Regarding Karp reductions, we get the following partial answer by combining the fact that #MONSAT is in $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$ (Remark 4.12) with Proposition 2.12:

$$\mathrm{R}_m^p(\#\mathrm{MONSAT}) \subseteq \mathrm{R}_m^p(\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}) \subsetneq \mathrm{R}_m^p(\mathsf{IF}_{\mathsf{t}}^{\prec}) \subsetneq \mathrm{R}_m^p(\mathsf{IF}_{\mathsf{p}}^{\prec}) \ .$$

Strict inclusions hold under the assumptions $\mathsf{P} \neq \mathsf{UP} \cap \mathsf{coUP}$ and $\mathsf{UP} \neq \mathsf{PP}$, respectively. Actually the above inclusions also hold for any reductions under which $\mathsf{IF}_{\mathsf{t}}^{\mathsf{LN}}$, $\mathsf{IF}_{\mathsf{t}}^{\prec}$, and $\mathsf{IF}_{\mathsf{p}}^{\prec}$ are downward closed. Of course, it is still open how $\mathrm{R}_m^p(\mathsf{IF}_{\mathsf{t}}^{\mathsf{rmed}})$ is related to

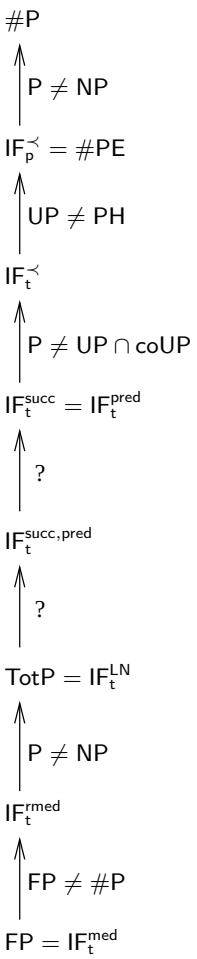

Figure 6.1: Inclusions among interval size function classes. Next to each arrow appears the assumption under which the corresponding inclusion is proper. It is open whether $\mathsf{TotP} = \mathsf{IF}_\mathsf{t}^\mathsf{succ}$, $\mathsf{TotP} = \mathsf{IF}_\mathsf{t}^\mathsf{succ,pred}$, or $\mathsf{IF}_\mathsf{t}^\mathsf{succ,pred} = \mathsf{IF}_\mathsf{t}^\mathsf{succ}$ imply unlikely class collapses.

$\mathrm{R}^p_m(\#\textsc{MonSat})$ and whether the first of the above inclusions is proper (under assumptions) or not.

On the other hand, since $\#\textsc{MonSat}$ is $\#\mathsf{P}$-complete under Cook reductions and classes $\mathsf{IF}^{\mathsf{rmed}}_{\mathsf{t}}$, $\mathsf{IF}^{\mathsf{LN}}_{\mathsf{t}}$, $\mathsf{IF}^{\prec}_{\mathsf{t}}$, $\mathsf{IF}^{\prec}_{\mathsf{p}}$, $\#\mathsf{P}$ are Cook-interreducible (by inclusions in Figure 6.1 and Corollary 5.29), we get that:

$$\mathrm{R}^p_T(\mathsf{IF}^{\mathsf{rmed}}_{\mathsf{t}}) = \mathrm{R}^p_T(\#\textsc{MonSat}) = \mathrm{R}^p_T(\mathsf{IF}^{\mathsf{LN}}_{\mathsf{t}}) = \mathrm{R}^p_T(\mathsf{IF}^{\prec}_{\mathsf{t}}) = \mathrm{R}^p_T(\mathsf{IF}^{\prec}_{\mathsf{p}}) \ .$$

Further important questions remain open such as determining, for classes studied in the present work, complete problems with respect to Karp reductions or other reductions under which these classes are closed downwards.

Another noteworthy aspect of our work concerns the apparent emergence of a hierarchy of interval size function classes between FP and $\#\mathsf{P}$. Indeed, by strengthening the feasibility constraint imposed on the underlying order as well as the nature of the order itself, ranging from a partial order having efficient adjacency checks ($\mathsf{IF}^{\prec}_{\mathsf{p}}$) to a total order having an efficiently computable median function ($\mathsf{IF}^{\mathsf{med}}_{\mathsf{t}} = \mathsf{FP}$), we obtain a hierarchy as depicted in Figure 6.1. It would be a challenging task to quantify the relative strength of the various feasibility constraints (possibly in terms of the amount of information that one can obtain about the order in polynomial time) and then to show that an increase in the amount of available information (which would correspond to a strengthening of the feasibility constraint) results in a more restricted class.

Acknowledgements

We are indebted to the anonymous referees for their careful review of the manuscript. Their pertinent remarks and suggestions induced a considerable improvement on the presentation of some of the technical parts of the paper. A preliminary version of this work appears in Bampas *et al.* (2009).

References

Carme Àlvarez & Birgit Jenner (1993). A very hard log-space counting class. *Theoretical Computer Science* **107**(1), 3–30.

Evangelos Bampas, Andreas-Nikolas Göbel, Aris Pagourtzis & Aris Tentes (2009). On the connection between interval size functions and path counting. In *Proceedings of the 6th Annual Conference on Theory and Applications of Models of Computation*, volume 5532 of *Lecture Notes in Computer Science*, 108–117.

Martin E. Dyer, Leslie Ann Goldberg, Catherine S. Greenhill & Mark Jerrum (2003). The relative complexity of approximate counting problems. *Algorithmica* **38**(3), 471–500.

Lane A. Hemaspaandra, Christopher M. Homan, Sven Kosub & Klaus W. Wagner (2007). The complexity of computing the size of an interval. *SIAM Journal on Computing* **36**(5), 1264–1300.

Lane A. Hemaspaandra & Mitsunori Ogihara (2002). *The Complexity Theory Companion*. Springer-Verlag Berlin Heidelberg.

Harald Hempel & Gerd Wechsung (2000). The operators min and max on the polynomial hierarchy. *International Journal of Foundations of Computer Science* **11**(2), 315–342.

Mark Jerrum, Alistair Sinclair & Eric Vigoda (2004). A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM* **51**(4), 671–697.

Richard M. Karp, Michael Luby & Neal Madras (1989). Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms* **10**(3), 429–448.

Aggelos Kiayias, Aris Pagourtzis, Kiron Sharma & Stathis Zachos (2001). Acceptor-definable counting classes. In *Proceedings of the 8th Panhellenic Conference on Informatics, Revised Selected Papers*, volume 2563 of *Lecture Notes in Computer Science*, 453–463.

Jingcheng Liu, Pinyan Lu & Chihao Zhang (2014). FPTAS for counting weighted edge covers. In *Proceedings of the 22nd Annual European Symposium on Algorithms*, volume 8737 of *Lecture Notes in Computer Science*, 654–665.

Aris Pagourtzis (2001). On the complexity of hard counting problems with easy decision version. In *Proceedings of the 3rd Panhellenic Logic Symposium*.

Aris Pagourtzis & Stathis Zachos (2006). The complexity of counting functions with easy decision version. In *Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science*, volume 4162 of *Lecture Notes in Computer Science*, 741–752.

Sanjeev Saluja, K. V. Subrahmanyam & Madhukar N. Thakur (1995). Descriptive complexity of #P functions. *Journal of Computer and System Sciences* **50**(3), 493–505.

Seinosuke Toda (1991). PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing* **20**(5), 865–877.

Seinosuke Toda & Osamu Watanabe (1992). Polynomial time 1-Turing reductions from #PH to #P. *Theoretical Computer Science* **100**(1), 205–221.

Leslie G. Valiant (1979a). The complexity of computing the permanent. *Theoretical Computer Science* **8**, 189–201.

Leslie G. Valiant (1979b). The complexity of enumeration and reliability problems. *SIAM Journal on Computing* **8**(3), 410–421.

Dror Weitz (2006). Counting independent sets up to the tree threshold. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, 140–149.

Manuscript received 22 January 2015

Evangelos Bampas
LIF, Aix-Marseille University and CNRS, France
evangelos.bampas@lif.univ-mrs.fr

Andreas-Nikolas Göbel
University of Oxford, UK
agob@cs.ox.ac.uk

Aris Pagourtzis
School of ECE, National Technical University of Athens, Greece
pagour@cs.ntua.gr

Aris Tentes
New York University, USA
tentes@cs.nyu.edu