A Parameterized Halting Problem

Yijia Chen[1] and Jörg Flum[2]

[1] Shanghai Jiaotong University, China, yijia.chen@cs.sjtu.edu.cn
[2] Albert-Ludwigs-Universität Freiburg, Germany,
joerg.flum@math.uni-freiburg.de

Abstract. The parameterized problem p-Halt takes as input a nondeterministic Turing machine $\mathbb{M}$ and a natural number n, the size of $\mathbb{M}$ being the parameter. It asks whether every accepting run of $\mathbb{M}$ on empty input tape takes more than n steps. This problem is in the class $\mathrm{XP}_{\mathrm{uni}}$, the class "uniform XP," if there is an algorithm deciding it, which for fixed machine $\mathbb{M}$ runs in time polynomial in n. It turns out that various open problems of different areas of theoretical computer science are related or even equivalent to p-Halt $\in \mathrm{XP}_{\mathrm{uni}}$. Thus this statement forms a bridge which allows to derive equivalences between statements of different areas (proof theory, complexity theory, descriptive complexity, ...) which at first glance seem to be unrelated. As our presentation shows, various of these equivalences may be obtained by the same method.

1. Introduction

Halting problems played a central role in *computability theory* right up from the beginning. In fact, in his seminal paper [33] Turing made precise the notion of algorithm by introducing the type of machine known as Turing machine and proved the first undecidability result: the undecidability of the halting problem for Turing machines:

Instance: A Turing machine $\mathbb{M}$.
Problem: Does $\mathbb{M}$ accept the empty input tape?

In *complexity theory* Cook [13] and Levin [28] showed that the halting problem

NTM-Halt
Instance: A nondeterministic Turing machine $\mathbb{M}$ and a string 1^n with $n \in \mathbb{N}$.
Problem: Does $\mathbb{M}$ accept the empty input tape in $\leq n$ steps?

is NP-complete (under polynomial time reductions). Comparing this problem with other NP-complete problems, Downey and Fellows remark in [16, page 236]:

> *... the point is that a nondeterministic Turing machine is such an opaque and generic object that it simply does not seem reasonable that we should be able to decide in polynomial time whether a given Turing machine has some accepting path. The Cook-Levin theorem therefore gives powerful evidence that* P $\neq$ NP.

Later on Chandra, Kozen, and Stockmeyer [7] introduced the notion of alternating Turing machine and showed that all levels of the Polynomial Hierarchy have natural complete problems that are halting problems for alternating Turing machines.

Also in *parameterized complexity* most complexity classes of parameterized intractable problems considered so far contain a more or less natural complete halting problem (complete under fpt-reductions). Depending on the class the machines are deterministic, nondeterministic, or alternating and they are single or multitape machines. The reader will find the completeness results, for example, in the textbook [20] on parameterized complexity. In this introduction we mention some of these results, however, otherwise not used in this paper.

The classes W[1], W[2], and W[P] are (among) the most important classes of parameterized intractable problems. In particular, for each of them there is a halting problem for nondeterministic Turing machines complete for it. In fact, the *short halting problem for nondeterministic single-tape Turing machines* p-Short-NSTM-Halt, the *short halting problem for nondeterministic (multitape) Turing machines* p-Short-NTM-Halt, and the *bounded halting problem for nondeterministic Turing machines* p-Bounded-NTM-Halt are complete for W[1], W[2], and W[P], respectively, where:

p-Short-NSTM-Halt
Instance: A nondeterministic single tape Turing machine $\mathbb{M}$ and $k \in \mathbb{N}$.
Parameter: k.
Problem: Does $\mathbb{M}$ accept the empty input tape in $\leq k$ steps?

p-Short-NTM-Halt
Instance: A nondeterministic multitape Turing machine $\mathbb{M}$ and $k \in \mathbb{N}$.
Parameter: k.
Problem: Does $\mathbb{M}$ accept the empty input tape in $\leq k$ steps?

p-Bounded-NTM-Halt
Instance: A nondeterministic Turing machine $\mathbb{M}$, $k \in \mathbb{N}$, and 1^n with $n \in \mathbb{N}$.
Parameter: k.
Problem: Does $\mathbb{M}$ accepts the empty input tape in $\leq n$ steps and using at most k nondeterministic steps?

These completeness results for W[1], W[2], and W[P] are due to Cai et al. [4], Cesati et al. [6], and Cesati [5], respectively. Concerning the W[1]-completeness of p-Short-NSTM-Halt, Downey and Fellows write in [16, page 236]:

> *It seems to us that if one accepts the philosophical argument that* **TURING MACHINE ACCEPTANCE** *[that is,* NTM-Halt*] is intractable, then the same reasoning would suggest that* **SHORT TURING MACHINE ACCEPTANCE** *[that is, p-Short-NSTM-Halt] is fixed parameter intractable.*

In the problem p-Bounded-NTM-Halt the parameter bounds the number of nondeterministic steps that are allowed in a run of length n, so small parameter means limited nondeterminism. However in the first two parameterized halting problems the parameter is the total number of steps considered in the given instance. Having in mind the original investigations which led to the introduction of the halting problem for (deterministic) Turing machines one hardly would argue that the parameter is small compared with the total size of an instance. In this context it is more natural to expect that the size of the Turing machine is small compared with the number of steps considered in a run of the machine on empty input tape.

If in p-Short-NSTM-Halt we replace the nondeterministic machines by alternating ones with the appropriate number of alternations we obtain complete parameterized halting problems for the different levels of the so-called A-hierarchy [19]. Finally, in this short report on known results concerning halting problems in parameterized complexity, let us mention at least one parameterized class with a halting problem for *deterministic* machines as complete problem. The problem

p-Exp-DTM-Halt
Instance: A Turing machine $\mathbb{M}$, $k \in \mathbb{N}$ and 1^n with $n \in \mathbb{N}$.
Parameter: k.
Problem: Does $\mathbb{M}$ accept the empty input tape in at most n^k steps?

is complete for the class XP.

We already mentioned that in halting problems the size of the machine is a reasonable parameter (reasonable in the sense of parameterized complexity). In this paper we consider the parameterized halting problem:

p-Halt
Instance: A nondeterministic Turing machine $\mathbb{M}$ and 1^n with $n \in \mathbb{N}$.
Parameter: $\|\mathbb{M}\|$, the size of $\mathbb{M}$.
Problem: Does every accepting run of $\mathbb{M}$ on empty input tape take more than n steps?

We introduced this parameterization of the halting problem in [9]; independently, it was introduced by Aumann and Domb [1]. Later on the complexity of this problem has also been studied by Monroe [30]. Note that the classical problem HALT underlying p-HALT essentially is the complement of the problem NTM-HALT considered above. In fact, $\langle \mathbb{M}, 1^n \rangle \in p\text{-HALT}$ means that there is no accepting run of $\mathbb{M}$ on empty input tape at all or if there is one, then all such runs take more than n steps.

We have seen above that (apparently) it makes a difference whether we consider single tape or multitape machines. Moreover, if we restrict the inputs of p-SHORT-NSTM-HALT to nondeterministic Turing machines with the cardinality of its alphabet bounded by some constant, then the problem becomes fixed-parameter tractable. As in p-HALT the size of the machine is the parameter, its complexity is robust against such changes (fixed alphabet versus arbitrary alphabet, single tape versus multitape, binary branching versus finite branching, ...); in fact, one easily verifies that any two such variants are fpt-equivalent.

It is easily seen that HALT (the classical problem underlying p-HALT) is coNP-complete. The algorithm that for every instance $\langle \mathbb{M}, 1^n \rangle$ of p-HALT systematically checks all possible runs of length $\leq n$ of $\mathbb{M}$ on empty input tape takes $|\mathbb{M}|^n$ steps approximately. The "small" parameter $|\mathbb{M}|$ is the base of the term $|\mathbb{M}|^n$ and the "big" n its exponent. The question arises whether we can reverse the roles of $|\mathbb{M}|$ and n, more precisely, whether there is an algorithm solving $\langle \mathbb{M}, 1^n \rangle \in p\text{-HALT}$ in time $n^{f(|\mathbb{M}|)}$ for some function $f : \mathbb{N} \to \mathbb{N}$, that is, whether

$$p\text{-HALT} \in \mathrm{XP}_{\mathrm{uni}}.$$

This problem is widely open. We conjecture that $p\text{-HALT} \notin \mathrm{XP}_{\mathrm{uni}}$; in fact, encouraged by the remarks of Downey and Fellows mentioned above, we are tempted to add:

> *The point is that a nondeterministic Turing machine is such an opaque and generic object that it simply does not seem reasonable that we should be able to decide whether every accepting run on empty input tape of a given nondeterministic Turing machine $\mathbb{M}$ takes more than n steps in time $n^{f(|\mathbb{M}|)}$ for some function f.*

In this paper first we report on what is known about the complexity of p-HALT (Section 3). Then we will see that the statement $p\text{-HALT} \in \mathrm{XP}_{\mathrm{uni}}$ is equivalent to other prominent open problems from different areas of theoretical computer science. More precisely, in Section 4 we show that it is equivalent to the existence of polynomially optimal proof systems. In Section 5 we show that $p\text{-HALT} \in \mathrm{XP}_{\mathrm{uni}}$ implies the existence of complete problems, first for the classical complexity class UP and then for the class of polynomial time equivalence relations under so-called equivalence reductions. We get a logic capturing the complexity class P (= PTIME) under the assumption $p\text{-HALT} \in \mathrm{XP}_{\mathrm{uni}}$ in Section 6. Even though we originally obtained these results using distinct arguments, in the meantime we realized that they can be obtained all by the same method: first we translate

the consequences of p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ mentioned so far into statements on listings (that is, on effective enumerations) and then, based on p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$, we apply a technique we call the *invariantization of listings*; this technique plays a central role in the present exposition. One could formulate the principle underlying this technique in an abstract way, and obtain our results as instances of a general theorem. however, we do not do so. Nevertheless, in Section 7 we introduce the notion of slicewise downward monotone parameterized problem and take a closer look on its role in the preceding results. In Section 8 we relate the assumption p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ to the complexity of deciding whether a hard valid first-order sentence has a proof of a given length. Finally, in Section 9 we show that p-HALT $\notin \mathrm{XP}_{\mathrm{uni}}$ is equivalent to the existence of hard sequences for algorithms deciding TAUT.

2. Some preliminaries

In this section we fix some notations and recall some basic definitions.

We let $\mathbb{N}[X]$ be the set of polynomials with natural numbers as coefficients. We denote the alphabet $\{0,1\}$ by Σ and the length of a string $x \in \Sigma^*$ by $|x|$. Let 1^n be the string consisting of n many 1s and let λ denote the empty string. We identify problems with subsets Q of Σ^*. We already remarked that the restriction to the alphabet Σ does not affect the complexity of the problem p-HALT. Sometimes statements containing a formulation like "there is a $d \in \mathbb{N}$ such that for all $x \in \Sigma^*$: $\ldots \leq |x|^d$" can be wrong for $x \in \Sigma^*$ with $|x| \leq 1$. We trust the reader's common sense to interpret such statements reasonably.

A problem $Q \subseteq \Sigma^*$ *has padding* if there is a function $pad : \Sigma^* \times \Sigma^* \to \Sigma^*$ computable in logarithmic space having the following properties:

(i) For any $x, y \in \Sigma^*$, $|pad(x,y)| > |x| + |y|$ and $\big(pad(x,y) \in Q \iff x \in Q\big)$.
(ii) There is a logspace algorithm which, given $pad(x,y)$ recovers y.

By $\langle \ldots \rangle$ we denote some standard logspace and linear time computable tupling function with logspace and linear time computable inverses.

If $\mathbb{A}$ is a (deterministic or nondeterministic) algorithm and $\mathbb{A}$ accepts $x \in \Sigma^*$, then we denote by $t_{\mathbb{A}}(x)$ the number of steps of a shortest accepting run of $\mathbb{A}$ on x; if $\mathbb{A}$ does not accept x, then $t_{\mathbb{A}}(x) := \infty$. By convention, $\infty > n$ for $n \in \mathbb{N}$. So we can state p-HALT in a more succinct way:

p-HALT	
Instance:	A nondeterministic Turing machine $\mathbb{M}$ and 1^n with $n \in \mathbb{N}$.
Parameter:	$\lvert\mathbb{M}\rvert$.
Problem:	Is $t_{\mathbb{M}}(\lambda) > n$?

By default, algorithms are deterministic. If an algorithm $\mathbb{A}$ on input x eventually halts and outputs a value, we denote it by $\mathbb{A}(x)$. We use deterministic and nondeterministic Turing machines with Σ as alphabet as our basic computational

model for algorithms (and we often use the notions "algorithm" and "Turing machine" synonymously). If necessary, we will not distinguish between a Turing machine and its code, a string in Σ^*. If $\mathbb{M}$ is a deterministic or nondeterministic Turing machine, then $L(\mathbb{M})$ is the language accepted by $\mathbb{M}$. We use Turing machines as acceptors and transducers. Even though we use formulations like "let $\mathbb{M}_1, \mathbb{M}_2, \ldots$ be an enumeration of *all* polynomial time Turing machines," from the context it will be clear that we only refer to acceptors (or that we only refer to transducers). We assume that a run of a nondeterministic Turing machine is determined by the sequence of its states.

A polynomial time deterministic or nondeterministic Turing machine $\mathbb{M}$ is *clocked* if (the code of) $\mathbb{M}$ contains a natural number time($\mathbb{M}$) such that $n^{\text{time}(\mathbb{M})}$ is a bound for the running time of $\mathbb{M}$ on inputs of length n. Of course, the function $\mathbb{M} \mapsto \text{time}(\mathbb{M})$ should be computable in logspace.

2.1. Parameterized complexity. Formally, we view *parameterized problems* as pairs (Q, κ) consisting of a classical problem $Q \subseteq \Sigma^*$ and a *parameterization* $\kappa : \Sigma^* \to \mathbb{N}$, which is required to be polynomial time computable. However, we will present parameterized problems in the form we did it for p-Halt and further parameterized problems in the Introduction.

We mainly consider the classes FPT_{uni} and XP_{uni} of *uniform parameterized complexity* and sometimes the classes FPT and XP of *strongly uniform parameterized complexity*. A parameterized problem (Q, κ) is in the class FPT_{uni} (or is *uniformly fixed-parameter tractable*) if $x \in Q$ is solvable by an algorithm running in time $\leq f(\kappa(x)) \cdot |x|^{O(1)}$ for some $f : \mathbb{N} \to \mathbb{N}$. The problem (Q, κ) is in the class XP_{uni} if $x \in Q$ is solvable by an algorithm running in time $\leq |x|^{f(\kappa(x))}$ for some $f : \mathbb{N} \to \mathbb{N}$.

If in the definition of FPT_{uni} and XP_{uni} we require the function f to be computable, then we get the corresponding classes FPT and XP.

The following inclusions hold between the four complexity classes of parameterized problems just introduced:

$$\begin{array}{ccccc} & & \text{FPT}_{\text{uni}} & & \\ & \subset\!\!\!\cdot & & \cdot\!\!\!\supset & \\ \text{FPT} & & & & \text{XP}_{\text{uni}} \\ & \cdot\!\!\!\supset & & \subset\!\!\!\cdot & \\ & & \text{XP} & & \end{array} \tag{1}$$

While the corresponding $\subseteq$-inclusions are trivial, the strict inclusions $\text{FPT} \subset \text{FPT}_{\text{uni}}$ and $\text{XP} \subset \text{XP}_{\text{uni}}$ are due to Downey and Fellows [15] and to Downey [17]. The strict inclusions $\text{FPT} \subset \text{XP}$ and $\text{FPT}_{\text{uni}} \subset \text{XP}_{\text{uni}}$ are easily obtained by showing that p-Exp-DTM-Halt $\in \text{XP} \setminus \text{FPT}_{\text{uni}}$ (cf. [20, Corollary 2.26]).

3. The complexity of p-Halt

In this section we report what we know on the complexity of p-Halt. We start with a simple observation. The problem Halt is in coNE even if the natural number n is given in binary. From that one easily obtains:

Theorem 1 ([9]). *If* $\mathrm{E} = \mathrm{NE}$ *(and hence if* $\mathrm{P} = \mathrm{NP}$*), then* p-HALT $\in \mathrm{FPT}$.

(Assuming $\mathrm{E} \neq \mathrm{NE}$), by (1) the most ambitious task is to show that p-HALT $\notin \mathrm{XP}_{\mathrm{uni}}$ and p-HALT $\notin \mathrm{FPT}$ should be the easiest one. We know:

Theorem 2 ([8]). *If* $\mathrm{NP}[\mathrm{TC}] \neq \mathrm{P}[\mathrm{TC}]$*, then* p-HALT $\notin \mathrm{FPT}$.

Here $\mathrm{NP}[\mathrm{TC}] \neq \mathrm{P}[\mathrm{TC}]$ means that $\mathrm{DTIME}(h^{O(1)}) \neq \mathrm{NTIME}(h^{O(1)})$ for all time constructible and increasing functions $h : \mathbb{N} \to \mathbb{N}$. If $\mathrm{NP}[\mathrm{TC}] \neq \mathrm{P}[\mathrm{TC}]$, then $\mathrm{P} \neq \mathrm{NP}$, even $\mathrm{E} \neq \mathrm{NE}$, as seen by taking as h the identity function and the function 2^n, respectively. On the other hand, it is not hard to see (cf. [8]) that the assumption "NP contains a P-bi-immune problem" and hence the so-called Measure Hypothesis imply $\mathrm{NP}[\mathrm{TC}] \neq \mathrm{P}[\mathrm{TC}]$.

The following idea underlies a proof of Theorem 2. Assume that p-HALT $\in$ FPT. Then, we have a *deterministic* algorithm deciding p-HALT, the parameterized halting problem for *nondeterministic* Turing machines. This yields a way (different from brute force) to translate nondeterministic algorithms into deterministic ones; a careful analysis of this translation shows that $\mathrm{NTIME}(h^{O(1)}) \subseteq \mathrm{DTIME}(h^{O(1)})$ for a suitable time constructible and increasing function h. For a detailed proof we refer the reader to [8].

One can refine the previous argument to get p-HALT $\notin \mathrm{XP}$; however one needs a complexity-theoretic assumption (apparently) stronger than $\mathrm{NP}[\mathrm{TC}] \neq \mathrm{P}[\mathrm{TC}]$, namely the assumption $\mathrm{NP}[\mathrm{TC}] \not\subseteq \mathrm{P}[\mathrm{TC}^{\log \mathrm{TC}}]$; it claims that $\mathrm{NTIME}(h^{O(1)}) \not\subseteq \mathrm{DTIME}(h^{O(\log h)})$ for every time constructible and increasing function $h : \mathbb{N} \to \mathbb{N}$. That is:

Theorem 3. *If* $\mathrm{NP}[\mathrm{TC}] \not\subseteq \mathrm{P}[\mathrm{TC}^{\log \mathrm{TC}}]$*, then* p-HALT $\notin \mathrm{XP}$.

The assumption "NP contains an E-bi-immune problem" implies the statement $\mathrm{NP}[\mathrm{TC}] \not\subseteq \mathrm{P}[\mathrm{TC}^{\log \mathrm{TC}}]$.

As mentioned, we do not know whether p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ or whether even p-HALT $\in \mathrm{FPT}_{\mathrm{uni}}$. However, from the point of view of nonuniform parameterized complexity the problem p-HALT is fixed-parameter tractable. Recall that a parameterized problem (Q, κ) is in the class $\mathrm{FPT}_{\mathrm{nu}}$ (or is *nonuniformly fixed-parameter tractable*) if there is a constant $c \in \mathbb{N}$, an arbitrary function $f : \mathbb{N} \to \mathbb{N}$, and for every $k \in \mathbb{N}$ an algorithm solving the (classical) problem

$$(Q, \kappa)_k := \big\{ x \in Q \;\big|\; \kappa(x) = k \big\}$$

in time $f(k) \cdot |x|^c$. The problem $(Q, \kappa)_k$ is called the kth *slice* of (Q, κ).

Proposition 4. *The problem* p-HALT *is in the class* $\mathrm{FPT}_{\mathrm{nu}}$.

Proof. Fix $k \in \mathbb{N}$; then there are only finitely many nondeterministic Turing machines $\mathbb{M}$ with $|\mathbb{M}| = k$, say, $\mathbb{M}_1, \ldots, \mathbb{M}_s$. Hence the algorithm $\mathbb{A}_k$ that on any instance $\langle \mathbb{M}, 1^n \rangle$ of p-HALT with $|\mathbb{M}| = k$ determines the i with $\mathbb{M} = \mathbb{M}_i$, and then accepts if and only if $t_{\mathbb{M}_i}(\lambda) > n$, decides the kth slice of p-HALT. It has running time $O(|\mathbb{M}| + n)$; thus it witnesses that p-HALT $\in \mathrm{FPT}_{\mathrm{nu}}$. □

The following lemma shows that p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ if there is an algorithm that accepts p-HALT and that runs in the time required by $\mathrm{XP}_{\mathrm{uni}}$ for instances $\langle \mathbb{M}, 1^n \rangle$, where $\mathbb{M}$ is a nondeterministic Turing machine which does not halt on the empty input tape.

Lemma 5. *If there is an algorithm $\mathbb{A}$ accepting p-HALT such that for all instances $\langle \mathbb{M}, 1^n \rangle$ with $t_{\mathbb{M}}(\lambda) = \infty$ we have $t_{\mathbb{A}}(\langle \mathbb{M}, 1^n \rangle) = n^{f(|M|)}$ for some function f, then p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$.*

Proof. Let $\mathbb{A}$ be as in the statement of the lemma. Let $\mathbb{B}$ be an algorithm that on input $\mathbb{M}$ computes $t_{\mathbb{M}}(\lambda)$ by systematically checking for $r = 0, 1, \ldots$ whether there is a run of length r accepting λ. Note that $\mathbb{B}$ does not stop on inputs $\mathbb{M}$ with $t_{\mathbb{M}}(\lambda) = \infty$.

Now we consider the algorithm $\mathbb{A}^*$ that on input $\langle \mathbb{M}, 1^n \rangle$ in parallel simulates $\mathbb{A}$ on input $\langle \mathbb{M}, 1^n \rangle$ and $\mathbb{B}$ on input $\mathbb{M}$. If $\mathbb{A}$ accepts, then $\mathbb{A}^*$ accepts. If $\mathbb{B}$ outputs $t_{\mathbb{M}}(\lambda)$, then $\mathbb{A}^*$ checks whether $t_{\mathbb{M}}(\lambda) > n$ and answers accordingly.

Clearly, $\mathbb{A}^*$ decides p-HALT. We still have to show that it runs in time polynomial in n for fixed nondeterministic Turing machine $\mathbb{M}$. By our assumption on $\mathbb{A}$, this is clear if $t_{\mathbb{M}}(\lambda) = \infty$; if $t_{\mathbb{M}}(\lambda) < \infty$, then eventually $\mathbb{B}$ will halt on input $\mathbb{M}$ and output $t_{\mathbb{M}}(\lambda)$. As the check $t_{\mathbb{M}}(\lambda) > n$ can be done in linear time, in this case the running time of $\mathbb{A}^*$ can be bounded by $O(n)$ (where the constant hidden in the Oh-notation depends on $\mathbb{M}$). □

Using the previous argument we show that the answer to the question "p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$?" would be the same if we only would require for an instance $\langle \mathbb{M}, 1^n \rangle$ of p-HALT that we get the correct answer if $t_{\mathbb{M}}(\lambda)$ is not near to n. Let us give a precise version of what we mean. Let $\rho : \mathbb{N} \to \mathbb{N}$ be a nondecreasing and polynomial time computable function when inputs and outputs are given in unary notation. We say that the approximation problem p-APP-HALT is in $\mathrm{XP}_{\mathrm{uni}}$ if there is a function $f : \mathbb{N} \to \mathbb{N}$ and an algorithm $\mathbb{A}$ that on every tuple $\langle \mathbb{M}, n \rangle$, where $\mathbb{M}$ is a nondeterministic Turing machine and $n \in \mathbb{N}$, runs in time $n^{f(|\mathbb{M}|)}$ and has the properties:

(i) if $t_{\mathbb{M}}(\lambda) = \infty$, then $\mathbb{A}$ accepts;
(ii) if $t_{\mathbb{M}}(\lambda) < \infty$ and $n \leq \frac{t_{\mathbb{M}}(\lambda)}{\rho(t_{\mathbb{M}}(\lambda))}$, then $\mathbb{A}$ accepts;
(iii) if $t_{\mathbb{M}}(\lambda) < \infty$ and $t_{\mathbb{M}}(\lambda) \cdot \rho(t_{\mathbb{M}}(\lambda)) \leq n$, then $\mathbb{A}$ rejects.

Thus, if $t_{\mathbb{M}}(\lambda) < \infty$, then the answer of $\mathbb{A}$ can be arbitrary for n with

$$\frac{t_{\mathbb{M}}(\lambda)}{\rho(t_{\mathbb{M}}(\lambda))} < n < t_{\mathbb{M}}(\lambda) \cdot \rho(t_{\mathbb{M}}(\lambda)).$$

Then:

Proposition 6. *p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ if and only if p-APP-HALT $\in \mathrm{XP}_{\mathrm{uni}}$.*

Proof. Clearly, every algorithm witnessing that p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ shows that p-APP-HALT $\in \mathrm{XP}_{\mathrm{uni}}$. Conversely, let $\mathbb{A}$ witness that p-APP-HALT $\in \mathrm{XP}_{\mathrm{uni}}$

and let $\langle \mathbb{M}, 1^n \rangle$ be an instance of p-HALT. We simulate $\mathbb{A}$ on $\langle \mathbb{M}, 1^{n \cdot \rho(n)} \rangle$. If $\mathbb{A}$ accepts, then, by (iii),

$$t_{\mathbb{M}}(\lambda) > n.$$

and hence, $\langle \mathbb{M}, 1^n \rangle \in p$-HALT. Otherwise, we know that $t_{\mathbb{M}}(\lambda) < \infty$, and we compute $t_{\mathbb{M}}(\lambda)$ by brute force and check whether $t_{\mathbb{M}}(\lambda) > n$ or not. □

4. Polynomially optimal propositional proof systems and p-Halt

By TAUT we denote the set of formulas of propositional logic that are tautologies. A *propositional proof system* in the sense of [14] is a polynomial time computable surjective function $p : \Sigma^* \to$ TAUT. If $p(w) = \alpha$, we say that w is a *p-proof* of α.

Let p and p' be propositional proof systems. A *simulation from p' to p* is a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ such that $p(f(w')) = p'(w')$ for all $w' \in \Sigma^*$. A propositional proof system p is *polynomially optimal* if for every propositional proof system p' there is a simulation from p' to p.

The quest for a polynomially optimal propositional proof system is an important open problem of proof theory. In [27] Krajíček and Pudlák conjectured that there is no polynomially optimal propositional proof system. It turns out that this conjecture is equivalent to p-HALT $\notin \mathrm{XP}_{\mathrm{uni}}$:

Theorem 7 ([10]). *There is a polynomially optimal propositional proof system if and only if p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$.*

In the proof of the implication from right to left, we will use the following simple result. By definition a *listing* is an effective enumeration. We denote by $\mathrm{PF}(\Sigma^*)$ and PF(TAUT) the set of all polynomial time computable functions from Σ^* to Σ^* and the set of all polynomial time computable functions from Σ^* to TAUT, respectively.

Proposition 8. *The following are equivalent:*

(a) There is a polynomially optimal propositional proof system.

(b) There is a listing of PF(TAUT) *by means of polynomial time Turing machines. By this we mean that there is a listing $\mathbb{M}_1, \mathbb{M}_2, \ldots$ of polynomial time Turing machines computing functions $h_1, h_2, \ldots$ from Σ^* to* TAUT *such that* $\mathrm{PF}(\text{TAUT}) = \{h_i \mid i \geq 1\}$.

Proof. (b) ⇒ (a): Let $\mathbb{M}_1, \mathbb{M}_2, \ldots$ and $h_1, h_2, \ldots$ be as in (b). By repeating machines $\mathbb{M}_i$ if necessary, we may assume that the function $i \mapsto \mathbb{M}_i$ is polynomial time computable. We fix a tautology α_0 and show that then the function $p : \Sigma^* \to$ TAUT is a polynomially optimal propositional proof system where

$$p(x) := \begin{cases} h_i(w), & \text{if } x = \langle i, w, c \rangle \text{ and } c \text{ is the computation of } \mathbb{M}_i \text{ on input } w \\ \alpha_0, & \text{otherwise.} \end{cases}$$

Then p is a propositional proof system: Our assumption on the function $i \mapsto \mathbb{M}_i$ and the presence of the computation c in the first case of the definition of p

guarantee its polynomial time computability. Moreover, every $\alpha \in \textsc{Taut}$ is in the range of p, as one of the h_is will be the constant function with value α. Furthermore, p is polynomially optimal: If p' is a further propositional proof system, then there is an $i \geq 1$ such that $p' = h_i$. Therefore, $w \mapsto \langle i, w, c\rangle$, where c is the computation of $\mathbb{M}_i$ on input w, is a simulation from p' to p.

(a) $\Rightarrow$ (b): Let p be a polynomially optimal propositional proof system computed by the polynomial time Turing machine $\mathbb{M}$ and let $\mathbb{M}_1, \mathbb{M}_2, \ldots$ be a listing of $\mathrm{PF}(\Sigma^*)$ by means of polynomial time Turing machines. If h_i denotes the function computed by $\mathbb{M}_i$, it is easy to verify that $\mathrm{PF}(\textsc{Taut}) = \{p \circ h_i \mid i \geq 1\}$ $\big($here $p \circ h_i$ is the function $x \mapsto p(h_i(x))\big)$. Then $\mathbb{M} \circ \mathbb{M}_1, \mathbb{M} \circ \mathbb{M}_2, \ldots$ is the required listing, where $\mathbb{M} \circ \mathbb{M}_i$ denotes a natural polynomial time Turing machine computing $p \circ h_i$. □

We first turn to a proof of the implication from right to left of Theorem 7. Thereby we use a technique, *the invariantization of listings*, that we shall use again and again in this paper; therefore, we give a detailed exposition here.

Lemma 9. *If p-*$\textsc{Halt} \in \mathrm{XP}_{\mathrm{uni}}$*, then there is a polynomially optimal propositional proof system.*

Proof. By the previous result it suffices to show that there exists a listing of $\mathrm{PF}(\textsc{Taut})$ by means of polynomial time Turing machines. However, it is undecidable whether a Turing machine computes a function from Σ^* to $\textsc{Taut}$. So we start with a listing of $\mathrm{PF}(\Sigma^*)$ by *clocked* polynomial time Turing machines, say

$$\mathbb{M}_1, \mathbb{M}_2, \ldots \tag{2}$$

We denote by h_i the function computed by $\mathbb{M}_i$. Thus, $\mathrm{PF}(\Sigma^*) = \{h_i \mid i \geq 1\}$. Using the hypothesis p-$\textsc{Halt} \in \mathrm{XP}_{\mathrm{uni}}$ we *invariantize* this listing in order to get a listing of $\mathrm{PF}(\textsc{Taut})$. For this purpose, for $h : \Sigma^* \to \Sigma^*$ and $n \geq 1$ we say that h is *n-tautological* if

$$h(w) \in \textsc{Taut} \text{ for all } w \text{ with } |w| \leq n$$

and define $h^{\mathrm{taut}} : \Sigma^* \to \Sigma^*$ by

$$h^{\mathrm{taut}}(w) := \begin{cases} h(w), & \text{if } h \text{ is } |w|\text{-tautological} \\ \alpha_0, & \text{otherwise.} \end{cases}$$

Then

(i) $h^{\mathrm{taut}} : \Sigma^* \to \textsc{Taut}$;
(ii) $h^{\mathrm{taut}} = h$ if $h : \Sigma^* \to \textsc{Taut}$;
(iii) if h is not n-tautological, then $h^{\mathrm{taut}}(w) = \alpha_0$ for all $|w| \geq n$;
(iv) if h is polynomial time computable, then so is h^{taut}.

Note that (iv) is an immediate consequence of (ii) and (iii). Hence,

$$h_1^{\mathrm{taut}}, h_2^{\mathrm{taut}}, \dots \tag{3}$$

is an enumeration of the elements of PF(TAUT). In fact, by (iv) all h_i^{taut} are polynomial time computable, by (i) their range is contained in TAUT, and by (ii) the enumeration contains all elements of PF(TAUT). But instead of the enumeration (3) of PF(TAUT) we aim at a listing of PF(TAUT) by means of *polynomial time Turing machines*. We show that there is an effective procedure assigning to every clocked polynomial time Turing machine $\mathbb{M}$ computing some $h \in \mathrm{PF}(\Sigma^*)$ a polynomial time Turing machine $\mathbb{M}^{\mathrm{taut}}$ computing h^{taut}. Then $\mathbb{M}_1^{\mathrm{taut}}, \mathbb{M}_2^{\mathrm{taut}}, \dots$ is the desired listing of PF(TAUT). For this purpose we need:

Claim: Assume that p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$, then there is an algorithm $\mathbb{B}$ that on input $\langle \mathbb{M}, 1^n \rangle$, where $\mathbb{M}$ is a clocked polynomial time Turing machine computing a function $h : \Sigma^* \to \Sigma^*$, and $n \in \mathbb{N}$ decides whether h is an n-tautological in time $n^{g(|M|)}$ for some $g : \mathbb{N} \to \mathbb{N}$.

With this Claim it is straightforward to present the program of a polynomial time Turing machine $\mathbb{M}^{\mathrm{taut}}$ computing h^{taut}, where $\mathbb{M}$ and h are as in the Claim. In fact, let $\mathbb{M}^{\mathrm{taut}}$ be the Turing machine that on input $w \in \Sigma^*$

> using the algorithm $\mathbb{B}$ of the Claim checks whether h is $|w|$-tautological; if so, it computes and outputs $\mathbb{M}(w)$ by simulating $\mathbb{M}$; otherwise, it outputs α_0.

So it only remains to show the Claim.

Proof of the Claim: Let $\mathbb{M}$ be a clocked polynomial time Turing machine computing a function $h : \Sigma^* \to \Sigma^*$. We show that we can decide whether h is n-tautological in the desired time by a reduction to the problem p-HALT. For this we assign to $\mathbb{M}$ a polynomial time nondeterministic Turing machine $\mathbb{M}^+$ such that (as a first approximation) we have

$$h : \Sigma^* \to \textsc{Taut} \iff \mathbb{M}^+ \text{ does not accept } \lambda. \tag{4}$$

To achieve this we take as $\mathbb{M}^+$ the machine that first guesses a string w, computes $\mathbb{M}(w)$ by simulating $\mathbb{M}$, and then checks if $\mathbb{M}(w)$ is a propositional formula; if not, it accepts, otherwise it guesses an assignment for $\mathbb{M}(w)$ and accepts if it does not satisfy $\mathbb{M}(w)$; otherwise it rejects. As $\mathbb{M}$ runs in $\leq |w|^{\mathrm{time}(\mathbb{M})}$ steps on input w, by standard means we can arrange $\mathbb{M}^+$ in such a way that for some polynomial $q \in \mathbb{N}[X]$ the machine $\mathbb{M}^+$ runs exactly $q(n)$ steps if in its first phase it guesses a string w of length n. As $q(n) < q(n+1)$, we get $\big($the fine-tuned version of (4)$\big)$

$$\mathbb{M} \text{ is } n\text{-tautological (more precisely, } h \text{ is } n\text{-tautological)} \iff \left\langle \mathbb{M}^+, 1^{q(n)} \right\rangle \in p\text{-HALT}$$

(note that we need the assumption that $\mathbb{M}$ is clocked to get $\mathbb{M}^+$ and the bound $q(n)$ in the desired effective way). As we assume that p-HALT $\in$ XP$_{\text{uni}}$, we know that whether $(\mathbb{M}^+, 1^{q(n)}) \in p$-HALT may be checked in time $q(n)^{f(|\mathbb{M}^+|)}$ for some function $f : \mathbb{N} \to \mathbb{N}$. As $q(n)^{f(|\mathbb{M}^+|)} = n^{g(|\mathbb{M}|)}$ for suitable $g : \mathbb{N} \to \mathbb{N}$, we are done. □

If $Q \subseteq \Sigma^*$, we write LIST(Q) if there is a listing of all subsets in P of Q by means of polynomial time Turing machines. We use this listing property to prove the implication from left to right of Theorem 7. It is known that:

Theorem 10 ([32]). *There is a polynomially optimal propositional proof system if and only if* LIST(TAUT).

In a first step we will show:

Lemma 11. *If* LIST(HALT)*, then* p-HALT $\in$ XP$_{\text{uni}}$.

Proof. Let $\mathbb{L}$ be a listing of the subsets in P of HALT by polynomial time Turing machines. As for every $\langle \mathbb{M}, 1^n \rangle \in p$-HALT, the set $\{ \langle \mathbb{M}, 1^n \rangle \}$ is a subset in P of HALT, the following algorithm $\mathbb{A}$ accepts p-HALT:

```
A   // a nondeterministic Turing machine M and 1^n with n ∈ N
1.  ℓ ← 1
2.  compute the ℓth machine listed by L
3.          simulate it on input ⟨M, 1^n⟩
4.                  if it accepts then accept
5.  ℓ ← ℓ + 1
6.  goto 2.
```

We want to show that $\mathbb{A}$ runs in time polynomial in n for fixed $\mathbb{M}$ with $t_{\mathbb{M}}(\lambda) = \infty$. Then our claim follows from Lemma 5.

If $\mathbb{M}$ does not halt on λ, then $\{ \langle \mathbb{M}, 1^n \rangle \mid n \in \mathbb{N} \}$ is a subset in P of HALT. Hence, there is a machine listed by $\mathbb{L}$, say the ℓ_0th one, that decides this set. Then Lines 2–4 (for $\ell = \ell_0$) show that the running time of $\mathbb{A}$ is polynomially bounded in n. □

Proof of Theorem 7: It remains to show the implication from left to right (the other one has already been proved by Lemma 9). As the problem HALT, the problem TAUT is coNP-complete and both problems have padding; hence, they are polynomially isomorphic. Thus,

$$\text{LIST(TAUT)} \iff \text{LIST(HALT)}. \tag{5}$$

If there is a polynomially optimal propositional proof system, then LIST(TAUT) by Theorem 10. Thus, LIST(HALT) by (5), hence p-HALT $\in$ XP$_{\text{uni}}$ by the previous result. □

Corollary 12. $\textsc{List}(\textsc{Halt}) \iff p\text{-}\textsc{Halt} \in \mathrm{XP}_{\mathrm{uni}}$.

Proof. Immediate by (5), Theorem 10, and Theorem 7. □

Later we will use the following simple observation.

Lemma 13. *If* $\textsc{List}(\textsc{Halt})$*, then* $\textsc{List}(Q)$ *for every* $Q \in \mathrm{coNP}$.

Proof. More generally, we show:

Assume Q' has padding and $\textsc{List}(Q')$. If $Q \leq_{\mathrm{pol}} Q'$, then $\textsc{List}(Q)$.

Here, $Q \leq_{\mathrm{pol}} Q'$ means that Q is polynomial time reducible to Q'. By the padding property we may assume that the polynomial time reduction $x \mapsto x'$ from Q to Q' is one-to-one and has a polynomial time computable inverse. Then, for every $X' \subseteq Q'$ in P, the set $X := \{x \mid x' \in X'\}$ is a subset in P of Q and every subset in P of Q is obtained in this way. Thus from a listing of the subsets in P of Q', we get a listing of the subsets in P of Q. □

5. Complete problems and p-Halt

In this section for some "semantically defined" complexity classes of classical problems we will see that to show that they contain no complete problem is at least as hard as it is to show that $p\text{-}\textsc{Halt} \notin \mathrm{XP}_{\mathrm{uni}}$. We first deal with complete problems for the class of polynomial time decidable equivalence relations under so-called equivalence reductions (Section 5.1) and then with complete problems for the class UP under polynomial time reductions (Section 5.2).

5.1. Complete P-equivalence relations. Problems concerning the algorithmic properties of equivalence relations arise throughout mathematics and theoretical computer science. Examples are to decide whether two finite graphs are isomorphic or to decide whether two lists of numbers are equivalent in the sense that they represent the same set.

If E and E' are equivalence relations on Σ^*, a polynomial time reduction from E to E' (in the usual sense of complexity theory) is a polynomial time computable function f such that

$$(x, y) \in E \iff f(x, y) \in E'$$

for all $x, y \in \Sigma^*$. Often, in the context of equivalence relations the notion of *equivalence reducibility* is more natural than that of polynomial time reducibility. We say that E *is equivalence reducible to* E' and write $E \leq_{\mathrm{eq}} E'$ if there is a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ such that

$$(x, y) \in E \iff \big(f(x), f(y)\big) \in E'$$

for all $x, y \in \Sigma^*$, that is, writing xEy for $(x, y) \in E$ and similarly for E',

$$xEy \iff f(x)E'f(y).$$

For example, compare the meaning of both notions of reductions if E is the relation of isomorphism between finite groups and E' that of isomorphism between finite graphs.

In [21] Fortnow and Grochow asked whether the class P(eq) of all polynomial time equivalence relations contain a complete problem under equivalence reductions, that is, whether there is an equivalence relation $E_0 \in \mathrm{P(eq)}$ such that $E \leq_{\mathrm{eq}} E_0$ for all $E \in \mathrm{P(eq)}$. We show:

Theorem 14. *If p-*HALT $\in \mathrm{XP}_{\mathrm{uni}}$*, then* P(eq) *contains a complete problem under equivalence reductions.*

To obtain this result we again want to use the technique of invariantization of listings. Hence, the first step yielding a proof of this theorem is a reformulation of its conclusion in terms of a listing. The bridge to listings is provided by the following result. The reader will find a proof in [3].

Proposition 15. *The following are equivalent:*
(a) P(eq) *contains a complete problem under equivalence reductions.*
(b) There is a listing of equivalence relations $\leq_{\mathrm{eq}}$*-cofinal in* P(eq)*; more precisely, there is a listing* $E_1, E_2, \ldots$ *of elements of* P(eq) *by means of clocked polynomial time Turing machines such that for every* $E \in \mathrm{P(eq)}$ *there is an* $i \geq 1$ *such that* $E \leq_{\mathrm{eq}} E_i$*.*

In contrast to Proposition 8, here a listing in terms of *clocked* machines is required; in the following proof of Theorem 14 an additional argument is needed to get such machines; otherwise the proof runs along the lines of that of Lemma 9. We say that a Turing machine $\mathbb{M}$ is a *Turing machine for tuples* if $\mathbb{M}$ first checks whether a given input is a tuple (that is, has the form $\langle x, y\rangle$ with $x, y \in \Sigma^*$) and immediately rejects in the negative case.

Proof of Theorem 14: As it is undecidable whether a Turing machine for tuples accepts an equivalence relation we start with a listing

$$\mathbb{M}_1, \mathbb{M}_2, \ldots \tag{6}$$

of all clocked polynomial time Turing machines for tuples. Hence, in general, the set $L(\mathbb{M}_i)$ of tuples accepted by $\mathbb{M}_i$, will not be an equivalence relation. We invariantize this listing. If T is a set of tuples and $n \in \mathbb{N}$, we say that T is an *n-equivalence relation* if the set of tuples of strings of length at most n, that is, if the set

$$\big\{\langle x, y\rangle \in T \;\big|\; x, y \in \Sigma^{\leq n}\big\}$$

is an equivalence relation on $\Sigma^{\leq n}$. Furthermore, we set

$$T^{\mathrm{eq}} := \big\{\, \langle x, y\rangle \in T \mid T \text{ is a } \max\{|x|, |y|\}\text{-equivalence relation}\big\} \cup \big\{\, \langle x, x\rangle \mid x \in \Sigma^*\big\}.$$

Then

(i) T^{eq} is an equivalence relation on Σ^*;
(ii) $T^{\mathrm{eq}} = T$ if T is an equivalence relation;
(iii) if T is not an n-equivalence relation, then T^{eq} has only finitely many equivalence classes with more than one element;
(iv) if $T \in \mathrm{P}$, then $T^{\mathrm{eq}} \in \mathrm{P}$.

Recall the listing (6) of all clocked polynomial time Turing machines for tuples. By (i)–(iv)

$$L(\mathbb{M}_1)^{\mathrm{eq}}, L(\mathbb{M}_2)^{\mathrm{eq}}, \ldots$$

is an enumeration of P(eq). But we aim at a listing of (a $\leq_{\mathrm{eq}}$-cofinal subset of) P(eq) by means of *clocked polynomial time Turing machines*. We show that there is an effective procedure assigning to every clocked polynomial time Turing machine $\mathbb{M}$ for tuples a clocked polynomial time Turing machine $\mathbb{M}^{\mathrm{eq}}$ such that

$$L(\mathbb{M})^{\mathrm{eq}} \leq_{\mathrm{eq}} L(\mathbb{M}^{\mathrm{eq}}).$$

Then $\mathbb{M}_1^{\mathrm{eq}}, \mathbb{M}_2^{\mathrm{eq}}, \ldots$ is the desired listing. We will show:

Claim: Assume that p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$, then there is an algorithm $\mathbb{B}$ that on input $\langle \mathbb{M}, 1^n \rangle$, where $\mathbb{M}$ is a clocked polynomial time Turing machine for tuples and $n \in \mathbb{N}$, decides whether $L(\mathbb{M})$ is an n-equivalence relation in time $n^{g(|M|)}$ for some $g : \mathbb{N} \to \mathbb{N}$.

Then it is straightforward to present the program of a clocked polynomial time Turing machine $\mathbb{M}^{\mathrm{eq}}$ (where $\mathbb{M}$ is as in the Claim) with

$$L(\mathbb{M}^{\mathrm{eq}}) := \left\{ \langle x, x \rangle \mid x \in \Sigma^* \right\} \cup \Big\{ \langle \langle x, 1^s \rangle, \langle y, 1^s \rangle \rangle \;\Big|\; \langle x, y \rangle \in L(\mathbb{M}),\ s \in \mathbb{N}, \\ \mathbb{B} \text{ accepts } \left\langle \mathbb{M}, 1^{|x|} \right\rangle \text{ in } \leq s \text{ steps and } \left\langle \mathbb{M}, 1^{|y|} \right\rangle \text{ in } \leq s \text{ steps} \Big\}.$$

For $s \geq g(|\mathbb{M}|)$, then the function $x \mapsto \left\langle x, 1^{|x|^s} \right\rangle$ is an equivalence reduction from $L(\mathbb{M})^{\mathrm{eq}}$ to $L(\mathbb{M}^{\mathrm{eq}})$. So it only remains to show the Claim.

Proof of the Claim: The proof parallels that of the claim in the proof of Lemma 9 in Section 4. Let $\mathbb{M}$ be a clocked polynomial time machine for tuples. We assign to $\mathbb{M}$ a polynomial time nondeterministic Turing machine $\mathbb{M}^+$ such that (as a first approximation) we have

$$L(\mathbb{M}) \text{ is an equivalence relation } \iff \mathbb{M}^+ \text{ does not accept } \lambda. \tag{7}$$

For this we take as $\mathbb{M}^+$ a machine that on empty input first guesses a string of the form $\langle r, x \rangle$, $\langle s, x, y \rangle$, or $\langle t, x, y, z \rangle$. Here r ("reflexivity"), s ("symmetry"), and t ("transitivity") are, say, the strings 00, 01, 10, respectively. If the string has the form $\langle r, x \rangle$, then $\mathbb{M}^+$ simulates $\mathbb{M}$ on input $\langle x, x \rangle$ and accepts if and only if $\mathbb{M}$ rejects; similarly, for strings $\langle s, x, y \rangle$ the machine $\mathbb{M}^+$ simulates $\mathbb{M}$ on input $\langle x, y \rangle$ and on input $\langle y, x \rangle$ and accepts if and only if $\mathbb{M}$ accepts $\langle x, y \rangle$ and rejects $\langle y, x \rangle$; it should be clear how $\mathbb{M}^+$ behaves on strings of the form $\langle t, x, y, z \rangle$.

As $\mathbb{M}$ runs in $\leq |w|^{\mathrm{time}(\mathbb{M})}$ steps on input w, by standard means we can arrange $\mathbb{M}^+$ in such a way that for some polynomial $q \in \mathbb{N}[X]$ the machine $\mathbb{M}^+$

runs exactly $q(n)$ steps if in its first phase it guesses a string w of length n. As $q(n) < q(n+1)$, we get (the fine-tuned version of (7))

$$L(\mathbb{M}) \text{ is an } n\text{-equivalence relation} \iff \left\langle \mathbb{M}^+, 1^{q(n)} \right\rangle \in p\text{-Halt}.$$

As we assume that p-Halt $\in \mathrm{XP}_{\mathrm{uni}}$, we know that $\left\langle \mathbb{M}^+, 1^{q(n)} \right\rangle \in p$-Halt may be checked in time $q(n)^{f(|\mathbb{M}^+|)}$ for some function $f : \mathbb{N} \to \mathbb{N}$. As $q(n)^{f(|\mathbb{M}^+|)} = n^{g(|\mathbb{M}|)})$ for suitable $g : \mathbb{N} \to \mathbb{N}$, we are done. □

5.2. UP-complete problems. Recall that a nondeterministic Turing machine $\mathbb{M}$ is *unambiguous* if for every $x \in \Sigma^*$ there is at most one accepting run of $\mathbb{M}$ on input x. UP is the class of problems accepted by an UP-*machine*, that is, by an unambiguous polynomial time nondeterministic Turing machine.

In this section we derive the following result showing that apparently it will be hard to show that UP contains no problem complete under polynomial time reductions. The result is due to Meßner and Torán [29] who have shown that UP contains a problem complete under polynomial time reductions if there is a polynomially optimal proof system. In virtue of Theorem 7 this is equivalent to:

Theorem 16. *If* p-Halt $\in \mathrm{XP}_{\mathrm{uni}}$*, then* UP *contains a problem complete under polynomial time reductions.*

In [29], the corresponding result is shown for the class of sparse sets in NP and further results of this type are given in [26, 24, 25]. We encourage the interested reader to apply our proof method to get these results. Therefore, we give again a quite detailed proof for UP, even though the proof just adapts the method used for equivalence relations to the present case.

Again, first we reformulate the conclusion of Theorem 16 in terms of listings. The reformulation is provided by:

Proposition 17 ([22, 26]). *The following are equivalent:*
(a) UP *contains a problem complete under polynomial time reductions.*
(b) There is a listing of problems $\leq_{\mathrm{pol}}$*-cofinal in* UP*, that is, there is a listing* $\mathbb{M}_1, \mathbb{M}_2, \ldots$ *of clocked* UP*-Turing machines such that for every* $Q \in$ UP *there is an* $i \geq 1$ *such that* $Q \leq_{\mathrm{pol}} L(\mathbb{M}_i)$.

Proof of Theorem 16: If $\mathbb{M}$ is a nondeterministic Turing machine and $n \in \mathbb{N}$ we say that $\mathbb{M}$ is *n-unambiguous* if for every $x \in \Sigma^{\leq n}$ there is at most one accepting run of $\mathbb{M}$ on input x. We set

$$L(\mathbb{M})^{\mathrm{unamb}} := \big\{ x \in L(\mathbb{M}) \;\big|\; \mathbb{M} \text{ is } |x|\text{-unambiguous} \big\}.$$

Then

(i) $L(\mathbb{M})^{\mathrm{unamb}} = L(\mathbb{M})$ if $\mathbb{M}$ is unambiguous;
(ii) if $\mathbb{M}$ is not n-unambiguous, then $L(\mathbb{M})^{\mathrm{unamb}}$ contains only strings of length less than n;

(iii) if $\mathbb{M}$ is a polynomial time nondeterministic Turing machine, then $L(\mathbb{M})^{\text{unamb}}$ is accepted by a UP-machine.

Let $\mathbb{M}_1, \mathbb{M}_2, \ldots$ be a listing of all clocked polynomial time nondeterministic Turing machines. Then, by (i)–(iii),

$$L(\mathbb{M}_1)^{\text{unamb}}, L(\mathbb{M}_2)^{\text{unamb}}, \ldots \tag{8}$$

is an enumeration of UP. But we aim at a listing of (a $\leq_{\text{pol}}$-cofinal subset of) UP by means of *clocked UP-machines*. We show that there is an effective procedure assigning to every clocked polynomial time nondeterministic Turing machine $\mathbb{M}$ a clocked UP-machine $\mathbb{M}^{\text{unamb}}$ such that

$$L(\mathbb{M})^{\text{unamb}} \leq_{\text{pol}} L(\mathbb{M}^{\text{unamb}}).$$

Then $\mathbb{M}_1^{\text{unamb}}, \mathbb{M}_2^{\text{unamb}}, \ldots$ is the desired listing. We will show:

Claim: Assume that p-HALT $\in \text{XP}_{\text{uni}}$, then there is an algorithm $\mathbb{B}$ that on input $\langle \mathbb{M}, 1^n \rangle$, where $\mathbb{M}$ is a clocked polynomial time nondeterministic Turing machine and $n \in \mathbb{N}$, decides whether $L(\mathbb{M})$ is n-unambiguous in time $n^{g(|M|)}$ for some $g : \mathbb{N} \to \mathbb{N}$.

With this Claim it is straightforward to present the program of a clocked UP-machine $\mathbb{M}^{\text{unamb}}$ (where $\mathbb{M}$ is as in the Claim) with

$$L(\mathbb{M}^{\text{unamb}}) := \left\{ \langle x, 1^s \rangle \;\middle|\; x \in L(\mathbb{M}) \text{ and } \mathbb{B} \text{ accepts } \left\langle \mathbb{M}, 1^{|x|} \right\rangle \text{ in } \leq s \text{ steps} \right\}.$$

Then, for $s \geq g(|\mathbb{M}|)$, the function $x \mapsto \langle x, 1^{|x|^s} \rangle$ is a polynomial time reduction from $L(\mathbb{M})^{\text{unamb}}$ to $L(\mathbb{M}^{\text{unamb}})$. So it only remains to show the Claim.

Proof of the Claim: Let $\mathbb{M}$ be a clocked polynomial time nondeterministic Turing machine. We assign to $\mathbb{M}$ a polynomial time nondeterministic Turing machine $\mathbb{M}^+$ such that (as a first approximation) we have

$$\mathbb{M} \text{ is unambiguous} \iff \mathbb{M}^+ \text{ does not accept } \lambda. \tag{9}$$

For this we take as $\mathbb{M}^+$ a machine that on empty input first guesses strings y, c_1, and c_2, then it checks whether c_1 and c_2 are distinct runs of $\mathbb{M}$ accepting y; if so $\mathbb{M}^+$ accepts, else it rejects. "Fine-tuning" as in the proof of the corresponding claim in the proof of Theorem 14 yields the statement. □

6. Logics capturing P and p-Halt

We start by recalling the concepts from logic we need.

Structures. A *vocabulary* τ is a finite set of relation symbols. Each relation symbol has an *arity*. A *structure* $\mathcal{A}$ of vocabulary τ, or *τ-structure* (or, simply structure), consists of a nonempty set A called the *universe*, and an interpretation $R^{\mathcal{A}} \subseteq A^r$ of each r-ary relation symbol $R \in \tau$. *All structures are assumed to have a finite*

universe. To avoid technicalities we assume in this section that all structures have as universe, for some $n \in \mathbb{N}$, the set $[n] := \{1, 2, \ldots, n\}$. Then, in a canonical way, we can identify structures with nonempty strings over Σ; in particular, $|\mathcal{A}|$ for a structure $\mathcal{A}$ is the length of the string $\mathcal{A}$. Moreover, then every structure $\mathcal{A}$ has a natural ordering $<_A$ on it.

In this section we deal with classes of structures. Thereby we always assume that all structures of a fixed class have the same vocabulary. But distinct vocabularies may correspond to distinct classes.

Logics and logics capturing P. For our purposes a *logic* L consists

- for every vocabulary τ of a set $L[\tau]$ of strings, the set of *L-sentences of vocabulary τ*, and of an algorithm that for every τ and every string ξ decides whether $\xi \in L[\tau]$ (in particular, $L[\tau]$ is decidable for every τ);
- of a *satisfaction relation* $\models_L$; if $(\mathcal{A}, \varphi) \in \models_L$ (written: $\mathcal{A} \models_L \varphi$), then $\mathcal{A}$ is a τ-structure and $\varphi \in L[\tau]$ for some vocabulary τ; furthermore, for each $\varphi \in L[\tau]$ the class
$$\textsc{Mod}_L(\varphi) := \{\mathcal{A} \mid \mathcal{A} \models_L \varphi\}$$
of *models of* φ is closed under isomorphism.

From now on, if we say "let φ be an L-sentence," we mean that, in addition to φ, a vocabulary τ with $\varphi \in L[\tau]$ is given.

Definition 18. Let L be a logic.

(a) *L is a logic for* P if for all classes S of structures closed under isomorphism (with respect to structures with universe of the form $[n]$ for some $n \in \mathbb{N}$) we have
$$S \in \mathrm{P} \iff S = \textsc{Mod}_L(\varphi) \text{ for some } L\text{-sentence } \varphi.$$

(b) *L captures* P if (a) holds and if there is an algorithm $\mathbb{A}$ deciding $\models_L$ (that is, for every structure $\mathcal{A}$ and L-sentence φ the algorithm $\mathbb{A}$ decides whether $\mathcal{A} \models_L \varphi$) and if moreover, $\mathbb{A}$ for every fixed φ runs in time polynomial in $|\mathcal{A}|$.

Hence, if L captures P, then for every L-sentence φ the algorithm $\mathbb{A}$ witnesses that $\mathrm{Mod}_L(\varphi) \in \mathrm{P}$. However, we do not necessarily know ahead of time the bounding polynomial.

(c) *L is an effectively captures* P if L captures P and if in addition to the algorithm $\mathbb{A}$ as in (b) there is a computable function that assigns to every L-sentence φ a polynomial $q \in \mathbb{N}[X]$ such that $\mathbb{A}$ decides whether $\mathcal{A} \models_L \varphi$ in $\leq q(|\mathcal{A}|)$ steps.

If there is no logic capturing P, then P $\neq$ NP (as Fagin [18] showed that there is a logic capturing NP). We prove that then even p-HALT $\notin \mathrm{XP}_{\mathrm{uni}}$ (cf. Theorem 1):

Theorem 19. *If p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$, then there is a logic capturing* P.

In addition, we want to show that under the assumption p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ the so-called invariant least fixpoint logic captures P.

The following well-known result yields the desired reformulation in terms of a listing. We include a proof as we shall make use of the logic associated with a listing. Here we say that a Turing machine $\mathbb{M}$ is a *Turing machine for structures* if (the code of) $\mathbb{M}$ contains a vocabulary τ and $\mathbb{M}$ first checks whether a given input is a τ-structure and immediately rejects in the negative case. We then also say that $\mathbb{M}$ is a τ-machine.

Proposition 20. *The following are equivalent:*
(a) There is a logic (effectively) capturing P.
(b) There is a listing of all classes in P *of structures closed under isomorphism by means of (clocked) polynomial time Turing machines for structures.*

Proof. (b) $\Rightarrow$ (a): Let the listing $\mathbb{L}$ with the enumeration

$$\mathbb{M}_1, \mathbb{M}_2, \ldots$$

be as in (b). We may assume that $|\mathbb{M}_1| < |\mathbb{M}_2| < \ldots$ by adding dummy lines to the programs of the machines if necessary. Then the logic $L(\mathbb{L})$ given by

$$L(\mathbb{L})[\tau] := \big\{\mathbb{M}_i \;\big|\; i \geq 1 \text{ and } \mathbb{M}_i \text{ is a } \tau\text{-machine}\big\}$$

and

$$\mathcal{A} \models_{L(\mathbb{L})} \mathbb{M} \iff \mathbb{M} \text{ accepts } \mathcal{A}$$

is a logic capturing P. If all the machines of the listing are clocked, then $L(\mathbb{L})$ effectively captures P.

Conversely, if L is a logic (effectively) capturing P and $\mathbb{A}$ is the algorithm deciding $\mathcal{A} \models_L \varphi$ in time $|\mathcal{A}|^{f(|\varphi|)}$ for some (computable) $f : \mathbb{N} \to \mathbb{N}$, denote by $\mathbb{M}_\varphi$ the (clocked) polynomial time Turing machine obtained by restricting $\mathbb{A}$ to inputs of the form $\langle \ldots, \varphi \rangle$. Then for any effective enumeration $\varphi_1, \varphi_2, \ldots$ of the sentences of L, the listing

$$\mathbb{M}_{\varphi_1}, \mathbb{M}_{\varphi_2}, \ldots$$

is the desired listing of all classes in P of structures closed under isomorphism.□

Proof of Theorem 19: Again we face the problem that it is undecidable whether a Turing machine for structures accepts a class closed under isomorphism. Therefore we start with a listing

$$\mathbb{M}_1, \mathbb{M}_2, \ldots \tag{10}$$

of *all* clocked polynomial time Turing machines for structures. In general, the class $L(\mathbb{M}_i) = \{\mathcal{A} \mid \mathbb{M}_i \text{ accepts } \mathcal{A}\}$ of structures accepted by $\mathbb{M}_i$ will not be closed under isomorphism. We apply the invariantization technique to get a listing as required by part (b) of Proposition 20.

If S is a class of structures and $n \in \mathbb{N}$, then we say that S is $n \cong$*-invariant* if for all isomorphic structures $\mathcal{A}$ and $\mathcal{B}$ whose universes have at most n elements we have

$$\mathcal{A} \in S \iff \mathcal{B} \in S.$$

We set

$$S^{\mathrm{inv}} := \big\{\mathcal{A} \in S \;\big|\; S \text{ is } |A| \cong\text{-invariant}\big\}. \tag{11}$$

We have

(i) S^{inv} is closed under isomorphism;
(ii) $S^{\mathrm{inv}} = S$ if S is closed under isomorphism;
(iii) if S is not $n \cong$-invariant, then all structures in S^{inv} have less than n elements;
(iv) if $S \in \mathrm{P}$, then $S^{\mathrm{inv}} \in \mathrm{P}$.

Thus,

$$L(\mathbb{M}_1)^{\mathrm{inv}}, L(\mathbb{M}_2)^{\mathrm{inv}}, \ldots$$

is an enumeration of all classes in P of structures closed under isomorphism $\big($in fact, the classes are in P by (iv), they are closed under isomorphism by (i), and all such classes occur by (ii)$\big)$.

We show that there is an effective procedure assigning to every clocked polynomial time Turing machine $\mathbb{M}$ for structures a (clocked) polynomial time Turing machine $\mathbb{M}^{\mathrm{inv}}$ for structures such that

$$L(\mathbb{M}^{\mathrm{inv}}) = L(\mathbb{M})^{\mathrm{inv}}. \tag{12}$$

Then $\mathbb{M}_1^{\mathrm{inv}}, \mathbb{M}_2^{\mathrm{inv}}, \ldots$ is the desired listing.

Claim: Assume that p-Halt $\in \mathrm{XP}_{\mathrm{uni}}$ (p-Halt $\in \mathrm{XP}$), then there is an algorithm $\mathbb{B}$ that on input $\langle \mathbb{M}, 1^n \rangle$, where $\mathbb{M}$ is a clocked polynomial time Turing machine for structures and $n \in \mathbb{N}$, decides whether $L(\mathbb{M})$ is $n \cong$-invariant in time $n^{g(|M|)}$ for some (computable) $g : \mathbb{N} \to \mathbb{N}$.

With this claim and the definition (11) of S^{inv} it is straightforward to present the program of a (clocked) polynomial time Turing machine $\mathbb{M}^{\mathrm{inv}}$ for structures satisfying (12).

Proof of the Claim: As in preceding proofs we reduce our problem to p-Halt. Let $\mathbb{M}$ be a clocked polynomial time Turing machine for structures. We assign to $\mathbb{M}$ a nondeterministic Turing machine $\mathbb{M}^+$ such that (as a first approximation) we have

$$L(\mathbb{M}) \text{ is closed under isomorphism} \iff \mathbb{M}^+ \text{ does not accept } \lambda. \tag{13}$$

For this we take as $\mathbb{M}^+$ a machine that on empty input first guesses strings $\mathcal{A}$, $\mathcal{B}$, and f, and accepts if $\mathcal{A}$ and $\mathcal{B}$ are structures, f is an isomorphism between $\mathcal{A}$ and $\mathcal{B}$, and $\mathbb{M}$ accepts $\mathcal{A}$ but rejects $\mathcal{B}$.

As $\mathbb{M}$ runs in $\leq |w|^{\mathrm{time}(\mathbb{M})}$ steps on input w, by standard means we can arrange $\mathbb{M}^+$ in such a way that for some polynomial $q \in \mathbb{N}[X]$ the machine $\mathbb{M}^+$ runs exactly $q(n)$ steps if in its first phase it guesses a structure $\mathcal{A}$ with n elements. As $q(n) < q(n+1)$, we get (the fine-tuned version of (13))

$$L(\mathbb{M}) \text{ is } n \cong\text{-invariant} \iff \left\langle \mathbb{M}^+, 1^{q(n)} \right\rangle \in p\text{-Halt}.$$

As we assume that p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ (p-HALT $\in \mathrm{XP}$), we know that $(\mathbb{M}^+, 1^{q(n)}) \in p$-HALT may be checked in time $q(n)^{f(|\mathbb{M}^+|)}$ for some (computable) function $f : \mathbb{N} \to \mathbb{N}$. As $q(n)^{f(|\mathbb{M}^+|)} = n^{g(|\mathbb{M}|)}$ for a suitable (computable) $g : \mathbb{N} \to \mathbb{N}$, we are done. □

In the previous proof we have shown the implication "(a) ⇒ (b)" of:

Proposition 21. *The following are equivalent:*
(a) p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ (p-HALT $\in \mathrm{XP}$)*;*
(b) There is an effective procedure assigning to every clocked polynomial time Turing machine $\mathbb{M}$ for structures a (clocked) polynomial time Turing machine $\mathbb{M}^{\mathrm{inv}}$ for structures such that

$$L(\mathbb{M}^{\mathrm{inv}}) = L(\mathbb{M})^{\mathrm{inv}}.$$

Proof. In the following we leave the proofs of the effective versions of our claims to the reader. It remains to prove the implication (b) ⇒ (a). For $k \geq 1$ let τ_k be the vocabulary $\{P_1, \ldots, P_k\}$ with unary relation symbols $P_1, \ldots, P_k$.

Let $\mathbb{M}$ be a nondeterministic Turing machine. We can assume that $[k]$ for some $k \geq 1$ is the set of states of $\mathbb{M}$. By our convention on nondeterministic machines every two distinct successor configurations of a given configuration of $\mathbb{M}$ have distinct states. Furthermore, here we assume that the starting state is not an accepting state. We let $S(\mathbb{M})$ be the class of τ_k-structures $\mathcal{B}$ satisfying (i) and (ii).

(i) The P_i's with $i \in [k]$ form a partition of the universe B of $\mathcal{B}$ and there is an $n \in \mathbb{N}$ such that $|P_i| = n$ for all $i \in [k]$.
(ii) If $i_1, \ldots, i_n$ (where n is according to (i)) are such that the jth element of B in the natural ordering $<_B$ of B is in P_{i_j}, then we require that neither $i_1, \ldots, i_n$ nor any initial segment of it are the states of a run accepting the empty input.

Clearly, $S(\mathbb{M}) \in \mathrm{P}$ and from $\mathbb{M}$ we get a clocked polynomial time machine $\mathbb{M}_0$ for τ_k-structures deciding $S(\mathbb{M})$, that is,

$$L(\mathbb{M}_0) = S(\mathbb{M}).$$

Furthermore, let $\mathcal{A}(\mathbb{M}, n)$ be the τ_k-structure with universe $[n \cdot k]$ and with $P_i := \{n \cdot (i-1)+1, \ldots, n \cdot (i-1)+n\}$ for all $i \in [k]$. Then $\mathcal{A}(\mathbb{M}, n) \in S(\mathbb{M})$ as the starting state is not an accepting one. Furthermore,

$$\mathcal{A}(\mathbb{M}, n) \in S(\mathbb{M})^{\mathrm{inv}} \ (= L(\mathbb{M}_0)^{\mathrm{inv}}) \iff \langle \mathbb{M}, 1^n \rangle \in p\text{-HALT},$$

as we obtain all possible sequences of states of length n by considering the isomorphic copies of $\mathcal{A}(\mathbb{M}, n)$. Thus, by our assumption (b), we have for the machine $\mathbb{M}_0^{\mathrm{inv}}$,

$$\mathbb{M}_0^{\mathrm{inv}} \text{ accepts } \mathcal{A}(\mathbb{M}, n) \iff \langle \mathbb{M}, 1^n \rangle \in p\text{-HALT}.$$

Therefore, the following algorithm $\mathbb{A}$ shows that p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$: On input $\mathbb{M}$, a nondeterministic Turing machine, and 1^n with $n \in \mathbb{N}$, it first computes the structure $\mathcal{A}(\mathbb{M}, n)$ and the clocked polynomial time machine $\mathbb{M}_0$; then applying the effective procedure of (b), it gets the machine $\mathbb{M}_0^{\mathrm{inv}}$; finally, it checks whether $\mathbb{M}_0^{\mathrm{inv}}$ accepts $\mathcal{A}(\mathbb{M}, n)$. □

6.1. The invariant least fixpoint logic. Recall that in the proof of Theorem 19 we started with a listing $\mathbb{M}_1, \mathbb{M}_2, \ldots$ of all clocked polynomial time Turing machine for structures and obtained under the hypothesis p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ (p-HALT $\in$ XP) a listing $\mathbb{M}_1^{\mathrm{inv}}, \mathbb{M}_2^{\mathrm{inv}}, \ldots$ of all classes in P of structures closed under isomorphism by means of (clocked) polynomial time Turing machines for structures. We denote this listing $\mathbb{M}_1^{\mathrm{inv}}, \mathbb{M}_2^{\mathrm{inv}}, \ldots$ by $\mathbb{L}$. Then the logic $L(\mathbb{L})$, the logic assigned to $\mathbb{L}$ in the proof of the direction "(b) ⇒ (a)" of Proposition 20, (effectively) captures polynomial time. In this section we present a more "logic-friendly" version of this logic.

For every vocabulary τ we let $\tau_< := \tau \cup \{<\}$, where $<$ is a binary relation symbol not in τ chosen in some canonical way. A logic L *captures* P *on ordered structures* if (a) and (b) of Definition 18 hold for *ordered* structures and classes of *ordered* structures. In Definition 18 (b), for fixed $\varphi \in L[\tau_<]$ the algorithm $\mathbb{A}$ must witness that the class of ordered models of φ is in P. It should be clear what we mean by a logic *effectively capturing* P *on ordered structures*.

Least fixpoint logic LFP is an extension of first-order logic obtained by adding an operator which allows to speak about the least fixpoint of monotone operations definable in the logic. We only need the following property of LFP.

Theorem 22. [23, 34] LFP *effectively captures* P *on ordered structures.*

If S is a class of $\tau_<$-structures and $n \in \mathbb{N}$, then S is n *$<$-invariant* if for all τ-structures $\mathcal{A}$ with $|A| \leq n$ and every orderings $<_1$ and $<_2$ of A we have

$$(\mathcal{A}, <_1) \in S \iff (\mathcal{A}, <_2) \in S.$$

We define the *invariant least fixpoint logic* $\mathrm{LFP}_{\mathrm{inv}}$ by: For every vocabulary τ we set

$$\mathrm{LFP}_{\mathrm{inv}}[\tau] := \mathrm{LFP}[\tau_<],$$

and we define the satisfaction relation by

$$\mathcal{A} \models_{\mathrm{LFP}_{\mathrm{inv}}} \varphi \iff \Big(\mathrm{MOD}_{\mathrm{LFP}}(\varphi) \text{ is } |A| <\text{-invariant and } (\mathcal{A}, <_A) \models_{\mathrm{LFP}} \varphi\Big); \tag{14}$$

recall that $<_A$ denotes the natural ordering on A.

If $\mathrm{MOD}_{\mathrm{LFP}}(\varphi)$ is not n $<$-invariant, then $\mathrm{MOD}_{\mathrm{LFP}_{\mathrm{inv}}}(\varphi)$ only contains structures with universe of cardinality less than n and hence is in P. Together with the fact that LFP captures P on ordered structures, this shows that $\mathrm{LFP}_{\mathrm{inv}}$ is a logic for P.

In the proof of Theorem 19 we started with a listing $\mathbb{L}_0$ of all clocked polynomial time Turing machines for structures. As LFP captures P on ordered

structures, in a certain sense the listing $\mathbb{L}_0$ corresponds to the sentences of LFP in the enlarged vocabularies $\tau_<$. We invariantized the listing $\mathbb{L}_0$ by using the concept of $n \cong$-invariance. As orderings correspond to permutations and hence to isomorphisms, in $\mathrm{LFP}_{\mathrm{inv}}$ this invariantization is taken care by the definition of its semantics in (14). Hence the following result is not surprising:

Theorem 23 ([31]). *(a)* p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ *if and only if* $\mathrm{LFP}_{\mathrm{inv}}$ *captures* P. *(b)* p-HALT $\in$ XP *if and only if* $\mathrm{LFP}_{\mathrm{inv}}$ *effectively captures* P.

Proof. For (a) it suffices to show that $\mathrm{LFP}_{\mathrm{inv}}$ captures P if and only if there is an effective procedure as stated in Proposition 21 (b).

Assume first that such a procedure exists. As LFP effectively captures P on ordered structures, for $\varphi \in \mathrm{LFP}_{\mathrm{inv}}[\tau] = \mathrm{LFP}[\tau_<]$, we obtain a clocked polynomial time machine $\mathbb{M}_\varphi$ for τ-structures with

$$L(\mathbb{M}_\varphi) := \big\{\mathcal{A} \;\big|\; (\mathcal{A}, <_A) \models_{\mathrm{LFP}} \varphi\big\}.$$

Then, one easily verifies that

$$L(\mathbb{M}_\varphi)^{\mathrm{inv}} = \mathrm{MOD}_{\mathrm{LFP}_{\mathrm{inv}}}(\varphi),$$

that is, $L(\mathbb{M}_\varphi^{\mathrm{inv}}) = \mathrm{MOD}_{\mathrm{LFP}_{\mathrm{inv}}}(\varphi)$. Hence, the algorithm that on input $\langle \mathcal{A}, \varphi\rangle$, first computes $\mathbb{M}_\varphi$, then $\mathbb{M}_\varphi^{\mathrm{inv}}$ and finally simulates $\mathbb{M}_\varphi^{\mathrm{inv}}$ on input $\mathcal{A}$, decides the satisfaction relation of $\mathrm{LFP}_{\mathrm{inv}}$ in the desired time.

Conversely, assume that $\mathrm{LFP}_{\mathrm{inv}}$ captures P. Let $\mathbb{M}$ be a clocked polynomial time Turing machine for τ-structures. Then

$$C := \Big\{(\mathcal{B}, <) \;\Big|\; \text{for some } \mathcal{A} \text{ we have: } \Big((\mathcal{B}, <) \cong (\mathcal{A}, <_A) \text{ and } \mathbb{M} \text{ accepts } \mathcal{A}\Big)\Big\}$$

is a class in P of ordered τ-structures closed under isomorphism (clearly, from $(\mathcal{B}, <)$ one can determine the unique structure $\mathcal{A}$ with $(\mathcal{B}, <) \cong (\mathcal{A}, <_A)$ in polynomial time). Hence, from $\mathbb{M}$ we effectively get a clocked polynomial time Turing machine $\mathbb{M}^*$ with $L(\mathbb{M}^*) = C$. Furthermore, it is well-known that from a clocked polynomial time Turing machine accepting a class in P of ordered τ-structures closed under isomorphism one effectively gets an $\mathrm{LFP}[\tau_<]$-sentence φ with $C = \mathrm{MOD}_{\mathrm{LFP}}(\varphi)$. Now, again it is routine to verify that $L(\mathbb{M})^{\mathrm{inv}} = \mathrm{MOD}_{\mathrm{LFP}_{\mathrm{inv}}}(\varphi)$. Thus, from the algorithm deciding the satisfaction relation and witnessing that $\mathrm{LFP}_{\mathrm{inv}}$ captures P, we can extract the algorithm assigning to a clocked polynomial time Turing machine $\mathbb{M}$ a polynomial time Turing machine $\mathbb{M}^{\mathrm{inv}}$ with $L(\mathbb{M}^{\mathrm{inv}}) = L(\mathbb{M})^{\mathrm{inv}}$. □

7. Slicewise downward monotone parameterized problems

We already mentioned in the Introduction that we do not want to present an abstract version of the invariantization technique available if p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ and underlying the previous proofs. However, in this section we want to point

out that hidden at the core of each of these proofs is a parameterized problem which (as we show here) is in $\mathrm{XP}_{\mathrm{uni}}$ if and only if p-Halt is in $\mathrm{XP}_{\mathrm{uni}}$.

The problem p-Halt is slicewise downward monotone. A parameterized problem (Q, κ) is *slicewise downward monotone* if Q is decidable, all elements of Q have the form $\langle x, 1^n \rangle$ with $x \in \Sigma^*$ and $n \in \mathbb{N}$, if $\kappa(\langle x, 1^n \rangle) = |x|$, and finally if the slices are downward monotone, that is, for all $x \in \Sigma^*$ and $n, n' \in \mathbb{N}$

$$\langle x, 1^n \rangle \in Q \text{ and } n' < n \text{ imply } \left\langle x, 1^{n'} \right\rangle \in Q.$$

The following slicewise downward monotone problems

$$p\text{-Taut}, \quad p\text{-Equiv}, \quad p\text{-Unamb}, \quad \text{and} \quad p\text{-}\cong\text{-Inv}$$

are hidden in our considerations on polynomially optimal proof systems, P(eq)-complete problems, UP-complete problems, and logics capturing P, respectively. Here

p-Taut
Instance: A clocked polynomial time Turing machine $\mathbb{M}$ and 1^n with $n \in \mathbb{N}$.
Parameter: $|\mathbb{M}|$.
Problem: Is $\mathbb{M}$ n-tautological?

p-Equiv
Instance: A clocked polynomial time Turing machine $\mathbb{M}$ for tuples and 1^n with $n \in \mathbb{N}$.
Parameter: $|\mathbb{M}|$.
Problem: Is $L(\mathbb{M})$ an n-equivalence relation?

p-Unamb
Instance: A clocked polynomial time nondeterministic Turing machine $\mathbb{M}$ and 1^n with $n \in \mathbb{N}$.
Parameter: $|\mathbb{M}|$.
Problem: Is $\mathbb{M}$ n-unambiguous?

p-$\cong$-Inv
Instance: A clocked polynomial time Turing machine $\mathbb{M}$ for structures and 1^n with $n \in \mathbb{N}$.
Parameter: $|\mathbb{M}|$.
Problem: Is $L(\mathbb{M})$ n $\cong$-invariant?

The *Claim*s in the proofs of Lemma 9, Theorem 14, Theorem 16, and Theorem 19 show that all these problem are in $\mathrm{XP}_{\mathrm{uni}}$ if p-Halt $\in \mathrm{XP}_{\mathrm{uni}}$. We show:

Theorem 24. *If one of the problems*

$$p\text{-Taut},\ p\text{-Equiv},\ p\text{-Unamb},\ p\text{-}\cong\text{-Inv},\ \textit{and}\ p\text{-Halt}$$

is in $\mathrm{XP}_{\mathrm{uni}}$*, then all are.*

To compare the complexity of parameterized problems we use standard notions of reductions of parameterized complexity theory that we recall first. Let (Q,κ) and (Q',κ') be parameterized problems. We write $(Q,\kappa) \leq_{\mathrm{fpt}} (Q',\kappa')$ if there is an *fpt-reduction* from (Q,κ) to (Q',κ'), that is, a mapping $R : \Sigma^* \to \Sigma^*$ with:

(1) For all $x \in \Sigma^*$ we have $(x \in Q \iff R(x) \in Q')$.
(2) $R(x)$ is computable in time $f(\kappa(x)) \cdot |x|^{O(1)}$ for some computable $f : \mathbb{N} \to \mathbb{N}$.
(3) There is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

We write $(Q,\kappa) \leq_{\mathrm{xp}} (Q',\kappa')$ if there is an *xp-reduction* from (Q,κ) to (Q',κ'), which is defined as $(Q,\kappa) \leq_{\mathrm{fpt}} (Q',\kappa')$ except that instead of (2) it is only required that $R(x)$ is computable in time $|x|^{f(\kappa(x))}$ for some computable $f : \mathbb{N} \to \mathbb{N}$. These are notions of reductions of the usual (strongly uniform) parameterized complexity theory. We shall use the following simple observation.

Lemma 25. *If* $(Q,\kappa) \leq_{\mathrm{xp}} (Q',\kappa')$ *and* $(Q',\kappa') \in \mathrm{XP}_{\mathrm{uni}}$*, then* $(Q,\kappa) \in \mathrm{XP}_{\mathrm{uni}}$.

Proof of Theorem 24. By the lemma and by the remark preceding the theorem it suffices to show that

$$p\text{-Halt} \leq_{\mathrm{fpt}} p\text{-Taut} \leq_{\mathrm{xp}} p\text{-Equiv} \leq_{\mathrm{xp}} p\text{-Unamb} \leq_{\mathrm{xp}} p\text{-}\cong\text{-Inv}.$$

p-Halt $\leq_{\mathrm{fpt}}$ p-Taut: Let $\mathbb{M}$ be a nondeterministic Turing machine. We choose $s(\mathbb{M}) \in \mathbb{N}$ such that all states of $\mathbb{M}$ are (coded by) strings of length $s(\mathbb{M})$. We fix a tautology α_0 and define $f : \Sigma^* \to \Sigma^*$ by

$$f(w) := \begin{cases} \lambda, & \text{if } w \text{ is the sequence of states of a run of } \mathbb{M} \text{ accepting } \lambda; \\ \alpha_0, & \text{otherwise.} \end{cases}$$

Of course, there is a polynomial time procedure assigning to $\mathbb{M}$ a clocked polynomial time Turing machine $\mathbb{M}'$ computing f. Then

$$\langle \mathbb{M}, 1^n \rangle \in p\text{-Halt} \iff \left\langle \mathbb{M}', 1^{n \cdot s(\mathbb{M})} \right\rangle \in p\text{-Taut},$$

so that $\langle \mathbb{M}, 1^n \rangle \mapsto \left\langle \mathbb{M}', 1^{n \cdot s(\mathbb{M})} \right\rangle$ is an fpt-reduction from p-Halt to p-Taut.

p-Taut $\leq_{\mathrm{xp}}$ p-Equiv: For a clocked polynomial time Turing machine $\mathbb{M}$ let $\mathbb{M}'$ be the Turing machine for tuples that accepts $\langle x, y \rangle$ if either $x = y$ or $\big(x = \lambda$ and $y = \langle w, v \rangle$, where $\mathbb{M}(w)$ is no propositional formula or ($\mathbb{M}(w)$ is a propositional formula and v a valuation which does not satisfy $\mathbb{M}(w)$)$\big)$. Again

we can assume that from $\mathbb{M}$ we get a clocked polynomial time such $\mathbb{M}'$ and that for some polynomial $q \in \mathbb{N}[X]$

$$\langle \mathbb{M}, 1^n \rangle \in p\text{-Taut} \iff \left\langle \mathbb{M}', 1^{q(n)} \right\rangle \in p\text{-Equiv}.$$

This yields the reduction $\langle \mathbb{M}, 1^n \rangle \mapsto \left\langle \mathbb{M}', 1^{q(n)} \right\rangle$ from p-Taut to p-Equiv, which is an xp-reduction and not an fpt-reduction as the degree of the polynomial q depends on time($\mathbb{M}$).

p-Equiv $\leq_{\text{xp}}$ p-Unamb: Let $\mathbb{M}$ be a clocked polynomial time machine for tuples. We consider the following clocked nondeterministic polynomial time machine $\mathbb{M}'$: It has an initial state which is left in the first step and cannot be visited again during any run on any input. From the initial state a direct transition to an accepting state is possible (independent of the symbol scanned by the head). Hence, $L(\mathbb{M}') = \Sigma^*$. Furthermore, all other runs on inputs which do not have the form $\langle r, x \rangle$, $\langle s, x, y \rangle$, or $\langle t, x, y, z \rangle$ will be rejecting. Here r ("reflexivity"), s ("symmetry"), and t ("transitivity") are the strings 00, 01, 10, respectively. On input $\langle r, x \rangle$ there is a run of $\mathbb{M}'$ which simulates $\mathbb{M}$ on input $\langle x, x \rangle$ and accepts if and only if $\mathbb{M}$ rejects; similarly, there is an additional accepting run of $\mathbb{M}'$ on input $\langle s, x, y \rangle$ if and only if $\big(\mathbb{M}$ accepts $\langle x, y \rangle$ and rejects $\langle y, x \rangle\big)$; finally, there is an additional accepting run of $\mathbb{M}'$ on input $\langle t, x, y, z \rangle$ if and only if $\big(\mathbb{M}$ accepts $\langle x, y \rangle$ and $\langle y, x \rangle$ but not $\langle x, z \rangle\big)$. In particular, we see that

$$L(\mathbb{M}) \text{ is an equivalence relation on } \Sigma^* \iff \mathbb{M}' \text{ is a UP-machine.}$$

Moreover, one can arrange matters in such a way that for some polynomial $q \in \mathbb{N}[X]$ we have

$$\langle \mathbb{M}, 1^n \rangle \in p\text{-Equiv} \iff \left\langle \mathbb{M}', 1^{q(n)} \right\rangle \in p\text{-Unamb}.$$

p-Unamb $\leq_{\text{fpt}}$ p-$\cong$-Inv: We assign to a string $w \in \Sigma^*$ a structure $\mathcal{A}(w)$ of vocabulary $\tau := \{<, P_0\}$, where

- the universe of $\mathcal{A}(w)$ is $[|w|]$;
- the binary $<$ is interpreted by the natural ordering on $[|w|]$;
- the unary P_0 is interpreted by the set of positions in w carrying a 0.

For $k \geq 1$ we introduce the vocabulary $\tau_k = \{U, V, P_1, \ldots, P_k, R\}$ with unary relation symbols $U, V, P_1, \ldots, P_k$ and a binary R. Let $\mathbb{M}$ be a clocked polynomial time nondeterministic Turing machine. We assume that $\mathbb{M}$ runs exactly $n^{\text{time}(\mathbb{M})}$ steps on inputs of length n. We set $q(n) := n^{\text{time}(\mathbb{M})}$. We let $\mathbb{M}'$ be a clocked polynomial time Turing machine for $\tau \cup \tau_k$-structures that accepts a structure $\mathcal{B}$ if for some $w \in \Sigma^*$ and $n := |w|$:

(i) (the interpretation of) U and V form a partition of B;
(ii) the τ-reduct on U is isomorphic to $\mathcal{A}(w)$;
(iii) the P_i's with $i \in [k]$ form a partition of the V-part and $|P_i| = q(n)$ for all $i \in [k]$;

(iv) R is an ordering of its field, this field is contained in the V-part and it has exactly exactly $q(n)$ elements;
(v) if $i_1, \ldots, i_{q(n)}$ are such that the mth element of the ordering R is in P_{i_m}, then $i_1, \ldots, i_{q(n)}$ is the sequence of states of a run of $\mathbb{M}$ accepting w;
(vi) if $j_1, \ldots, j_{q(n)}$ are such that the mth element of V in the natural ordering on B is in P_{i_m}, then either $(i_1, \ldots, i_{q(n)}) = (j_1, \ldots, j_{q(n)})$ or $j_1, \ldots, j_{q(n)}$ is not the sequence of states of a run of $\mathbb{M}$ accepting w.

It is easy to see that $\mathbb{M}$ is an unambiguous machine if and only if the class of structures accepted by $\mathbb{M}'$ is closed under isomorphism. We leave the rest of the argument to the reader. □

We close this section by showing that some results we proved for p-HALT hold for all slicewise downward monotone parameterized problems. The proof of the first result is obtained by the obvious modifications in that of Proposition 4.

Proposition 26. *Every slicewise downward monotone parameterized problem is in the class* $\mathrm{FPT}_{\mathrm{nu}}$.

If (Q, κ) is slicewise downward monotone and $x \in \Sigma^*$, we set

$$s(x) := \min\bigl\{n \bigm| n \in \mathbb{N} \text{ and } \langle x, 1^n\rangle \notin Q\bigr\}.$$

If $\langle x, 1^n\rangle \in Q$ for all $n \in \mathbb{N}$, then we set $s(x) := \infty$. Note that for $(Q, \kappa) = p$-HALT and every nondeterministic Turing machine we have $s(\mathbb{M}) = t_{\mathbb{M}}(\lambda)$. Hence, the following lemma generalizes Lemma 5. As its proof runs along the same lines we omit it here.

Lemma 27. *Let (Q, κ) be slicewise downward monotone. If there is an algorithm $\mathbb{A}$ accepting (Q, κ) such that for all instances $\langle x, 1^n\rangle$ with $s(x) = \infty$ we have $t_{\mathbb{A}}(\langle x, 1^n\rangle) = n^{f(|M|)}$ for some function f, then $(Q, \kappa) \in \mathrm{XP}_{\mathrm{uni}}$.*

With this lemma we show the following generalization of Lemma 11, which will be used in the next section.

Lemma 28. *Let (Q, κ) be slicewise downward monotone. If* LIST(Q)*, then* $(Q, \kappa) \in \mathrm{XP}_{\mathrm{uni}}$.

Proof. Let $\mathbb{L}$ be a listing of the subsets in P of Q by polynomial time Turing machines. As for every $\langle x, 1^n\rangle \in Q$, the set $\{\langle x, 1^n\rangle\}$ is a subset in P of Q, the following algorithm $\mathbb{A}$ accepts Q:

```
A   // x ∈ Σ* and 1^n with n ∈ N

1.  ℓ ← 1
2.  compute the ℓth machine listed by L
3.          simulate it on input ⟨x, 1^n⟩
4.                  if it accepts then accept
5.  ℓ ← ℓ + 1
6.  goto 2.
```

We want to show that $\mathbb{A}$ runs in time polynomial in n for fixed x with $s(x) = \infty$. Then our claim follows from Lemma 27.

If $s(x) = \infty$, then $\{\langle x, 1^n\rangle \mid n \in \mathbb{N}\}$ is a subset in P of HALT. Hence, there is a machine listed by $\mathbb{L}$, say the ℓ_0th one, that decides this set. Then Lines 2–4 (for $\ell = \ell_0$) show that the running time of $\mathbb{A}$ is polynomially bounded in n. □

8. The length of first-order proofs and p-Halt

By the undecidability of first-order logic we know that there is no computable bound on the length of shortest proofs of valid sentences of first-order logic.[3] Mathematicians' experience seems to indicate that various valid sentences φ of first-order logic only have quite long proofs, say, proofs superpolynomial in $|\varphi|$. How hard is it to decide whether such a hard valid sentence has a proof of a length less than a given bound? Corollary 32 will show that this problem is not decidable in polynomial time if p-HALT $\notin \mathrm{XP}_{\mathrm{uni}}$. First we have to make precise the preceding question. By "hard valid sentences" we mean valid sentences like the Four Color Theorem or Fermat's Last Theorem, but also statements like $\mathrm{P} \neq \mathrm{NP}$ or the Riemann Hypothesis. Of course, we do not know whether these last two statements are valid sentences; hence the following promise problem could be viewed as the appropriate precise version of our question (note that its promise is equivalent to assuming that either φ is not valid or that φ is valid and has no short proof). Let $\iota : \mathbb{N} \to \mathbb{N}$ be a nondecreasing, unbounded, and computable function.

PROMISE-EXP-GÖDEL$_\iota$
Instance: A first-order sentence φ having no proof of length $< |\varphi|^{\iota(|\varphi|)}$ and 1^n with $n \geq |\varphi|^{\iota(|\varphi|)}$.
Problem: Does every proof of φ have length $> n$?

One could also consider the following (plain) problem.

EXP-GÖDEL$_\iota$
Instance: A first-order sentence φ and 1^n with $n \geq |\varphi|^{\iota(|\varphi|)}$.
Problem: Does every proof of φ have length $> n$?

Clearly, EXP-GÖDEL$_\iota$ is in coNP; Buhrman and Hitchcock [2] have shown that sparse problems are not coNP-hard unless the polynomial hierarchy collapses. This implies (cf. [8]):

Lemma 29. *Assume that the polynomial hierarchy does not collapse. Then the problems* PROMISE-EXP-GÖDEL$_\iota$ *and* EXP-GÖDEL$_\iota$ *are not* coNP*-hard (for the problem* PROMISE-EXP-GÖDEL$_\iota$ *this means that the set of instances of the problem that satisfy the promise and are positive instances is not* coNP*-hard).*

[3] Here we refer to any reasonable sound and complete proof calculus for first-order logic. However, we do not allow proof calculi, which admit all first-order instances of propositional tautologies as axioms (as then it would be difficult to recognize correct proofs if $\mathrm{P} \neq \mathrm{NP}$).

If the two problems are not coNP-hard, how do we convince ourselves that the two problems are intractable? For this purpose we consider a further slicewise downward monotone parameterized problem, namely

p-GÖDEL
Instance: A first-order sentence φ and 1^n with $n \in \mathbb{N}$.
Parameter: $|\varphi|$.
Problem: Does every proof of φ have a length $> n$?

We establish the following relationship to the previous problems:

Proposition 30 ([8]). *Let ι be a nondecreasing, unbounded, and computable function. If* PROMISE-EXP-GÖDEL$_\iota$ *or* EXP-GÖDEL$_\iota$ *is decidable in polynomial time, then* p-GÖDEL $\in$ FPT.

Proof. Assume that the algorithm $\mathbb{A}$ decides PROMISE-EXP-GÖDEL$_\iota$ in polynomial time. Then the following algorithm $\mathbb{G}$ shows that p-GÖDEL $\in$ FPT: Given an arbitrary instance $\langle \varphi, 1^n \rangle$ of p-GÖDEL, by brute force $\mathbb{G}$ checks whether a shortest proof of φ has length $s(\varphi) < |\varphi|^{\iota(|\varphi|)}$; if so, it checks whether $s(\varphi) > n$ or not and answers accordingly; otherwise, if $n < |\varphi|^{\iota(|\varphi|)}$, it accepts and if $n \geq |\varphi|^{\iota(|\varphi|)}$ (and hence the promise of PROMISE-EXP-GÖDEL$_\iota \in$ P is fulfilled), it simulates $\mathbb{A}$ on $\langle \varphi, 1^n \rangle$ and answers accordingly.

As the "brute force check" can be done in time $\leq f(|\varphi|)$ for a suitable computable f, the algorithm $\mathbb{G}$ witnesses that p-GÖDEL $\in$ FPT. □

We show:

Theorem 31. p-GÖDEL $\in \mathrm{XP}_{\mathrm{uni}}$ *if and only if* p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$.

From the two previous results we get:

Corollary 32. *If* p-HALT $\notin \mathrm{XP}_{\mathrm{uni}}$*, then the problems* PROMISE-EXP-GÖDEL$_\iota$ *and* EXP-GÖDEL$_\iota$ *are not polynomial time decidable.*

Proof of Theorem 31. Assume first that p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$. Then LIST(HALT) (by Corollary 12) and thus, by Lemma 13, LIST(GÖDEL), where GÖDEL denotes the classical problem underlying p-GÖDEL. Therefore, p-GÖDEL $\in \mathrm{XP}_{\mathrm{uni}}$ by Lemma 28.

Now assume that p-GÖDEL $\in \mathrm{XP}_{\mathrm{uni}}$. By standard means one can show (e.g., [8, Lemma 7]) that there exists a $d \in \mathbb{N}$ and a polynomial time algorithm that assigns to every nondeterministic Turing machine $\mathbb{M}$ a first-order sentence $\varphi_{\mathbb{M}}$ such that for $n \in \mathbb{N}$

$$\left\langle \varphi_{\mathbb{M}}, 1^{n^d} \right\rangle \in p\text{-GÖDEL} \Longrightarrow \langle \mathbb{M}, 1^n \rangle \in p\text{-HALT}. \tag{15}$$

Moreover,

$$\varphi_{\mathbb{M}} \text{ has a proof} \Longrightarrow \mathbb{M} \text{ accepts the empty input tape.} \tag{16}$$

Now assume that $\mathbb{G}$ is an algorithm that witnesses p-GÖDEL $\in \mathrm{XP}_{\mathrm{uni}}$. Let $d \in \mathbb{N}$ be as above. We present an algorithm $\mathbb{A}$ showing that p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$. On an instance $\langle \mathbb{M}, 1^n \rangle$ of p-HALT the algorithm $\mathbb{A}$ first computes $\varphi_{\mathbb{M}}$ and then runs two algorithms in parallel:

- an algorithm that on input $\mathbb{M}$, by brute force, computes $t_{\mathbb{M}}(\lambda)$ (the least n such that $\mathbb{M}$ on empty input tape has an accepting run of length n);
- the algorithm $\mathbb{G}$ on input $\left\langle \varphi_{\mathbb{M}}, 1^{n^d} \right\rangle$.

If the brute force algorithm halts outputting $t_{\mathbb{M}}(\lambda)$, then $\mathbb{A}$ checks whether $n < t_{\mathbb{M}}(\lambda)$, answers accordingly, and halts. Assume now that $\mathbb{G}$ halts. If $\mathbb{G}$ accepts $\left\langle \varphi_{\mathbb{M}}, 1^{n^d} \right\rangle$ $\big($and hence $\langle \mathbb{M}, 1^n \rangle \in p$-HALT by (15)$\big)$, then $\mathbb{A}$ accepts. If $\mathbb{G}$ rejects $\left\langle \varphi_{\mathbb{M}}, 1^{n^d} \right\rangle$, then $\mathbb{A}$ continues the simulation of the "brute force algorithm."

The algorithm $\mathbb{A}$ decides p-HALT: note that if $\mathbb{G}$ rejects $\left\langle \varphi_{\mathbb{M}}, 1^{n^d} \right\rangle$, then $\left\langle \varphi_{\mathbb{M}}, 1^{n^d} \right\rangle \notin p$-GÖDEL; in particular, $\varphi_{\mathbb{M}}$ has a proof, and therefore $\mathbb{M}$ accepts the empty input tape by (16), so that in this case the computation of the brute force algorithm eventually will output $t_{\mathbb{M}}(\lambda)$, and $\mathbb{A}$ will answer correctly.

We still have to show that for fixed nondeterministic Turing machine $\mathbb{M}$ the algorithm $\mathbb{A}$ runs in time polynomial in n on inputs of the form $\langle \mathbb{M}, 1^n \rangle$. We consider two cases.

$\mathbb{M}$ *halts on empty input tape*: Then an upper bound for the running time is given by the time that the brute force algorithm needs to compute $t_{\mathbb{M}}(\lambda)$ (and the time for the check whether $n < t_{\mathbb{M}}(\lambda)$); hence we have an upper bound of the form $n^{c_{\mathbb{M}}}$.

$\mathbb{M}$ *does not halt on empty input tape*: Then, by (16), we have $\left\langle \varphi_{\mathbb{M}}, 1^{n^d} \right\rangle \in p$-GÖDEL; hence an upper bound is given by the running time of $\mathbb{G}$ on input $\left\langle \varphi_{\mathbb{M}}, 1^{n^d} \right\rangle$. □

Similarly as we did for p-HALT at the end of Section 3, one can show that the answer to the question "p-GÖDEL $\in \mathrm{XP}_{\mathrm{uni}}$?" would be the same if we only would require for an instance $\langle \varphi, 1^n \rangle$ of p-GÖDEL that we get the correct answer if $s(\varphi)$, the length of a shortest proof of φ, is not near to n.

9. Hard sequences for algorithms and p-Halt

Recall that an algorithm $\mathbb{O}$ deciding a problem $Q \subseteq \Sigma^*$ is *almost optimal* if for every algorithm $\mathbb{A}$ deciding Q there is a polynomial $p_{\mathbb{A}} \in \mathbb{N}[X]$ such that for every $x \in Q$

$$t_{\mathbb{O}}(x) \leq p_{\mathbb{A}}(t_{\mathbb{A}}(x) + |x|). \tag{17}$$

Note that nothing is required for $x \notin Q$.

In [27] it was shown that

TAUT has an almost optimal algorithm $\iff$
there is a polynomially optimal propositional proof system.

Hence, by Theorem 7,

$$\textsc{Taut} \text{ has an almost optimal algorithm} \iff p\text{-}\textsc{Halt} \in \mathrm{XP}_{\mathrm{uni}}. \qquad (18)$$

Let $\mathbb{A}$ be an algorithm deciding a problem Q. A sequence $(x_s)_{s\in\mathbb{N}}$ of strings x_s in Q is *hard for* $\mathbb{A}$ if the function $1^s \mapsto x_s$ is computable in polynomial time and the sequence $\big(t_{\mathbb{A}}(x_s)\big)_{s\in\mathbb{N}}$ is not polynomially bounded in s. Clearly, if $\mathbb{A}$ is polynomial time, then $\mathbb{A}$ has no hard sequences. Furthermore, an almost optimal algorithm for Q has no hard sequences either. In fact, if $(x_s)_{s\in\mathbb{N}}$ is a hard sequence for an algorithm, then one can polynomially speed up it on $\{x_s \mid s \in \mathbb{N}\}$, so it cannot be almost optimal. We show:

Theorem 33. *Every algorithm deciding* Taut *has a hard sequence if and only if* p-Halt $\notin \mathrm{XP}_{\mathrm{uni}}$.

Proof. If p-Halt $\in \mathrm{XP}_{\mathrm{uni}}$, then Taut has an almost optimal algorithm $\big($by (18)$\big)$; we have just remarked that an almost optimal algorithm has no hard sequence.

It remains to show the implication from right to left. So assume that p-Halt $\notin \mathrm{XP}_{\mathrm{uni}}$. Then, by Lemma 5, for every algorithm $\mathbb{A}$ deciding p-Halt there is a nondeterministic machine $\mathbb{M}(\mathbb{A})$ with $t_{\mathbb{M}(\mathbb{A})}(\lambda) = \infty$ such that $\mathbb{A}$ restricted to instances of the form $\langle \mathbb{M}(\mathbb{A}), 1^n\rangle$ is not polynomial time.

Now let $\mathbb{C}$ be any algorithm deciding Taut and let $\mathbb{S}$ be a polynomial time reduction from Halt to Taut. Then the algorithm $\mathbb{C}\circ\mathbb{S}$ that on input x first computes $\mathbb{S}(x)$ and then simulates $\mathbb{C}$ on input $\mathbb{S}(x)$, decides Halt. By the previous observation, $\mathbb{C}\circ\mathbb{S}$ restricted to instances of the form $\langle \mathbb{M}(\mathbb{C}\circ\mathbb{S}), 1^n\rangle$ is not polynomial time; hence, $\mathbb{C}$ restricted to instances of the form $\mathbb{S}\big(\langle \mathbb{M}(\mathbb{C}\circ\mathbb{S}), 1^n\rangle\big)$ is not polynomial time. As $1^s \mapsto \mathbb{S}\big(\langle \mathbb{M}(\mathbb{C}\circ\mathbb{S}), 1^s\rangle\big)$ is computable in polynomial time, the sequence $\big(\mathbb{S}\big(\langle \mathbb{M}(\mathbb{C}\circ\mathbb{S}), 1^s\rangle\big)\big)_{s\in\mathbb{N}}$ is a hard sequence for $\mathbb{C}$. □

10. Summary, generalizations and extensions of the results

Summarizing we present a theorem which contains statements from different areas of theoretical computer science we have shown to be equivalent to p-Halt $\in \mathrm{XP}_{\mathrm{uni}}$.

Theorem 34. *The following are equivalent:*
(1) p-Halt $\in \mathrm{XP}_{\mathrm{uni}}$*;*
(2) There is a polynomially optimal propositional proof system;
(3) $\mathrm{LFP}_{\mathrm{inv}}$ *captures* P*;*
(4) p-Gödel $\in \mathrm{XP}_{\mathrm{uni}}$*;*
(5) There are algorithms deciding Taut *without hard sequences.*

In this expository article we only derived consequences of or statements equivalent to "p-Halt $\in \mathrm{XP}_{\mathrm{uni}}$." There are various extensions of these equivalences, which arise from questions like "what do p-Halt $\in$ XP, p-Halt $\in$ FPT, or p-Halt $\in \mathrm{FPT}_{\mathrm{uni}}$ mean for these related problems?" Further complexity classes have been considered in [12].

Here we report what the effect of changing membership of p-HALT in the class $\mathrm{XP}_{\mathrm{uni}}$ by a different class means for the equivalence (18). By this equivalence, p-HALT $\in \mathrm{XP}_{\mathrm{uni}}$ if and only if there is an algorithm $\mathbb{O}$ deciding TAUT such that for every further algorithm $\mathbb{A}$ deciding TAUT there is a polynomial $p_{\mathbb{A}} \in \mathbb{N}[X]$ such that for every tautology α

$$t_{\mathbb{O}}(\alpha) \leq p_{\mathbb{A}}(t_{\mathbb{A}}(\alpha) + |\alpha|). \tag{19}$$

The statement p-HALT $\in$ XP is equivalent to the existence of an effective procedure assigning to an algorithm $\mathbb{A}$ deciding TAUT a polynomial $p_{\mathbb{A}}$ satisfying (19). And p-HALT $\in \mathrm{FPT}_{\mathrm{uni}}$ means that for some d the polynomials $p_{\mathbb{A}}$ may be chosen of degree $\leq d$. If, in addition, they may be chosen effectively, this means that p-HALT $\in$ FPT.

References

1. Y. Aumann and Y. Dombb. Fixed structure complexity. In *Proceedings of the 3rd International Workshop on Parameterized and Exact Computation (IWPEC 2008)*, M. Grohe and R. Niedermeier (eds.), Lecture Notes in Computer Science 5018, 31–42, 2008.
2. H. Buhrman and J. M. Hitchcock. NP-hard sets are exponentially dense unless coNP $\subseteq$ NP/poly. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC'08)*, pp. 1-7, 2008. *Electronic Colloquium on Computational Complexity (ECCC'08)*, Report TR08-022, available at `http://eccc.hpi-web.de/eccc-local/Lists/TR-2008.html`
3. S. Buss, Y. Chen, J. Flum, S. Friedman, and M. Müller. Strong isomorphism reductions in complexity theory. To appear in *Jour. Symb. Logic.*
4. L. Cai, J. Chen, R. Downey, and M. Fellows On the parameterized complexity of short computation and factorization. *Archive for Mathematical Logic,* 36:321-337, 1997.
5. M. Cesati. The Turing way to parameterized complexity. *Journal of Computer and System Sciences,* 67:654–685, 2003.
6. M. Cesati and M. Di Ianni. Computation models for parameterized complexity. *Mathematicall Logical Quarterly,* 43:179–202, 1997.
7. A.K. Chandra, D. Kozen, and L.J. Stockmeyer, Alternation. In *Journal of the ACM*, 28:114–133, 1981. pages 77–90, 1977.
8. Y. Chen and J. Flum. On the complexity of Gödel's proof predicate. *The Journal of Symbolic Logic* 75, 239–254, 2009.
9. Y. Chen and J. Flum. A logic for PTIME and a parameterized halting problem. In *Fields of Logic and Computation*, Lecture Notes in Computer Science 6300, 251–276, 2010.
10. Y. Chen and J. Flum. On p-optimal proof systems and logics for PTIME. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP'10, Track B)*, Lecture Notes in Computer Science 6199, 321–332, 2010.
11. Y. Chen and J. Flum. On slicewise monotone parameterized problems and optimal proof systems for TAUT. In *Proceedings of the 19th EACSL Annual Conference in Computer Science Logic (CSL'10)*, Lecture Notes in Computer Science 6247, 200–214, 2010.

12. Y. Chen and J. Flum. Listings and logics. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS'11)*, 165–174, 2011.
13. S. Cook. The complexity of theorem proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151–158, 1971.
14. S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44:36–50, 1979.
15. R. Downey and M. Fellows. Fixed-parameter tractability and commpleteness III: Some structurl aspects of the W-hierarchy. In *Complexity Theory* (ed. K. Ambos-Spies et al.), 166-191, 1993
16. R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.
17. R. Downey. Private communication.
18. R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, 43–73, 1974.
19. J. Flum and M. Grohe. Fixed-parameter tractability, definability, and model checking. *SIAM Journal on Computing,* 31:113–145, 2001.
20. J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer-Verlag, 2006.
21. L. Fortnow and J. Grochow. Complexity classes of equivalence problems revisited, `arXiv:0907.4775v1` [cs.CC], 2009.
22. J. Hartmanis and L. Hemachandra. Complexity classes without machines: On complete languages for UP. *Theoretical Computer Science 58* , 129–142, 1988.
23. N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986.
24. J. Köbler and J. Messner. Complete problems for promise classes by optimal proof systems for test sets. In *Proceedings of the 13th IEEE Conference on Computational Complexity (CCC' 98)*, 132–140, 1998.
25. J. Köbler, J. Messner, and J. Torán. Optimal proof systems imply complete sets for promise classes. *Information and Computation*, 184:71–92, 2003.
26. W. Kowalczyk. Some connections between presentability of complexity classes and the power of formal systems of reasonning. In *Proceedings of Mathematical Foundations of Computer Science 1984 (MFCS'84)*, Lecture Notes in Computer Science 176, 364–369, 1984.
27. J. Krajíček and P. Pudlák. Propositional proof systems, the consistency of first order theories and the complexity of computations. *The Journal of Symbolic Logic*, 54:1063–1088, 1989.
28. L. Levin. Universal search problems. *Problems of Information Transmission*, 9(3):265-266, 1973. In Russian; English translation in: B.A.Trakhtenbrot. A survey of Russian approaches to perebor (brute-force search) algorithms. *Annals of the History of Computing*, 6(4):384-400, 1984.
29. J. Messner and J. Torán. Optimal proof systems for propositional logic and complete sets. In *Proceedings of the 15th Annual Symposium of Theoretical Aspects of Computer Science (STACS'98)*, Lecture Notes in Computer Science 1373, 477–487, 1998.
30. H. Monroe. Speedup for natural problems and noncomputability. *Theoretical Computer Science*, 412(4-5):478–481, 2011
31. A. Nash, J. Remmel, and V. Vianu. PTIME queries revisited. In *Proceedings of the 10th International Conference on Database Theory (ICDT'05)*, Lecture Notes in Computer Science 3363, 274–288, 2005.
32. Z. Sadowski. On an optimal propositional proof system and the structure of easy subsets. *Theoretical Computer Science*, 288(1):181–193, 2002.

33. A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Society*, 2:230–265, 1936.
34. M.Y. Vardi. The complexity of relational query languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC'82)*, 137–146, 1982.