Succinctness as a source of complexity in logical formalisms

Georg Gottlob*, Nicola Leone, Helmut Veith

Institut für Informationssysteme, Technische Universität Wien, A-1040 Wien, Paniglgasse 16, Austria

Received 2 December 1997, received in revised form 24 August 1998
Communicated by S.N. Artemov

Abstract

The often observed complexity gap between the expressiveness of a logical formalism and its exponentially harder expression complexity is proven for all logical formalisms which satisfy natural closure conditions. The expression complexity of the prefix classes of second-order logic can thus be located in the corresponding classes of the weak exponential hierarchies; further results about expression complexity in database theory, logic programming, nonmonotonic reasoning, first-order logic with Henkin quantifiers and default logic are concluded. The proof method illustrates the significance of quantifier-free interpretations in descriptive complexity theory. © 1999 Elsevier Science B.V. All rights reserved.

AMS classification: 03C13; 03D15; 03C85

Keywords: Expression complexity; Descriptive complexity theory; Query languages; Succinct problems; Quantifier-free reductions; Second order logic

1. Introduction

In [49], Vardi introduced data complexity and expression complexity as complexity measures for query languages over relational databases, and thus, for logics over finite structures. Data complexity measures the complexity of checking a fixed property over varying structures; expression complexity on the other hand measures the complexity of checking varying expressions over a fixed structure. The data complexity of a logic is uniquely determined by its expressive power. Thus, two languages with the same expressive power will always have the same data complexity. Hence, data complexity is syntax-independent, i.e., merely depends on expressive power.

Expression complexity however depends on the syntax of the language, and therefore, it is in general not possible to determine the expression complexity from the expressive

* Corresponding author. E-mail: gottlob@dbai.tuwien.ac.at.
power. Indeed, both the syntax and the semantics of a logical language influence its expression complexity.

Nevertheless, the typical behaviour of the expression complexity was often observed [12, 49] to resemble the following pattern:

**Observation.** If the expressive power of a language $\mathcal{L}$ captures a complexity class $C$, then the expression complexity of $\mathcal{L}$ is exponentially harder than $C$.

The main result of this paper shows that all languages satisfying simple uniformity properties indeed match this pattern. Intuitively, we say that a language is uniformly closed if the following operations are LOGSPACE-computable:

- replacing input relations by positive Boolean combinations of input relations,
- replacing variables by tuples of variables.

Thus, for complexity classes $C$ in the polynomial hierarchy (and similarly, for syntactic complexity classes [35] in general) the following holds:

**Theorem.** If $\mathcal{L}$ captures $C$, and $\mathcal{L}$ is uniformly closed, then $\mathcal{L}$ has expression complexity at least $E(C)$.

Here, $E(C)$ is the class which corresponds to $C$ in the weak exponential hierarchy, cf. Section 2.4.

For the proof of our main results, we require techniques from finite model theory and structural complexity; to put it in a nutshell, we shall use that every syntactic complexity class $C$ [35] is characterized by a leaf language [5], and every leaf language definable problem in turn is characterized by a succinct circuit problem up to quantifier-free interpretations [52]. Problems complete under quantifier-free interpretations then can be used as base classes for succinct formula problems [51]. Since succinct formula problems are classes of formulas, it is possible to derive the conditions for uniform closure which facilitate the representation of succinct formula problems within logical languages. Thus, the well-known complexity of the succinct formula problem [51] is a lower bound for the expression complexity of all uniformly closed languages capturing $C$.

**Combined complexity** [49] measures the complexity of all pairs of structures and satisfied expressions. It is clear that the lower bounds for expression complexity are also lower bounds for combined complexity. With respect to LOGSPACE reductions, expression complexity and combined complexity are usually equivalent. Typically, however, there are better upper bounds for expression complexity than for combined complexity to the effect that expression complexity can be sharpened to completeness in complexity classes like $\text{ETIME}$ (also known as $\text{ETIME}$) which are not closed under LOGSPACE reductions (and therefore not syntactical).

The intuitive reason is that the complexity of evaluating a formula $\phi$ over a structure $\mathcal{U}$ can often be described by a polynomial of the form $n^m$, where $n$ is linear in $|\mathcal{U}|$ and $m$ is linear in $|\phi|$. 
For data complexity, this amounts to $n^{\text{const}}$, while for expression complexity we obtain a bound of $\text{const}^m$, as in the definition of the complexity class $E$ and its derived classes. For combined complexity however, $n^m$ obtains the form $\text{const}^{m\log n}$ as in the definition of $\text{EXP}$.

In [S], a uniform method to prove computational lower bounds for decidable theories such as Presburger Arithmetic or the first-order theory of finite cyclic groups has been presented. Our method resembles the work of Compton and Henson [8] on an abstract level in the sense that both papers exploit interpretations as a means to obtain lower bounds. However, both the methods and the aims of Compton and Henson [8] are different from ours, since (1) we consider the complexity of the satisfaction relation over finite structures rather than the complexity of logical theories, and (2) we use quantifier-free interpretations not to reduce theories, but as a combinatorial tool whose intrinsic complexity yields the gap between expressiveness and expression complexity.

Our starting point will be the proof that the expression complexity of $\sum_k^1$ formulae is $\text{NE}^{\text{L}^{\text{Space}}} \text{-complete under LOGSPACE reductions. Using this result and its paradigmatic methodology, corresponding results for } \Pi_k^1 \text{ and } \Theta_k^1 \text{ follow.}$

Henkin quantifiers, introduced by Henkin [25], are important in both model theory and theoretical linguistics [33, 41, 48]. The expressibility of Henkin quantifiers over finite models has been recently characterized [3, 18]. We show that first-order logic with Henkin quantifiers has expression complexity $\text{LinSpace}^{\text{NP}}$.

Infinitary logic $\mathcal{L}_{\infty,\omega}$ has attracted interest in finite model theory [9], since it provides a common framework for many important finitary logics, such as transitive closure logic. Here, we show that transitive closure logic has expression complexity $\text{NLinSpace}$.

Default logic is one of the most used languages in Artificial Intelligence. It was proposed by Reiter [39] as a language incorporating default conclusions; such conclusions are drawn unless counter evidence is available. In [6], default logic was proposed as a database query language, and it was shown that it is able to express relevant database queries that cannot be modelled in traditional languages. We show that first-order default logic, under credulous inference modality, has expression complexity $\text{NE}^{\text{NP}}$ by an embedding of skolemized $\Sigma_2^1$ into first-order default logic.
Given the evidence of facts in the outer world, abduction aims at concluding previous facts from which an underlying theory infers the known facts. That is, in pulling modus ponens upward down, abduction strives to restore the cause from the consequence. Abduction was investigated thoroughly by Pierce [37] and has found several applications in computer science [26, 30, 31, 38]. The complexity of propositional abduction was settled in [10] and the data complexity of abduction over logic programs was solved in [11]. In this paper, we apply our results to expression complexity problems related to abduction.

1.1. Structure of the paper

Section 2 briefly introduces basic notions from finite model theory and complexity theory. The weak exponential hierarchies are shortly explored and an exact definition of expression complexity is provided. Since the aim of this paper is to stress the logical aspects of expression complexity, we defer the proofs about the weak exponential hierarchies to the appendix.

Section 3 introduces the new notion of convex problems, and investigates their appropriateness for succinct representation. In Section 4, we apply succinct problems to settle the expression complexity for the levels of second order logic. This proof is worked out in detail which allows us in Section 5 to identify those syntactic properties of logical languages which are sufficient to show hardness of the expression complexity. The results about Henkin logic, transitive closure logic, default logic and abduction are obtained in Section 6 by applying or extending the new methods.

2. Preliminaries

2.1. Notation

Given a binary string $w$, $|w|$ denotes its length, and $\text{int}(w)$ denotes the positive integer whose binary representation is $w$. Given an integer number $n$, $\text{bin}(n)$ denotes its binary representation. Tuples of variables are denoted by boldface letters $x$.

We assume some familiarity with basic complexity classes such as LOGSPACE (or, for brevity $L$), NLOGSPACE, PTIME, NP, LinSpace, and PSPACE. For an introduction, consult [35]. We shall need two types of reductions: LOGSPACE reductions which we denote by $\leq_{\text{LOGSPACE}}$, and quantifier-free reductions (to be defined in Section 2.2) which we denote by $\leq_{\text{qf}}$.

2.2. Descriptive complexity

A signature is a sequence $\tau = (P_1^{a_1}, \ldots, P_k^{a_k})$ of relation symbols with associated arities $a_1, \ldots, a_k$. A finite structure over $\tau$ is a tuple $\mathfrak{U} = (n, P_1^{a_1}, \ldots, P_k^{a_k})$, where $n = \{0, \ldots,$
n - 1}, and \( P_i \subseteq n^{a_i} \) is called the universe (or domain) of \( \mathcal{A} \), and denoted \(|\mathcal{A}|\).

The set of all finite structures over \( \tau \) is denoted by \( \text{Struct}(\tau) \). Let \( \mathcal{A}, \mathcal{B} \in \text{Struct}(\tau) \), s.t. \(|\mathcal{A}| = |\mathcal{B}|\). Then \( \mathcal{A} \subseteq \mathcal{B} \) if \( P_i \subseteq P_i^{a_i} \) for \( 1 \leq i \leq k \). A computational problem over signature \( \tau \) is a set \( \Pi \subseteq \text{Struct}(\tau) \), s.t. \( \Pi \) is closed under isomorphisms. \( \Pi \) is monotone, if \( \mathcal{A} \subseteq \mathcal{B} \) and \( \mathcal{A} \in \Pi \) implies \( \mathcal{B} \in \Pi \). \( \tau^{(l)} = (P_1^{a_1}, \ldots, P_k^{a_k}) \) is called the \( l \)-ary variant (or vectorisation) of \( \tau \). For a problem \( T \), let \( T^{(l)} \) denote the problem \( T \) over \( l \)-tuples, i.e. over universe \( n \times \cdots \times n \). Due to closure under isomorphisms, \( l \)-tuples can be understood as numbers from \( \{0, \ldots, n^l - 1\} \). Formally, \( T^{(l)} \subseteq \text{Struct}(\tau^{(l)}) = \text{Struct}(\tau)^{(l)} \).

Let \( \mathcal{A} \) be a structure, and \( U \) be a subset of \(|\mathcal{A}|\). Then the restriction of \( \mathcal{A} \) to \( U \), in symbols \( \mathcal{A} \upharpoonright U \), is the structure \( (U, P_1^{a_1} \cap U^{a_1}, \ldots, P_k^{a_k} \cap U^{a_k}) \). First order logic \( \text{FO}(\tau) \) is the language of all first-order sentences over signature \( \tau \) with logical predicates for equality = and successor \( s(_, -) \), and two constants \( 0, \text{max} \) denoting the minimal and maximal element with respect to the successor relation. (We shall see later that for expression complexity, one can in fact always suppose without loss of generality that all domains are ordered, i.e., equipped with either a linear order or a successor relation.) A quantifier-free formula is positive, if contains only the propositional connectives \( \land \) and \( \lor \). The set of all finite models of a formula \( \Psi \) is denoted by \( \text{Mod}_c(\Psi) \). Two formulae \( \Psi \) and \( \Phi \) are equivalent if they have the same finite models, i.e., \( \text{Mod}_c(\Psi) = \text{Mod}_c(\Phi) \). If a logic \( \mathcal{L} \) expresses exactly those sets of structures which are recognized by Turing machines of a computational class \( C \), we say that \( \mathcal{L} \) captures \( C \) and write \( \mathcal{L} = C \).

Let \( \phi(x_1, \ldots, x_n) \) be a formula with free variables \( x_1, \ldots, x_n \), and let \( \mathcal{A} \) be a structure. Then \( \phi^{\mathcal{A}} \) denotes the \( n \)-ary relation \( \{(d_1, \ldots, d_n) \mid \mathcal{A} \vDash \phi(d_1, \ldots, d_n)\} \).

Given signatures \( \tau, \sigma \) and a natural number \( k \), a \( k \)-ary interpretation \( I \) of \( \tau \) into \( \sigma \) is a definition of the \( \sigma^{(k)} \) relations in terms of \( \tau \). For a structure \( \mathcal{A} \in \text{Struct}(\tau) \), \( I(\mathcal{A}) \) denotes the structure over \( \sigma^{(k)} \) which is defined by \( I \). Let \( T \subseteq \text{Struct}(\tau), \ S \subseteq \text{Struct}(\sigma) \) be problems, and let \( \mathcal{L} \) be a syntactic fragment of first-order logic. We say that \( T \) is \( \mathcal{L} \)-reducible to \( S \) if there exist an interpretation \( I \) of \( \tau \) into \( \sigma \), s.t \( I \) is described by \( \mathcal{L} \)-formulas, and for all \( \mathcal{A} \in \text{Struct}(\tau) \), \( \mathcal{A} \in T \) iff \( I(\mathcal{A}) \in S^{(k)} \), with \( k \) being the arity of \( I \). By restricting the logic \( \mathcal{L} \) for the interpretations we obtain low-level reductions: A quantifier-free reduction [27] is a reduction whose defining formulas are quantifier-free.

Let \( \phi \) be a formula over signature \( \tau \), and let \( R_1, \ldots, R_r \) be a sequence of relational symbols from \( \tau \) with associated arities \( b_1, \ldots, b_r \). Suppose that for each \( R_i \) we have a quantifier-free defining formula \( \phi_i(x_1, \ldots, x_{b_i}) \) over some signature \( \sigma_i \). Then

\[
\rho[R_1/\phi_1, \ldots, R_r/\phi_r]
\]

is the formula obtained from replacing each occurrence of an \( R_i(x) \) in \( \rho \) by its defining formula \( \phi_i(x) \).
2.3. Expression complexity

In [49] Vardi defined three important notions of complexity for database query languages.

- **Data complexity** is the complexity of recognizing the models of a given expression.
- **Expression complexity** is the complexity of recognizing the expressions which are satisfied by a given structure.
- **Combined complexity** is the complexity of recognizing the satisfaction relation on finite structures.

**Example 1.** We consider different situations where complexity is of interest to the database theorist. Suppose that a street map is described by a binary adjacency relation.

- **Data complexity.** What is the complexity of recognizing if street maps are connected?
- **Expression complexity.** Given a map of Moscow, what is the complexity of the queries a mathematician can ask in transitive closure logic?
- **Combined complexity.** What is the overall complexity, if the mathematician uses transitive closure logic on arbitrary maps?

This definition easily extends to logical languages in general. Thus, when we speak of a logic, we have in mind a large number of logical formalisms, in particular model-theoretic logics, database query languages, and knowledge representation languages from Artificial Intelligence. Previous results about both data and expression complexity for various languages can be found in [6, 12, 13, 49]. Let us now turn to a formal definition.

Let $\tau$ be a relational vocabulary, and $\mathcal{L}(\tau)$ be a language with vocabulary $\tau$. We suppose that formulas have canonic representations as strings, which come with their syntax, and that finite structures can be represented e.g. by the characteristic sequences of the domain relations. Thus, the notion of length is well-defined, and algorithms can be used to modify formulas and structures. Most reasonable syntactic variations of the logics do not affect complexity issues; in Section 5.3, Example 7, we shall however consider an artificial language where formulas are represented as strings in a non-standard way.

To formally define expression complexity and combined complexity, we can now consider the sets

$$E_{\mathcal{U}, \mathcal{L}(\tau)} = \{ \phi \in \mathcal{L}(\tau) \mid \mathcal{U} \models \phi \}$$

and

$$C_{\mathcal{L}(\tau)} = \{ (\mathcal{U}, \phi) \in \text{Struct}(\tau) \times \mathcal{L}(\tau) \mid \mathcal{U} \models \phi \}$$

**Definition 1.** Let $D$ be a complexity class, and $\mathcal{L}$ be a logic. $\mathcal{L}$ has

- **Expression complexity $D$** if
(i) there exist $\mathcal{U}, \tau$, such that $E_{\mathcal{U}, \mathcal{P}(\tau)}$ is $D$-complete under LOGSPACE reductions, and,
(ii) for all $\mathcal{U}$ and $\tau$, $E_{\mathcal{U}, \mathcal{P}(\tau)}$ is contained in $D$.

- Combined complexity $D$, if
  (i) there exists $\tau$, such that $C_{\mathcal{P}(\tau)}$ is $D$-complete under LOGSPACE reductions, and,
  (ii) for all $\tau$, $C_{\mathcal{P}(\tau)}$ is contained in $D$.

In the proofs, we shall often refer to clause (i) as hardness, and to clause (ii) as membership. If only clause (i) is fulfilled, we say that $L$ has expression complexity (resp. combined complexity) at least $D$.

It follows immediately that the expression complexity is always a lower bound of the combined complexity. Since in most cases both notions coincide, it will then be sufficient to prove membership for $C_{\mathcal{P}(\tau)}$, and hardness for some $E_{\mathcal{U}, \mathcal{P}(\tau)}$.

**Remark.** Since expression complexity is measured with respect to arbitrary fixed structures, we may in the hardness proofs without loss of generality assume that the structure under consideration is ordered.

### 2.4. The weak exponential hierarchies

In this section, we recall the definition of the polynomial hierarchy $PH$ and its exponential counterparts, the weak exponential hierarchies $EH$ and $ExpH$.

Recall that $PSPACE$ is the class of problems decidable in polynomial space, and $LinSpace$ is the class of problems decidable in linear space. Note that $LinSpace$ is not closed under polynomial time reductions.

An Oracle Turing Machine is a Turing Machine (deterministic or nondeterministic) which is equipped with an infinite oracle set $C$ and a distinguished oracle tape. In one step, the machine can decide if the word on the oracle tape belongs to $C$. $P^C$ (resp. $NP^C$) are the classes of decision problems solvable by some deterministic (resp. nondeterministic) Turing machine in polynomial time with an oracle set from $C$. The size of the oracle tape is unrestricted, hence the oracle tape content is potentially as large as the running time. (This oracle model is referred to as the Ladner–Lynch model [34].) The polynomial hierarchy ($PH$) is the natural closure of $NP$ under oracle computation [16, 35]. The classes $\Delta^p_k$, $\Sigma^p_k$, and $\Pi^p_k$ of the polynomial time hierarchy ($PH$) are defined as follows:

$$\Delta^p_0 = \Sigma^p_0 = \Pi^p_0 = \Theta^p_0 = P$$

and for all $k \geq 0$,

$$\Delta^p_{k+1} = P^{\Sigma^p_k}, \quad \Sigma^p_{k+1} = NP^{\Sigma^p_k},$$

$$\Pi^p_{k+1} = co-\Sigma^p_{k+1}, \quad \Theta^p_{k+1} = L^{\Sigma^p_k}.$$

The polynomial hierarchy $PH$ is defined as $\bigcup_{k=0}^{\infty} \Sigma^p_k$. 

Two computational classes, namely,

$$E = \bigcup_{k} \text{DTIME}[2^{kn}] \quad \text{and} \quad \text{EXP} = \bigcup_{k} \text{DTIME}[2^{n^k}]$$

are commonly referred to as exponential time. E is more common in structural complexity theory [23, 40] because it is defined by the natural bound $\text{DTIME}[(2^n)^k] = \text{DTIME}[2^{kn}]$ obtained by applying a polynomial time algorithm to an instance of size $2^n$. On the other hand, EXP is closed under polynomial time reductions, while E is not.

Starting from E and EXP, two different exponential hierarchies have been defined: the weak exponential hierarchy (EH) [21], composed of the levels E, NE, E$^{NP}$, NE$^{NP}$, etc.; and, the weak EXP hierarchy (ExpH) [24], consisting of EXP, NEXP, EXP$^{NP}$, NEXP$^{NP}$, etc. Both weak exponential hierarchies have characterizations in terms of the alternating Turing machines of Chandra et al. [7]. In particular, EH models alternating Turing machines with a bounded number of $2^{kn}$-sized alternation blocks, and ExpH models alternating Turing machines with a bounded number of $2^{n^k}$-sized alternation blocks. These hierarchies are referred to as weak in opposition to the strong exponential hierarchy — E, NE, P$^{NE}$, NP$^{NE}$, etc. Hemachandra [24] has proven that the strong exponential hierarchy collapses to P$^{NE}$, its $\Delta_2$ level [24]. We refine the exponential hierarchies by exhibiting natural intermediate classes of EH corresponding to the $\Theta_k^P$ levels of PH; later we shall see that there exist natural complete problems for these classes.

Thus, the weak exponential time hierarchy (EH) is a hierarchy symmetric to the polynomial one, where E and NE play the roles of P and NP, respectively:

$$E_0^P = F.\Sigma_0^P = E.\Pi_0^P = F.$$ 

and for all $k \geq 0$,

$$E_{k+1}^P = E_{k+1}^P, \quad E_{k+1}^P = \text{co-}E_{k+1}^P,$$

$$E_{k+1}^P = \Theta_{k+1}^P = \text{LinSpace}^{\Sigma_{k+1}^P}.$$ 

EH is equal to $\bigcup_{k=0}^{\infty} E_{k+1}^P$. Note that our definition of $E_{k+1}^P$ is new and will be justified by the results to follow.

A similar exponential hierarchy, called weak EXP hierarchy (ExpH) [24], is defined considering EXP, NEXP and PSPACE as the base classes of the hierarchy in place of E, NE, and LinSpace respectively.

Let $C$ be a class in PH, then $F(C)$ and $Exp(C)$ denote the classes at the corresponding level of the exponential hierarchies. From the Alternating Time Hierarchy Theorem [29] it immediately follows that $C$ is a proper subclass of $E(C)$ and $Exp(C)$. In the appendix, we prove several properties of the weak exponential hierarchies. In particular, completeness in EH implies completeness in ExpH:
Theorem 19 (Appendix). If a problem \( A \) is complete for \( E(C) \) under LOGSPACE reductions, then it is also complete for \( \text{Exp}(C) \) under LOGSPACE reductions.

Thus, \( E(C) \)-completeness is very close to \( \text{Exp}(C) \) completeness from the point of view of complexity theory; in particular, it follows that a problem is hard for \( E(C) \) iff it is hard for \( \text{Exp}(C) \). However, the distinction between \( E(C) \) and \( \text{Exp}(C) \) still is meaningful, because an \( E(C) \) algorithm may be considerably faster than an \( \text{Exp}(C) \) algorithm. Moreover, as mentioned in the introduction, the distinction between \( E(C) \) and \( \text{Exp}(C) \) often marks the natural border between the expression complexity of a language and its combined complexity.

A mapping due to Balcazar et al. [2] allows to reduce the complexity of problems in EH to PH by an exponential blow-up of the instance size.

Definition 2 (Balcazar et al. [2]). Let \( A \) be a decisional problem. The long version \( \text{long}(A) \) of \( A \) is the following problem:

\[
\text{long}(A) = \{ w \in \{0,1\}^* \mid \text{bin}(|w|) \in 1A \}
\]

where \( 1A = \{ w \in \{0,1\}^* \mid w \in A \} \).

Thus, for each instance \( w \in A \), we include the whole \( \{0,1\}^{\text{int}(1w)} \) in \( \text{long}(A) \). Therefore,

\[
\text{long}(A) = \bigcup_{w \in A} \{0,1\}^{\text{int}(1w)}.
\]

Since membership in \( \text{long}(A) \) is determined by an algorithm whose running time is measured with respect to an exponentially larger instance length, complexity is expected to get exponentially lower. For a class \( C \), \( \text{long}(C) \) denotes \( \{ \text{long}(A) \mid A \in C \} \).

For the complexity upgrading technique, we shall be interested in triples \( C, X, Y \) of complexity classes, such that \( \text{long}(X) \subseteq C \), and \( Y \) is the closure of \( X \) under LOGSPACE reductions.

Paradigmatic examples of such triples are given by \( C, E(C), \text{Exp}(C) \) where \( C \) is a class of the polynomial hierarchy.

Theorem 17 (Appendix). For all classes \( C \) in the polynomial hierarchy, it holds that

\[
\text{long}(A) \in C \iff A \in E(C).
\]

Fig. 2 summarizes some results about exponential downgrading by \( \text{long} \).
3. The easy way to obtain hard problems

In this section we describe a method to obtain complete problems which can be used in hardness proofs for expression complexity in a possibly easy way. First, we define a class of problems (convex problems) which can without loss of generality be restricted to such instances where the domain size is a power of 2. Then we show that such problems give rise to a particularly easy and logically appealing form of succinct problems which can be used as starting point for hardness proofs of expression complexity.

3.1. Succinct data representation

It is well-known that the computational complexity of a problem depends on how problem instances are encoded as inputs to a Turing machine. Galperin and Wigderson [15] investigated problems whose instances are not given straightforward, but are themselves encoded by Boolean circuits: The succinct version $s(A)$ of $A$ is the class of Boolean circuits which describe true instances of $A$. They show that such a succinct representation of a problem can be exponentially harder. Papadimitriou and Yannakakis [36] and Balcazar et al. [1, 2] developed a general upgrading theorem deriving completeness of the succinct problem in a complexity class $C$ under polynomial time reductions from completeness in an exponentially easier class under LOGTIME reductions where problems are given in their usual representation. In [11] the assumption was relaxed to POLYLOGTIME reductions, and in [52] the conclusion was sharpened to completeness under monotone projection reductions, a very restricted form of quantifier-free reductions.

Our attention in this paper will be mainly focused on a restricted form of succinct representation: A propositional problem is a problem whose instances are defined by Boolean formulas.

Example 2. Consider a propositional formula $\phi(x_1, \ldots, x_{2n})$. Then $\phi$ describes a graph with $2^n$ vertices $0, \ldots, 2^n - 1$ whose adjacency matrix is defined by $\phi$: to compute its entry for $(a, b)$, we assign the bits of the binary notation of $a$ and $b$ to $x_1, \ldots, x_n$ and $x_{n+1}, \ldots, x_{2n}$, respectively.

For a problem $A$, let $p(A)$ denote the propositional version of $A$. First results about propositional problems were obtained by Wagner [53]. He showed that several tractable
combinatorial problems turn into complete problems at different levels of the polynomial hierarchy under propositional representation. In [51], a general upgrading theorem like for succinct circuit problems was developed. It derives completeness under LOGSPACE reductions from completeness under quantifier-free reductions. Since we do not need those results in their full generality, we defer a formal definition of p(A) to Section 3.

A syntactic complexity class [35] is a complexity class which has a complete problem under PTIME reductions and is closed under PTIME reductions [5]. In [4, 52] it was shown that the syntactic complexity classes all contain a problem of the form s(A) which is complete under quantifier-free reductions. This, together with the fact that the formulas for propositional representation (like \( \phi \) in Example 2) can be expressed within every reasonable logic, will be the key to our upgrading results.

### 3.2. Convex problems

For a structure \( \mathcal{A} \in \text{Struct}(\tau) \), the Harzf Graph \( H(\mathcal{A}) \) [20] is the undirected binary graph \((\mathcal{A}, E)\) where

\[
E = \{(a, b) \mid a \neq b, R \in \tau, \mathcal{A} \models R(\ldots, a, \ldots, b, \ldots)\}.
\]

The active domain \( \text{dom}(\mathcal{A}) \) is the set of non-isolated vertices in \( H(\mathcal{A}) \).

**Definition 3.** A problem \( \Pi \) is convex if for all structures \( \mathcal{A} \),

\[
\mathcal{A} \in \Pi \iff \mathcal{A} \upharpoonright \text{dom}(\mathcal{A}) \in \Pi.
\]

It is easy to see that a problem is convex iff its characteristic property does not change when new isolated domain elements are added to a structure.

**Example 3.** 3-colorability of graphs is convex because adding isolated vertices does not interfere with 3-colorability. Hamiltonicity is not convex because a graph with an isolated vertex can never be Hamiltonian.

**Lemma 1.** Let \( \Pi \) be convex, and \( d \geq 1 \). Then \( \Pi \) and \( \Pi^{(d)} \) are mutually reducible by quantifier-free reductions.

**Proof.** \( \Pi^{(d)} \) is reducible to \( \Pi \) by definition. As for the other direction, \( \Pi \) can be embedded into \( \Pi^{(d)} \) by mapping \( x \) to the tuple \((0, \ldots, 0, x)\) because \( \Pi \) is convex. □

The main reason to study convex problems is the fact that we can always suppose without loss of generality, that the domain size of an instance under investigation is of the form \( 2^k \); suppose an instance \( \mathcal{A} \) of a convex problem \( \Pi \) has size \( d \) s.t. \( 2^k - 1 < d < 2^k \). Then we can add \( 2^k - d \) new domain elements, i.e., a pure set of size \( 2^k - d \), without
changing membership in \( \Pi \). This property of convex problems is similar to closure under padding in structural complexity theory.

Fortunately, using convex problems is no restriction in general, as the following theorem shows. We say that a complexity class is syntactic [35] if it is closed under polynomial time reductions, and contains a complete problem under polynomial time reductions.

**Theorem 1.** Let \( C \) be a syntactic complexity class. Then there exists a problem \( \Pi \), such that

1. \( \Pi \) is \( C \)-complete under projection reductions, and
2. \( \Pi \) is convex.

**Proof.** In [51, Corollary 6], it was shown that for every syntactic complexity class there exists a succinct circuit problem \( \Pi \) (which is obtained using the leaf language of \( C \)) which is complete for \( C \) under projection reductions.

\( \Pi \) is defined over signature \( \zeta = (\land^3, \lor^3, \neg^2, \triangleleft^1, \square^1, \top^1, \bot^1) \), where \( \land \) and \( \lor \) denote conjunction and disjunction respectively, \( \neg \) denotes negation, \( \triangleleft \) the input gates, and \( \square \) the output gate. \( \top \) and \( \bot \) denote the constant one and zero gates, respectively. Let for example \( \mathcal{C} \in \text{Struct}(\zeta) \). Then \( \mathcal{C} \models \land(8,5,3) \) means that the circuit encoded by \( \mathcal{C} \) computes gate number 8 as the conjunction of gates 5 and 3. A structure \( \mathcal{U} \in \text{Struct}(\zeta) \) represents a boolean circuit if it contains an output gate and each gate is either an input gate or is computed from other gates, such that there are no cycles.

The definition in [51] of circuit problems does not require that all domain elements in fact carry gates. Since membership in \( \Pi \) depends only on the function computed by the circuit, a new domain element can be added without changing membership in \( \Pi \). We conclude that \( \Pi \) is convex. \( \square \)

**Remark.** Using the notion of implicit circuits [51], it is even possible to strengthen completeness to monotone projection reductions. Since the definition of implicit circuits is less straightforward, and the result is of technical interest only, we refrain from stating an explicit proof.

### 3.3. Succinct representation of convex problems

The strategy followed in the hardness proofs of this paper is to exploit the fact that most logical formalisms are able to express positive Boolean combinations of input relations; therefore, it is possible to reproduce the effect of succinctness within the logical formalism.

For convex problems, we obtain a very nice and natural definition of succinctness:

**Definition 4.** Let \( \tau = (R_1^{a_1}, \ldots, R_r^{a_r}) \), and let \( d \geq 1 \). Then a \( \tau \)-description of arity \( d \) is a tuple

\[
\Phi = (\phi_1, \ldots, \phi_r)
\]
of propositional formulas such that every \( \phi_i \) contains \( a_i \) vectors \( x_1, \ldots, x_{a_i} \), where each \( x_j \) is a vector of \( d \) Boolean variables \( x_{j,1}, \ldots, x_{j,d} \). Every \( \tau \)-description \( \Phi \) defines a \( \tau \)-structure

\[
\operatorname{gen}(\Phi) = (2^d, \operatorname{gen}(\phi_1), \ldots, \operatorname{gen}(\phi_r))
\]

where

\[
\operatorname{gen}(\phi_i) = \{ (\operatorname{int}(b_1), \ldots, \operatorname{int}(b_d)) \mid \phi_i(b_1, \ldots, b_d) = 1 \}.
\]

Let \( \Pi \subseteq \operatorname{Struct}(\tau) \) be a convex problem. Then the propositional version \( p(\Pi) \) of \( \Pi \) is defined as

\[
p(\Pi) = \{ \Phi \mid \operatorname{gen}(\Phi) \in \Pi \}.
\]

**Example 4.** Let \( \tau = (E^2, C^1) \) be a signature for two-colored graphs, and consider the \( \tau \)-description \( \Phi = (\phi_E, \phi_U) \) of arity 3 where \( \phi_E(x_1, x_2, x_3, y_1, y_2, y_3) \) is

\[
(x_1 \equiv y_1 \land x_2 \equiv y_2) \lor (\neg x_3 \land \neg y_3)
\]

and \( \phi_U(x_1, x_2, x_3) \) is

\[
(x_1 \equiv x_2) \land (x_2 \equiv x_3).
\]

Then the colored graph \( \operatorname{gen}(\Phi) \) has a set of vertices \( \{0, 1, 2, 3, 4, 5, 6, 7\} \), or, in binary notation, \( \{0, 1\}^3 \). The edge relation and the coloring are shown in Fig. 3.

It has been shown in [50] that succinct versions of convex problems indeed are special cases of a more general definition in [51] which allows to represent domain sizes which are not a power of 2 and vectorized instances of problems in a direct way.

Therefore, we can use the results of [51]; the most important step in theorems of this style is the following lemma:

**Lemma 2** (Veith [51]; Propositional Conversion Lemma). Let \( A \) and \( B \) be convex. If \( A \equiv \text{qf} \ R \) then \( p(A) \equiv \text{LOGSPACE} \ p(B) \).

With the Conversion Lemma, it is a small step to obtain the Upgrading Theorem:

**Theorem 2** (Veith [51]). Let \( C_1 \) and \( C_2 \) be complexity classes, such that \( \text{long}(C_1) \subseteq C_2 \), and let \( B \) be convex. If \( B \) is \( C_2 \)-hard under quantifier-free reductions then \( p(B) \) is \( C_1 \)-hard under \text{LOGSPACE} reductions.

By combining these results we obtain the following theorem:

**Theorem 3.** Let \( C \) be a class in \( \text{PH} \). Then there exists a problem \( \Pi_C \) such that

1. \( \Pi_C \) is complete for \( C \) under quantifier-free reductions;
Fig. 3. The graph gen(Φ). The colored vertices are underlined.

2. \( \Pi_C \) is convex;
3. \( p(\Pi_C) \) is complete for \( E(C) \) and \( \text{Exp}(C) \) under \text{LOGSPACE} reductions.

**Proof.** Since all classes in the PH are syntactic, by Theorem 1 there exists a convex problem \( \Pi_C \) which is complete for \( C \) under quantifier-free reductions. From Theorem 17 we conclude that \( \text{long}(E(C)) \subseteq C \), and thus, by virtue of Theorems 2 and 19, hardness follows. To prove membership in \( E(C) \), we recall that \( p(\Pi_C) \) is a propositional problem. Since a formula of size \( n \) has less than \( n \) input variables, the structure described by the formula can be computed in deterministic time \( 2^n \) by a Turing machine \( M \). Since there is a \( C \) algorithm for \( \Pi_C \) by the first proposition of the theorem, we conclude that there is an \( E(C) \) algorithm for \( p(\Pi_C) \) obtained from composing \( M \) with a \( C \)-machine. Membership in \( \text{Exp}(C) \) follows because \( E(C) \subseteq \text{Exp}(C) \).

4. Second-order logic – a case study

In this section, we determine the expression complexity of fragments of second-order logic. The proofs are intended as a case study; we shall write them in sufficient detail to extract those general properties of logical languages which are needed to generalize the proof in Section 5.

Technically, we show that \( \Sigma_k \) has expression complexity \( E \Sigma_k' \). The membership part of the completeness proof will be easy and direct. To prove hardness, we apply the complexity upgrade technique presented in the previous section.
4.1. Second-order logic

A second-order prenex formula is a formula $\Psi$ of the form $Q_1X_1Q_2X_2\ldots Q_kX_k : \phi$ where $X_i, 1 \leq i \leq k$ is a tuple of predicate variables, $\phi$ is a first-order formula with signature $\tau \cup \{X_1, \ldots, X_k\}$, and $Q_i, 1 \leq i \leq k$ is a quantifier from $\{\forall, \exists\}$.

The prenex fragments $\Sigma^1_k$ and $\Pi^1_k$ are defined in the usual way. The fragment $\Theta^1_k$ consists of second-order formulas where all subformulas with a leading second-order quantifier are $\Sigma^1_k$ or $\Pi^1_k$. Thus, the fragment $\Theta^1_k$ is the natural first-order closure of $\Sigma^1_k$.

**Theorem 4** (Fagin [14]; Stockmeyer [46]; Gottlob [18]). $\Sigma^1_k$ captures $\mathcal{C}_a$ and captures $\mathcal{I}_a$. Over ordered structures, $\Theta^1_k$ captures $\Theta^0_k$.

4.2. Expression complexity of second-order logic

**Theorem 5.** $\Sigma^1_k$ has expression complexity $EC^\Sigma^\varnothing_k$.

**Proof.** Hardness: Let $\mathfrak{B}$ be the fixed structure $\langle(0, 1), T^\mathfrak{B}, F^\mathfrak{B}\rangle$, where $T^\mathfrak{B} = \{1\}$ and $F^\mathfrak{B} = \{0\}$. Note that with every propositional formula in negation normal form we can associate a corresponding positive quantifier-free first-order formula by replacing every occurrence of a positive literal $x$ by $T(x)$ and every occurrence of a negative literal $\neg x$ by $F(x)$. Thus, truth value assignments to the propositional variables can be identified with assignments to first-order variables over $\mathfrak{B}$.

We shall consider the problem $E_{\mathfrak{B}, \Sigma^1_k} = \langle\Phi E \Sigma^1_k 1 \mathfrak{B}, \mathfrak{B}\rangle$ that is the set of $\Sigma^1_k$ formulae satisfied by $\mathfrak{B}$. The complexity of recognizing $E_{\mathfrak{B}, \Sigma^1_k}$ is a lower bound for the expression complexity of the logic $\Sigma^1_k$. By Theorem 3, there exists a convex problem $\Xi := \Pi_{\Sigma^1_k} \subseteq \text{Struct}(\zeta)$, such that $p(\Xi)$ is complete for $EC^\Sigma_k$ under LOGSPACE reductions. Since $\Sigma^1_k$ captures $\Sigma^0_k$, there exists a $\Sigma^1_k$ formula $\rho$ over $\zeta$ such that $\Xi = \text{Mod}(\rho)$.

We will show that $p(\Xi) \leq \text{LOGSPACE} E_{\mathfrak{B}, \Sigma^1_k}$. The reduction will be a LOGSPACE computable function $f$ mapping a $\zeta$-description $\Phi$ to a $\Sigma^1_k$ formula $f(\Phi)$ such that

$$\mathfrak{B} \models f(\Phi) \iff \Phi \in p(\Xi)$$

and therefore, by definition of the succinct problems,

$$\mathfrak{B} \models f(\Phi) \iff \text{gen}(\Phi) \in \Xi \iff \text{gen}(\Phi) \models \rho.$$

Going from the left to the right, we see that in order to decide $\text{gen}(\Phi) \models \rho$, $f(\Phi)$ will have to encode two steps of computation: First, the structure $\text{gen}(\Phi)$ described by the $\zeta$-description $\Phi$ has to be determined; second, $\text{gen}(\Phi)$ has to be checked for satisfying $\rho$.

Recall that $\zeta = (\land, \lor, \neg, \land, \lor, T^1, \bot^1)$, and consider a $\zeta$-description $\Phi = (\phi_1, \ldots, \phi_t)$ of arity $d$. For every $\phi_t$, we construct a first-order formula $\psi_t$ by transforming $\phi_t$ into negation normal form, and then replacing every occurrence of a literal $\neg x$ by $F(x)$ and of a literal $x$ by $T(x)$. Thus, the $\psi_t$ are quantifier-free positive first-order formulas.
involving only the predicates \( T \) and \( F \), and obviously \( \psi_i \) is LOGSPACE computable from \( \phi_i \). Evidently, the propositional assignments to the variables in \( \phi_i \) are in one-one correspondence to the assignments to the first-order variables in \( \psi_i \), i.e., for all \( d \)-tuples \( b_1, \ldots, b_3 \) of values 0, 1 it holds that

\[
\text{gen}(\phi) \models \Pi (\text{int}(b_1), \text{int}(b_2), \text{int}(b_3)) \iff \mathcal{B} \models \psi_i(b_1, b_2, b_3) \tag{*}
\]

and similarly for the other relations in \( \zeta \). In the following, we shall write out only the case of \( \Pi \); the other relations in \( \zeta \) are treated completely analogously.

Thus, \( \psi_i^{\mathbb{B}} \) and \( \Pi^{\text{ent}}(\phi) \) are isomorphic; however, \( \psi_i^{\mathbb{B}} \) has arity \( 3d \), while \( \Pi^{\text{ent}}(\phi) \) has the original arity 3. Therefore, we have to increase the arities in the formula \( \rho \) by a factor \( d \), that is, we have to go from \( \rho \) with \( \text{Mod}(\rho) = \Xi \) to a new formula \( \rho^{(d)} \) such that
\[
\text{Mod}(\rho^{(d)}) = \Xi^{(d)}.
\]

Given \( \rho, \rho^{(d)} \) can be computed easily by a LOGSPACE transducer:

1. Replace every first-order variable \( y \) in \( \rho \) by a vector \( y_1, \ldots, y_d \).
2. Replace every vector equality \( y_1, \ldots, y_d = z_1, \ldots, z_d \) by \( \bigwedge_{j=1}^{d} y_j = z_j \).

From (*) it follows that for the closed formula \( \rho \),

\[
\text{gen}(\Phi) \models \rho \iff \mathcal{B} \models \psi_i^{\mathbb{B}}, \ldots, \psi_i^{\mathbb{B}} \models \rho^{(d)}
\]

and this in turn is equivalent to

\[
\mathcal{B} \models \rho^{(d)}[[\forall/\psi_1, \ldots, \exists/\psi_1]].
\]

Thus, we set \( f(\Phi) = \text{hard}_k(\Phi) \). Note that \( \text{hard}_k(\Phi) \) is over vocabulary \( (T^1, F^1) \). \( f \) is LOGSPACE computable, because \( \rho \) is constant, and all the intermediate steps were shown to be LOGSPACE computable.

Membership: Let \( \mathcal{A} \models \exists P_1, \ldots P_p : \Phi \) be an instance, where \( \mathcal{A} = (n, R_1^{\mathbb{B}}, \ldots, R_n^{\mathbb{B}}) \) is fixed. Without loss of generality we may assume that \( \text{ar}(P_1) \) is linear in the size of the formula, because otherwise we may add a dummy predicate variable of sufficient arity. An NE algorithm can guess relations \( P_1, \ldots, P_p \) over domain \( k \) for the relation symbols \( P_1, \ldots, P_p \), and ask the oracle for

\[
(n, R_1^{\mathbb{B}}, \ldots, R_n^{\mathbb{B}}, P_1, \ldots, P_p) \models \Phi.
\]

By our assumption on the relation arities, the size of the guessed relations is exponential in the formula size, whence \( \alpha \) can be solved by a \( \Pi_{k-1}^{P} \) oracle. \( \square \)

Remark. (1) The proof does not depend on any specific properties of the signature \( \zeta \). Instead of using abstractly generated complete problems over \( \zeta \) for syntactic complexity classes, it may sometimes be more instructive to use more natural problems, as we have done in [19]. Complete problems for the levels of the polynomial hierarchy which involve quantified Boolean formulas can be found for example in [42].
Similarly, one has to turn to concrete problems if the logic under investigation does not capture a syntactic class. We shall investigate such logics in Section 6.

**Theorem 6.** \( \Sigma^i_k \) has combined complexity \( \text{Exp} \Sigma^i_k \).

**Proof.** By virtue of Theorems 5 and 19 it remains to show membership. Since the domain size is not fixed now, the algorithm used in the proof of Theorem 5 has to guess relations of size \( m^m \), which is bounded above by \( 2^{m^2} \). We conclude that the algorithm is in NEXP. \( \square \)

**Remark.** Since hardness for combined complexity follows from hardness for expression complexity, and membership proofs for combined complexity are just observations from the proofs for expression complexity, we shall concentrate on expression complexity further on.

5. **Closure properties – the general case**

In this section, we consider which properties of second order logic we needed in the proof of Theorem 5. In Section 6 we shall use these properties to derive the announced general conditions for the complexity gap.

5.1. **Uniform interpretation closure**

In first-order logic, it is a trivial syntactic operation to replace a relation symbol by a formula; to capture this phenomenon for arbitrary logics, we use the following definition:

**Definition 5.** A logic \( L \) is uniformly interpretation closed if relation symbols in formulas of \( L \) can be replaced by quantifier-free positive formulas over signature \((T',F')\) in LOGSPACE, i.e., if for all signatures \( \tau = (R^{n_1},\ldots,R^{n_r}) \) there exists a LOGSPACE computable mapping

\[
(\rho, \phi_1,\ldots,\phi_r) \rightarrow \rho[R^1/\phi_1,\ldots,R^r/\phi_r]
\]

such that for all structures \( A \in \text{Struct}(T,F) \), \( L(\tau) \)-expressions \( \rho \), and quantifier-free positive first-order \((T',F')\)-formulas \( \phi_1,\ldots,\phi_r \) it holds that

\[
(A, \phi_1^n,\ldots,\phi_r^n) \models \rho \iff A \models \rho[R^1/\phi_1,\ldots,R^r/\phi_r].
\]

5.2. **Uniform vector closure**

The second closure property which we investigate is also motivated by the proof of Theorem 5, and exemplified by first-order logic.
Let \( \tau = (R_1^{\omega}, \ldots, R_p^{\omega}) \), and \( \Pi \subseteq \text{Struct}(\tau) \), such that \( \Pi = \text{Mod}(\phi) \), where \( \phi \) is a first-order sentence. Recall that \( \Pi^{(k)} \) denotes the vectorized problem \( \Pi \), i.e., those instances of \( \Pi \), where the domain consists of \( k \)-tuples. Formally, \( \Pi^{(k)} \) has a signature \( \tau^{(k)} = (R_1^{d_1 k}, \ldots, R_p^{d_p k}) \).

In first-order logic, \( \phi^{(k)} \) is the formula obtained from \( \phi \) by replacing, for each variable \( x \) occurring in \( \phi \), every occurrence of \( x \) by the \( k \)-tuple of variables \( x_1, \ldots, x_k \). In the resulting formula, each vector equation \( x_1, \ldots, x_k = y_1, \ldots, y_k \) is replaced by \( \bigwedge_{i=1}^{k} x_i = y_i \). Then \( \Pi^{(k)} = \text{Mod}(\phi^{(k)}) \).

This motivates the second closure property:

\textbf{Definition 6.} A logic \( \mathcal{L} \) is uniformly vector closed if there exists a LOGSPACE computable mapping

\[(\phi, w) \mapsto \phi^{[|w|]} \]

such that

\[\text{Mod}(\phi^{[|w|]} = \text{Mod}(\phi^{(|w|)})\]

for all binary strings \( w \) and \( \mathcal{L} \) expressions \( \phi \).

\textbf{Remark.} Since the arity of a description is of the same order as its size, we require that the number of variables is given by the size of a string to obtain a useful notion of LOGSPACE mapping.

We say that a logic is uniformly closed if it is both uniformly interpretation closed and uniformly vector closed.

\subsection*{5.3. Criteria for expression complexity}

Our first main result links the existence of convex hard problems to the expression complexity of a language:

\textbf{Theorem 7.} Let \( \mathcal{L} \) be uniformly closed. If \( \mathcal{L} \) defines a convex problem \( \Pi_C \) which is complete for a class \( C \) under quantifier-free reductions, then \( \mathcal{L} \) has expression complexity at least \( D \), for every \( D \) such that \( C \subseteq \text{long}(D) \).

In particular, if \( C \) is a class in the polynomial hierarchy, then \( \mathcal{L} \) has expression complexity at least \( E(C) \) and \( \text{Exp}(C) \).

\textbf{Proof.} Let \( \Pi_C = \text{Mod}(\rho) \subseteq \text{Struct}(\tau) \), where \( \tau = (R_1, \ldots, R_p) \). By Theorem 2 the succinct version \( p(\Pi_C) \) is hard for \( E(C) \) under LOGSPACE reductions. Now we can proceed analogously as in the hardness part of the proof of Theorem 5.

Let \( \mathfrak{B}, \Phi = (\phi_1, \ldots, \phi_p) \), and \( \psi_1, \ldots, \psi_p \) be like in the proof of Theorem 5. Then the mapping

\[f : \Phi \mapsto \rho^{(d)}[R_1/\psi_1, \ldots, R_p/\psi_p]\]
is a reduction from $p(\Pi_C)$ to $E_{\phi,\varphi}$ whose complexity is a lower bound for the expression complexity of $L'$. Since $L'$ is uniformly closed, $f$ is computable in LOGSPACE.

Our second main result finally links the expression complexity of languages capturing complexity classes to their expression complexity:

**Theorem 8.** If $L$ captures a syntactic complexity class $C$, and $L$ is uniformly closed, then $L$ has expression complexity at least $D$, for every $D$ such that $C \subseteq \text{long}(D)$.

In particular, if $C$ is a class in the polynomial hierarchy, then $L$ has expression complexity at least $E(C)$ and $\text{Exp}(C)$.

**Proof.** By Theorem 1 there exists a convex problem $\Pi_C$ which is complete for $C$ under quantifier-free reductions. Since $L$ captures $C$, there exists a $\rho \in L'$ such that $\text{Mod}(\rho) = \Pi_C$. An application of Theorem 7 yields the result. □

**Example 5.** Since $\Pi_k^1$ captures $\Pi_k^P$, and is uniformly closed, Theorem 8 says that its expression complexity is at least $E\Pi_k^P$. A straightforward membership proof shows that $\Pi_k^P$ indeed has expression complexity $E\Pi_k^P$.

The same holds for $\Theta_k^1$ and $E\Theta_k^P$: as mentioned earlier, we can for expression complexity always suppose that the domain is ordered, and therefore the capturing results of [18] can be applied.

**Example 6.** Consider strict$\Sigma_k^1$, i.e. the fragment of $\Sigma_k^1$, where the first-order part is universal. Since strict$\Sigma_k^1$ is properly contained in $\Sigma_k^1$, there is no hope to apply Theorem 8 in a straightforward manner.

However, strict$\Sigma_k^1$ can define convex problems which are hard for NP under quantifier-free reductions. Consider for example the problem 3COL, i.e., the problem to decide if the vertices of a graph can be colored by 3 colors in such a way that any two adjacent vertices have different colors. Here, the colors correspond to unary relations $R, G, B$. Formally, we define

$$3\text{COL} = \{(n,E) \mid \exists R,G,B \phi\}$$

where $\phi$ is

$$\forall x (x \text{ is colored by one of } R, G, B) \land \forall xy (E(x,y) \rightarrow x \text{ and } y \text{ have different colors}).$$

Evidently, 3COL is convex and in strict$\Sigma_k^1$, and in fact it even is in monadic NP. In [45], it was shown that 3COL is NP-complete under quantifier-free reductions.

strict$\Sigma_k^1$ is obviously uniformly closed. By Theorem 7, it follows that the expression complexity of strict$\Sigma_k^1$ is at least NE. Since full $\Sigma_k^1$ has expression complexity NE, we conclude that strict$\Sigma_k^1$ has expression complexity NE.
The following example shows that there exist odd syntactic modifications of known languages which destroy the behavior we expect for expression complexity:

**Example 7.** By the classical result in [49], first-order logic has expression complexity PSPACE. Consider now the language FOOL, a version of first-order logic: $\text{FOOL} = \{ \phi \circ \text{pad}(\phi) | \phi \text{ is FO} \}$, where $\text{pad}(\phi) = 1^{2^{\text{size}(\phi)}}$, i.e., each formula is followed by a lot of foolish bits. The semantics is straightforward: $\mathcal{A} \models \phi$ iff $\mathcal{A} \models \phi \circ \text{pad}(\phi)$. Although FOOL is as expressive as FO, the exponential blow-up causes $E_{\text{FOOL}} \subseteq \text{LOGSPACE}$. Here, the vectorization is not computable in LOGSPACE, because the vectorized formula becomes exponentially larger than the original.

5.4. Uniform reducibility

Once we have settled the expression complexity of a language, we can reduce the expression complexity of this language to the expression complexity of another language in the following sense:

**Definition 7.** Let $\mathcal{L}_1, \mathcal{L}_2$ be two languages, such that the signature of $\mathcal{L}_1$ is contained in the signature of $\mathcal{L}_2$. We say that $\mathcal{L}_1$ is uniformly reducible to $\mathcal{L}_2$, in symbols $\mathcal{L}_1 \leq \mathcal{L}_2$, if there exists a LOGSPACE computable mapping $f: \mathcal{L}_1 \rightarrow \mathcal{L}_2$ such that for all finite structures $\mathcal{A}$ and all expressions $\phi \in \mathcal{L}_1$, $\mathcal{A} \models \phi$ iff $\mathcal{A} \models f(\phi)$

**Theorem 9.** Suppose that $\mathcal{L}_1 \leq \mathcal{L}_2$, and that $\mathcal{L}_1$ has expression complexity at least $C$. Then $\mathcal{L}_2$ has expression complexity at least $C$.

**Proof.** By definition, the expression complexity of $\mathcal{L}_1$ is witnessed by a set $E_{\text{W}, \mathcal{L}_1}$. By the assumption of the theorem, there exists a LOGSPACE reduction to the set $E_{\text{W}, \mathcal{L}_2}$ which is a lower bound for the expression complexity of $\mathcal{L}_2$.

**Example 8.** Consider the fragment skolemized$\Sigma_1^1$ which consists of $\Sigma_1^1$ formulas in Skolem Normal Form

$$\exists P \forall x \exists y \phi$$

where $\phi$ is quantifier-free. As shown in [32], the Skolemization procedure can be easily done in LOGSPACE, and thus $\Sigma_1^1 \leq \text{skolemized} \Sigma_1^1$. By Theorems 5 and 9 it follows that skolemized$\Sigma_1^1$ has expression complexity NE. Analogous results hold for all skolemized$\Sigma_k^1$.

**Example 9.** The expression complexity of skolemized$\Sigma_1^1$ can also be concluded by a reduction from strict$\Sigma_1^1$ to skolemized$\Sigma_1^1$. Such a reduction is trivially obtained by the identical mapping, and therefore hardness is proven. Membership on the other hand is an immediate consequence of Theorem 5.
6. Applications to logic, databases and AI

6.1. Henkin quantifiers

Henkin [25] quantifiers are a natural generalization of first-order formulas. Consider an expression

$$(\forall u^x \exists v^y \forall x^y) \phi(u, v, x, y),$$

where $\phi$ is a first-order formula with free variable vectors $u, v, x, y$. The semantics of the above sentence is defined by the following second-order sentence:

$$\exists F_1, F_2: (\forall x^y \exists v^y F_1(x, y)) \land (\forall x^y \exists v^y F_2(x, y)) \land "F_1 \text{ and } F_2 \text{ are functions, and } (\forall u^x, v^x, x, y: F_1(u, v) \land F_2(x, y) \rightarrow \phi(u, v, x, y)"

In other words, there exist distinct Skolem functions for the variables in $v, y$ which depend only on $u, x$, respectively.

We shall investigate two Henkin languages corresponding to the classes $\Sigma_1$ and $\Theta_1$: Prenex Henkin Logic, and its first-order closure (termed first-order logic with Henkin quantifiers) no Henkin quantifier occurs in the scope of another Henkin quantifier.

**Theorem 10.** Prenex Henkin logic has expression complexity $\text{NE}$. First-order logic with Henkin quantifiers has expression complexity $\text{LinSpace}^{\text{NE}} = \text{Exp}\Theta_1^p$.

**Proof.** Let us consider the prenex case first. For membership, we just have to translate a prenex Henkin formula into a second-order $\Theta_1^p$ formula by writing out the Skolem functions explicitly according to formula $(\dagger)$. This operation preserves the length of the input up to a constant factor, and thus, membership in $\text{NE}$ follows.

For hardness, we consider the problem 3COL previously used in Example 7. The following formula is a modification of a formula in [3], and defines 3COL by a Prenex Henkin formula over domains of size at least 3:

$$\left( \forall x_1 \exists y_1 \right) \exists abc \ a \neq b \land a \neq c \land b \neq c \land y_1 = a \lor y_1 = b \lor y_1 = c \land x_1 = x_2 \rightarrow y_1 = y_2 \land E(x_1, x_2) \rightarrow y_1 \neq y_2$$

It is easy to see that $E$ can be replaced by a formula over $(T^1, F^1)$, and that variables can be replaced by tuples; thus the prenex Henkin formulas are uniformly closed. and we conclude hardness for $\text{NE}$ by Theorem 7.

For the full first-order case the result follows analogously by using the fact that over ordered structures, $\Theta_1^{\text{FO}}$ is captured by the first-order closure of 3COL [18, 43, 44].
6.2. Transitive closure logic

Transitive closure logic has been studied in depth in the field of finite model theory; variants of this logic have been shown to capture the classes LOGSPACE and NLOGSPACE respectively [27]; for simplicity, we consider only unnested transitive closure logic here.

Transitive closure logic is obtained by closing the syntax of first-order logic under the following rule:

Let $\phi(x, y)$ be a first-order formula where $x$ and $y$ are free, and let $u, v$ be two new variables. Then the formula

$$\text{TC}(\lambda x y. \phi)[u, v]$$

is a formula of transitive closure logic, and the variables $x, y$ are bound by the $TC$ operator. The semantics is defined as follows:

$$\models \text{TC}(\lambda x y. \phi)[u, v] \iff \text{there is a path in } (n, \phi^n) \text{ from } u^n \text{ to } v^n.$$

In addition, transitive closures over graphs on tuples are allowed, that is, we add operators $\text{TC}(\lambda x, y. \phi)[u, v]$ where $x, y, u, v$ are vectors of $k$ variables each.

Theorem 11 (Immerman [27]). Over ordered structures, transitive closure logic captures NLOGSPACE. Moreover, the respective decision problem GAP, i.e., the problem to decide if max is reachable from 0, is complete for NLOGSPACE under quantifier-free reductions.

It immediately follows that the data complexity of the language is NLOGSPACE. Expression complexity is settled by the following theorem.

Theorem 12. Transitive closure logic has expression complexity NLinSpace.

Proof. Membership: Since NLinSpace is closed under complement [28, 47], any alternating LinSpace algorithm with a constant number of alternations collapses to NLinSpace. In a first step, the algorithm moves all negations in front of atoms or $TC$ operators. This can be easily done in LinSpace. Then the algorithm works in the usual deterministic way for first-order formulas, i.e., by simulating quantifiers by loops. If the algorithm reaches an occurrence of $TC(\phi)$ or $\neg TC(\phi)$, then it changes into an existential (resp. universal) state, and obtains the transitive closure by repeatedly evaluating $\phi$.

Hardness: We know from Theorem 11 that GAP is complete for NLOGSPACE under quantifier free reductions. It is evident that GAP is convex because isolated domain elements cannot destroy existing paths in a graph. Since transitive closure logic captures NLOGSPACE, it expresses GAP, and since it is uniformly closed, we conclude hardness for PSPACE with Theorem 7. □
Remark. Since NLOGSPACE is not a syntactic complexity class (i.e., it is not closed under polynomial time reductions, unless \( \text{PTIME} = \text{NLOGSPACE} \)) we could not apply Theorem 8; thus, we did not get a NLOGSPACE complete problem for free, but had to use GAP.

6.3. Default logic

Default logic, first proposed by Reiter in [39], is the most popular knowledge representation language in the field of Artificial Intelligence. Interestingly, as shown in [6], default logic can be used as a powerful query language for databases. In this section, by using our methodology and the expressiveness results of [6], we derive very simply the expression complexity of default logic.

6.3.1. Propositional default logic

A propositional default theory is a pair \( (W, D) \) where \( W \) is a finite set of propositional sentences and \( D \) is a set of defaults. A default is an expression of the form

\[
\frac{\alpha : M_\gamma}{\omega}
\]

where \( \alpha, \gamma, \omega \) are propositional formulas. \( \alpha \) is called the prerequisite, \( \gamma \) is called the justification and \( \omega \) the consequence of the default. (In the literature, there are usually several justifications, but for the purpose of our exposition this language is sufficient.) The intended interpretation of a default is: If \( \alpha \) holds, and there is no counter evidence for \( \gamma \), then \( \omega \) can be inferred by default.

An extension of \( (W, D) \) is a set of propositional formulas which is based on a series of default conclusions starting from \( W \), and which is closed under both propositional inference and default application. In formal terms, \( E \) is an extension of \( (W, D) \) if \( E = \bigcup_{i \geq 0} E_i \), where \( E_0 = W \) and \( E_{i+1} = \{ \gamma | (\alpha : M_\beta / \gamma) \in D, \alpha \in E_i, \neg \beta \notin E \} \cup \text{Cons}(E_i) \). Here, \( \text{Cons}(E_i) \) denotes the propositional deductive closure.

A default theory in general does not have a unique extension. This gives rise to credulous and skeptical semantics. We say that \( (W, D) \) entails \( \phi \) credulously (resp. skeptically) if \( \phi \) is contained in all (some) extensions of \( (W, D) \). Note that extensions are infinite sets. Formally, we write \( (W, D) \vdash \text{sk} \phi \) (or \( (W, D) \vdash \text{cr} \phi \)). We call the corresponding decision problems DEF-sk and DEF-cr.

Theorem 13 (Gottlob [17]). DEF-cr is \( \Sigma^p_2 \)-complete and DEF-sk is \( \Pi^P_2 \)-complete.

6.3.2. Full default logic

We next show how to extend the propositional definitions to the first-order case. In the following, the term formula will subsume both ordinary first-order and default formulas.
Let $\mathcal{A} \in \text{Struct}(\tau)$ be a structure. Then the completion $\text{COMP}(\mathcal{A})$ of $\mathcal{A}$ is defined as $\{ R(b) \mid \mathcal{A} \models R(b), \mathcal{A} \models \neg R(c), c \in \mathcal{A} \} \cup \{ \neg R(b) \mid \mathcal{A} \models R(b), R \in \tau, b \in \mathcal{A} \} \cup \{ \neg R(b) \mid \mathcal{A} \models R(b), R \in \tau, b \in \mathcal{A} \} \cup \{ \neg R(b) \mid \mathcal{A} \models R(b), R \in \tau, b \in \mathcal{A} \}$, i.e., the Herbrand model corresponding to $\mathcal{A}$.

Let $\phi(x)$ be a formula or rule with free variables $x$. Then $\text{INST}_{\mathcal{A}}(\phi(x))$ is the set of formulas obtained by replacing tuples of domain elements from $\mathcal{A}$ for $x$. For a set $S$ of formulas and rules, let $\text{INST}_{\mathcal{A}}(S) = \bigcup_{\phi \in S} \text{INST}_{\mathcal{A}}(\phi)$. Thus, for every set $S$ and structure $\mathcal{A}$, $\text{INST}_{\mathcal{A}}(S)$ is a set of closed formulas equivalent to $S$ over $\mathcal{A}$.

Let us turn to full first-order formulas. Over a fixed domain, first-order formulas can be seen as large quantifier-free formulas with the quantifiers replaced by conjunctions and disjunctions. This gives rise to the following definition:

**Definition 8.** Let $\psi$ be a default formula over signature $\tau$, $\mathcal{A} \in \text{Struct}(\tau)$. Then $\text{prop}_{\mathcal{A}}(\psi)$ denotes the formula obtained from $\psi$ by recursively replacing each subformula $\forall x \psi(x)$ by $\bigwedge_{c \in \mathcal{A}} \psi(c)$ and $\exists x \psi(x)$ by $\bigvee_{c \in \mathcal{A}} \psi(c)$. Then we define $\text{ground}_{\mathcal{A}}(\phi) = \text{INST}_{\mathcal{A}}(\text{prop}_{\mathcal{A}}(\phi))$.

As with $\text{INST}$, this definition generalizes to sets of formulas. Now it is easy to define the semantics for first-order default logic: Given a model $\mathcal{A}$, and a first-order default theory $(W, D)$, we consider the extensions of the propositional default theory

$$\text{ground}_{\mathcal{A}}((W, D)) = \langle \text{COMP}(\mathcal{A}) \cup \text{ground}_{\mathcal{A}}(W), \text{ground}_{\mathcal{A}}(D) \rangle.$$  

Hence, we can define entailment for both skeptical and credulous reasoning:

$$\mathcal{A} \models_{(W, D)} \phi \iff \text{ground}_{\mathcal{A}}((W, D)) \models \text{ground}_{\mathcal{A}}(\phi).$$

It is easy to show that first-order default logic with credulous semantics captures $\Sigma_2^P$. From [6] we know that the quantifier-free fragment captures $\Sigma_2^P$, therefore we only have to prove the membership part by showing that first-order default logic credulous entailment can be polynomially reduced to DEF-cr. This follows immediately from the fact that the computation of $\text{ground}_{\mathcal{A}}$ is polynomial in $|\mathcal{A}|$ and from the definition of entailment above. Thus we obtain:

**Theorem 14.** First-order default logic with credulous semantics captures $\Sigma_2^P$.

In a similar way we can obtain results about expression complexity. For a default $d$, let $|d|$ denote the cumulated length of its constituting first-order formulas.

**Theorem 15.** First-order default logic under credulous semantics has expression complexity $\text{NE}^{\text{NP}}$. Hardness even holds for the quantifier-free case.

**Proof.** Hardness: From Example 8 we know that skolemized $\Sigma_2^1$ has expression complexity $\text{NE}^{\text{NP}}$. The capturing proof for quantifier-free default logic from [6] features a LOGSPACE mapping $f$ which associates with each skolemized $\Sigma_2^1$ formula $\phi$ a default
theory $f(\phi)$, s.t. $\mathcal{U} \models \phi$ iff $\mathcal{U} \vdash f(\phi) V$, for a fixed literal $V$. Therefore, skolemized $\Sigma^1_2 \leq$ Credulous Default Logic, and hardness follows from Theorem 9.

**Membership:** Note that the computation of $\text{ground}_{\mathcal{U}}(\langle W, D \rangle)$ for fixed $\mathcal{U}$ is exponential in $|\langle W, D \rangle|$. Therefore, each instance of the expression complexity problem can be deterministically reduced to an exponential size instance of DEF-cr, hence its complexity is bounded above by NE$^N$.

6.4. Logic programming

In this section we show that languages for logic programming typically satisfy the closure properties considered in Section 5 and, as a consequence, easy proofs for the expression complexity of logic programming languages follow from our theorems.

Let us first consider uniform interpretation closure. Suppose there is a program $LP$ which works on input relations $\tau = (P_1, \ldots, P_n)$, and that there are positive quantifier-free definitions $\psi_1, \ldots, \psi_n$ of the $\tau$ relations in terms of $(T^1, F^1)$.

Then, $\psi_i$ can be easily translated into a logic program $R_i$ by inductively rewriting all subformulas $\phi$ of $\psi_i$ into program rules: If $\phi$ is of the form $\phi_1(x) \land \phi_2(x)$, we write

$$R_{\phi}(x) : = R_{\phi_1}(x), R_{\phi_2}(x)$$

and if $\phi$ is of the form $\phi_1(x) \lor \phi_2(x)$, we write

$$R_{\phi}(x) : = R_{\phi_1}(x), \quad R_{\phi}(x) : = R_{\phi_2}(x).$$

If $\phi(x)$ is a literal, we add the rule

$$R_{\phi}(x) : = \phi(x).$$

Since all logic programming semantics agree on non-recursive negation-free programs, the new program $LP \cup \bigcup_{i=1}^n R_i$ serves as $LP[P_1/\psi_1, \ldots, P_n/\psi_n]$. The procedure just described obviously works in LOGSPACE.

For uniform vector closure, let $d$ be a natural number, and consider an arbitrary logic program $LP$ where all equalities $x = y$ are replaced by a new relation $E(x, y)$, and all inequalities by $I(x, y)$. For every variable $x$, let $x = x_1, \ldots, x_d$ be a vector of $d$ new variables. Then we obtain a program $LP^{(d)}$ by replacing all occurrences of a variable $x$ by $x$, and set

$$LP^{(d)} = LP^{(d)} \left[ E \bigwedge_{i=1}^d x_i = y_i, \quad I / \bigwedge_{i=1}^d x_i = y_i \right].$$

Obviously, this operation is possible for most logic programming formalisms. The following two examples show how one gets easy proofs of the expression complexity results in [12, 49]:
Example 10. It is well known that DATALOG with stratified negation captures P over ordered structures [49]. By the above arguments, DATALOG is uniformly closed, and we immediately conclude with Theorem 8 that DATALOG has expression complexity ETIME, as previously shown in [49].

Example 11. We know from [11] that Disjunctive Datalog captures $\Sigma^P_2$ under the stable model semantics. By Theorem 8 it follows that Disjunctive Datalog has expression complexity $\text{Exp } \Sigma^P_2$, as previously shown in [11].

6.5. Abduction

A function-free logic programming abduction problem is described by a tuple $(H, M, LP, \models)$, where $H$ is a finite set of predicates ('hypotheses'), $M$ is a finite set of ground literals ('manifestations'), and $LP$ is a function-free logic program for the inference parameter $\models$.

Let $S$ be a subset of the ground instantiations of the predicates from $H$ by constants from $LP$ and $M$. Then $S$ is called a solution of $(H, M, LP, \models)$ iff $LP \cup S \models M$.

Moreover, one can restrict the set of solutions to its minimal elements with respect to set inclusion ($\subseteq$) or cardinality ($\ll$). For a given set or subset of solutions, one can distinguish if a given hypothesis is necessary, i.e., occurs in all solutions or if it is just relevant, i.e., occurs in some solution. In [11], data complexity of the main decision problems arising from abductive reasoning is investigated for different kind of inference operators (stable brave/cautious, well-founded). For all those problems, the data complexity was settled somewhere in the first levels of the polynomial hierarchy.

By Theorem 7, the hardness results for the expression complexity of abductive reasoning over logic programs follow immediately. Membership follows easily from [12].

We can summarize the results for abduction with stable model semantics under cautious inference operator, as corollaries to [12]:

Theorem 16. Under stable cautious inference, the expression complexity of deciding if for a logic programming abduction instance

- there exists a solution, is $\text{E}\Sigma^P_2$.
- a given hypothesis is $\subseteq$-relevant, is $\text{E}\Sigma^P_2$.
- a given hypothesis is $\ll$-relevant, is $\text{E}\Theta^P_2$.
- a given hypothesis is $\subseteq$-necessary, is $\text{E}\Pi^P_2$.
- a given hypothesis is $\ll$-necessary, is $\text{E}\Theta^P_2$.

Similar results can be easily derived for the other logic-based abduction formalisms.

Acknowledgements

We would like to thank Marc Spielmann for valuable comments.
Appendix A. The weak exponential hierarchies

A.1. PH vs. EH

Theorem 17. For all classes $C$ in the polynomial hierarchy, it holds that

$$\text{long}(A) \in C \iff A \in E(C).$$

Proof. We prove the statement for $k \geq 1$; then, the case $k = 0$ will be clear. Consider first the case that $C = \Delta_k^P$ ($E(C) = \text{EA}_k^P$).

If $A \in \text{EA}_k^P$, we have to prove that, given a binary string $w$, a DTM with an oracle in $C_k$ can decide whether $w \in \text{long}(A)$ in time polynomial in $|w|$. By definition, $w \in \text{long}(A) \iff \text{bin}(|w|) \in 1A$; hence, to decide whether $w$ is in $\text{long}(A)$, we can check if $\text{bin}(|w|) \in 1A$. Now, $x = \text{bin}(|w|)$ can be clearly computed in polynomial time in $|w|$. Since $A$ is in $\text{EA}_k^P$, $1A$ is in $\text{EA}_k^P$ as well. Thus (by definition of $\text{EA}_k^P$) a DTM with an oracle in $C_k$ can decide whether $x \in 1A$ in time exponential in $|x|$. Hence, such a DTM can decide whether $x \in 1A$ in time polynomial in $|w|$, as $|w| \geq 2^{|x|+1}$.

If $\text{long}(A) \in \text{EA}_k^P$, then $A \in \text{EA}_k^P$. Let $x$ be a binary word. From Definition 2, $x \in A$ iff all binary words of length $\text{int}(1|x|)$ are in $\text{long}(A)$. The construction of a word $w$ of such a length can be done in time $2^{|x|+1}$. Then, since $\text{long}(A) \in \text{EA}_k^P$, a DTM with an oracle in $C_k$ can decide whether $w \in \text{long}(A)$ in time polynomial in $|w|$. Since $|w| = 2^{|x|+1}$, such a DTM can decide whether $w \in \text{long}(A)$ in time exponential in $|x|$. This proof trivially generalizes to the cases of $\Pi_k^P$ and $\Sigma_k^P$. As for $\text{Exp}_k^P$, the proof is obtained by replacing exponential and polynomial time with linear and logarithmic space, respectively. □

A.2. EH vs. ExpH

It is well known that, even if $E$ has a lower time bound than EXP, problems hard under LOGSPACE reductions for $E$ are hard for EXP as well [29]. This result extends to arbitrary pair of analogue classes of EH and ExpH.

Theorem 18. Given a problem $A$ in $\text{Exp}(x)$, there exists a problem $B$ in $E(x)$ s.t. $A$ and $B$ are mutually reducible by LOGSPACE translations.

Proof. For $\text{Exp}(x) = \text{EXP}$ and $\text{Exp}(x) = \text{NEXP}$ the result has been already proven. We next show that the standard padding technique applies also to the general case. Assume first that $\text{Exp}(x) = \text{NEXP}^C$. Let $B \in \text{NTIME}[2^n]^C$. Let $\odot$ denote string concatenation. Construct $B = \{w(I) \odot 1^n - 2/1 | I \in A\}$, where $w(I)$ is a self-delimiting encoding of $I$, s.t. $|w(I)| = 2 \times |I|$. (For example $w(x_1, \ldots, x_n) = (x_1, 0, x_2, 0, \ldots, x_n, 1)$.) Let $M$ be the NTM for $A$. Then we construct a NTM $M'$, deciding $B$ in time $O(2^{kn})$, as follows. On an instance of $B$, $M'$ first erases the redundant bits and reconstructs $w$ (which is linear in $|B| = n^k$); then $M'$ simulates $M$. Since $M$ has complexity $\text{NTIME}[2^n]^C$, the
complexity of $M'$ is only NTIME[$2^n$]$^C$, as the input size $|B|$ is $n^k$. It is straightforward to check that $A$ and $B$ are mutually reducible by LOGSPACE reductions.

The cases $\text{Exp}(x) = \text{EXP}^C$ and $\text{Exp}(x) = \text{co-NEXP}^C$ are proven analogously. As for $\text{Exp}(x) = \text{PSPACE}^C$, the proof is obtained by interchanging NTIME[$2^n$]$^C$ and NTIME[$n^k$] by $\text{DSPACE}[n^k]$ and $\text{DSPACE}[n]$, respectively. □

**Theorem 19.** Let $E(x)$ be a class in EH and $\text{Exp}(x)$ its analogue class in ExpH. If problem $A$ is complete for $E(x)$ under LOGSPACE reductions, then it is also complete for $\text{Exp}(x)$ under LOGSPACE reductions.

**Proof.** We have to show that all $\text{Exp}(x)$ problems are reducible to $A$ by LOGSPACE reductions. Let $D$ be an arbitrary problem in $\text{Exp}(x)$. By virtue of Theorem 18, there exists some problem $B$ in $E(x)$, s.t. $D \leq_m B$. On the other hand, by hypothesis, $B \leq_m A$. Hence, $D \leq_m A$. □

We conclude that it is sufficient to prove completeness results for the classes in EH to obtain completeness in both hierarchies. It follows from the time hierarchy Theorems by Hartmanis and Stearns [22] that the converse implication does not hold.

**References**


