In: Proc. 11th International Symposium on Fundamentals of Computation Theory (FCT '97), B.S. Chlebus and L. Czaja (eds), LNCS 1279, pp. 1–18, Springer, 1997.

The Complexity Class Θ_2^p: Recent Results and Applications in AI and Modal Logic

Thomas Eiter[1] and Georg Gottlob[2]

[1] Institut für Informatik, Universität Giessen, Arndtstrasse 2, D-35392 Giessen, Germany. Email: eiter@informatik.uni-giessen.de
[2] Institut für Informationssysteme, TU Wien, Paniglgasse 16, A-1040 Wien, Austria. Email: gottlob@dbai.tuwien.ac.at

Abstract. This paper surveys some results involving bounded query classes in the context of structural and descriptive complexity theory, and reviews concrete complete problems in AI and modal logic.

1 Introduction

The study of bounded query classes, in which the access to an oracle for solving a problem is subject to certain restrictions, has received considerable attention within the last fifteen years, see [43, 47, 21, 37, 7, 6, 36, 48, 2, 34] and references therein. Different forms of restrictions, such as imposing an upper bound on the number of queries to the oracle, or prescribing the mode in which the oracle may be accessed, e.g. sequential versus parallel, have been investigated.

A particular interesting and relevant among these classes is $\mathrm{P}^{\mathrm{NP}}[O(\log n)]$, which is the class of problem decidable in polynomial with logarithmically many calls to an NP oracle, measured in the size of the input. This class, also denoted by Θ_2^p, was initially defined to characterize the complexity of optimization problems which did not precisely fit into slots of the polynomial hierarchy.

It appears that Θ_2^p can be defined in many different ways, and is the proper class for characterizing the complexity of a number of practical problems. For this reason, Θ_2^p has become significant both in complexity theory and its applications.

In this paper, we review some recent results on characterizations of Θ_2^p and describe applications of this class. In particular, we focus on AI and modal logic, and present several Θ_2^p-complete problems in these areas. It is worth pointing out that part of them does not involve optimization in their formulation. Some results are new, including an improvement of a result in [31].

2 Preliminaries

We assume familiarity with basic complexity theory, see e.g. [35].

Recall that P^{NP} (resp. L^{NP}), is the class of problems that can be solved in polynomial time (resp., logarithmic work space) with an oracle in NP. The class P^{NP} constitutes the class Δ_2^p of the polynomial hierarchy (PH).

For a family of functions $\mathcal{F}$ and C^{NP}, where C is P or L, let $C^{\mathrm{NP}}[\mathcal{F}]$ denote the class of problems in C^{NP} solvable under the constraint that for some $f \in \mathcal{F}$, on each input of length n at most $f(n)$ oracle calls are made; similarly, let $C^{\mathrm{NP}}_{\|[\mathcal{F}]}$ denote the class of problems in C^{NP} solvable with $f(n)$ rounds of parallel oracle queries.

In particular, $C^{\mathrm{NP}}[O(\log n)]$ is the class of problems solvable in polynomial time with $O(\log n)$ oracle queries, and $C^{\mathrm{NP}}_{\|[k]}$, $k \geq 1$, is the class of problems solvable with k rounds of parallel queries to NP oracles, which for $k = 1$ is usually denoted $C^{\mathrm{NP}}_{\|}$.

We focus here on the class $\Theta^p_2 = \mathrm{P}^{\mathrm{NP}}[O(\log n)]$; the classes $\Theta^p_k = \mathrm{P}^{\Sigma^p_k}[O(\log n)]$, $k \geq 2$, are defined similarly by replacing the $\mathrm{NP} = \Sigma^p_1$ oracle with an Σ^p_k oracle.

2.1 A brief review of Θ^p_2

The restriction of P^{NP} by imposing a logarithmic bound on the number of oracle calls was first considered by Papadimitriou and Zachos [43]. Later, this and related classes have been extensively studied in many papers, e.g. [47, 48, 20, 21, 37, 7, 6, 36, 2, 34], to mention a few.

Θ^p_2 is very robust and has many characterizations under different models of computation. In particular, the following are known. Let $\mathrm{P}^{\mathrm{NP}}_{tt}$ denote the closure of NP under polynomial-time truth table reductions.

Proposition 2.1 (see [48, 6, 38]) *$\Theta^p_2 = \mathrm{P}^{\mathrm{NP}}_{tt} = \mathrm{L}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}}[O(\log n)] = \mathrm{L}^{\mathrm{NP}}[O(\log n)] = \mathrm{P}^{\mathrm{NP}}_{\|} = \mathrm{L}^{\mathrm{NP}}_{\|} = \mathrm{P}^{\mathrm{NP}}_{\|[k]}$, for any $k \geq 1$.*

Many other characterizations exist, some of which are described in this paper.

Note that it is not known whether $\Delta^p_2 = \Theta^p_2$ implies P = NP, or, less strict, the collapse of PH. On the other hand, for the functional analogues, $F\Delta^p_2 = F\Theta^p_2$ implies P = NP [37]. Recent results indicate that the functional versions of the classes in Proposition 2.1 behave quite different from their decisional counterparts, cf. [34].

The class Θ^p_2 has natural complete problems, which often involve optimization and counting. We mention here some well-known examples.

- UOCSAT [36]: Given a SAT instance in CNF, is the largest subset of clauses that are simultaneously satisfiable unique ?
- ODD-CLIQUE [37, 47]: Given a graph, is the maximum clique size odd ?
- ODD-COLORING [47]: Given a graph, is its chromatic number odd?
- PARITY(A) [47, 48]: Let A be any NP-complete problem. Decide, given instances $I_1, \ldots, I_n$ of A, whether the number of all j such that $\chi(I_j) = 1$ (i.e., the number of Yes-instances I_j) is odd.

Somewhat surprisingly, PARITY(A) is Θ^p_2-hard even under the strong assertion that $\chi(I_j) = 0 \Rightarrow \chi(I_{j+1}) = 0$, for all $i = 1, \ldots, n-1$ [47]. This gives a powerful tool for proving Θ^p_2 lower bounds.

3 NP Dags and Trees

In this section, we survey recent results on problems and complexity classes definable by interdependent queries to an oracle in NP. How the NP queries depend on each other is specified by a directed graph G. In Section 3.1, we deal with acyclic (dags) G, and in Section 3.2 with trees G. Applications in modal and epistemic logic will be discussed in Section 6.

3.1 NP DAGs

We define classical propositional logic over the language $\mathcal{L}$ based on a countable set of propositional variables and the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\equiv$, and $\oplus$ (exclusive disjunction), and the constants $\top$ and $\bot$ for truth and falsity, respectively.

A truth value assignment maps each propositional variable to $\{true, false\}$. The truth values $true$ and $false$ will be identified with 1 and 0, respectively.

We divide a directed acyclic graph (DAG) into levels as follows. Level 0, the bottom level, consists of all vertices with no incoming arcs. Level $i+1$ consists of all vertices whose incoming arcs connect only to levels $\leq i$ and to at least one vertex of level i. Nodes without outgoing arcs are called *terminal nodes*. The *depth* of a DAG is equal to the number of its levels minus one. The level of a node a is denoted by $level(a)$. The depth of a DAG G is denoted by $depth(G)$.

Intuitively, an NP DAG consists of an acyclic network (circuit) of nodes, each representing a parametric query to an NP oracle, such that parts of the query string at node N depend on the oracle answers of N's immediate predecessor nodes. Thus a directed arc $\langle N_1, N_2\rangle$ in an NP DAG symbolizes that the oracle query at node N_2 depends on the oracle answer at node N_1, which is an *input node* for N_2. An oracle query at a node N can be evaluated as soon as the queries of all input nodes for N have been evaluated. Each NP DAG has a unique distinguished node called the *result node*. An NP DAG is positively answered iff the oracle query at its result node is positively answered.

Definition 3.1 *The Problem* DAGS(SAT) *is defined as follows.*
INSTANCE*: triple $\langle Var, G, F_r\rangle$ given by*

- *a set of propositional variables $Var = \{v_1, \ldots, v_n\}$, called* linking variables*;*
- *a DAG $G = \langle V, E\rangle$. The nodes $V = \{F_1, \ldots, F_n\}$ represent (not necessarily distinct) propositional formulas, which are identified with their names F_i. Each formula F_i has associated the linking variable v_i, also referred to by $[F_i]$, and contains the linking variables $\{v_j \mid \langle F_j, F_i\rangle \in E\}$; moreover, F_i may have "private" propositional variables not appearing elsewhere.*
- *a distinguished terminal node F_r called the result node.*

QUESTION: *Does F_r evaluate to true under the following inductively defined truth value assignment σ to $v_1, \ldots, v_n$: $\sigma(v_i)$ is true if the formula F_i' is satisfiable, where F_i' is obtained from F_i by replacing each v_j occurring in F_i by $\top$ if $\sigma(v_j) = true$ and by $\bot$ if $\sigma(v_j) = false$; otherwise, $\sigma(v_i)$ is false.*

An NP *DAG is an instance of* DAGS(SAT)*. The class* **DAGS**(**NP**) *consists of all decision problems polynomially transformable to* DAGS(SAT)*.*

Many well-known problems are in **DAGS**(**NP**). E.g., PARITY(SAT) is in this class, as it can be polynomially transformed into DAGS(SAT).

Indeed, translate each PARITY(SAT)-instance consisting of m SAT instances $F_1, \ldots, F_m$ into an NP DAG with m bottom nodes $F_1 \ldots, F_m$ and an additional node $F_r = v_1 \oplus v_2 \oplus \cdots \oplus v_m$ which encodes the parity function over $[F_1], \ldots, [F_m]$. F_r is the result node and the graph G of the DAGS(SAT)-instance has as arcs $(F_1, F_r), \ldots, (F_m, F_r)$. Note that this graph is a tree of depth 1.

Hence, DAGS(SAT) is Θ_2^p-hard. In fact, its complexity is slightly higher.

Theorem 3.1 ([26]) DAGS(SAT) *is* Δ_2^p *complete.*

Corollary 3.2 **DAGS**(**NP**)$= \Delta_2^p$.

The above theorem and its corollary are closely related to a recent result independently found by M. Vardi and reported in [31].

Definition 3.2 *Let* $\mathrm{DAGS_k}$(SAT) *be the* NP *dags whose depth is bounded by a constant* k*, and let* $\mathbf{DAGS_k}$(**NP**) *the class of problems that are polynomial-time transformable into such dags. Moreover, let* $\mathbf{DAGS^0}$(**NP**) $= \bigcup_k \mathbf{DAGS_k}(\mathbf{NP})$.

Theorem 3.3 ([26]) *For every* $k \geq 1$, $\mathrm{DAGS_k}$(SAT) *is* Θ_2^p*-complete, and* $\mathbf{DAGS_k}(\mathbf{NP}) = \Theta_2^p = \mathbf{DAGS^0}(\mathbf{NP})$.

Membership follows from Proposition 2.1 and the observation that any NP DAG in $\mathrm{DAGS_k}$(SAT) can be evaluated with $k+1$ rounds of parallel NP queries, and hardness since PARITY(SAT), as follows from above, is in $\mathbf{DAGS_1}(\mathbf{NP})$.

3.2 NP trees

An NP tree is an NP DAG which is a rooted tree such that each node reaches the root, i.e., arcs are directed bottom up.

Definition 3.3 TREES(SAT) *is the restriction of* DAGS(SAT) *to* NP *trees, and* **TREES**(**NP**) *the class of all problems polynomial-time transformable into* TREES(SAT)*.*

The main result of [26] is the following:

Theorem 3.4 ([26]) TREES(SAT) *is* Θ_2^p *complete.*

While Θ_2^p-hardness is easy (see above) the rather involved membership proof has two main steps. In the first step, a given NP tree T with n nodes is transformed in polynomial time into an equivalent NP DAG $D(T)$ of logarithmic depth which satisfies the following property: From an arbitrary node N in $D(T)$, no more than $2\lceil \log n \rceil$ other nodes in $D(T)$ are reachable via a directed path.

The transformation uses as an intermediate construct an NP *hypertree*, i.e., a tree whose (hyper-)nodes are Boolean combinations of parametric NP instances. First, T is transformed into a hypertree $H(T)$ such that each hypernode corresponds to a node of T. A *fusion* operation between connected hypernodes allows to merge two connected hypernodes into a single hypernode under certain conditions. By successive fusions, $H(T)$ is transformed in polynomial time into an equivalent hypertree $H'(T)$ of logarithmic depth. Finally, the resulting $H'(T)$ is transformed into the desired DAG $D(T)$. In the second step, advanced census techniques are used to evaluate the DAG $D(T)$ in polynomial time with a logarithmic number of queries to an NP oracle. Since $\mathrm{P}^{\mathrm{NP}}[O(\log n)] = \Theta_2^p$, membership in Θ_2^p follows. As an immediate corollary, we get:

Corollary 3.5 $\mathbf{TREES}(\mathbf{NP}) = \Theta_2^p$.

4 Relativized Boolean Circuits

Relativized Boolean circuits and the corresponding reductions were first studied in [10, 49]. Like NP DAGs, Boolean circuits relativized by an NP oracle describe sophisticated dependencies between NP queries. The main differences are:

- Circuits are *uniform*, in our case, *logspace-uniform*, models of computation. I.e., a language A is recognized by a family $\{C_n\}$ of circuits of a specific type if there exists a logspace machine which for each input size n generates a circuit C_n of this type that recognizes exactly all words of length n in A.
- Various types of circuits can be defined according to different parameters such as depth-bounds and fan-in bounds.

In this section, we review some recent results on AC and NC circuits relativized by an oracle in NP, which can be found in detail in [9, 42, 41].

4.1 Relativized AC

The (unrelativized) class $\mathbf{AC^i}$ consists of all problems solvable by logspace-uniform Boolean circuits with $p(n)$ gates and having depth $O(\log^i(n))$ for a polynomial p, and where the gates have unbounded fan-in. Here, the depth of circuit is the length of the longest path in it. In particular, $\mathbf{AC^0}$ has constant depth. Although the computational power of such circuits is limited, important problems such as evaluation of relational first-order formulas are in $\mathbf{AC^0}$.

Relativizations of $\mathbf{AC^0}$ with NP sets have been investigated in [10, 1], and of $\mathbf{AC^i}$, $i \geq 0$, with NP sets in [9]. Briefly,

Definition 4.1 *For any $k \geq 0$, denote by $\mathbf{AC^k}(\mathbf{NP})$ the set of all languages A recognizable by logspace uniform $\mathbf{AC^k}$ circuits with oracle gates for a set $B \in \mathrm{NP}$, where each oracle gate counts 1 in depth and size.*

In [9], a strong relationship between $\mathbf{AC^i}(\mathbf{NP})$ and polynomial time Turing machines with bounded calls to an NP oracle has been established.

Theorem 4.1 ([9]) *For every $k \geq 1$, $\mathrm{P}^{\mathrm{NP}}[O(\log^k n)] = \mathbf{AC^{k-1}}(\mathbf{NP})$.*

In particular, $\mathbf{AC^0}(\mathbf{NP}) = \Theta_2^p$. Intuitively, the evaluation of an $\mathbf{AC^0}(\mathbf{NP})$ circuit on an input can be trivially transformed into an NP DAG whose depth is bounded by a constant, and hence by Theorem 3.3 each problem in $\mathbf{AC^0}(\mathbf{NP})$ is in $\mathbf{DAGS^0}(\mathbf{NP})$. On the other hand, PARITY(SAT) can be easily expressed in $\mathbf{AC^0}(\mathbf{NP})$, which gives hardness.

Moreover, the following interesting results are shown in [9].

Theorem 4.2 ([9]) *For all $k \geq 1$, $\mathrm{P}^{\mathrm{NP}}[O(\log^k n)]$ is the class of languages which are reducible to* NP *with circuits of size $O(\log^k n)$.*

Theorem 4.3 ([9]) *There is a recursive set A such that for any integer $k \geq 1$, $\mathrm{P}^{\mathrm{NP}(A)}[O(\log^k n)] \subset \mathrm{P}^{\mathrm{NP}(A)}[O(\log^{k+1} n]$.*

Thus, the classes $\mathrm{P}^{\mathrm{NP}}[O(\log^k n)]$, $k \geq 1$, are separated by a recursive oracle.

4.2 $\mathbf{NC^i}(\mathbf{NP})$

The (unrelativized and logspace-uniform) class $\mathbf{NC^i}$ consists of all problems solvable by logspace-uniform constant-bounded fan-in Boolean circuits with $p(n)$ gates, having depth $O(\log^i(n))$ for a polynomial p.

For a relativized version of $\mathbf{NC^i}$, the notion of "constant-bounded fan-in" for circuits with oracle-gates must be redefined. Ideed, a fan-in bounded by a constant c is not meaningful in this context, as an NP oracle gate should get words of polynomial length as input (words of constantly many bits are trivially solvable in constant time). To settle this problem, one usually trades unbounded fan-in for additional depth [49, 35]. This is achieved by adopting that an oracle gate with m inputs contributes $\lceil \log m \rceil$ to the circuit depth. Intuitively, this reflects that any `and`- and `or`-gate with fan-in m can be replaced by an equivalent circuit of fan-in 2 having size polynomial in m and depth logarithmic in m. Applying this to oracle gates is a bit artificial, but seems to be most reasonable for generalizing $\mathbf{NC}$-circuits to the relativized setting.

Definition 4.2 *For any $k \geq 0$, denote by $\mathbf{NC^k}(\mathbf{NP})$ the set of all languages A recognizable by logspace uniform $\mathbf{NC^k}$ circuits with oracle gates for a set $B \in$ NP, where each oracle gate having fan-in m counts $\lceil \log m \rceil$ in depth and 1 in size.*

By results of [1], $\Theta_2^p \subseteq \mathbf{NC^1}(\mathbf{NP})$. Combining this with Theorem 4.1, we get $\mathbf{AC^0}(\mathbf{NP}) = \Theta_2^p \subseteq \mathbf{NC^1}(\mathbf{NP})$. Castro and Seara [9] asked whether the $\subseteq$ sign can be turned into an equality; recently, this was answered affirmatively:

Theorem 4.4 ([42, 41]) *For each integer k, $\mathbf{AC^k}(\mathbf{NP}) = \mathbf{NC^{k+1}}(\mathbf{NP})$.*

As an immediate corollary, we get:

Corollary 4.5 $\mathbf{AC^0}(\mathbf{NP}) = \Theta_2^p = \mathbf{NC^1}(\mathbf{NP})$.

The equality $\mathbf{AC^0}(\mathbf{NP}) = \mathrm{P}^{\mathrm{NP}}[O(\log n)] = \mathbf{NC^1}(\mathbf{NP})$ also follows as a corollary of Theorem 3.4. In fact, any $\mathbf{NC^1}$ DAG G with input w can be transformed in polynomial time into an equivalent $\mathbf{NP}$ tree, e.g. as follows.

Denote by $anc(N)$ the set of all nodes that reach N by a directed path and let $pred(N) = anc(N) - \{N\}$. Given an instance of $\mathbf{NC^1}(\mathbf{NP})$, transform it into an NP DAG G by converting each boolean node to a bogus NP-oracle node and by putting each input bit into a bottom node. If the result node N of G has at least two children C_1, C_2 with $anc(C_1) \cap anc(C_2) \neq \emptyset$, decouple all children C_i of N by first attaching to each C_i a private copy of the sub-DAG D_i spanned by $pred(C)$, and then deleting every D_i. Applying this procedure recursively to all children of N, the resulting DAG T is a tree equivalent to G. For each leaf-root path $N_1, \ldots, N_k = N$ in T, the sum $\sum_i \log(in(N_i))$, where $in(N_i)$ is the in-degree of N_i, is $O(\log n)$. Hence, the product $\prod_i in(N_i)$ is bounded by a polynomial. Thus, starting from the root, only polynomially many choices for a (reversed) path to a leaf exist. Hence T, has polynomial size and the transformation is polynomial.

5 Complete Problems in AI and Logic Programming

In this section, we present some known and new Θ_2^p-complete problems in AI and logic programming. Many further such problems can be found.

5.1 Theory revision and counterfactual reasoning

Theory revision deals with incorporating new information, given by a formula φ, into a logical theory (set of formulas) T. This is accomplished by an operator $\circ$, which, given T and φ, produces the revised theory $T \circ \varphi$. Many revision operators $\circ$ have been proposed which adhere to the principle of minimality of change, cf. [40, 14].

In this context, an important problem is deciding whether a formula ψ follows from T after revision by formula φ ($T \circ \varphi \models \psi$); this is also known as the *Ramsey Test* for evaluating counterfactual statements of the form "If φ would hold, then ψ would be true."

According to Dalal's operator $\circ_D$ [11] for propositional theories, the revision $T \circ_D \varphi$ is logically described by those models M of φ that result from some model M' of T by switching the truth value of as few atoms as necessary. The following result is known.

Theorem 5.1 ([14]) *Given propositional T, φ, and ψ, deciding $T \circ_D \varphi \models \psi$ is Θ_2^p-complete.*

To see the upper bound, one can compute the minimal number of switches to transform some model M' of T into a model M of φ in a binary search with an NP oracle, and then check with the oracle whether every model M of φ obtained this way satisfies φ. Hardness can be shown by a reduction of UOCSAT.

Further revision operators are now known to be Θ_2^p-complete, cf. [40, 39].

5.2 Null value problem

In the context of deductive databases, knowledge is often represented in clausal theories. A *disjunctive deductive database* (DDDB) D is a collection of clauses

$$A_1 \vee \cdots \vee A_n \leftarrow B_1, \ldots, B_m, \qquad n \geq 1, m \geq 0,$$

where the A_i and the B_j are atoms in a function-free language; it is *ground*, if no variables occur in the clauses.

Given a predicate letter R, D has a *null value on* R, if there exist distinct tuples of constants $\mathbf{c}_1, \ldots, \mathbf{c}_k$, $k \geq 2$, in the language, such that D logically implies the clause $C = R(\mathbf{c}_1) \vee \cdots \vee R(\mathbf{c}_k)$, but does not imply any proper subclause C' of C [4]. Intuitively, C describes the range of a null value on R.

The complexity of the null value problem is as follows.

Theorem 5.2 *Given a ground disjunctive deductive database D and a predicate R, deciding whether D has a null value in R is Θ_2^p-complete.*

Proof. (Sketch) For the membership part, we exploit the following observation. Let A be the set of all tuples $\mathbf{c}$ such that $D \not\models R(\mathbf{c})$, and let $C = R(\mathbf{c}_1) \vee \cdots \vee R(\mathbf{c}_k)$ where $A = \{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$. Then, it holds that D has a null value on R if and only if $D \models C$.

The problem can thus be solved in polynomial time with two rounds of parallel queries to a SAT oracle as follows. In the first round, check whether $D \cup \{\neg R(\mathbf{c})\}$ is satisfiable, i.e., $D \not\models R(\mathbf{c})$, for all $\mathbf{c}$; the result gives A. Then, construct C from A and check whether $D \cup \{\neg C\}$ is satisfiable, i.e., $D \not\models C$. Hence, the problem is in $\mathrm{P}^{\mathrm{NP}}_{\|[2]}$, which by Proposition 2.1 means it is in Θ_2^p.

To prove hardness, we provide a reduction from PARITY(SAT); w.l.o.g., in instances $I_1, \ldots, I_n$ all I_j are in CNF on disjoint sets of propositional variables and satisfy $\chi(I_j) = 0 \Rightarrow \chi(I_{j+1}) = 0$.

Introduce new propositional letters (0-ary predicates) a_j and constants c_j, $j = 1, \ldots, n$. For each $j = 1, \ldots, n$, set up a database D_i which contains the clauses of I_j, to each of which a_j is added, plus the clause $R(c_j) \leftarrow a_j$ if j is odd, and the clause $R(c_j) \vee R(c_{j-1}) \leftarrow a_j$, if j is even.

It is easy to see that $D_j \models a_j$ if and only if $\chi(I_j) = 0$. Hence, clearly for odd j, $D_j \models R(c_j)$ iff $\chi(I_j) = 0$, and for even j, $D_j \models R(c_j) \vee R(c_{j-1})$ iff $\chi(I_j) = 0$.

Let then $D = \bigcup_j D_j$. It can be shown D has a null value on R iff $D \models R(c_j) \vee R(c_{j-1})$ and $D \not\models R(c_j)$, $D \not\models R(c_{j-1})$ for some j iff the maximal j such that $\chi(I_j) = 1$ is odd. This proves Θ_2^p-hardness. ■

Notice that, unlike with other Θ_2^p-complete problems, no numbers or optimization criterion is explicit in the problem statement.

5.3 Logic programming semantics

The preferred answer set semantics [5] is an attempt to formally integrate priorities into the answer sets semantics of logic programming [22]. A propositional *extended logic program* (ELP) P is a collection of rules

$$A \leftarrow B_1, \ldots, B_n, \sim C_1, \ldots, \sim C_k,$$

where A, B_i, and C_j are classical propositional literals, i.e., a propositional atom p or a negated atom $\neg p$. The *reduct of* P by a set S of classical literals, denoted P^S, is the collection of rules obtained from P by

1. first deleting all rules in which a literal $\sim L$ occurs where $L \in S$, and then
2. removing all literals $\sim L$ from the remaining rules.

The set of consequences of a $\sim$-free program P, denoted by $Cn(P)$, is the smallest set of all classical literals which is closed under the rules of P and logically closed, i.e., either consistent or the set of all literals. Then, a set of literals S is an *answer set* of P, iff $S = Cn(P^S)$ holds.

For example, the program $P = \{\neg a \leftarrow \sim b,\ b \leftarrow \sim \neg a\}$ has two answer sets, namely $S_1 = \{\neg a\}$ and $S_2 = \{b\}$; program $P = \{\neg a \leftarrow,\ a \leftarrow\ \}$ has the inconsistent answer set $S = \{a, \neg a\}$; however, $P = \{a \leftarrow \sim a\}$ has no answer set.

In [5], priorities of rules are modeled through a linear order $<$ on the rules of P; formally, an ELP with priorities is thus an ordered set $(P, <)$.

Among the answer sets, strongly preferred answer sets are distinguished according to $<$ as follows. Call any program P in which no literals B_i occur *prerequisite-free*. For any ordered such ELP $(P, <) = r_1 < r_2 < \cdots < r_n$, let $C_<(P)$ be the smallest logically closed set of literals containing $\bigcup_i S_i$, where

$$S_0 = \emptyset,$$
$$S_i = \begin{cases} S_{i-1} \cup \{A\}, \text{if } r_i = A \leftarrow \sim C_1, \ldots, \sim C_k \text{ and } C_j \notin S_{i-1}, \text{ for all } j; \\ S_{i-1} \qquad\qquad \text{otherwise.} \end{cases}$$

The *dual reduct of* P by a set of literals S, denoted S^P, is defined as the collection of all rules obtained from P by

1. first deleting all rules with a "prerequisite" B_i such that $B_i \notin S$, and then
2. removing all prerequisites B_i from all remaining rules.

Definition 5.1 *An answer set A is* strongly preferred, *if $A = C_<(A^P)$.*

For an example, we construct for a collection of propositional clauses $\mathcal{C} = \{C_1, \ldots, C_m\}$ on variables $x_1, \ldots, x_n$ a program P and an order $<$ such that the strongly preferred answer sets of $(P, <)$ correspond to the satisfying truth assignments of C.

The program P involves the variables x_i, y_i, $i = 1, \ldots, n$, a, b, and *unsat*, and has $m + 4n + 3$ rules ordered as follows. The first $2n$ rules are

$$x_i \leftarrow x_i \ < \ y_i \leftarrow y_i;$$

the next $2n$ rules are

$$x_i \leftarrow \sim y_i \ < \ y_i \leftarrow \sim x_i, \quad i = 1, \ldots, n.$$

Next follows for each clause $C_i = L_{i,1} \vee \cdots \vee L_{i,k}$ a rule

$$unsat \leftarrow L^*_{i,1}, \ldots, L^*_{i,k},$$

where $L^*_{i,j} = x_\ell$ if $L_{i,j} = \neg x_\ell$, and $L^*_{i,j} = y_\ell$ if $L_{i,j} = x_\ell$. The last three rules are

$$a \leftarrow \sim b \ < \ b \leftarrow \sim a \ < \ b \leftarrow unsat, \sim b.$$

Notice that this program does not involve classical negation. It can be shown that the strongly preferred answer sets correspond 1-1 to the satisfying truth assignments of C. Hence, the following is easily established.

Theorem 5.3 *Given a finite ordered ELP $(P, <)$, deciding whether a strongly preferred answer set exists is* NP*-complete.*

As a program with answer sets may have no strongly preferred answers sets [5], the weaker concept of preferred answer set is introduced as follows.

Let the distance between two orders $<$ and $<'$, denoted $D(<, <')$, be the smallest number of transpositions (exchanges of two elements) needed to transform $<$ into $<'$. The *preference violation degree* of an answer set A for P w.r.t. $<$, denoted $pvd_{(P,<)}(A)$, is the minimal $D(<, <')$ over all $<'$ such that A is a strongly preferred answer set of $(P, <')$.

Definition 5.2 *An answer set A of P is a* preferred answer set of $(P, <)$ *iff $pvd_{(P,<)}(A) = pvd(P, <)$, where $pvd(P, <)$ is the minimal $pvd_{(P,<)}(A)$ over all answer sets A of P.*

In the example above, we have $pvd(P, <) = 0$ if C is satisfiable, and $pvd(P, <) = 1$ if C is unsatisfiable; indeed, if the rules $a \leftarrow \sim b$ and $b \leftarrow \sim a$ are swapped, then the resulting $(P, <')$ has always a strongly preferred answer set. If $pvd(P, <) = 0$, then no answer set contains $unsat$, while if $pvd(P, <) = 1$, then every answer set contains $unsat$. Based on this observation, one can show that computing a preferred answer set is Θ^p_2-hard.

For inference, we obtain the following result. A literal L *is inferred from* (P, L), denoted $(P, <) \vdash L$, iff L belongs to all preferred answer sets of $(P, <)$.

Theorem 5.4 *Deciding, given a finite ordered ELP $(P, <)$ and a literal L, whether $(P, <) \vdash L$ is Θ^p_2-complete.*

Proof. (Sketch) The problem is in Θ^p_2: $pvd(P, <)$ is computable in a binary search over $[0, n-1]$ using an NP oracle, by querying whether an answer set for P and an order $<'$ exist such that $D(<, <') \leq k$ and A is a strongly preferred answer set for $(P, <')$. Once $pvd(P, <)$ is known, check with a call to the NP oracle whether $L \in A$ for each answer set A such that $pvd_{(P,<)}(A) = pvd(P, <)$. Overall, this requires $O(\log n)$ oracle calls.

The hardness part is shown by a reduction from PARITY(SAT). Let $I_1, \ldots, I_n$ be SAT instances on disjoint sets of variables such that $\chi(I_j) = 0 \Rightarrow \chi(I_{j+1}) = 1$, for all $j = 1, \ldots, n-1$. For each I_j, construct an ordered ELP $(P_i, <_i)$. Let $P = \bigcup_i P_i \cup Q$, where $Q = \{odd \leftarrow \sim unsat_1, \ldots, \sim unsat_{2k+1}, unsat_{2k+2}, \ldots, unsat_n \mid 1 \leq 2k+1 \leq n\}$ in which odd is a new variable, and let $<$ be an order on P compatible with each $<_i$ and such that Q is a final segment of $<$.

Any preferred answer set A of $(P, <)$ contains $unsat_j$ iff $\chi(I_j) = 0$, for every I_j; hence, $odd \in A$ iff for some k, $I_1, \ldots, I_{2k+1}$ are the Yes-instances of $I_1, \ldots, I_n$. Thus, $(P, <) \vdash odd$ iff $I_1, \ldots, I_n$ is a Yes-instance of PARITY(SAT). ■

5.4 Stationary default logic

Default logic [45] is one of the most prominent formalisms for reasoning with incomplete information. It augments a classical theory W by a set of default rules of the form $\frac{\alpha\,:\,\beta}{\gamma}$, which are to be read "If α is provable and $\neg\beta$ is not provable, then γ is concluded." The meaning of a default theory $T = (W, D)$ is given by extensions, which are fixpoints of an operator Γ_T on sets of classical formulas defined as follows. $\Gamma_T(S)$ is the smallest set of formulas such that

- $S \subseteq \Gamma_T(S)$;
- $Th(\Gamma_T(S)) = \Gamma_T(S)$, i.e., $\Gamma_T(S)$ is deductively closed; and
- if $\frac{\alpha\,:\,\beta}{\gamma} \in D$, $\alpha \in \Gamma_T(S)$, $\neg\beta \notin S$, then $\gamma \in \Gamma_T(S)$.

A set of formulas E is an *extension* of T, if $E = \Gamma_T(S)$ holds. In general, no, one, or even several extensions E may exist; a formula φ is cautiously inferred from T, denoted $T \vdash \varphi$, if φ belongs to all extensions E. As shown in [25], deciding $T \vdash \varphi$ for given T and φ is Π_2^p-complete in the propositional case.

Several variants of default logic have been proposed in order to remedy semantical problems and the multiple extension problem. In [44], the operator $\Gamma_T^2(S) = \Gamma_T(\Gamma_T(S))$ is suggested to replace $\Gamma_T(S)$, and a *stationary extension* of T defined a set of formulas satisfying

$$E \subseteq \Gamma_T(E) \qquad \text{and} \qquad E = \Gamma_T^2(E) = \Gamma_T(\Gamma_T(E)).$$

Since Γ_T is antimonotone, Γ_T^2 is monotone and thus has a least fixpoint $lfp(\Gamma_T^2)$, which coincides with the least stationary extension of T. Notice that in general, T does not have a least standard extension.

The problem of cautiously inferring φ from all stationary extensions of T, denoted $T \vdash_{stat} \varphi$, is Δ_2^p-complete [27], and thus easier than $T \vdash \varphi$ unless PH collapses. Intuitively, $T \vdash_{stat} \varphi$ is equivalent to $\varphi \in lfp(\Gamma_T^2)$, and a representation of $lfp(\Gamma_T^2)$ is computable by the usual lfp construction in polynomial time with an NP oracle.

An important syntactical fragment of default logic are *normal default theories*, in which each default is *normal*, i.e., of form $\frac{\alpha\,:\,\beta}{\beta}$. For this fragment, inference has the following complexity.

Theorem 5.5 ([27]) *Let $T = (W, D)$ be a normal propositional default theory and φ a formula. Deciding whether $T \vdash_{stat} A$ is Θ_2^p-complete.*

Intuitively, this result is explained by the fact that $lfp(\Gamma_T^2)$ is reached in constantly many steps. Notice that also this problem has no numbers or optimization criterion explicit in its formulation.

5.5 Abduction

Abduction is a natural way of reasoning studied in depth by Ch.S. Peirce. Intuitively, it aims at finding an explanation for observations based on a background theory; for example, given b and the clause $a \supset b$, by abduction a is concluded. Abduction has many applications [15], of which diagnosis is most prominent.

More formally, let H and O be sets of propositional atoms, called hypotheses, and observation, respectively, and let T be a propositional theory. Then $E \subseteq H$ is called an *abductive explanation* for O, if $T \cup E$ is satisfiable, and $T \cup S \models O$; E is *parsimonious*, if no abductive explanation E' exists such that $|E'| < |E|$.

Note that different parsimonious explanations might exist. A major abductive reasoning is thus deciding whether a hypothesis h is *relevant*, i.e., belongs to some parsimonious explanation.

Theorem 5.6 ([15]) *Let sets H, O of propositional atoms, a set T of propositional formulas, and an atom $h \in H$ be given. Then, deciding whether h is relevant is Θ_3^p-complete; if T is a set of Horn clauses (i.e., clauses with at most one positive literal), then it is Θ_2^p-complete.*

Further Θ_k^p-completeness results occur in abduction where the background knowledge is captured by a logic program. In fact, in the framework of [16], Θ_k^p-complete relevance tests for $k = 2, 3$ have been exhibited.

6 Complete Problems in Modal Logic

In this section we review complexity results on modal and epistemic logics that are obtained as applications of the structural complexity results in Section 3.

6.1 Carnap's modal logic

Carnap's modal logic [8, 24], which we call **C** here, is a nonstandard modal logic substantially different from the better-known Lewis-systems. It provides a possibility operator $\Diamond$ and a necessity operator $\Box$, and has the usual syntax of propositional modal logic. A subformula $\Diamond\psi$ of a formula φ evaluates to *true* if ψ is a consistent formula, and a subformula $\Box\psi$ evaluates to *true* iff ψ is valid. Each modal subformula of φ is evaluated independently of its context in φ.

Validity in **C** can be formally defined as follows. Say that an occurrence of a subformula γ in a formula φ is *strict*, if it is not in the scope of a modal operator.

Definition 6.1 *Validity in logic* **C**, $\models_{\mathbf{C}} \varphi$, *is inductively defined as follows:*

- *If φ is a modal-free formula, then $\models_{\mathbf{C}} \varphi$ iff φ is a tautology;*
- *If $\varphi = \Box\psi$ then $\models_{\mathbf{C}} \varphi$ iff ψ^+ is a tautology;*
- *If $\varphi = \Diamond\psi$ then $\models_{\mathbf{C}} \varphi$ iff ψ^+ is satisfiable;*
- *In all other cases, $\models_{\mathbf{C}} \varphi$ iff φ^+ is a tautology,*

where ψ^+ (φ^+) results from ψ (φ) by replacing each strict occurrence of subformulas of the form $\Diamond\mu$ or $\Box\mu$ by true if $\models_{\mathbf{C}} \mu$ and by false otherwise.

Intuitively, $\Box$ is a strong provability operator which expresses $\models_{\mathbf{C}}$ within the object language. $\Box\varphi$ is valid (and provable) if φ is provable. On the other hand, if φ is not provable, then $\neg\Box\varphi$ is valid (and provable). Accordingly, in $\mathbf{C}$ each formula φ of type $\Box\psi$ or $\Diamond\psi$ is *determinate*, i.e., either $\models_{\mathbf{C}} \varphi$ or $\models_{\mathbf{C}} \neg\varphi$ holds; as usual, $\Diamond$ can be seen shorthand for $\neg\Box\neg$. E.g., the formulas $\Diamond p$, $\Diamond\neg p$, $\neg\Box p$, $\Box q \to \Box r$ are all valid in $\mathbf{C}$; none of them is valid in any standard system of modal logic.

The next theorem tells the complexity of validity testing in logic $\mathbf{C}$.

Theorem 6.1 ([26]) *Given a modal formula φ, deciding $\models_{\mathbf{C}} \varphi$ is Θ_2^p complete.*

This theorem follows from Theorem 3.4 by exploiting the similarity between NP trees and the standard formula trees. Note that the Boolean closure of NP, corresponds to the fragment of $\mathbf{C}$ having modal nesting depth 1.

The validity problem for logic $\mathbf{C}$ is an interesting Θ_2^p-complete problem, because it is simple and does not involve any numerical or optimality property.

6.2 Zero-One laws for Kripke structures

The 0-1-law for classical predicate logic [23, 19] states that every property definable without function symbols is either true with probability 1 (almost surely true) or false with probability 1 (almost surely false) on finite structures. Analogous 0-1-laws for various types of Kripke structures have recently been obtained by Halpern and Kapron [31], by establishing relationships to Carnap's logic.

Let `ALL` (resp, `R`; `RT`; `RST`) denote the set of all Kripke structures where the accessibility relation is arbitrary (resp., reflexive; reflexive and transitive; reflexive, symmetric and transitive). Note that the sets `ALL`, `R`, `RT`, and `RST` characterize the modal logics **K**, **T**, **S4**, and **S5**, respectively.

Let $S \in \{\mathtt{ALL}, \mathtt{R}, \mathtt{RT}, \mathtt{RST}\}$ be a class of Kripke structures. For any modal formula φ, let $\nu_n^S(\varphi)$ express the ratio between the number of structures in S with state-space $\{1, \ldots, n\}$ satisfying φ over the number of *all* structures in S with state space $\{1, \ldots, n\}$. (For a more precise definition of $\nu_n^S(\varphi)$ refer to [31].) Let $\nu^S(\varphi) = \lim_{n\to\infty} \nu_n^S(\varphi)$. Thus $\nu^S(\varphi)$ expresses the asymptotic probability that the formula φ is satisfied by an arbitrarily chosen Kripke structure in S, when the state space of the Kripke structure is finite but grows towards infinity.

The 0-1-law for structure validity in S holds if for all modal formulas φ, we have either $\nu^S(\varphi) = 0$ or $\nu^S(\varphi) = 1$. We say that a formula φ is *almost surely S-valid* if $\nu^S(\varphi) = 1$. Intuitively, φ is almost surely S-valid if the probability that φ is valid in an arbitrarily chosen S Kripke-structure is 1.

Halpern and Kapron [31] have shown that the 0-1-law for structure validity holds for each $S \in \{\mathtt{ALL}, \mathtt{R}, \mathtt{RT}, \mathtt{RST}\}$, and have characterized the set of almost-sure formulas for S and investigated their complexity.

Almost-sure `RST`-validity is equivalent to **S5**-provablity and thus co-NP-complete. For `ALL` and `R` the following theorem was shown.

Proposition 6.2 ([31]) *A propositional modal formula φ is almost surely `ALL`-valid iff φ is almost surely `R`-valid iff $\models_{\mathbf{C}} \varphi$.*

(Note that [31] does not explicitly refer to **C**, but independently reconstructs it for ad hoc use.) This result gives a new very attractive meaning to logic **C**, which was *almost surely* not foreseen by Carnap. The next result relates almost-sure structure validity in **S4** to logic **C**.

Proposition 6.3 ([31]) *Almost-sure* `RT`*-validity of formula φ is polynomial-time transformable into almost-sure* `ALL`*-validity of a formula ψ.*

By Propositions 6.2,6.3 and by Theorem 6.1, we thus get a precise complexity characterization of almost-sure structure validity in **K**, **T**, and **S4**.

Theorem 6.4 *Deciding almost-sure structure validity for modal logics* **K**, **T**, *and* **S4** *is Θ_2^p-complete.*

6.3 On a further open problem by Halpern and Kapron

As reported in [31], M. Vardi has shown that deciding almost-sure structure validity for modal logics **K** and **T** is Δ_2^p-complete if formulas are represented succinctly as DAGs, i.e., if multiple occurrences of common subformulas may be represented by a single node. This result in essence corresponds to the (independently obtained) Theorem 3.1 stating that DAGS(SAT) is Δ_2^p-complete.

For **S5**, the problem is reported co-NP-complete in [31], while for **S4**, it remained open; a Θ_2^p-hardness and a Δ_2^p upper bound are provided in [31], and the authors note this gap [31, p. 176]. We close it by establishing Θ_2^p-completeness.

Theorem 6.5 *Deciding almost-sure validity for modal logic* **S4** *under DAG representation is Θ_2^p-complete.*

Proof. (Sketch) Hardness was shown in [31]. For membership, as shown in [31], almost-sure validity of a formula φ in **S4** is equivalent to $\models_{\mathbf{C}} \psi$, where $\psi = \varphi^0 \wedge \varphi^1 \wedge \varphi^2$, and φ^i, for $i = 0, 1, 2$, are inductively defined as follows:

- $p^i = p$ for any propositional atom p;
- $(\varphi \wedge \psi)^i = \varphi^i \wedge \psi^i$;
- $(\neg\varphi)^i = \neg(\varphi^i)$;
- $(\Box\varphi)^i = \begin{cases} \varphi^0 & \text{if } i = 0, \\ \varphi^i \wedge \Box(\varphi^{i-1}) & \text{if } i = 1, 2. \end{cases}$

Notice that in φ^i the nesting depth of $\Box$ is bounded by i.

Given a DAG for a formula φ, it is not difficult to show that a DAG for each φ^i, $i = 0, 1, 2$ can be constructed from it in polynomial time (in fact, in logspace). A simple (though not optimal) method is to proceed inductively from subformulas μ, where the only critical case is $(\Box\mu)^i$; here, make a copy μ^* of μ (i.e., of its DAG), create an $\wedge$-node as the root of the DAG for $(\Box\mu)^i$, whose sons are the root of the DAG for μ^i and a $\Box$-node linked to $(\mu^*)^{i-1}$. An easy calculation shows that the size of the DAG for φ^i is $O(n^i s)$, where n is the number of $\Box$ in φ and s is the size of the DAG for φ.

The resulting DAGs can be easily combined to a DAG D_ψ for ψ. Notice that in D_ψ, any path contains at most two $\Box$-nodes; for deciding $\models_{\mathbf{C}} \psi$, D_ψ can thus be easily transformed into an NP DAG whose depth is bounded by a constant, by using a node $F_{\Box\mu}$ for each subformula $\Box\mu$, a node F_ψ for ψ, and a result node F_r. Hence, by Theorem 3.3 deciding $\models_{\mathbf{C}} \psi$ is in Θ_2^p, and the result follows. ∎

7 Logical Characterizations of Θ_2^p

The efforts in linked complexity theory and logic has fostered the field of *descriptive complexity theory* [33, 13, 30], which aims at describing the expressive power of logics and query languages via complexity classes.

For any finite signature σ of relation symbols, every sentence Φ in a logical language $\mathcal{L}$ on σ naturally defines a property over any class $\mathcal{K}$ of structures for σ, given by the collection of those structures in $\mathcal{K}$ on which Φ is true. Here, a property is any subclass of $\mathcal{K}$; it is generic, if it is closed under isomorphisms. E.g., 3-colorability of graphs is a generic property.

A language $\mathcal{L}$ is said to *capture* a complexity class $\mathbf{C}$ over $\mathcal{K}$, if evaluating each fixed formula in $\mathcal{L}$ on any structure in $\mathcal{K}$ is in $\mathbf{C}$, and if every generic property over $\mathcal{K}$ that is recognizable in $\mathbf{C}$ is definable in $\mathcal{L}$; see [33, 30] for details. By default, $\mathcal{K}$ is the class $\mathcal{F}in$ of all finite structures.

Note that capturing of $\mathbf{C}$ by $\mathcal{L}$ is a stronger result than $\mathbf{C}$-completeness of evaluating fixed formulas from $\mathcal{L}$. The latter means that *some* $\mathbf{C}$-hard property is definable in $\mathcal{L}$, while capturing means that *all* (not only hard) $\mathbf{C}$-properties are. In general, $\mathcal{L}$ may allow to define even non-recursive properties but fail to define very simple ones.

A seminal result was Fagin's Theorem [18] stating that existential second-order logic (Σ_1^1) captures NP. Many important complexity classes can be characterized by variants of first-order logic or higher order logics.

One such variant is the extension by *generalized quantifiers*. Generalized quantifiers (see e.g. [13, 28]) are devices for expressing higher order properties that are usually not first-order definable. The semantics of a generalized quantifier Q is given by the set of structures on which Q evaluates to true.

A prominent example are Henkin (or branching) quantifiers H [32]. These are two-dimensional quantifiers for expressing subtle interdependencies among $\exists$- and $\forall$-quantified variables that are not expressible in standard logic. E.g., the formula $\binom{\forall x_1\ \exists y_1}{\forall x_2\ \exists y_2}\varphi(x_1, x_2, y_1, y_2)$, with a leading Henkin quantifier says that there exist functions f_1 and f_2 such that for each x_1 and x_2, $\varphi(x_1, x_2, f_1(x_1), f_2(x_2))$ is true. The salient feature here is that $y_1 = f(x_1)$ does not depend on x_2 and $y_2 = f(x_2)$ does not depend on x_1.

For testing graph properties, one can add a *3-colorability* operator $3COL$ to first-order logic, where $3COLx, y\varphi(x, y)$ means that the graph on the domain elements whose edges are described by $\varphi(x, y)$ is 3-colorable. Similarly, a *Hamiltonian path* quantifier HAM expressing that a graph has a Hamiltonian path between specified vertices, could be added.

Usually, one considers not only a single quantifier Q of a given type, but a *vectorized* family $\mathbf{Q} = \{Q^1, Q^2, \ldots\}$ of quantifiers, where in Q^i instead of single elements i-tuples of elements occur. For example, $3COL^2$ applies to a graph whose vertices are 2-tuples of elements. Denote by FO ($\mathbf{Q}$) the extension of first-order logic with Q (see e.g. [13, 28] for precise definitions).

The class Θ_2^p has the following logical characterizations. Let $\mathcal{O}$ be the class of ordered finite structures, i.e., the finite structures equipped with a linear order on the elements.

Theorem 7.1 ([46]) *Over $\mathcal{O}$, both* FO(**HAM**) *and* FO(**3COL**) *capture Θ_2^p.*

This result was generalized as follows. Denote with $\mathbf{Q}^+$ the fragment of FO($\mathbf{Q}$) containing all formulas $Q\mathbf{x}\varphi(\mathbf{x})$, where $Q \in \mathbf{Q}$ and φ is first-order.

Theorem 7.2 ([28]) *For any $\mathbf{Q}$, if $\mathbf{Q}^+$ captures* NP *over $\mathcal{O}$, then* FO($\mathbf{Q}$) *captures Θ_2^p over $\mathcal{O}$.*

This result and the fact that $\mathbf{H}^+$ captures NP on $\mathcal{F}in$ settles the open issue of the expressive power of FO($\mathbf{H}$) in [3].

Corollary 7.3 ([28]) FO($\mathbf{H}$) *captures Θ_2^p on $\mathcal{O}$.*

Further logics capturing Θ_2^p on $\mathcal{O}$ are e.g. the first order closure of Σ_1^1 [29] or stratified modular logic programming with stable negation [17].

What about $\mathcal{F}in$? Capturing Θ_2^p there is difficult or even impossible:

Theorem 7.4 ([12]) *If any family $\mathbf{Q}$ of* NP*-evaluable quantifiers* FO($\mathbf{Q}$) *captures Θ_2^p on $\mathcal{F}in$, then the Boolean hierarchy over* **NEXPTIME** *collapses.*

This means that the issue of a logic for Θ_2^p (in the sense of [30]) is a nontrivial open problem, akin to the issue of a logic for P [30].

8 Conclusions

We have considered the class Θ_2^p and surveyed a number of different characterizations and complete problems of this class. In particular, we focussed on problems in AI and logic programming. It appears that this class is of particular relevance for characterizing the complexity many problems in applications, and provides interesting insights in their computational nature.

The rich variety of characterizations and the robustness of Θ_2^p support the view that this class and its natural generalizations Θ_k^p should be considered as constitutional classes of the PH, as suggested in [48].

However, this extension of PH suffers from one minor problem that remains to be resolved: While any collapse of regular neighbored members of the PH would imply a collapse of the entire PH, no such consequence is known for a collapse of Θ_2^p and Δ_2^p, or, more general, for Θ_k^p and Δ_k^p, $k \geq 2$.

Also the consequences of a collapse of the hierarchy $\mathrm{P}^{\mathrm{NP}}[O(\log^k n)]$, $k \geq 1$ ($= \mathbf{AC^{k-1}(NP)} = \mathbf{NC^k(NP)}$) within Δ_2^p are not clear. Moreover, it would be interesting to know natural complete problems for its classes from the realm of applications, for any $k \geq 2$.

References

1. E. Allender and C. Wilson. Width-bounded Reducibility and Binary Search over Complexity Classes. In *Proc. IEEE Conf. Structure in Complexity Theory*, pp. 122–129, 1990.
2. R. Beigel. Bounded Queries to SAT and the Boolean Hierarchy. *TCS*, 84(2):199–223, 1991.
3. A. Blass and Y. Gurevich. Henkin Quantifiers and Complete Problems. *Annals of Pure and Applied Logic*, 32:1–16, 1986.
4. P. A. Bonatti and T. Eiter. Querying Disjunctive Databases Through Nonmonotonic Logics. *TCS*, 160:321–363, 1996.
5. G. Brewka. Preferred Answer Sets. In *Proc. Workshop on Logic Programming and Knowledge Representation*, October 1997, forthcoming.
6. S. Buss and L. Hay. On Truth-Table Reducibility to SAT. *Information and Computation*, 91:86–102, 1991.
7. J.-Y. Cai, *et al.* The Boolean Hierarchy I: Structural Properties. *SIAM J. Computing*, 17(6):1233–1252, 1988.
8. R. Carnap. *Meaning and Necessity.* The University of Chicago Press, 1947.
9. J. Castro and C. Seara. Characterizations of Some Complexity Classes Between Θ_2^p and Δ_2^p. In *Proc. STACS '92*, LNCS 577, pp. 305–317, 1992.
10. A. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM J. Computing*, 13:423–439, 1984.
11. M. Dalal. Investigations into a Theory of Knowledge Base Revision: Preliminary Report. In *Proc. AAAI '88*, pages 475–479, 1988.
12. A. Dawar, G. Gottlob, and L. Hella. Capturing Relativized Complexity Classes without Order. *Mathematical Logic Quarterly*, to appear.
13. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1995.
14. T. Eiter and G. Gottlob. On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. *Art. Intelligence*, 57(2–3):227–270, 1992.
15. T. Eiter and G. Gottlob. The Complexity of Logic-Based Abduction. *JACM*, 42(1):3–42, January 1995.
16. T. Eiter, G. Gottlob, and N. Leone. Abduction From Logic Programs: Semantics and Complexity. *TCS*, 1997.
17. T. Eiter, G. Gottlob, and H. Veith. Modular Logic Programming and Generalized Quantifiers. In *Proc LPNMR '97*, LNCS, 1997, forthcoming.
18. R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp, editor, *Complexity of Computation*, pp. 43–74. AMS, 1974.
19. R. Fagin. Probabilities on Finite Models. *J. Symbolic Logic*, 41:50–58, 1976.
20. W. Gasarch. The Complexity of Optimization Functions. Technical Report 1652, Dept. of Computer Science, University of Maryland, 1986.
21. W. Gasarch, L. A. Hemachandra, and A. Hoene. On Checking Versus Evaluation of Multiple Queries. In *Proc. MFCS '90*, LNCS 452, pp. 261–268, 1990.
22. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
23. Y. Glebskii *et al.* Range and Degree of Realizability of Formulas in the Restricted Predicate Calculus. *Kibernetika*, pp. 17–28, 1969.
24. G. Gottlob. Survey on a Carnapian Extension of S5. In E. Orlowska, editor, *Logic At Work, Essays Dedicated to the Memory of Helena Rasiowa.* To appear.
25. G. Gottlob. Complexity Results for Nonmonotonic Logics. *J. Logic and Computation*, 2(3):397–425, 1992.

26. G. Gottlob. NP Trees and Carnap's Modal Logic. *JACM*, 42(2):421–457, 1995. Abstract Proc. FOCS '93, 42–52.
27. G. Gottlob. The Complexity of Propositional Default Reasoning Under the Stationary Fixed Point Semantics. *Information and Computation*, 121(1):81–92, 1995.
28. G. Gottlob. Relativized Logspace and Generalized Quantifiers over Ordered Finite Structures. *J. Symbolic Logic*, 1997. Abstract Proc. IEEE LICS '95, 65–78.
29. G. Gottlob, N. Leone, and H. Veith. Second-Order Logic and the Weak Exponential Hierarchies. In *Proc. MFCS '95*, LNCS 969, pp. 66–81, 1995. Full paper CD-TR 95/80, Information Systems Dept, TU Wien.
30. Y. Gurevich. Logic and the Challenge of Computer Science. In E. Börger, editor, *Trends in Theoretical Computer Science*, ch 1. Computer Science Press, 1988.
31. J. Halpern and B. Kapron. Zero-One Laws for Modal Logic. *Annals of Pure and Applied Logic*, 69:157–193, 1994. Abstract Proc. IEEE LICS '92.
32. L. Henkin. Some Remarks on Infinitely Long Formulas. In *Infinitistic Methods, Proc. Symp. on Foundations of Mathematics*, pp. 167–183. Warsaw, Panstwowe Wydawnictwo Naukowe and Pergamon Press, 1961.
33. N. Immerman. *Descriptive Complexity Theory.* 1997, Springer, to appear.
34. B. Jenner and J. Toran. Computing Functions with Parallel Queries to NP. *TCS*, 141:175–193, 1995.
35. D. S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen (ed), *Handbook of TCS* A, ch. 2, 1990.
36. J. Kadin. $P^{NP[O(\log n)]}$ and Sparse Turing-Complete Sets for NP. *JCSS*, 39:282–298, 1989.
37. M. Krentel. The Complexity of Optimization Problems. *JCSS*, 36:490–509, 1988.
38. R. Ladner and N. Lynch. Relativization Questions about Logspace Computability. *Mathematical Systems Theory*, 10:19–32, 1976.
39. P. Liberatore. The Complexity of Iterated Belief Revision. In *Proc. ICDT '97*, LNCS 1186, 276–290, 1997.
40. B. Nebel. How Hard is it to Revise a Belief Base ? TR No. 83, Inst. für Informatik, Univ. Freiburg, Germany, August 1996. In: *Handbook on Defeasible Reasoning and Uncertainty Management Systems. Volume 2: Belief Change*, D. Dubois and H. Prade (eds), to appear.
41. M. Ogihara. Equivalence of $\mathrm{NC^k}$ and $\mathrm{AC^{k-1}}$ closures of NP and other classes. *Information and Computation*, 120:55–58, 1995.
42. M. Ogiwara. $\mathbf{NC^k(NP)} = \mathbf{AC^{k-1}(NP)}$. In *Proc. STACS '94*, LNCS 775, pp. 313–324, 1994.
43. C. Papadimitriou and S. Zachos. Two Remarks on the Power of Counting. In *Proc. 6th GI Conf. on TCS*, LNCS 145, pp. 269–276, 1983.
44. H. Przymusinska and T. Przymusinski. Stationary default extensions. *Fundamenta Informaticae*, 21(1/2):67–87, 1994.
45. R. Reiter. A Logic for Default Reasoning. *Artificial Intelligence*, 13:81–132, 1980.
46. I. Stewart. Logical Characterizations of Bounded Query Classes, I + II. *Fundamenta Informaticae*, 18:65–92 + 93–105, 1993.
47. K. Wagner. More Complicated Questions about Maxima and Minima, and Some Closures of NP. *TCS*, 51:53–80, 1987. Abstract *Proc. ICALP '86*, 434–443.
48. K. Wagner. Bounded Query Classes. *SIAM J. Computing*, 19(5):833–846, 1990.
49. C. Wilson. Relativized NC. *Mathematical Systems Theory*, 20:13–29, 1987.

This article was processed using the LaTeX macro package with LLNCS style