Ann Math Artif Intell (2007) 51:123–165
DOI 10.1007/s10472-008-9086-5

Complexity results for answer set programming with bounded predicate arities and implications

Thomas Eiter · Wolfgang Faber · Michael Fink · Stefan Woltran

Published online: 26 February 2008
© Springer Science + Business Media B.V. 2008

Abstract Answer set programming is a declarative programming paradigm rooted in logic programming and non-monotonic reasoning. This formalism has become a host for expressing knowledge representation problems, which reinforces the interest in efficient methods for computing answer sets of a logic program. The complexity of various reasoning tasks for general answer set programming has been amply studied and is understood quite well. In this paper, we present a language fragment in which the arities of predicates are bounded by a constant. Subsequently, we analyze the complexity of various reasoning tasks and computational problems for this fragment, comprising answer set existence, brave and cautious reasoning, and strong equivalence. Generally speaking, it turns out that the complexity drops significantly with respect to the full non-ground language, but is still harder than for the respective ground or propositional languages. These results have several implications, most importantly for solver implementations: Virtually all currently available solvers have exponential (in the size of the input) space requirements even for programs with bounded predicate arities, while our results indicate that for those programs polynomial space should be sufficient. This can be seen as a manifestation of the "grounding bottleneck" (meaning that programs are first instantiated and

Some results in this paper have been presented in preliminary form at KR 2004 [15].

T. Eiter · M. Fink (✉) · S. Woltran
Institute of Information Systems, Vienna University of Technology, Vienna, Austria
e-mail: michael@kr.tuwien.ac.at

T. Eiter
e-mail: eiter@kr.tuwien.ac.at

S. Woltran
e-mail: stefan@kr.tuwien.ac.at

W. Faber
Department of Mathematics, University of Calabria, Rende, Italy
e-mail: faber@mat.unical.it

then solved) from which answer set programming solvers currently suffer. As a final contribution, we provide a sketch of a method that can avoid the exponential space requirement for programs with bounded predicate arities.

Keywords Answer set programming · Computational complexity

Mathematics Subject Classifications (2000) 68N17 · 68Q17

1 Introduction

In the last years, *answer set programming* (ASP) [36, 43, 51, 56], also called A-Prolog [3, 26], has emerged as a declarative programming paradigm which has its roots in logic programming and non-monotonic reasoning. It is well-suited for modeling and solving problems which involve common sense reasoning, and has been fruitfully applied to a range of applications including data integration, configuration, diagnosis, reasoning about actions and change, and many others; see [68] for a recent survey of applications. Furthermore, a number of extensions of the ASP core language, which goes back to the seminal paper by Gelfond and Lifschitz [27], have been developed, which aim at increasing the expressiveness of the formalisms and/or providing convenient constructs for application-specific problem representation; see e.g. [52] for a more recent account of such extensions.

The most important and natural such extension is the usage of disjunction in the head of rules, leading to disjunctive ASP. The semantics of programs with disjunctive rule heads goes back to the pioneering work of Jack Minker, who was the first to consider this construct in the context of logic programming and databases. He proposed to consider the minimal Herbrand models of a logic program with disjunctive rule heads as the semantics of such programs, and formulated the generalized closed world assumption (GCWA) [44] in order to overcome problems of Reiter's closed world assumption (CWA) [58] over disjunctive databases. This was complemented with SLI-resolution for handling indefinite clauses [50]. Since then, Jack Minker was driving the research on disjunctive logic programming and made many further important contributions, including [23, 47, 61, 62]. A first climax of his work was the early state-of-the-art monograph [40], which was complemented with several influential surveys [45, 46, 48, 49]. Today, disjunctive ASP is, besides Core ASP, the most prominent class of ASP and of disjunctive logic programming alike.

The basic idea of ASP is to encode solutions to a problem into the intended models of a non-monotonic logic program, in a way such that the solutions are described in terms of rules and constraints rather than that a concrete algorithm is specified how to single out the solutions. The problem encoding is then fed into an ASP solver, which computes some or multiple answer set(s) of the program, from which the solutions of the problem can easily be read off. Alternatively, ASP-solvers may also be used to answer queries on the given problem encodings. Advanced ASP solvers such as Smodels, DLV, GnT, Cmodels, Clasp, or ASSAT (see Asparagus homepage: http://asparagus.cs.uni-potsdam.de/), are able to deal with large problem instances; a recent demonstration effort of the potential of ASP was made at the ASP solver competition [24] which took place within the latest edition of LPNMR in May 2007.

The success of ASP is based on a variety of diverse sophisticated algorithms and techniques for evaluating non-monotonic logic programs, which in turn draw from

results about the computational complexity of reasoning tasks from such programs. The latter is quite well understood, and detailed pictures of the complexity for ASP and major extensions are given in [4, 5, 10, 13, 35, 42], for ground (propositional) and non-ground programs (with variables, but without function symbols, i.e., the Datalog case); see [8] for a survey. Consistency checking of non-ground (disjunctive) ASP programs has complexity NEXP^{NP}, i.e., non-deterministic exponential time with an oracle for NP, and thus provides a powerful host for problem solving. In the propositional case, the complexity is lowered by one exponential, and is Σ_2^P-complete. The resource requirements of current implementations, with DLV being the foremost of the disjunctive ASP systems, match these theoretical worst case bounds, and use at most exponential space in the non-ground case but only polynomial space in the ground case.

In this paper, we consider non-ground programs under the restriction that the arities of predicates are bounded by some constant. This restriction is very relevant in practice, since often only a few arguments in a predicate are sufficient to express relationships between objects. Furthermore, often complex relationships can be broken up by splitting the predicates into smaller predicates.

As it turns out, for programs with bounded arities the complexity is (much) lower than for general programs. In fact, we show that such programs have complexity within the polynomial hierarchy, and are thus far cheaper to evaluate than unrestricted programs.

Example 1 Consider the problem of reachability over paths of length k in a directed graph. This problem can be represented by an ASP program in the following way, assuming that each arc (v_1, v_2) of the input graph is given by $e(v_1, v_2)$:

$$p_k(X_1, X_k) \text{ :- } e(X_1, X_2), \ldots, e(X_{k-1}, X_k).$$
$$reachable(X, Y) \text{ :- } p_k(X, Y).$$
$$reachable(X, Y) \text{ :- } reachable(X, Z), p_k(Z, Y).$$

Note that the first rule will in general give rise to an exponential number (in k) of body literals that all give rise to the same head atom. For instance, for a graph as depicted in Fig. 1, where $k = 2n + 1$, there are 2^n paths from v_1 to v_{n+1}, and therefore 2^n different ways for deriving $p_k(v_1, v_{n+1})$.

If the graph is given as an input, current ASP grounding methods will not generate ground versions of the first rule as an optimization. However, if the graph depends on some nondeterministic subprogram (for instance if this property is to be verified on subgraphs of an input graph), these optimizations can not be applied, and an

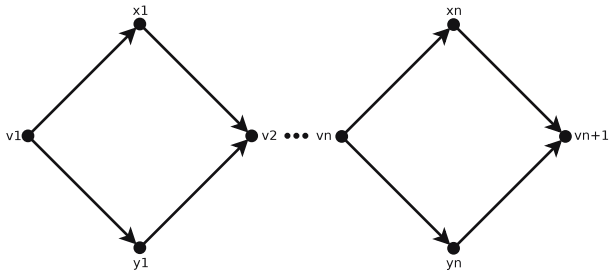

Fig. 1 Example graph

exponential number of ground rules will be created (cf. Section 6.1), while the results in this paper indicate that this is not necessary.

In this paper, we study the complexities of the following decision problems:

Answer set existence:	Given a program $\mathcal{P}$, decide whether $\mathcal{P}$ has some answer set.
Brave reasoning:	Given a program $\mathcal{P}$, and a ground literal a, decide whether a is true in some answer set of $\mathcal{P}$.
Cautious reasoning:	Given a program $\mathcal{P}$, and a ground literal a, decide whether a is true in all answer sets of $\mathcal{P}$.
Strong equivalence:	Given programs $\mathcal{P}$ and $\mathcal{Q}$, decide whether, for each program $\mathcal{R}$ it holds that $\mathcal{P} \cup \mathcal{R}$ and $\mathcal{Q} \cup \mathcal{R}$ have the same answer sets.

This enumeration includes the three canonical reasoning tasks (answer set existence, brave reasoning, and cautious reasoning) which have been amply considered in the literature, as well as strong equivalence [37]. We consider the fourth task, deciding strong equivalence, to be a central decision problem due to its fundamental role for a replacement property of logic programs. In particular, whenever $\mathcal{P}$ occurs as a subprogram in the context of another program, we can replace $\mathcal{P}$ without changing the semantics (i.e., without changing the answer sets of the overall program) only by programs which are strongly equivalent to $\mathcal{P}$. Note that due to the non-monotonic semantics such a replacement property cannot be obtained by just comparing the answer sets of the subprograms which are subject to replacement. The replacement property is important for modularization and optimization of logic programs, and its semantic and complexity properties have been investigated for ground and non-ground programs [16, 18, 19, 38, 55, 66].

Our main contributions and results on the above issues are briefly summarized as follows:

- We provide a complexity characterization of deciding answer set existence, brave reasoning, and cautious reasoning for programs with bounded arities. The analysis covers the most commonly considered syntactically restricted classes of programs, namely stratified negation [2] and head-cycle-free disjunction [4], as well as weak constraints [7]. Roughly speaking, it turns out that the complexities of these tasks are one level above the complexities of the corresponding problems for propositional programs, and range from NP resp. co-NP to Δ_4^P. Intuitively, this is explained by the fact that models of programs with bounded arities are of polynomial size, while checking whether an interpretation is closed under the rules of the program is intractable (more precisely, co-NP-complete).
- We also provide a complexity characterization of deciding strong equivalence of programs with bounded arities. As it turns out, this problem is Π_2^P-complete for almost all classes of programs which we consider, as compared to co-NEXP-completeness in the general case (resp. co-NP-completeness for the ground case). As a by-product, we provide a novel characterization for deciding strong equivalence between programs containing weak constraints.
- Based on our results and other results on answer set programming in the literature, we sketch how non-ground programs with bounded arities can be transformed in polynomial time to logic programs for which exponentially large intermediate results of the computation using current standard methods is avoided.

Our results extend and complement previous results on the complexity of answer set programming in the literature. They are significant since, unfortunately, current ASP solvers do not always run in polynomial space, even for families of instances which can be solved in PSPACE. They highlight the fact that the current ASP systems suffer from an inherent *grounding bottleneck*: even if DLV, Smodels' grounder Lparse, and other tools employ intelligent grounding techniques to keep the ground instantiation of a program as small as possible, and apply furthermore also program transformations for that, there are cases for which the produced grounding is not polynomial but the program could be solved in polynomial space. As shown by our results, the grounding bottleneck may be overcome by a different system architecture for an important class of programs.

The structure of the remainder of this paper is as follows. In Section 2, we give the necessary preliminaries and recall previous results. Section 3 then considers the answer set existence problem, followed by Section 4 in which brave and cautious reasoning are considered. In Section 5, we consider strong equivalence. Section 6 contains a discussion of the results, including possible transformation of some reasoning tasks to propositional ASP, and Section 7 discusses related work and concludes the paper.

2 Preliminaries and previous results

In this section, we first give a brief overview of the syntax and semantics of disjunctive datalog under the answer sets semantics [28]; for further background, see [12, 35].

An *atom* is an expression $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity $n \geq 0$ and each t_i is either a variable or a constant. A *(classical) literal* l is an atom a (in this case, it is *positive*), or a negated atom $\neg a$ (in this case, it is *negative*). Given a literal l, its *complement* $\neg l$ is defined as $\neg a$ if $l = a$ and a if $l = \neg a$. A set L of literals is said to be *consistent* if, for every literal $l \in L$, $\neg l \notin L$.

A *(disjunctive) rule* r is of the form

$$a_1 \vee \cdots \vee a_n \text{ :- } b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m.$$

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, and where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are literals. We refer to "¬" as *strong negation* and to "not" as *default negation*, or simply as *negation*. The *head* of r is the set $H(r) = \{a_1, \ldots, a_n\}$; and the *body* of r is $B(r) = \{b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m\}$. Furthermore, $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$.

A rule r is called *fact* if $m = 0$, $n > 0$, in which case the symbol :- is usually omitted; (integrity) *constraint* if $n = 0$; r is *normal* if $n \leq 1$, *definite* if $n = 1$, *(proper) disjunctive* if $n > 1$, and *positive* if $k = m$; a positive and normal rule is called *Horn*.

A weak constraint [7] is an expression wc of the form

$$:\sim b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m. [w : l]$$

where $m \geq k \geq 0$ and $b_1, \ldots, b_m$ are literals, while $weight(wc) = w$ (the *weight*) and l (the *level*) are positive integer constants or variables. For convenience, w and/or l may be omitted and are set to 1 in this case. The sets $B^+(wc)$, and $B^-(wc)$, are defined as for rules. A weak constraint is called *positive* or *Horn* iff $B^-(wc) = \emptyset$.

A *program* $\mathcal{P}$ is a finite set of rules and weak constraints. $Rules(\mathcal{P})$ denotes the set of rules and $WC(\mathcal{P})$ the set of weak constraints in $\mathcal{P}$. $w^{\mathcal{P}}_{\max}$ and $l^{\mathcal{P}}_{\max}$ denote

the maximum weight and maximum level over $WC(\mathcal{P})$, respectively. Programs are normal (resp., definite, disjunctive, positive, Horn) if all of their rules and weak constraints enjoy this property. Horn programs without constraints and strong negation are *definite Horn*.

Example 2 One of the standard examples for employing non-ground answer-set programs is the problem of three-colorability of graphs, which we briefly review here. One constructs a program $\mathcal{P}_{3col}$ such that its answer sets provide all valid three-colorings of a graph which is given via additional facts over predicates $v(\cdot)$ and $e(\cdot, \cdot)$, specifying the vertices and respectively edges of the graph. Disjunctions can be used to guess a coloring, i.e., they assign either *red*, *green*, or *blue* to each vertex, and constraints are employed to eliminate those colorings which do not result in a valid coloring, i.e., where connected vertices are assigned the same color. These ideas yield the program $\mathcal{P}_{3col}$, given as follows:

$$
\begin{aligned}
&red(X) \vee green(X) \vee blue(X) \text{ :- } v(X).\\
&\text{ :- } red(X), red(Y), e(X, Y).\\
&\text{ :- } green(X), green(Y), e(X, Y).\\
&\text{ :- } blue(X), blue(Y), e(X, Y).
\end{aligned}
$$

One can now employ weak constraints, to penalize red vertices, for instance. To this end, consider the program $\mathcal{P}_{2col}$ given by the program $\mathcal{P}_{3col}$ together with the simple weak constraint

$$:\sim red(X).\ [1 : 1].$$

This expression counts the number of red vertices in each valid three-coloring, and thus only those answer sets are selected which have a minimum number of red vertices among all valid colorings. Thus $\mathcal{P}_{2col}$ can be seen as an approximation for the 2-coloring problem of graphs, as well. We will sketch how such programs are semantically treated in more detail below.

For any program $\mathcal{P}$, let $U_\mathcal{P}$ be the set of all constants appearing in $\mathcal{P}$ (if no constant appears in $\mathcal{P}$, an arbitrary constant is added to $U_\mathcal{P}$); let $B_\mathcal{P}$ be the set of all ground literals constructible from the predicate symbols appearing in $\mathcal{P}$ and the constants of $U_\mathcal{P}$. Moreover, $Ground(\mathcal{P}, U)$ is the set of rules $r\sigma$ obtained by applying, to each rule and weak constraint $r \in \mathcal{P}$, all possible substitutions σ from the variables[1] in $\mathcal{P}$ to elements of U. We call $Ground(\mathcal{P}, U_P)$ the grounding of $\mathcal{P}$, and write $Ground(\mathcal{P})$ as a shorthand for $Ground(\mathcal{P}, U_P)$. $U_\mathcal{P}$ is usually called the *Herbrand Universe* of $\mathcal{P}$ and $B_\mathcal{P}$ the *Herbrand Literal Base* of $\mathcal{P}$.

Taxonomy of logic programs Starting from Horn programs without weak constraints, we define classes DL[L] with $\{L\} \subseteq \{\text{not}_s, \text{not}, \vee_h, \vee, w\}$. This set is used to indicate the (possibly combined) admission of

not_s: (default) negation; the program remains stratified;
not: unrestricted (default) negation;

[1]This includes also variables appearing in levels or weights of weak constraints.

$\vee_h$: disjunction; the program remains head-cycle free;
$\vee$: unrestricted disjunction;
w: weak constraints.

Recall that stratified negation, not$_s$, cf. [2, 57], allows only a layered use of default negation not, such that for each instantiation of a rule, each default negated literal must be in a lower layer than all head literals; all head literals must be in the same layer; and each positive body literal must be in the same or a lower layer than the head literals.[2] As well, in head-cycle-free disjunction, $\vee_h$, cf. [4], for short HCF, no different head literals of any rule instance positively depend mutually on each other (a head literal $a \in H(r)$ depends on a literal b, if $b \in B^+(r)$, or some literal $c \in B^+(r)$ depends on b).

Thus, for instance, DL$[\vee_h, \text{not}_s]$ contains all HCF stratified programs without weak constraints, and DL = DL$[\vee, \text{not}, w]$ is the full language of all logic programs.

Further classes of logic programs (see, e.g., [8]) would be interesting to study as well. In this article, our focus remains along the lines of the above taxonomy since it covers the best known classes.

Semantics A ground rule r is *satisfied* by a consistent set of literals I iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. I satisfies a ground program $\mathcal{P}$, if each $r \in \mathcal{P}$ is satisfied by I. A ground (weak) constraint c is *violated* by I, iff $B^+(c) \subseteq I$ and $B^-(c) \cap I = \emptyset$; it is satisfied otherwise. A non-ground rule r (resp., a program $\mathcal{P}$) is satisfied by a ground interpretation I iff I satisfies all groundings of r (resp., $Ground(\mathcal{P})$). A non-ground (weak) constraint c is violated by I iff I violates at least one grounding of c.

Recall that for $\mathcal{P} \in$ DL$[\vee, \text{not}]$, a consistent set $I \subseteq B_\mathcal{P}$ is an *answer set*[3] of $\mathcal{P}$ iff it is a subset-minimal set satisfying the *Gelfond-Lifschitz reduct*

$$\mathcal{P}^I = \{H(r) \text{ :- } B^+(r). \mid I \cap B^-(r) = \emptyset, r \in Ground(\mathcal{P})\}.$$

For $\mathcal{P} \in$ DL$[\vee, \text{not}, w]$, a consistent set $I \subseteq B_\mathcal{P}$ is an *(optimal) answer set* of $\mathcal{P}$ iff I is an answer set of $Rules(\mathcal{P})$ and $H^\mathcal{P}(I)$ is minimal among all the answer sets of $Rules(\mathcal{P})$, where the penalization $H^\mathcal{P}(I)$ for weak constraint violations is defined as follows:

$$\begin{aligned} H^\mathcal{P}(I) &= \textstyle\sum_{i=1}^{l^\mathcal{P}_{\max}} \left(f_\mathcal{P}(i) \cdot \sum_{w \in N_i^\mathcal{P}(I)} weight(w) \right); \\ f_\mathcal{P}(1) &= 1, \quad \text{and} \\ f_\mathcal{P}(n) &= f_\mathcal{P}(n-1) \cdot |WC(Ground(\mathcal{P}))| \cdot w^\mathcal{P}_{\max} + 1 \quad \text{for } n > 1; \end{aligned}$$

where $N_i^\mathcal{P}(I)$ denotes the set of the weak constraints with level i in $Ground(\mathcal{P})$ violated by I. For any program $\mathcal{P}$, we denote the set of its (optimal) answer sets by $\mathcal{AS}(\mathcal{P})$.

Intuitively, the weights of the constraints are summarized separately in different levels, and an answer set of $Rules(\mathcal{P})$ is optimal if it has a minimum weight among all the answer sets of $Rules(\mathcal{P})$ on the highest level used, i.e. on level $l^\mathcal{P}_{\max}$. If some

[2] Hence, strictly speaking we consider local stratification.

[3] Note that we only consider *consistent answer sets*, while in [28] also the inconsistent set of all possible literals can be a valid answer set.

answer sets of $Rules(\mathcal{P})$ have the same minimal weight at level $l^{\mathcal{P}}_{\max}$, then the weights on level $l^{\mathcal{P}}_{\max} - 1$ are compared, and so on. This level-by-level evaluation is in fact captured by the above functions. For a more thorough motivation of these concepts, see [7].

Example 3 For a brief illustration of the semantics, consider the programs $\mathcal{P}_{3col}$ and $\mathcal{P}_{2col}$ from Example 2. Observe that $Rules(\mathcal{P}_{2col}) = \mathcal{P}_{3col}$.

Consider the simple graph $G = \{v(1).\ v(2).\ v(3).\ e(1,2).\ e(2,3).\}$. Then, each answer set of $\mathcal{P}_{3col} \cup G$ contains the input graph G together with one of the valid colorings:

$$\begin{array}{ll}
\{red(1), green(2), red(3)\} & \{red(1), green(2), blue(3)\} \\
\{red(1), blue(2), red(3)\} & \{red(1), blue(2), green(3)\} \\
\{green(1), red(2), green(3)\} & \{green(1), red(2), blue(3)\} \\
\{green(1), blue(2), red(3)\} & \{green(1), blue(2), green(3)\} \\
\{blue(1), red(2), green(3)\} & \{blue(1), red(2), blue(3)\} \\
\{blue(1), green(2), red(3)\} & \{blue(1), green(2), blue(3)\}
\end{array}$$

This is also the collection of answer sets of $Rules(\mathcal{P}_{2col} \cup G)$. Given an interpretation $I \in \mathcal{AS}(Rules(\mathcal{P}_{2col} \cup G))$, the weak constraint $:\sim red(X).\ [1:1]$ yields for $H^{\mathcal{P}}(I)$ just the number of red vertices in I. Hence, for our simple example graph G from above, the program $\mathcal{P}_{2col} \cup G$ derives only the following answer sets:

$$I_1 = G \cup \{green(1), blue(2), green(3)\} \quad I_2 = G \cup \{blue(1), green(2), blue(3)\}.$$

which have a total penalization of $H^{\mathcal{P}}(I_1) = H^{\mathcal{P}}(I_2) = 0$.

Note that the computation of the penalty thus relies on the groundings of the weak constraints. We therefore restrict, unless stated otherwise, the use of weak constraints in a program $\mathcal{P}$ in such a way that (1) the number of ground instances of weak constraints and (2) the highest level index, $l^{\mathcal{P}}_{\max}$, both remain polynomial in the size of $\mathcal{P}$. This ensures that computation and comparison of penalizations can be done in polynomial time. Relaxing this restriction is discussed in Section 6.3.

The formal definition of the decision problems for program classes DL[L], where $\{L\} \subseteq \{\text{not}_s, \text{not}, \vee_h, \vee, w\}$, studied in this paper is given as follows:

- Answer set existence: Given a program $\mathcal{P} \in$ DL[L], decide whether $\mathcal{AS}(\mathcal{P}) \neq \emptyset$.
- Brave reasoning: Given $\mathcal{P} \in$ DL[L] and a ground atom a, decide whether $a \in I$ holds for at least one $I \in \mathcal{AS}(\mathcal{P})$.
- Skeptical reasoning: Given $\mathcal{P} \in$ DL[L] and a ground atom a, decide whether $a \in I$ holds for each $I \in \mathcal{AS}(\mathcal{P})$.
- Strong equivalence: Given $\mathcal{P}_1, \mathcal{P}_2 \in$ DL[L], decide whether $\mathcal{AS}(\mathcal{P}_1 \cup \mathcal{P}) = \mathcal{AS}(\mathcal{P}_2 \cup \mathcal{P})$, for each further program $\mathcal{P}$.

For some classes of programs we make use of the following proposition which is immediate from the well-known result that any normal stratified program has at most one answer set:

Proposition 1 *For any* $\mathcal{P} \in$ DL[L] *with* $\{L\} \subseteq \{w, \text{not}_s\}$, $|\mathcal{AS}(\mathcal{P})| \leq 1$.

Table 1 Complexity of answer set existence for propositional fragments of DL

[a]Trivial without constraints and strong negation.

	{}[a]	{not$_s$}[a]	{not}
{}	P	P	NP
{$\vee_{\mathbf{h}}$}	NP	NP	NP
{$\vee$}	NP	Σ_2^P	Σ_2^P

Hence, weak constraints are immaterial in such programs and we obtain:

Corollary 1 *For $\mathcal{P} \in \mathrm{DL}[L]$ with $\{L\} \subseteq \{w, \mathrm{not}_s\}$, $\mathcal{AS}(\mathcal{P}) = \mathcal{AS}(Rules(\mathcal{P}))$.*

Previous results We assume that the reader is acquainted with NP-completeness and basic notions of complexity theory, and refer to [34, 54] for further background.

As mentioned in the Introduction, previous work on the complexity of ASP mostly considered the case of propositional programs. Tables 1, 2 and 3 provide a complete overview of the complexity of answer set checking, brave and cautious reasoning, respectively, as well as strong equivalence for the propositional variants of the language fragments considered in this paper. All entries in the tables refer to completeness results. For the results on strong equivalence, we refer to [18, 19] (so far, strong equivalence in combination with weak constraints has not been considered); Tables 1 and 2 are taken from [35]. In these tables, the rows specify the form of disjunction allowed (in particular, {} = no disjunction), whereas the columns specify the support for default negation and weak constraints. So the field in row R and column C indicates DL$[L]$, where $\{L\} = R \cup C$.

For the canonical reasoning problems in the general non-ground case, the complexity of brave and cautious reasoning in general increases by one exponential compared to the according results in the propositional case. In particular, the results shift from P to EXP, NP to NEXP, Δ_2^P to EXP$^{\mathrm{NP}}$, Σ_2^P to NEXP$^{\mathrm{NP}}$, etc. These complexity results in the non-ground case have been derived e.g. in [12, 13] for disjunctive programs and several of its fragments, and in general are obtained by means of complexity upgrading techniques as described in [12, 29]. Also the complexity of the strong equivalence problem increases by one exponential [16, 18] resulting in co-NEXP-completeness in general, and in EXP-completeness for Horn programs.

3 Complexity of answer set existence

A key issue to the complexity results in this article is the following lemma.

Lemma 1 *Given a program $\mathcal{P}$ and a consistent set $I \subseteq B_{\mathcal{P}}$ of literals, deciding whether I satisfies $\mathcal{P}^I$ is in* co-NP.

Table 2 Complexity of brave and cautious reasoning for propositional fragments of DL

Brave/cautious	{}	{**w**}	{not$_s$}	{not$_s$, **w**}	{not}	{not, **w**}
{}	P	P	P	P	NP/co-NP	Δ_2^P
{$\vee_{\mathbf{h}}$}	NP/co-NP	Δ_2^P	NP/co-NP	Δ_2^P	NP/co-NP	Δ_2^P
{$\vee$}	Σ_2^P/co-NP	Δ_3^P	Σ_2^P/Π_2^P	Δ_3^P	Σ_2^P/Π_2^P	Δ_3^P

Table 3 Complexity of strong equivalence for propositional fragments of DL

	{}	{not$_\mathbf{s}$}	{not}
{}	P	co-NP	co-NP
{$\vee_\mathbf{h}$}	co-NP	co-NP	co-NP
{$\vee$}	co-NP	co-NP	co-NP

Proof For deciding the complementary problem, it suffices to guess a ground substitution θ from the variables in $\mathcal{P}$ to $U_\mathcal{P}$ and a rule $r \in \mathcal{P}$, and to check whether I does not satisfy $(r\theta)^I$. Here, $r\theta$ denotes the ground instance of r obtained by applying the substitution θ to its variables. Given r, θ, and I, one can obviously compute $(r\theta)^I$ in polynomial time, and checking whether $I \not\models (r\theta)^I$ is also tractable. Therefore, the complementary problem is in NP, which proves co-NP-membership. □

In fact, the problem is also co-NP-hard, since an NP-complete subsumption problem (see, e.g., Lemma 1 in [17]) can easily be encoded as its complementary problem. Note also that in the propositional case, the problem is polynomial.

If we constrain the programs to have the arities of predicates bounded by some constant, we just need to consider interpretations which are polynomial in the size of the input program, when examining candidates for answer sets. Notice that this holds for any reasonably succinct representation of interpretations, in particular it is true not only for bitmap representations (i.e., a bit representation of I with one bit encoding $a \in I$ for each ground atom a) but also for set representations (i.e., an explicit enumeration of all ground atoms $a \in I$). Hence, we can guess such interpretations, which is the main reason for corresponding decision problems to remain within the polynomial hierarchy. In contrast to the propositional case, however, the grounding stays exponential, as illustrated in Example 1. We address this issues also with respect to optimization techniques ("intelligent grounding") in Section 6.1.

We proceed to derive complexity results for answer set checking for different classes of programs with bounded arities step-by-step. Note that to show program classes $\mathrm{DL}[L_1] \subseteq \mathrm{DL}[L_2] \subseteq \ldots \subseteq \mathrm{DL}[L_k]$ to be complete for a complexity class C, it suffices to prove C-hardness for $\mathrm{DL}[L_1]$ and C-membership for $\mathrm{DL}[L_k]$. The complexity results we obtain for answer set existence are summarized in Theorem 1 (see also Table 4) at the end of the section.

We start with an informal discussion. As sketched above, guessing an interpretation $I \subseteq B_\mathcal{P}$ and deciding whether I satisfies $\mathcal{P}^I$ is in Σ_2^P. Since with an oracle for this problem we can also decide minimality of I, we obtain Σ_3^P-membership

Table 4 Complexity of answer set existence under bounded predicate arities

	{}[a]	{not$_\mathbf{s}$}[a]	{not}
{}	co-NP	Δ_2^P	Σ_2^P
{$\vee_\mathbf{h}$}	Σ_2^P	Σ_2^P	Σ_2^P
{$\vee$}	Σ_2^P	Σ_3^P	Σ_3^P

All entries are completeness results

[a]Trivial without constraints and strong negation.

in general (cf. Lemma 3). In case of DL[$\vee$] programs, since we are interested in mere answer set existence, due to monotonicity we can omit checking for minimality and thus stay within Σ_2^P (cf. Lemma 3). For DL[not] programs, we cannot build on monotonicity but we can guarantee minimality by not only guessing I, but also a founded proof of I with respect to $\mathcal{P}^I$. This serves the same purpose and explains the remaining Σ_2^P entries in Table 4 (cf. also Lemma 2). Similarly, given a DL[] program we do not guess an interpretation but a founded proof of inconsistency (that can be checked in polynomial time) and obtain the co-NP result (cf. Lemma 2). The only exception from the one level shift in general is for DL[not$_s$] programs, for which propositional answer set existence is tractable, while under bounded predicates it jumps to the second level of PH. In this case, we cannot do better than checking for inconsistency stratum by stratum, which requires a polynomial number of NP oracle calls. Therefore, the Δ_2^P result (cf. Lemma 2).

Lemma 2 *Answer set existence is in* co-NP *for* DL[] *programs, in* Δ_2^P *for* DL[not$_s$] *programs, and in* Σ_2^P *for* DL[not] *programs.*

Proof For DL[] programs we can guess a polynomial-size founded proof of inconsistency as follows. We guess either a ground substitution of a program constraint or a (polynomial size) inconsistent set of ground literals together with (a polynomial number of) polynomial length founded proofs Pr_a for every guessed ground literal a. Each proof is a sequence of rule applications $r_1\theta_1, \ldots, r_k\theta_k$ which derives a starting from scratch. The proofs can be checked in polynomial time, and constitute witnesses that the respective ground literal must be contained in any answer set of the program. Hence, we conclude that the guessed interpretation is a subset of any answer set. Since, the interpretation is a ground substitution of a program constraint or inconsistent, it follows that the program does not have an answer set. Therefore, the complementary problem is in NP. This proves co-NP-membership in case of DL[] programs.

Given a DL[not$_s$] program $\mathcal{P}$, we first compute a stratification of $\mathcal{P}$ in polynomial time. Starting with the lowest stratum and $I_0 = \emptyset$, we proceed stratum by stratum and consider the subset $\mathcal{P}_s$ of rules in $\mathcal{P}$ that do not contain literals from higher strata. We check whether $\mathcal{P}_s^{I_s}$ is consistent, as examined above for DL[] programs, by means of an NP oracle call. In case of consistency, with a further oracle call we guess a new polynomial size partial interpretation $I_{s+1} \supseteq I_s$ together with a polynomial length founded proof Pr_a for every guessed ground literal $a \in I_{s+1}$. Each proof is a sequence of rule applications $r_1\theta_1, \ldots, r_k\theta_k$ which derives a in $\mathcal{P}_s^{I_s}$ starting from scratch. With I_{s+1} we continue with the next stratum. Since this procedure requires two oracle calls per stratum and the number of strata is polynomially bounded, the problem is in Δ_2^P.

Given a normal program $\mathcal{P}$ we first guess a (polynomial-size) interpretation I of $\mathcal{P}$. I is an answer set of $\mathcal{P}$, iff (1) I satisfies the reduct $\mathcal{P}^I$, and (2) I is minimal in satisfying $\mathcal{P}^I$. We can check the minimality of I by providing, for each atom $a \in I$, a founded proof Pr_a which is a sequence of rule applications $r_1\theta_1, \ldots, r_k\theta_k$ which derives a starting from scratch, where default negation is evaluated w.r.t. I. Since $\mathcal{P}^I$ is Horn, as above for DL[] programs, the proofs constitute witnesses that the respective ground literal must be contained in any answer set of $\mathcal{P}^I$. Therefore, I is minimal if a founded proof can be provided for each atom $a \in I$. Furthermore, because $\mathcal{P}^I$ is Horn, the number of steps required to derive a is at most the number of

atoms in I, which is obviously polynomial in the size of the original problem. Hence, we can guess such proofs Pr_a for all $a \in I$ together with I at once and check them in polynomial time. From Lemma 1, we know that (1) is in co-NP and can thus be checked with an NP oracle. This proves that we can check (1) and (2) with one NP oracle call, respectively, thus the problem is in Σ_2^P. □

Lemma 3 *Answer set existence is in* Σ_3^P *for* DL[$\vee$, not] *programs and in* Σ_2^P *for* DL[$\vee$] *programs.*

Proof In general, we can guess a (polynomial-size) interpretation I of $\mathcal{P}$ and check that it is an answer set for $\mathcal{P}$. Clearly, I is an answer set for $\mathcal{P}$ iff (1) I satisfies $\mathcal{P}^I$ and (2) there does not exist some $I' \subset I$ which satisfies $\mathcal{P}^I$. By Lemma 1, given I, (1) is in co-NP. The complement of (2), given an interpretation I, is in Π_2^P: we further guess $I' \subset I$ and use an NP oracle for the co-NP check. Hence, both (1) and (2) can be accomplished by means of an oracle for $\text{NP}^{\text{NP}} = \Sigma_2^P$ problems. This proves Σ_3^P membership of answer set existence.

For a DL[$\vee$] program $\mathcal{P}$, it is sufficient to guess a (polynomial-size) interpretation I of $\mathcal{P}$ and check that it is founded (but not necessarily minimal) and whether I satisfies $\mathcal{P}$. The argument here is as follows: If I is founded and I satisfies $\mathcal{P}$, then, no $I' \supset I$ is an answer set of $\mathcal{P}$ due to minimality. Thus, we can conclude that either I itself is an answer set of $\mathcal{P}$, or that there exists a subset of I that is an answer set of $\mathcal{P}$. Both conclusions serve our purpose of deciding answer set existence. To check whether I is founded, we need to provide, for each atom $a \in I$, a founded proof Pr_a which is a sequence of rule applications $r_1\theta_1, \ldots, r_k\theta_k$ which derives a starting from scratch. Since $\mathcal{P}$ is positive, the number of steps required to derive a is at most the number of atoms in I, which is obviously polynomial in the size of the original problem. Hence, we can guess such proofs Pr_a for all $a \in I$ with one call to an NP oracle and check them in polynomial time. Furthermore, given I, checking whether I satisfies $\mathcal{P}$ is in co-NP (see Lemma 1), and can thus be decided by a second call to an NP oracle. Therefore the overall problem is in $\text{NP}^{\text{NP}} = \Sigma_2^P$. □

We next prove corresponding hardness results, where we employ that C-hardness for normal programs is immediate from C-hardness for HCF programs, due to a faithful polynomial-time rewriting of HCF programs to equivalent normal programs [4]. Unless stated otherwise, this technique is implicitly used throughout the article.

Lemma 4 *Answer set existence is* co-NP*-hard for* DL[] *programs.*

Proof This result is an immediate consequence from conjunctive query evaluation, which is NP-complete (see [1]): Given a query a :- B and a database DB, deciding whether the query fires and derives atom a is NP-complete. This holds even if all involved predicates have arity bounded by a constant. Consider $\mathcal{P} = DB \cup \{\text{:- } B.\}$. Obviously, $\mathcal{P}$ is Horn and polynomial in size of the database and the query involved. It is also easily seen that $\mathcal{P}$ has no answer set iff a :- B evaluates to true under DB. □

Lemma 5 *Answer set existence is* Δ_2^P*-hard for* DL[not_s] *programs.*

Proof We prove the result by means of a reduction from deciding the least bit of the lexicographic maximum satisfying truth assignment for a CNF $C = c_1 \wedge \cdots \wedge c_k$ over atoms $X = \{x_1, \ldots, x_n\}$, which is Δ_2^P-complete, cf. [54]. A lexicographic maximum satisfying truth assignment is a truth assignment satisfying C that is maximal when interpreted as a binary number $x_1 x_2 \ldots x_n$. W.l.o.g., each $c_i = L_{i,1} \vee L_{i,2} \vee L_{i,3}$ contains three literals and C is known to be satisfiable.

We construct a program containing facts of ternary predicates describing the (2^3–1 possible) satisfying truth assignments for each clause c_i. For example, if $c_1 = x_1 \vee \neg x_2 \vee x_3$, we add

$$c_1(0,0,0).\ c_1(0,0,1).\ c_1(0,1,1).\ c_1(1,0,0).\ c_1(1,0,1).\ c_1(1,1,0).\ c_1(1,1,1).$$

Furthermore, we introduce a fact $true(1)$., and for each atom $x_i \in X$, we add a predicate $val_{x_i}(V)$ and rules

$$val_{x_i}(1) \text{ :- } c_1(\bar{t}_1), \ldots, c_k(\bar{t}_k), true(V_i), val_{x_{i-1}}(V_{i-1}), \ldots, val_{x_1}(V_1).$$
$$val_{x_i}(0) \text{ :- not } val_{x_i}(1).$$

where $\bar{t}_j = V_{l_1}, V_{l_2}, V_{l_3}, 1 \leq j \leq k$, given that the atoms of literal $L_{j,1}$, $L_{j,2}$, and $L_{j,3}$ are x_{l_1}, x_{l_2}, and x_{l_3}, respectively. Note that as usual, $val_{x_{i-1}}(V_{i-1}), \ldots, val_{x_1}(V_1)$ is empty if $i = 1$. For the example clause $c_1 = x_1 \vee \neg x_2 \vee x_3$, if we assume that it is the only clause in C, i.e., $k = 1$, then the transformation yields the following rules:

$$val_{x_1}(1) \text{ :- } c_1(V_1, V_2, V_3), true(V_1).$$
$$val_{x_1}(0) \text{ :- not } val_{x_1}(1).$$
$$val_{x_2}(1) \text{ :- } c_1(V_1, V_2, V_3), true(V_2), val_{x_1}(V_1).$$
$$val_{x_2}(0) \text{ :- not } val_{x_2}(1).$$
$$val_{x_3}(1) \text{ :- } c_1(V_1, V_2, V_3), true(V_3), val_{x_2}(V_2), val_{x_1}(V_1).$$
$$val_{x_3}(0) \text{ :- not } val_{x_3}(1).$$

Let us denote the resulting program by $\mathcal{P}_{lmax}$. Note that $\mathcal{P}_{lmax}$ is definite and stratified. The maximum satisfying truth assignment for C is computed in the layers of $\mathcal{P}_{lmax}$, and encoded by $val_{x_i}(b_i)$ in the unique answer set I of $\mathcal{P}_{lmax}$. At the bottom $val_{x_1}(1)$ is derived iff $C\theta$ for $\theta = \{x_1/1\}$ is satisfiable. Otherwise, $val_{x_1}(0)$ is derived. Next, depending on the value of $val_{x_1}(b_1)$, $val_{x_2}(1)$ is derived iff $C\theta$ for $\theta = \{x_1/b_1, x_2/1\}$ is satisfiable, otherwise $val_{x_2}(0)$ is derived, and so on. Thus, $val_{x_n}(1)$ is in I iff the least bit of the maximum satisfying assignment is 1, and $val_{x_n}(0)$ is in I otherwise.

For proving the result, we just need to add the constraint :- $val_{x_n}(0)$. to $\mathcal{P}_{lmax}$. Call the resulting program $\mathcal{P}$. Obviously, $\mathcal{P}$ remains stratified and has an answer set iff the last bit of the maximum satisfying assignment is 1. Since $\mathcal{P}$ is polynomial in size of C, Δ_2^P-hardness follows. □

Lemma 6 *Answer set existence is* Σ_2^P*-hard for* DL[$\vee_h$] *programs.*

Proof The proof is by reduction of the evaluation problem for a QBF of the form $\Phi = \forall X \exists Y c_1 \wedge \cdots \wedge c_k$, where the c_i are clauses over $X \cup Y$. This problem is Π_2^P-hard, even if all clauses have size 3. The reduction presented here is similar to the "classic" reduction of such formulas to the problem of brave reasoning over disjunctive programs. In particular, we construct a program $\mathcal{P}$ for each QBF Φ of

above form, such that $\mathcal{P}$ does not have an answer set iff Φ is true. The construction is as follows:

First, set up a disjunctive fact

$$t(x_i) \vee f(x_i). \qquad \text{for each } x_i \in X \tag{1}$$

using x_i as a constant. For each clause $c_i = L_{i,1} \vee L_{i,2} \vee L_{i,3}$, we introduce a predicate whose arity is the number of variables from Y. We then define, by rules, which truth assignments to these variables make the clause true, given the truth of the variables from X in c_i, This is best illustrated by examples. Suppose we have $c_1 = x_1 \vee \neg x_2 \vee y_3$. Then, we introduce $c_1(V)$, where the argument V is reserved for the truth assignments to y_3, and define:

$$\begin{array}{ll} c_1(0) \text{ :- } t(x_1). & c_1(1) \text{ :- } t(x_1). \\ c_1(0) \text{ :- } f(x_2). & c_1(1) \text{ :- } f(x_2). \\ c_1(1) \text{ :- } f(x_1), t(x_2). & \end{array}$$

Informally, this states that clause c_1 is satisfied, if either x_1 is true or x_2 is false, and in both cases the value of the Y-variable is irrelevant. Or, x_1 is false and x_2 is true and the Y-variable is true as well. As another example, consider $c_2 = x_2 \vee \neg y_1 \vee y_5$. Here, we introduce $c_2(V_1, V_2)$, and define:

$$\begin{array}{lll} c_2(0,0) \text{ :- } t(x_2). & c_2(0,1) \text{ :- } t(x_2). & \\ c_2(1,0) \text{ :- } t(x_2). & c_2(1,1) \text{ :- } t(x_2). & \\ c_2(0,0) \text{ :- } f(x_2). & c_2(0,1) \text{ :- } f(x_2). & c_2(1,1) \text{ :- } f(x_2). \end{array}$$

Now set up a rule which corresponds to evaluating the formula $\exists Y c_1 \wedge \cdots \wedge c_k$ for a given assignment to X:

$$w \text{ :- } c_1(\bar{Y}_1) \wedge \cdots \wedge c_k(\bar{Y}_k). \tag{2}$$

where $\bar{Y}_i$, $1 \leq i \leq k$, is a vector representing the variables from Y occurring in c_i, put at proper position. In the case above, we have $c_1(Y_3)$ and $c_2(Y_1, Y_5)$.

Let us call the program built so far $\mathcal{P}_{QBF}$; it will also be used in some of the subsequent proofs. Note that $\mathcal{P}_{QBF}$ is positive, disjunctive, and HCF, as well as polynomial in the size of the underlying QBF. The functioning of $\mathcal{P}_{QBF}$ is as follows: The disjunctive clauses (1) generate a truth assignment to X, and the remaining clauses check whether $\exists Y c_1 \wedge \cdots \wedge c_k$ is true under this assignment, deriving w if so.

For proving Lemma 6, consider the addition of the constraint :- w., and let $\mathcal{P}$ be the resulting program. Then, $\mathcal{P}$ has an answer set iff there exists a truth assignment to X such that $\exists Y c_1 \wedge \cdots \wedge c_k$ is not true under this assignment, i.e., iff Φ is false. □

Lemma 7 *Answer set existence is* Σ_3^P*-hard for* DL$[\vee, \text{not}_s]$ *programs.*

Proof Consider an existential QBF $\Phi = \exists X_1 \forall X_2 \exists Y c_1 \wedge \cdots \wedge c_k$, take $\mathcal{P}_{QBF}$ from the proof of Lemma 6, but now with $X = X_1 \cup X_2$, and add rules p :- w. for each ground atom $p \in B^+ \setminus \{w, t(x_i), f(x_i) \mid x_i \in X_1\}$, making the program non-HCF, where B^+ is the set of all positive literals in $B_{\mathcal{P}_{QBF}}$. Note that the resulting program, denoted $\mathcal{P}$, remains polynomial in size of Φ. Intuitively, $\mathcal{P}$ guesses a truth assignment σ for the atoms X_1, by means of the disjunctive clauses (1) related to X_1. Then, for each of these truth assignments, the remaining clauses behave like the program in Lemma 6

for $\Phi' = \forall X_2 \exists Y (c_1 \wedge \cdots \wedge c_k)\sigma$. In particular, the remaining disjunctive clauses (1), i.e., those related to X_2, extend the truth assignment to X_2, and the clauses (2) check whether $\exists Y c_1 \wedge \cdots \wedge c_k$ is true under this assignment, deriving w if so. The additional rules p :- w., by means of answer-set minimality, ensure that w is in an answer set iff Φ' is true, i.e., iff every extension of σ to a truth assignment on X_2 derives w.

Now extend $\mathcal{P}$ by the constraint :- not w. Then, every answer set of this extended program has to contain w. Therefore, the extended program has an answer set iff there exists a truth assignment to X_1 such that Φ' is true, i.e., iff Φ is true. □

This concludes the necessary bounds to derive a full picture of the complexity of deciding answer set existence for programs with bounded arities. We summarize our results:

Theorem 1 *The complexity of deciding answer set existence under bounded predicate arities is given by the respective entries in Table 4.*

These results show, that if we move from ground (i.e., propositional) programs to non-ground programs but allow only predicates with small arity, the complexity of the language moves up *only one level in the polynomial hierarchy (PH), but not more*. Thus, unless we use growing predicate arities, we can not encode problems above PH, e.g., as commonly believed, PSPACE-complete problems. On the other hand, it means that an exponential-size grounding-at-once can be avoided. Furthermore, a number of the problems can be polynomially mapped to ASP with disjunctive propositional programs (harboring Σ_2^P/Π_2^P complexity), avoiding grounding.

We note that the results remain valid if we just restrict the arities of the intensional predicates, i.e., those occurring in the heads of non-facts, and predicates of non-ground atoms in disjunctive facts. Intuitively, any answer set S has then polynomial size modulo a fixed part—i.e., the extensional database given by a grounding of the non-disjunctive facts wrt. $U_{\mathcal{P}}$—while checking rule compliance of a candidate answer set S is co-NP-complete rather than polynomial as in the ground case.

4 Complexity of brave and cautious reasoning

In what follows, we now extend our results to brave and skeptical reasoning from non-ground programs having predicates with bound arities. Compared to answer set existence, weak constraints are now of relevance to decide the respective reasoning task. Our results are summarized in Table 5.

Table 5 Complexity of brave and cautious reasoning under bounded predicate arities

Brave/cautious	{}	{w}	{not$_s$}	{not$_s$, w}	{not}	{not, w}
{}	D^{P*}/NP	D^{P*}/NP	Δ_2^P	Δ_2^P	Σ_2^P/Π_2^P	Δ_3^P
{$\vee_h$}	Σ_2^P/Π_2^P	Δ_3^P	Σ_2^P/Π_2^P	Δ_3^P	Σ_2^P/Π_2^P	Δ_3^P
{$\vee$}	Σ_3^P/Π_2^P	Δ_4^P	Σ_3^P/Π_3^P	Δ_4^P	Σ_3^P/Π_3^P	Δ_4^P

All entries are completeness results.

*Without constraints and strong negation (= definite Horn) the complexity is NP.

In what follows, we informally summarize some remarks for the different classes of programs under consideration, and afterwards give the formal proofs.

The D^P results for Horn programs (cf. Lemma 8 and 9) are explained as follows: The class D^P contains the decision problems whose *yes* instances are characterized by the conjunction of an NP property and an independent co-NP property. The co-NP part is needed to show that no contradiction is derivable (which vanishes for definite Horn programs), while the NP part stems from a foundedness (minimality) check.

As for answer set existence, again we have slightly higher complexity for stratified normal programs (cf. Lemma 15), since we must evaluate a polynomial number of NP problems according to the strata of the program, i.e., two NP oracle calls per stratum are needed.

In the presence of weak constraints (cf. Lemma 14–17), the upper bounds follow easily since the oracles at hand are powerful enough to guess and check answer sets for the respective class of programs without weak constraints. Thus, one may first compute the cost of an optimal answer set in a binary search, and then decide the problem with a single oracle call.

The only peculiarity in Theorem 2 is for DL[$\vee$], for which brave reasoning is one level higher than cautious reasoning (cf. Lemma 11 and 13). However, also this is carried over from the propositional case in which a similar gap exists, see Table 2. This gap can explained by the fact that minimality is not important for cautious reasoning in this case, while it is for brave reasoning. These results (also Π_3^P-hardness when negation is involved, cf. Lemma 13) can be proved similar to Lemma 7, where Σ_3^P-hardness of answer set existence for disjunctive programs with negation was shown.

We proceed with a more formal elaboration, starting with the D^P and NP entries in Table 5.

Lemma 8 *Brave reasoning is in* D^P *for Horn programs and in* NP *for definite Horn programs. Cautious reasoning is in* NP *for Horn programs in general.*

Proof For brave reasoning, we do not need to guess an interpretation I, but instead can guess a polynomial-size founded proof Pr_a for the query literal a, as described in Lemma 2. The proof is a sequence of rule applications $r_1\theta_1, \ldots, r_k\theta_k$ which derives a starting from scratch and can be checked in polynomial time. If the program is definite Horn, consistency and answer set existence is guaranteed. Therefore, Pr_a constitutes a witness that the query literal must be contained in the answer set of the program. This proves NP-membership of brave reasoning for definite Horn programs.

If constraints or strong negation are present, we need an additional, independent co-NP-check to ensure that no constraint is violated and obtain D^P-membership in this case. More precisely, Pr_a, which we can guess and check in NP, constitutes a witness that the query literal must be contained in the answer set of the program iff the program is consistent. That is, a further independent check for answer set existence is required. As proved in Lemma 2, this problem is in co-NP for DL[] programs and thus for Horn programs. Consequently, brave reasoning for Horn programs is in D^P.

Concerning cautious reasoning, it is sufficient to guess and check a polynomial-size founded proof for either the query a, or a constraint violation, or inconsistency

in order to witness cautious consequence of a. In particular, we guess either (1) a polynomial length founded proof Pr_a for the query literal a, or (2) a ground substitution of a program constraint together with a polynomial length founded proof Pr_a for every guessed ground literal a, or (3) a polynomial-size inconsistent set of ground literals together with polynomial length founded proof Pr_a for every guessed ground literal a. In all three cases the respective proofs Pr_a can be checked in polynomial time and constitute a witness that (1) the query literal is contained in every answer set of $\mathcal{P}$ (which is trivial if no answer set exists due to inconsistency), that (2) no answer set exists due to constraint violation, hence the cautious query can be answered trivially, and that (3) no answer set exists due to inconsistency and the cautious query can be answered trivially. The test decides cautious reasoning for Horn programs and is in NP. □

Lemma 9 *For definite Horn programs without weak constraints, both brave and cautious reasoning are* NP*-hard. For Horn programs without weak constraints, brave reasoning is* D^P*-hard.*

Proof The results are inherited from (bounded) conjunctive queries as used in the proof of Lemma 4. Indeed, consider a conjunctive query a :- B over a database DB. Then a :- B evaluates to true under DB iff the *unique* answer set of the definite Horn program $DB \cup \{a$:- $B.\}$ contains a. For the D^P-hardness result, consider two conjunctive queries a_1 :- B_1, a_2 :- B_2 with $a_1 \neq a_2$, and two databases DB_1, DB_2 over disjoint alphabets not containing a_1 or a_2. Then, a_1 :- B_1 evaluates to true under DB_1 and a_2 :- B_2 evaluates to false under DB_2 iff a_1 is a brave consequence of the (non-definite) Horn program $DB_1 \cup DB_2 \cup \{a_1$:- B_1. :- a_2, $B_2.\}$. □

We proceed with the upper bounds for the remaining program classes without weak constraints.

Lemma 10 *For* DL$[\vee_h$, not$]$ *programs brave reasoning is in* Σ_2^P*, and cautious reasoning is in* Π_2^P.

Proof Recall that we can employ the rewriting technique to normal programs [4]. Therefore, let us consider a normal program $\mathcal{P}$. For brave reasoning, we proceed by guessing a (polynomial-size) interpretation I of $\mathcal{P}$ that contains the query literal. We then need to check whether I is an answer set of $\mathcal{P}$, i.e., iff (1) I satisfies the reduct $\mathcal{P}^I$, and (2) I is minimal in satisfying $\mathcal{P}^I$. As in the proof of Lemma 2, we can check the minimality of I by providing, for each atom $a \in I$, a founded proof Pr_a which is a sequence of rule applications $r_1\theta_1, \ldots, r_k\theta_k$ which derives a starting from scratch, where default negation is evaluated w.r.t. I. Since $\mathcal{P}^I$ is Horn, the proofs constitute witnesses that the respective ground literal must be contained in any answer set of $\mathcal{P}^I$. Therefore, I is minimal if a founded proof can be provided for each atom $a \in I$. Furthermore, because $\mathcal{P}^I$ is Horn, the number of steps required to derive a is at most the number of atoms in I, which is obviously polynomial in the size of the original problem. Hence, we can guess such proofs Pr_a for all $a \in I$ together with I at once and check them in polynomial time. From Lemma 1, we know that (1) is in co-NP and can thus be checked with an NP oracle. This proves that we can check (1) and (2) with one NP oracle call, respectively, hence brave reasoning is in Σ_2^P.

The complementary problem, i.e., cautious reasoning, can be decided by guessing a (polynomial-size) interpretation I of $\mathcal{P}$ that does not contain the query literal, and checking that I is an answer set of $\mathcal{P}$ as above. This proves Π_2^P-membership of cautious reasoning. □

Lemma 11 *For* DL[$\vee$, not] *programs brave reasoning is in* Σ_3^P*, and cautious reasoning is in* Π_3^P*; for* DL[$\vee$] *programs, cautious reasoning remains in* Π_2^P*.*

Proof The argumentation follows the corresponding proof for answer set existence from Lemma 3: Guess a (polynomial-size) interpretation I such that the query atom a is in I and check that I is an answer set for $\mathcal{P}$, which can be accomplished by means of an oracle for Σ_2^P problems. This shows Σ_3^P-membership for brave reasoning. The Π_3^P-membership for skeptical reasoning follows immediately by using the complementary problem and the same argumentation as above, but using $a \notin I$, instead of $a \in I$.

For a DL[$\vee$] program $\mathcal{P}$, the complementary problem of skeptical reasoning can be decided by guessing I with $a \notin I$ and check whether I is founded (but not necessarily minimal) for $\mathcal{P}$ and also satisfies $\mathcal{P}$ (since then, there is also an answer set $I' \subseteq I$ of $\mathcal{P}$ with $a \notin I'$). The checks can be accomplished by means of calls to an NP oracle (since they are in NP and co-NP, respectively, see also Lemma 1). □

The following results give the matching lower bounds.

Lemma 12 *For* DL[$\vee_h$] *programs brave reasoning is* Σ_2^P*-hard, and cautious reasoning is* Π_2^P*-hard.*

Proof Π_2^P-hardness follows from the reduction in the proof of Lemma 6 with a slight variation:

For proving the lemma, we create a maximal interpretation if w holds as follows. Let B^+ be the set of all positive literals in $B_{\mathcal{P}_{QBF}}$, and add rules p :- w. for each ground atom $p \in B^+ \setminus \{w\}$ to $\mathcal{P}_{QBF}$. Call the resulting program $\mathcal{P}$. Note that if we derive w from $\mathcal{P}_{QBF}$, any element from B^+ can be derived in $\mathcal{P}$. Therefore, w is a cautious consequence of the program $\mathcal{P}$ used iff the formula $\Phi = \forall X \exists Y c_1 \wedge \cdots \wedge c_k$ is true.

We obtain the dual Σ_2^P-hardness result for brave reasoning by adding the disjunctive fact $u \vee w$. to $\mathcal{P}$, where u is a fresh atom, and asking whether u is a brave consequence of the resulting program; this is the case iff w is not a cautious consequence of the original program. □

Lemma 13 *For* DL[$\vee$] *programs, brave reasoning is* Σ_3^P*-hard, and cautious reasoning is* Π_2^P*-hard. For* DL[$\vee$, not$_s$] *programs, cautious reasoning is* Π_3^P*-hard.*

Proof Σ_3^P-hardness of brave reasoning follows from the construction that has been used in the proof of Lemma 7, where Σ_3^P-hardness of answer set existence for disjunctive programs with negation was shown. In fact, w is a brave consequence of the program $\mathcal{P}$ there iff $\Phi = \exists X_1 \forall X_2 \exists Y c_1 \wedge \cdots \wedge c_k$ is true. Since $\mathcal{P}$ is positive, the result follows. Cautious reasoning for this fragment, however, is in Π_2^P, since to disprove a cautious consequence it is sufficient to find some (not necessarily

subset-minimal) interpretation I which satisfies $\mathcal{P}$ and does not contain the query; such I can be guessed and checked with an NPoracle in polynomial time.

If negation is involved, we obtain Π_3^P-hardness of cautious inference by a simple reduction of the complement of brave reasoning of the atom w as above, by adding the stratified rule w' :- not w., where w' is a fresh atom, and asking whether w' is a cautious consequence. □

We conclude with reasoning problems involving weak constraints.

Lemma 14 *For* DL[$\vee$, not, w] *programs, both brave and cautious inference are in* Δ_4^P, *and for* DL[not, w] *programs, both inference tasks are in* Δ_3^P.

Proof In case of DL[$\vee$, not, w] programs, by means of an oracle for Σ_3^P problems, we can we guess a (polynomial-size) interpretation I of *Rules*($\mathcal{P}$), check that it is an answer set for *Rules*($\mathcal{P}$), namely that it satisfies *Rules*($\mathcal{P}$)^I (which is in co-NP) and that no $I' \subset I$ satisfies *Rules*($\mathcal{P}$)^I (which is in Π_2^P), and compute its cost with respect to $WC(\mathcal{P})$ (which is in P). Therefore, we can compute the cost of an optimal answer set by a binary search (between zero and maximum violation) by a polynomial number of oracle calls. With a further oracle call we can ask whether there exists an optimal answer set I such that $a \in I$, respectively $a \notin I$, and hence decide cautious inference, respectively brave inference. Since we used a polynomial number of independent Σ_3^P oracle calls, the problem is in Δ_4^P.

For DL[not, w] programs we proceed in the same way, but in this case an oracle for Σ_2^P problems is sufficient (see also Lemma 3). Therefore, we obtain Δ_3^P-membership. □

Lemma 15 *For* DL[not$_s$, w] *programs, both brave and cautious inference are* Δ_2^P*-complete, where hardness holds also for* DL[not$_s$].

Proof Membership holds, since the number of strata is polynomially bounded. Hardness can be shown by the same construction as in the proof of Lemma 5 but instead of adding :- $val_{x_n}(0)$. to $\mathcal{P}_{lmax}$, we check here whether $val_{x_n}(1)$ is contained in the (unique) optimal answer set of $\mathcal{P}_{lmax}$. □

Lemma 16 *For* DL[$\vee_h$, w] *programs, both inference tasks are* Δ_3^P*-hard.*

Proof Consider the open QBF $\Phi[X] = \exists Y c_1 \wedge \cdots \wedge c_k$, with $X = \{x_1, \ldots, x_n\}$. Deciding the last bit of the lexicographic maximum assignment to the atoms $x_1, \ldots, x_n$ making $\Phi[X]$ false is Δ_3^P-complete.

Consider now the program $\mathcal{P}$ which extends $\mathcal{P}_{QBF}$ by the weak constraints $:\sim w.\,[: n+1]$ and $:\sim f(x_i).\,[: n-i+1]$ for each $i \in \{1, \ldots, n\}$. As in previous proofs, $\mathcal{P}$ is positive and HCF. The answer sets of *Rules*($\mathcal{P}$) correspond to all possible truth assignments to X and contain w iff $\Phi[X]$ evaluates to true under the corresponding guess for X. Now we are interested in those assignments making $\Phi[X]$ false and w.l.o.g. we assume that at least one such assignment exists. The intuition of the weak constraints then is as follows: If w is in an answer set of *Rules*($\mathcal{P}$) then the highest penalty is given. For the remaining ones, we first eliminate those where x_1 is set to false, then those where x_2 is set to false, and so on. The unique optimal answer set

of $\mathcal{P}$ thus corresponds to the lexicographic maximum assignment to X which makes $\Phi[X]$ false. Hence, via both brave and cautious reasoning, we can decide the last bit of this assignment. □

Lemma 17 *For* DL$[\vee, w]$ *programs, both inference tasks are* Δ_4^P*-hard.*

Proof The proof is similar to the one of Lemma 16; the differences mirror the lifting between the proofs of Lemmas 6 and 7, respectively. In fact, consider the open QBF $\Phi[X_1] = \forall X_2 \exists Y c_1 \wedge \cdots \wedge c_k$ with $X_1 = \{x_1, \ldots, x_n\}$. Deciding the last bit of the lexicographic maximum satisfying truth assignment to the atoms $x_1, \ldots, x_n$ for $\Phi[X_1]$ is Δ_4^P-complete. Again, w.l.o.g. we assume that at least one such assignment exists. Consider *Rules*$(\mathcal{P})$, where $\mathcal{P}$ is as in Lemma 7 and extend this program by adding a fresh atom q to the head of all rules. Let $\mathcal{Q}$ denote the extended program, which is positive and disjunctive, but not HCF. By the extension, compared to $\mathcal{P}$, the program $\mathcal{Q}$ has one additional answer set $\{q\}$. Due to minimality, the other answer sets do not contain q, which means that they are the same as those for $\mathcal{P}$. Therefore, as for $\mathcal{P}$ in the proof of Lemma 7, we obtain that w is in an answer set of $\mathcal{Q}$ iff $\Phi' = \forall X_2 \exists Y (c_1 \wedge \cdots \wedge c_k)\sigma$ is true for the respective truth assignment σ on X_1, i.e., iff $\Phi[X_1]\sigma$ is true.

We then add weak constraints $:\sim \text{not}\, w.\ [:n+1]$ and $:\sim f(x_i).\ [:n-i+1]$ for each $i \in \{1, \ldots, n\}$, giving the highest penalty if w is not in the answer set. Since by assumption at least one satisfying assignment, i.e. at least one answer set containing w, exists, this in particular eliminates the answer set $\{q\}$, and all answer sets such that $\Phi[X_1]\sigma$ is false. From the remaining ones, which all are satisfying assignments (i.e., which all contain w), the weak constraints first eliminate those, where x_1 is set to false, then those where x_2 is set to false, and so on. Thus, we get that the optimal answer set of the resulting program corresponds to the maximal truth assignment to variables X_1 such that $\Phi[X_1]$ is true. Both brave and cautious reasoning therefore allow to decide the last bit of this assignment. Hence, we derive Δ_4^P-hardness. □

This concludes the collection of results for the forthcoming theorem which summarizes the complexity results for brave and cautious reasoning under bounded (intensional) predicate arities.

Theorem 2 *The complexity of brave and cautious reasoning under bounded predicate arities is given by the respective entries in Table 5.*

As before, these results show, that if we move from ground (i.e., propositional) programs to non-ground programs but allow only predicates with small arity, the complexity of the language moves up one level in the polynomial hierarchy.

5 Complexity of strong equivalence

In this section, we provide several results for strong equivalence. We start by considering the problem without weak constraints. Afterwards we introduce a notion for strong equivalence between programs possibly containing weak constraints as well. To the best of our knowledge, the problem of strong equivalence has not been

considered in combination with weak constraints, so far. For the sake of simplicity, we omit strong negation in this section; the complexity behavior of the strong equivalence problems considered is not affected by this.

5.1 Strong equivalence without weak constraints

In the setting without weak constraints, the problem of strong equivalence is to test, given two programs $\mathcal{P}_1, \mathcal{P}_2$, whether the answer sets of $\mathcal{P}_1 \cup \mathcal{P}$ and $\mathcal{P}_2 \cup \mathcal{P}$ coincide for any further program, i.e., set of rules, $\mathcal{P}$. We denote the test for strong equivalence between $\mathcal{P}_1$ and $\mathcal{P}_2$ by $\mathcal{P}_1 \equiv_s \mathcal{P}_2$. We restrict the problem to subclasses by assuming that $\mathcal{P}_1$ and $\mathcal{P}_2$ are given from such a subclass DL[L], $w \notin \{L\}$. However, we do not apply this restriction analogously to the context programs $\mathcal{P}$. Indeed, such restrictions to disjunction-free, negation-free, or even Horn programs do not make any difference; this is due to the fact that strong equivalence between two programs $\mathcal{P}_1, \mathcal{P}_2$, can be decided by taking only unary programs (that are programs built from simple rules of the form $a.$ or a :- b.) as context programs $\mathcal{P}$ into account [16]. In contrast, syntactic concepts which are defined over an entire program (stratification, head cycles) allow for different ways how to incorporate such restrictions to the comparison. For instance, one can define that either only context programs $\mathcal{P}$ are considered which separately satisfy the restriction or the combined programs $\mathcal{P}_1 \cup \mathcal{P}$ and $\mathcal{P}_2 \cup \mathcal{P}$ jointly have to satisfy the respective syntactic restriction. Moreover, there are several ways to concretely define the latter kind of equivalence, by considering questions, like what should happen if $\mathcal{P}_1 \cup \mathcal{P}$ is, say stratified, but $\mathcal{P}_2 \cup \mathcal{P}$ is not. We leave these issues for future work and mention that questions of this kind have been partly addressed in related work [18, 53].

Hence, we formally define the problem of deciding strong equivalence with respect to a given class DL[L] as follows: Given programs $\mathcal{P}_1, \mathcal{P}_2$ from DL[L] (with $w \notin \{L\}$), does each further program $\mathcal{P}$ from DL[$\vee$, not] satisfy $\mathcal{AS}(\mathcal{P}_1 \cup \mathcal{P}) = \mathcal{AS}(\mathcal{P}_2 \cup \mathcal{P})$? Towards our complexity analysis, this definition allows to draw immediate conclusions for different classes as before, i.e., for showing that for program classes DL[L_1] $\subseteq$ DL[L_2] $\subseteq \ldots \subseteq$ DL[L_k], deciding strong equivalence between DL[L_i] programs is complete for a complexity class C, it suffices to prove C-hardness for DL[L_1] and C-membership for DL[L_k].

Towards our results, let us first recall an important concept. Strong equivalence can be decided via a certain model-theoretic characterization, so-called SE-models [66], which have been defined for non-ground programs in [16]. Roughly speaking an SE-model for a program $\mathcal{P}$ is a pair (X, Y) of ground interpretations $X \subseteq Y$, such that Y satisfies a grounding of $\mathcal{P}$ and X satisfies the respective reduct with respect to Y. We use a slightly adapted version for our purpose: Given a program $\mathcal{P}$, let $U_\mathcal{P}^+$ be the *extended* Herbrand Universe which is built as usual, but taking an additional set of constants into account, i.e, $U_\mathcal{P}^+ = U_\mathcal{P} \cup \{c_1, \ldots, c_n\}$ where $c_1, \ldots, c_n$ are new constants disjoint from $U_\mathcal{P}$ and n is the maximal number of different variables occurring in a rule (or weak constraint) of $\mathcal{P}$. Moreover, let $B_\mathcal{P}^+$ be the set of all ground literals built from the predicate symbols in $\mathcal{P}$ and the constants $U_\mathcal{P}^+$. Note that in case $\mathcal{P}$ is given over bounded arities, the size of $B_\mathcal{P}^+$ remains polynomial in $\mathcal{P}$, and for a propositional program $\mathcal{P}$, B_P^+ is the set of the (0-ary) predicates in $\mathcal{P}$.

The following result follows implicitly from the characterization in [16].

Proposition 2 *Let $\mathcal{P}_1, \mathcal{P}_2$ be programs without weak constraints and $U = U^+_{\mathcal{P}_1 \cup \mathcal{P}_2}$. Then $\mathcal{P}_1 \equiv_s \mathcal{P}_2$ holds iff $SE_U(\mathcal{P}_1) = SE_U(\mathcal{P}_2)$ where, for $i \in \{1, 2\}$, $SE_U(\mathcal{P}_i)$ contains each pair (X, Y), $X \subseteq Y \subseteq B^+_{\mathcal{P}_1 \cup \mathcal{P}_2}$, such that Y satisfies Ground$(\mathcal{P}_i, U)$, and X satisfies Ground$(\mathcal{P}_i, U)^Y$.*

From now on, we call elements from $SE_U(\cdot)$ SE-models, hence SE-models here are always implicitly understood relative to a strong-equivalence problem $\mathcal{P}_1 \equiv_s \mathcal{P}_2$.

Example 4 We illustrate the characterization using parts of our example for graph-coloring as introduced in Example 2. Indeed, in $\mathcal{P}_{3col}$ the disjunctive guess r

$$red(X) \vee green(X) \vee blue(X) \text{ :- } v(X).$$

can be replaced by its so-called shift, $r^{\rightarrow}$, given by

$$\begin{aligned} red(X) &\text{ :- } v(X), \text{not } green(X), \text{not } blue(X).\\ green(X) &\text{ :- } v(X), \text{not } red(X), \text{not } blue(X).\\ blue(X) &\text{ :- } v(X), \text{not } green(X), \text{not } red(X). \end{aligned}$$

without changing the answer sets for any input graph. However, one may ask whether r can faithfully be replaced by $r^{\rightarrow}$ in *any program*. By definition the latter holds iff $\mathcal{P}_1 = \{r\}$ is strongly equivalent to $\mathcal{P}_2 = \{r^{\rightarrow}\}$.

In terms of the characterization, we have $U = U^+_{\mathcal{P}_1 \cup \mathcal{P}_2} = \{c\}$ and $B^+_{\mathcal{P}_1 \cup \mathcal{P}_2} = \{v(c), red(c), green(c), blue(c)\}$. One can check that the program Ground$(\mathcal{P}_2, U)$—which is easily obtained by replacing variable X by constant c—has $(\{v(c)\}, \{v(c), r(c), g(c), b(c)\}) \in SE_U(\mathcal{P}_2)$ which is however not contained in $SE_U(\mathcal{P}_1)$. Thus, by Proposition 2, $\mathcal{P}_1 = \{r\}$ and $\mathcal{P}_2 = \{r^{\rightarrow}\}$ are not strongly equivalent, and thus r cannot be faithfully replaced by $r^{\rightarrow}$ in any program. For instance, consider

$$\mathcal{P} = \{v(c).\, r(X) \text{ :- } g(X).\, g(X) \text{ :- } b(X).\, b(X) \text{ :- } r(X).\}.$$

Then, $\{v(c), r(c), g(c), b(c)\} \in \mathcal{AS}(\mathcal{P} \cup \{r\})$, while $\mathcal{AS}(\mathcal{P} \cup \{r^{\rightarrow}\}) = \emptyset$.

The membership results for strong equivalence problems of program class DL[L] are as follows.

Lemma 18 *Strong equivalence between* DL[] *programs is in* NP*, even for Horn programs with constraints.*

Proof For Horn programs, we have $\mathcal{P}_1 \equiv_s \mathcal{P}_2$ iff $\mathcal{P}_1 \equiv \mathcal{P}_2$, i.e., iff $\mathcal{P}_1$ and $\mathcal{P}_2$ are classically equivalent (see, e.g., [16]). Thus, we can decide the problem by checking $\mathcal{P}_1 \models r$, for each $r \in \mathcal{P}_2$; and $\mathcal{P}_2 \models s$, for each $s \in \mathcal{P}_1$, where $\models$ stands for classical semantic consequence. Each such test, say $\mathcal{P}_1 \models r$, holds iff for each possible ground substitution θ for the head of r, $\mathcal{P}_1 \cup B(r\theta) \models H(r\theta)$ holds. Note that only a polynomial number of different such ground substitutions are possible, since the head is a single atom with bounded arity, but there may be variables left in $B(r\theta)$. It remains to get rid of these remaining variables in $B(r\theta)$, since then we can directly make use of the skeptical reasoning problem for Horn programs, which is contained in NP (cf. Lemma 8), even for Horn programs with constraints. In fact, substituting the remaining variables in $B(r\theta)$ by disjoint constants from $U^+_{\mathcal{P}_1 \cup \mathcal{P}_2} \setminus U_{\mathcal{P}_1 \cup \mathcal{P}_2}$, provides a sufficient test for $\mathcal{P}_1 \cup B(r\theta) \models H(r\theta)$, which follows from the well-known strong

monotonicity property of plain datalog: given a datalog program $\mathcal{P}$, a database D and a ground atom a on universe U, for any mapping $\rho : U \to U$, we have $\mathcal{P} \cup D \models a$ implies $(\mathcal{P} \cup D)\rho \models a\rho$. This shows that a polynomial number of independent NP-tests decides $\mathcal{P}_1 \equiv_s \mathcal{P}_2$. Thus the problem itself is in NP. □

Lemma 19 *Strong equivalence between* DL[not, $\vee$] *programs is in* Π_2^P.

Proof We guess a pair (X, Y) with $X \subseteq Y \subseteq B^+_{\mathcal{P}_1 \cup \mathcal{P}_2}$ and check whether it is an SE-model of exactly one of the programs. Recall that the size of (X, Y) is polynomial in $\mathcal{P}_1 \cup \mathcal{P}_2$. Given (X, Y), $\mathcal{P}_1$, $\mathcal{P}_2$ (and thus also $U = U^+_{\mathcal{P}_1 \cup \mathcal{P}_2}$), checking $(X, Y) \in SE_U(\mathcal{P}_i)$ is in co-NP. This can be seen by similar arguments as used for Lemma 1. Thus we solve the problem in nondeterministic polynomial time with two calls to an NP-oracle for SE-model checking. □

We now turn to the matching lower-bound results.

Lemma 20 *Strong equivalence between* DL[] *programs is* NP*-hard, even for definite Horn programs.*

Proof We again use the NP-complete problem of conjunctive query evaluation (see also proof of Lemma 4). Given a query a :- B and a database DB, consider programs $\mathcal{P}_1 = DB \cup \{a \text{ :- } B.\}$ and $\mathcal{P}_2 = DB \cup \{a \text{ :- }.\}$. Obviously, $\mathcal{P}_1, \mathcal{P}_2$ are definite Horn and polynomial in size of the database plus the query. To see that $\mathcal{P}_1 \equiv_s \mathcal{P}_2$ holds iff a :- B evaluates to true under DB, one can use the fact that for positive programs $\mathcal{P}_1, \mathcal{P}_2$, $\mathcal{P}_1 \equiv_s \mathcal{P}_2$ holds iff for each arbitrary further database F, $\mathcal{AS}(\mathcal{P}_1 \cup F) = \mathcal{AS}(\mathcal{P}_2 \cup F)$ [16]. The result is then easily seen. □

Concerning the remaining results, note that for the problem of strong equivalence we are not allowed to exploit the fact that normal and head-cycle-free programs are closely related (as we did in the previous sections). Indeed, the polynomial-time rewriting of HCF programs to equivalent normal programs [4], is not faithful with respect to strong equivalence, as is witnessed by Example 4. However, to obtain Π_2^P-hardness for positive HCF programs, a straightforward adaption of previous constructions is sufficient.

Lemma 21 *Strong equivalence between* DL[$\vee_h$] *programs is* Π_2^P*-hard.*

Proof Once more, we use a mapping from QBFs of the form $\Phi = \forall X \exists Y c_1 \wedge \cdots \wedge c_k$ to equivalence problems. For a given Φ of that form, consider the programs $\mathcal{P}_1 = \mathcal{P}_{QBF} \cup \{w \text{ :- } t(x_i),\ f(x_i).\mid x_i \in X\}$, where $\mathcal{P}_{QBF}$ is the program already used in the proof of Lemma 6, and $\mathcal{P}_2$ which is obtained from $\mathcal{P}_1$ by replacing Rule (2) by w :- . The function of $\mathcal{P}_{QBF}$ has already been discussed in previous proofs. The additional rules w :- $t(x_i)$, $f(x_i)$. guarantee that all ground models (not only minimal ones) not containing w refer to a valid assignment for the variables X. Then, it can be seen that $\mathcal{P}_1 \equiv_s \mathcal{P}_2$ holds iff Φ is true. In particular, we have $\mathcal{P}_1 \not\equiv_s \mathcal{P}_2$ iff there exists a model I of $\mathcal{P}_1$ not containing w. That is, there exists an assignment to X, such that all possible assignments to Y make $c_1 \wedge \cdots \wedge c_k$ false, i.e. Φ is false. Both programs are

positive HCF and polynomial-time constructible with respect to the size of Φ. Since the evaluation problem for QBFs having the form of Φ is Π_2^P-complete, the result follows. □

For the case of stratified programs, some additional construction is required, for which we reuse ideas from the co-NP-hardness proof for ground stratified programs in [18].

Lemma 22 *Strong equivalence between* DL[not_s] *programs is* Π_2^P*-hard.*

Proof Again consider any QBF of the form $\Phi = \forall X \exists Y c_1 \wedge \cdots \wedge c_k$. The construction for the two programs $\mathcal{P}_1, \mathcal{P}_2$ is now as follows, starting again from $\mathcal{P}_{QBF}$:

1. In both programs, remove the disjunctions (1) and add $\{w \text{ :- } t(x_i), f(x_i). \mid x_i \in X\}$.
2. In both programs, add now for each rule w :- B. a new rule v :- B., where v is a new fixed atom.
3. Add to $\mathcal{P}_1$ the rule w :- not v. and to $\mathcal{P}_2$ the rule v :- not w.

The two resulting programs work as follows. First suppose Φ is true, then each model of $\mathcal{P}_1$, resp. $\mathcal{P}_2$, contains w and v. Hence, the only differing rules w :- not v. and v :- not w. do not come into play and $\mathcal{P}_1 \equiv_s \mathcal{P}_2$ follows quite easily. Now suppose Φ is false. Then, we have an interpretation I, which satisfies both $\mathcal{P}_1 \setminus \{w \text{ :- not } v.\}$ and $\mathcal{P}_2 \setminus \{v \text{ :- not } w.\}$, but does neither contain w nor v. Assume now the interpretation $I \cup \{w\}$. Clearly, $I \cup \{w\}$ satisfies both $\mathcal{P}_1$ and $\mathcal{P}_2$, but for $\mathcal{P}_1$, the fact w is included in the reduct with respect to $I \cup \{w\}$, while the rule v :- not w. from $\mathcal{P}_2$ gets completely deleted within the reduct. Hence, the models of the respective reducts now have to differ, i.e. I satisfies the reduct built from $\mathcal{P}_2$, but does not satisfy the corresponding reduct built from $\mathcal{P}_1$ which contains the fact w. Using Proposition 2, we arrive at $\mathcal{P}_1 \not\equiv_s \mathcal{P}_2$. This shows that $\mathcal{P}_1 \equiv_s \mathcal{P}_2$ holds iff Φ is true. Both programs are stratified disjunction-free and polynomial-time constructible with respect to the size of Φ. Π_2^P-hardness thus follows. □

5.2 Strong equivalence and weak constraints

We extend our results to programs with weak constraints. As already mentioned, so far strong equivalence combined with weak constraints has not been considered in the literature, and different ways to define this problem are possible. Here, we basically reuse the definition of the traditional strong equivalence problem as follows: Given programs $\mathcal{P}_1, \mathcal{P}_2$, then $\mathcal{P}_1 \equiv_s \mathcal{P}_2$ holds iff, for each further program $\mathcal{P}$, $\mathcal{AS}(\mathcal{P}_1 \cup \mathcal{P}) = \mathcal{AS}(\mathcal{P}_2 \cup \mathcal{P})$. Note that this definition now allows for weak constraints in $\mathcal{P}_1, \mathcal{P}_2$, but also in all context programs $\mathcal{P}$, and we thus compare the optimal answer sets of each program extension. Consequently, this notion of strong equivalence also yields the most natural way to obtain equivalence for substitution in logic programming with weak constraints.

To define strong equivalence problems with respect to subclasses, we restrict the compared programs (in analogy to the previous subsection) but leave the context ranging over arbitrary programs. Formally the problem of deciding strong equivalence with respect to a given class DL[L] is now as follows: Given programs $\mathcal{P}_1, \mathcal{P}_2$ from DL[L], does each further program $\mathcal{P}$ from DL[$\vee$, not, w] satisfy $\mathcal{AS}(\mathcal{P}_1 \cup \mathcal{P}) =$

$\mathcal{AS}(\mathcal{P}_2 \cup \mathcal{P})$? Note that to decide strong equivalence between $\mathcal{P}_1, \mathcal{P}_2 \in \mathrm{DL}[L]$ programs where $w \notin \{L\}$, we can safely ignore the weak constraints in the possible context programs. In fact, this is due to the following observations: (1) if $\mathcal{AS}(\mathcal{P}_1 \cup \mathcal{P}) = \mathcal{AS}(\mathcal{P}_2 \cup \mathcal{P})$ holds for all $\mathcal{P} \in \mathrm{DL}[\vee, \mathrm{not}]$, then we can add any additional weak constraints to the context $\mathcal{P}$, and since the same weak constraints are added to both programs, the same answer sets are selected; (2) if $\mathcal{AS}(\mathcal{P}_1 \cup \mathcal{P}) = \mathcal{AS}(\mathcal{P}_2 \cup \mathcal{P})$ for all $\mathcal{P} \in \mathrm{DL}[\vee, \mathrm{not}, w]$, then the same holds for all $\mathcal{P} \in \mathrm{DL}[\vee, \mathrm{not}]$, simply since $\mathrm{DL}[\vee, \mathrm{not}] \subseteq \mathrm{DL}[\vee, \mathrm{not}, w]$. Thus, our more general definition coincides with the traditional definition as long as no weak constraints are present in the compared programs.

We next provide a novel characterization to decide strong equivalence involving weak constraints by properly generalizing the characterization from Proposition 2. Roughly speaking, we have to additionally test whether the possible candidates for answer sets of program extensions pairwise show the same difference in their penalties. In the following, we assume for the compared programs $\mathcal{P}_1$, $\mathcal{P}_2$, that $f_{\mathcal{P}_1}(i) = f_{\mathcal{P}_2}(i)$ holds, for each $i > 0$ [for the exact definition of $f_{\mathcal{P}}(\cdot)$ see Section 2]. Note that this is not a serious restriction, since we can add, for instance, dummy weak constraints $:\sim b_i, \mathrm{not}\ b_i.\ [w_{\max}^{\mathcal{P}_1} : 1]$, $1 \leq i \leq |WC(\mathcal{P}_1)|$, to $\mathcal{P}_2$, and, vice versa, weak constraints $:\sim b_i, \mathrm{not}\ b_i.\ [w_{\max}^{\mathcal{P}_2} : 1]$, $1 \leq i \leq |WC(\mathcal{P}_2)|$, to $\mathcal{P}_1$. Then, both resulting programs possess the same number of weak constraints and share the same maximum weight, but the extensions do neither change the semantics of the two programs, nor result in an exponential blow up of the problem size. Thus, we assume that this kind of "pre-processing" has already taken place.

Moreover, we need the following concept: Let $\mathcal{P}$ be a program, Y, Z ground interpretations and $i > 0$, then

$$\delta_i^{\mathcal{P}}(Y, Z) = \sum_{w \in N_i^{\mathcal{P}}(Y)} \mathit{weight}(w) - \sum_{w \in N_i^{\mathcal{P}}(Z)} \mathit{weight}(w)$$

denotes the difference on penalization between Y and Z in $\mathcal{P}$ on level i.

Lemma 23 *For any programs $\mathcal{P}_1$, $\mathcal{P}_2$ from class* DL[not, $\vee$, w], *$\mathcal{P}_1 \equiv_s \mathcal{P}_2$ holds iff jointly $\mathit{Rules}(\mathcal{P}_1) \equiv_s \mathit{Rules}(\mathcal{P}_2)$ and for all ground interpretations $Y, Z \subseteq B^+_{\mathcal{P}_1 \cup \mathcal{P}_2}$ satisfying $\mathit{Rules}(\mathcal{P}_1 \cup \mathcal{P}_2)$, $\delta_i^{\mathcal{P}_1}(Y, Z) = \delta_i^{\mathcal{P}_2}(Y, Z)$ holds for each level $i > 0$.*

Proof If. First suppose, $\mathit{Rules}(\mathcal{P}_1) \equiv_s \mathit{Rules}(\mathcal{P}_2)$ does not hold. Hence, there exists a ground interpretation Y, and a set of rules $\mathcal{P}$, such that without loss of generality $Y \in \mathcal{AS}(\mathit{Rules}(\mathcal{P}_1) \cup \mathcal{P}) \setminus \mathcal{AS}(\mathit{Rules}(\mathcal{P}_2) \cup \mathcal{P})$. We now can penalize all other answer sets for $\mathit{Rules}(\mathcal{P}_1) \cup \mathcal{P}$ in order to ensure that Y becomes an optimal answer set for some program extension. For instance, take $\mathcal{P}' = \mathcal{P} \cup \{:\sim \mathrm{not}\ y.\ [M : 1] \mid y \in B_{\mathcal{P}_1 \cup \mathcal{P}} \setminus Y\}$, where M is a sufficiently large weight. In fact, this penalizes each answer set of $\mathit{Rules}(\mathcal{P}_1 \cup \mathcal{P})$ different from Y, having in mind that no $Y' \subset Y$ can be answer set of $\mathit{Rules}(\mathcal{P}_1 \cup \mathcal{P})$. Then, Y is an optimal answer set of $\mathcal{P}_1 \cup \mathcal{P}'$ while Y is not an optimal answer set of $\mathcal{P}_2 \cup \mathcal{P}'$. This yields $\mathcal{P}_1 \not\equiv_s \mathcal{P}_2$. Now suppose, $\mathit{Rules}(\mathcal{P}_1) \equiv_s \mathit{Rules}(\mathcal{P}_2)$ holds but there exist ground interpretations Y, Z, satisfying $\mathit{Rules}(\mathcal{P}_1 \cup \mathcal{P}_2)$, such that, for some i, $\delta_i^{\mathcal{P}_1}(Y, Z) \neq \delta_i^{\mathcal{P}_2}(Y, Z)$. Now, take the program

$$\mathcal{P} = \{a \vee b :\text{-}\ .\} \cup \{y :\text{-}\ a.\mid y \in Y\} \cup \{z :\text{-}\ b.\mid z \in Z\} \cup \{wc_i \mid 1 \leq i \leq l_{\max}^{\mathcal{P}_1 \cup \mathcal{P}_2}\}$$

where wc_i is a weak constraint of the form $:\sim B_i.\ [\delta_i^{\mathcal{P}_1}(Y, Z) : i]$ where B_i is the sequence of the elements from Y if $\delta_i^{\mathcal{P}_1}(Y, Z) < 0$ and the sequence of the elements from Z, otherwise; a, b are new propositional atoms. Then, Y and Z are the only answer sets of $Rules(\mathcal{P}_i \cup \mathcal{P})$ for $i \in \{1, 2\}$. Observe that $\mathcal{P}_1 \cup \mathcal{P}$ now puts the same penalty to Y and Z at each level, and thus Y and Z are both also the optimal answers sets of $\mathcal{P}_1 \cup \mathcal{P}$. It remains to show that $\mathcal{P}_2 \cup \mathcal{P}$ possesses only one optimal answer set. To this end, let i be the maximal level, such that $\delta_i^{\mathcal{P}_1}(Y, Z) \neq \delta_i^{\mathcal{P}_2}(Y, Z)$. Then, by definition of $\mathcal{P}$, Y and Z remain differently penalized on level i in $\mathcal{P}_2 \cup \mathcal{P}$. Moreover, the weak constraints from levels lower than i cannot equalize this difference by definition of $f_{\mathcal{P}_2 \cup \mathcal{P}}(\cdot)$. Thus Y and Z possess different sums of penalties for the violated weight constraints in $\mathcal{P}_2 \cup \mathcal{P}$ having level $\leq i$. On all higher levels $j > i$ the penalties are equalized for Y and Z by definition of $\mathcal{P}$, since we assumed $\delta_j^{\mathcal{P}_1}(Y, Z) = \delta_j^{\mathcal{P}_2}(Y, Z)$, for all $j > i$. Hence, either Y or Z is an optimal answer set of $\mathcal{P}_2 \cup \mathcal{P}$, but not both of them. This shows $\mathcal{P}_1 \not\equiv_s \mathcal{P}_2$.

Only-If. For $\mathcal{P}_1 \not\equiv_s \mathcal{P}_2$, suppose without loss of generality that Y is an optimal answer set of $\mathcal{P}_1 \cup \mathcal{P}$ but not an optimal answer set of $\mathcal{P}_2 \cup \mathcal{P}$. If Y is not even an answer set of $Rules(\mathcal{P}_2 \cup \mathcal{P})$ we obtain $Rules(\mathcal{P}_1) \not\equiv_s Rules(\mathcal{P}_2)$. So suppose Y is answer set of $Rules(\mathcal{P}_2 \cup \mathcal{P})$ but not optimal. Hence, there exists an answer set Z of $\mathcal{P}_2 \cup \mathcal{P}$, such that $H^{\mathcal{P}_2 \cup \mathcal{P}}(Z) < H^{\mathcal{P}_2 \cup \mathcal{P}}(Y)$. Now since Y is optimal for $\mathcal{P}_1 \cup \mathcal{P}$, we obtain that Z is either no answer set of $Rules(\mathcal{P}_1 \cup \mathcal{P})$ or $H^{\mathcal{P}_1 \cup \mathcal{P}}(Z) \geq H^{\mathcal{P}_1 \cup \mathcal{P}}(Y)$. In the former case, we again get $Rules(\mathcal{P}_1) \not\equiv_s Rules(\mathcal{P}_2)$. In the latter case, first observe that both Y and Z satisfy $Rules(\mathcal{P}_1 \cup \mathcal{P}_2)$. Moreover, by the assumption that $f_{\mathcal{P}_1}(i) = f_{\mathcal{P}_2}(i)$ holds for all $i > 0$, we get that for any program $\mathcal{P}$, $\hat{f}_{\mathcal{P}}(i) := f_{\mathcal{P}_1 \cup \mathcal{P}}(i) = f_{\mathcal{P}_2 \cup \mathcal{P}}(i)$, for all $i > 0$. So, let for any ground interpretation X,

$$\hat{H}^{\mathcal{P}}(X) = \sum_{i=1}^{l_{\max}^{\mathcal{P}}} \left(\hat{f}_{\mathcal{P}}(i) \cdot \sum_{w \in N_i^{\mathcal{P}}(X)} weight(w) \right);$$

and, in addition, for $j \in \{1, 2\}$,

$$\hat{H}^{\mathcal{P}_j;\mathcal{P}}(X) = \sum_{i=1}^{l_{\max}^{\mathcal{P}_j}} \left(\hat{f}_{\mathcal{P}}(i) \cdot \sum_{w \in N_i^{\mathcal{P}_j}(X)} weight(w) \right).$$

Then, for any ground interpretation X, $j \in \{1, 2\}$, and any program $\mathcal{P}$,

$$H^{\mathcal{P}_j \cup \mathcal{P}}(X) = \hat{H}^{\mathcal{P}_j;\mathcal{P}}(X) + \hat{H}^{\mathcal{P}}(X).$$

Hence, using the assumptions $H^{\mathcal{P}_1 \cup \mathcal{P}}(Z) \geq H^{\mathcal{P}_1 \cup \mathcal{P}}(Y)$ and $H^{\mathcal{P}_2 \cup \mathcal{P}}(Z) < H^{\mathcal{P}_2 \cup \mathcal{P}}(Y)$, we obtain:

$$\left(\hat{H}^{\mathcal{P}_1;\mathcal{P}}(Z) - \hat{H}^{\mathcal{P}_1;\mathcal{P}}(Y)\right) \geq \left(\hat{H}^{\mathcal{P}}(Y) - \hat{H}^{\mathcal{P}}(Z)\right) > \left(\hat{H}^{\mathcal{P}_2;\mathcal{P}}(Z) - \hat{H}^{\mathcal{P}_2;\mathcal{P}}(Y)\right);$$

and thus

$$\hat{H}^{\mathcal{P}_1;\mathcal{P}}(Z) - \hat{H}^{\mathcal{P}_1;\mathcal{P}}(Y) \neq \hat{H}^{\mathcal{P}_2;\mathcal{P}}(Z) - \hat{H}^{\mathcal{P}_2;\mathcal{P}}(Y).$$

It can be checked that then, $\delta_i^{\mathcal{P}_1}(Y, Z) = \delta_i^{\mathcal{P}_2}(Y, Z)$ cannot hold for all $i > 0$. This concludes the proof. □

We illustrate this characterization by comparing two different variants of the 2-coloring approximation program, already used in previous examples.

Example 5 Consider the program $\mathcal{P}_{2col}$ from Example 2 and a slightly changed program $\mathcal{P}^2_{2col}$ where the weak constraint

$$:\sim red(X).\ [1:1]$$

from $\mathcal{P}_{2col}$ is replaced by a weak constraint with different penalization:

$$:\sim red(X).\ [2:1].$$

One can check that $\mathcal{P}_{2col}$ and $\mathcal{P}^2_{2col}$ provide the same optimal answer sets for each added graph, and also for each added program without weak constraints. Indeed, we have that $Rules(\mathcal{P}_{2col}) = Rules(\mathcal{P}^2_{2col})$ and thus $Rules(\mathcal{P}_{2col}) \equiv_s Rules(\mathcal{P}^2_{2col})$. However, do the answer sets also coincide when we add programs which might contain also weak constraints?

For instance, consider our example graph $G = \{v(1).\ v(2).\ v(3).\ e(1,2).\ e(2,3).\}$ and the extension program

$$\mathcal{P} = G \cup \{:\sim green(X).\ [1:1]\}.$$

Then, $\mathcal{P}_{2col} \cup \mathcal{P}$ equally penalizes red and green vertices, and consequently has in its answer sets those colorings with a maximal number of blue vertices, i.e.,

$$\{blue(1), red(2), blue(3)\} \quad \text{and} \quad \{blue(1), green(2), blue(3)\}.$$

For program $\mathcal{P}^2_{2col} \cup \mathcal{P}$, the different weights additionally make green vertices preferred over red vertices. Thus the only answer set of $\mathcal{P}^2_{2col} \cup \mathcal{P}$ is G together with

$$\{blue(1), green(2), blue(3)\}.$$

Hence, $\mathcal{P}_{2col}$ and $\mathcal{P}^2_{2col}$ are not strongly equivalent to each other.

This effect is reflected in our characterization as follows by the fact that for the interpretations $Y = G \cup \{blue(1), red(2), blue(3)\}$ and $Z = G \cup \{blue(1), green(2), blue(3)\}$, we have $\delta^{\mathcal{P}_{2col}}(Y, Z) = 0$ while $\delta^{\mathcal{P}^2_{2col}}(Y, Z) = 1$.

Using this characterization, we obtain the following result as a complexity upper bound.

Lemma 24 *Strong equivalence between* DL[not, $\vee$, w] *programs is in* Π^P_2.

Proof We already know that deciding $Rules(\mathcal{P}_1) \equiv_s Rules(\mathcal{P}_2)$ is in Π^P_2. The remaining check involving the comparison of the classical models with respect to their weights in the different levels is also in Π^P_2. This can be seen as follows. For the complementary problem one can guess ground interpretations Y, Z, and check whether they both satisfy $\mathcal{P}_1 \cup \mathcal{P}_2$, but violate $\delta_i^{\mathcal{P}_1}(Y, Z) = \delta_i^{\mathcal{P}_2}(Y, Z)$ for some i. The first check is in co-NP (cf. Lemma 1). For the second check, one has to sum up the penalties of the weak constraints which are violated by the respective ground interpretations. Since only a polynomial number of ground weak constraints can occur (due to our restriction on the usage of weak constraints), this computation is possible in polynomial time. We obtain an algorithm for the problem which guesses

Table 6 Complexity of strong equivalence under bounded predicate arities

SE	{}	{w}	{not$_s$}	{not$_s$, w}	{not}	{not, w}
{}	NP	NP-hard; in Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P
{$\vee_h$}	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P
{$\vee$}	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P	Π_2^P

Unless stated otherwise, all entries are completeness results.

Y, Z and then runs in polynomial time using several calls to NP-oracles. This shows that the complement of the weight comparison is in Σ_2^P. Hence, the entire problem is contained in Π_2^P. □

By similar argumentation, one can obtain co-NP-membership for the ground case. Thus, for all classes except DL[w] we obtain completeness results which show that weak constraints do not add any complexity under our notion of strong equivalence and we can extend Table 3 in Section 2 with co-NP-entries for all classes DL[L, w], where co-NP-hardness for DL[L] holds.

Currently, it is unknown whether any of the given bounds for deciding strong equivalence between non-ground DL[w] programs with bounded arities is tight, and whether for propositional programs the problem is tractable.

We summarize our results:

Theorem 3 *The complexity of strong equivalence under bounded predicate arities is given by the respective entries in Table 6.*

6 Discussion and implications

In this section, we provide a discussion of the complexity results obtained in this work and outline some of their implications. Perhaps the most important result from an application point of view is Theorem 2, which implies that reasoning tasks over programs with bounded arities are feasible in polynomial space. In particular, the theorem also implies that polynomial reductions to propositional ASP (with disjunction) from HCF programs with bounded arities exist. However, examining competitive ASP systems, we can observe that they currently do not respect this complexity bound, the reason being that they create a ground program which is equivalent to the input, and which in general has exponential size even for programs with bounded arities. In Section 6.1, we provide a brief account on the reasons for this behavior and the nature of the programs that cause it. Furthermore, in Section 6.2 we provide ideas and a sketch for alternative methods that better match the complexity results. Finally, in Section 6.3 we discuss related and further issues.

6.1 Exponential grounding for programs with bounded arity

In this section we provide empirical evidence that current ASP systems do require exponential space for reasoning with programs with bounded predicate arities. In particular, we show that all competitive ASP grounders, that is the grounding module

of DLV [35] (version 2007-10-11), Lparse [65] (version 1.0.17), and GrinGo [25] (version 0.0.1), produce a ground program which is exponential in the size of the input program.

Example 6 Let us first consider the family of programs as considered in Example 1. If the input graph is represented simply by facts, then all three systems are able to fully evaluate the program—that is, they produce only a set of facts (which is not exponential in the input size). However, if the input graph depends on a nondeterministic predicate, which is defined by unstratified negation or disjunction, then this optimized evaluation strategy is no longer applicable. We note that this is not at all an uncommon situation, for example if one wants to compute reachability over paths of length k over subgraphs of a given graph, this is exactly what will happen.

For simplicity, we have added a generic nondeterministic predicate on which the atoms defining the input graph depends in order to simulate such a situation and to prevent the systems to apply the optimization.

We consider programs of Example 1 with graphs as in Fig. 1, where $k = 2n + 1$, and each arc (v_1, v_2) of the input graph is specified by a rule $e(v_1, v_2) \text{ :- } a.$ Additionally, we add the following rules:

$$a \text{ :- not } b. \quad b \text{ :- not } a. \quad \text{ :- not } a.$$

There is one problem: Neither Lparse nor GrinGo will accept this program as is, because these systems only support a restricted language in which each variable in a rule must occur in a positive atom, the predicate of which is defined essentially by a nonrecursive program. In order to obtain a program that meets the syntactic restrictions imposed by Lparse and GrinGo, we consider the following modification of the program:

$$\begin{aligned}
&a \text{ :- not } b. \quad b \text{ :- not } a. \quad \text{ :- not } a.\\
&f(V_1, V_2) \text{ :- } e(V_1, V_2), a.\\
&p_k(X_1, X_k) \text{ :- } e(X_1, X_2), \ldots, e(X_{k-1}, X_k), f(X_1, X_2), \ldots, f(X_{k-1}, X_k).\\
&\mathit{reachable}(X, Y) \text{ :- } p_k(X, Y), e(X, V_1), e(V_2, Y).\\
&\mathit{reachable}(X, Y) \text{ :- } \mathit{reachable}(X, Z), p_k(Z, Y), e(X, V_1), e(V_2, Z), e(V_3, Y).
\end{aligned}$$

The main difference to the original program is that the graph encoding is done by facts in order to be able to provide domain predicates. Then, a new predicate f duplicates the graph, but also depends on a nondeterministic predicate a, thus simulating situations in which subgraphs are to be considered, or in which the graph structure depends on some nondeterministic assumptions. The predicate p_k is then defined by means of f, but adding also corresponding atoms composed with the predicate e, which serve as domain predicates for the variables $X_1, \ldots, X_k$. Furthermore, for defining reachability we have added domain predicates for the variables X, Y, Z in the last two rules. Here, the knowledge that X has to occur as a source vertex in some arc and that Y has to occur as a target vertex in some arc has been exploited. Z must occur both as source and target node in arcs, and we chose its occurrence as target node for providing a domain predicate. The variables V_1, V_2, V_3 just serve for stating that an appropriate arc must exist.

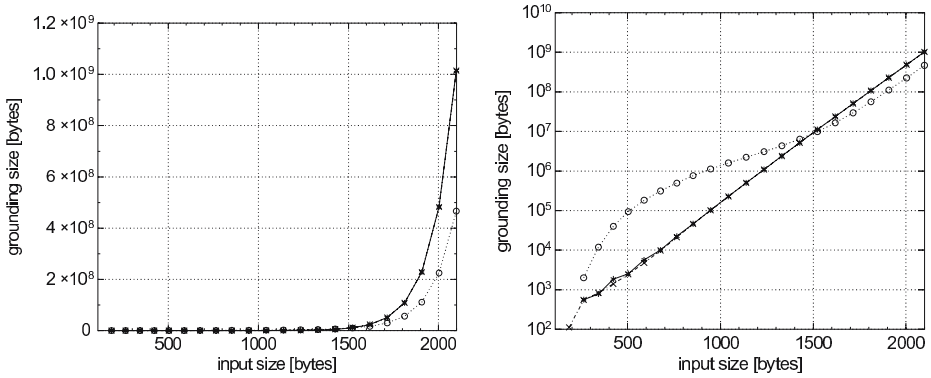

Fig. 2 Grounding size in relation to program size for DLV (*dashed line, crosses*), GrinGo (*solid line, stars*) and Lparse (*dotted line, circles*) for n between 1 and 22. Left in linear scale, right in logarithmic scale

When we compare the size of the input programs with the size of the ground programs for this class of programs in Fig. 2, we can clearly observe an exponential behavior for all three systems. Lparse appears to perform better than the other two systems, however this is only due to the output format: While GrinGo and DLV output valid ground logic programs, Lparse produces a more concise numerical format. Indeed, when we look at the number of ground rules that are produced in Fig. 3, we can see that Lparse actually always creates more rules in the tested range than the other two systems DLV and Gringo, which produce almost equal programs. We have also tried DLV with the version that is not accepted by Lparse and GrinGo and obtained a similar behavior.

In any case, the crucial observation is that all systems have an exponential behavior for this class of programs.

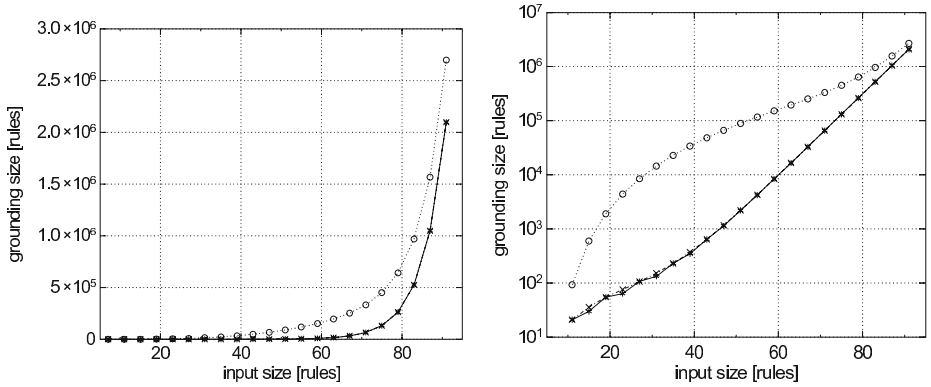

Fig. 3 Ground rules produced by DLV (*dashed line, crosses*), GrinGo (*solid line, stars*) and Lparse (*dotted line, circles*) for n between 1 and 22. Left in linear scale, right in logarithmic scale

Let us now look at another, more generic class of programs.

Example 7 Consider the following class of programs with predicate arity bounded by 1:

$$
\begin{aligned}
&e(0).\ e(1).\\
&p(X) \text{ :- } e(X), \text{not } q(X).\\
&q(X) \text{ :- } e(X), \text{not } p(X).\\
&r \text{ :- } p(X_1), \ldots, p(X_k).
\end{aligned}
$$

One would expect an exponential grounding also in this case. DLV , however, applies an optimization and transforms the program to

$$
\begin{aligned}
&e(0).\ e(1).\\
&p(X) \text{ :- } e(X), \text{not } q(X).\\
&q(X) \text{ :- } e(X), \text{not } p(X).\\
&r \text{ :- } p', \ldots, p'.\\
&p' \text{ :- } p(X).
\end{aligned}
$$

where p' is a new predicate, exploiting the fact that the variables $X_1, \ldots X_k$ have a single occurrence in the original rule. The transformed program always produces 9 ground rules, independent of k; only the size of the penultimate rule (which is already ground) depends on k.

Lparse, on the other hand, refuses to ground this program because it is not domain-restricted. GrinGo does accept this kind of programs and indeed produces an exponential ground program.

In order to obtain a uniform input program to all systems, which as a side-effect also impedes DLV from applying the optimization, we consider the following (strongly equivalent) variant of the program:

$$
\begin{aligned}
&e(0).\ e(1).\\
&p(X) \text{ :- } e(X), \text{not } q(X).\\
&q(X) \text{ :- } e(X), \text{not } p(X).\\
&r \text{ :- } p(X_1), \ldots, p(X_k), e(X_1), \ldots, e(X_k).
\end{aligned}
$$

Lparse already produces around 70 MB of ground output for $k = 20$, and exceeds 1 GB of main memory consumption for $k = 22$, producing around 300 MB of ground output. DLV consumes less memory, but still produces about 17 MB of ground output for $k = 20$, stays just below 1 GB of main memory consumption when producing about 70 MB of ground output for $k = 22$. Gringo uses only a few megabytes of main memory, but produces more than 500 MB of ground output (the difference to Lparse is due to the more concise output format of Lparse). Figure 4 illustrates the exponential space behavior of the systems.

Interestingly, all three systems produce essentially the same amount of ground rules, as evidenced in Fig. 5. In this case, the smaller size (in bytes) of the ground output produced by DLV is due to the fact that DLV produces shorter ground

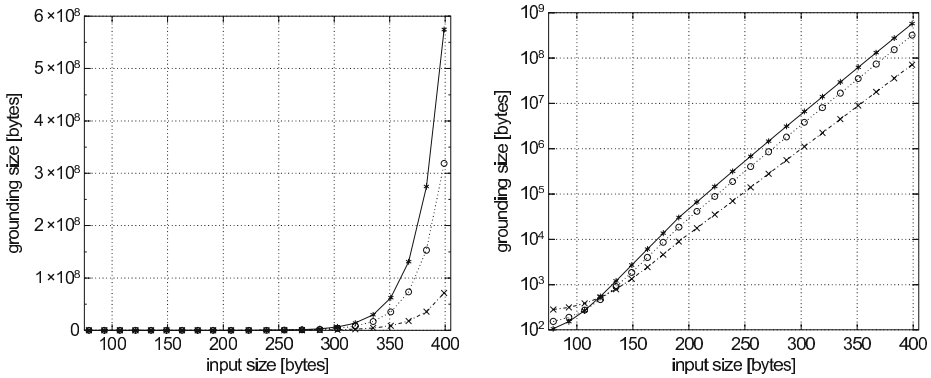

Fig. 4 Grounding size in relation to program size for DLV (*dashed line, crosses*), GrinGo (*solid line, stars*) and Lparse (*dotted line, circles*) for k between 1 and 22. Left in linear scale, right in logarithmic scale

instantiations of the last rule, as it omits ground atoms for $e(X_1), \ldots, e(X_k)$ as an optimization. In fact, for these programs this even compensates the more concise output format of Lparse.

Indeed, the ground programs provided by Lparse will contain $6 + 2^k$ rules: The two facts are included directly, the rules $p(X)$:- $e(X)$, not $q(X)$. and $p(X)$:- $e(X)$, not $q(X)$. give rise to two ground rules each (one for each fact of predicate e), and 2^k ground rules for the final rule. Note that the number of variables per rule (k) is not bounded, and therefore depends on the size of the input.

We observe that in programs as in these examples an exponential grounding is generated by the systems, while only a small part of it is relevant for the computation. Indeed, our result shows that this part fits in polynomial space for bounded arities. It is worth noting that even techniques like magic sets (cf. [21]) cannot reduce the grounding size for this kind of programs. Note that the predicate arities in Example 7

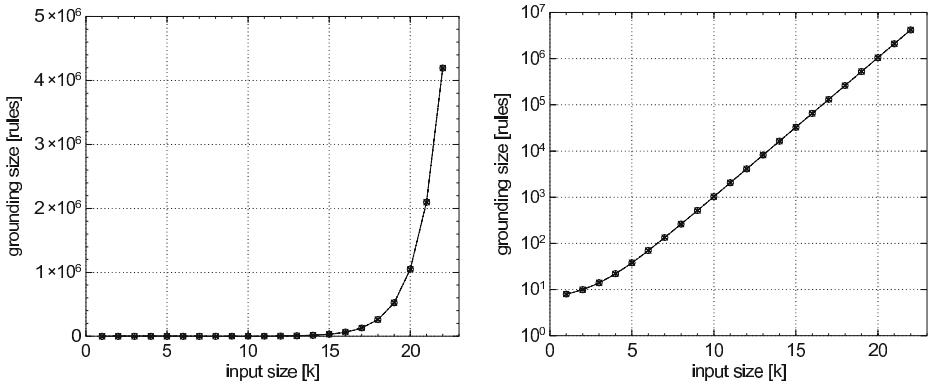

Fig. 5 Ground rules produced by DLV , GrinGo, and Lparse for k between 1 and 22. Left in linear scale, right in logarithmic scale

are not only bounded; these programs are actually monadic. For monadic programs, the complexity of the reasoning tasks is in fact the same as for propositional programs (cf. Section 6.3).

As a final note, Example 7 evidences that there exist programs for which GrinGo produces an exponential-sized ground program while DLV does not, thanks to an optimization. Programs of this kind are, however, never in the language fragment accepted by Lparse. Moreover, all programs accepted by Lparse, and for which Lparse produces an exponential grounding, will also cause DLV and Gringo to generate exponential-sized ground programs.

6.2 Avoiding exponential space

In this section, we provide indications on how solvers can avoid exponential space for reasoning tasks on programs with bounded predicate arities. The first observation is that simply grounding the input will in general not achieve this task, as evidenced in Section 6.1.

One possibility is therefore using top–down techniques for query answering, which intuitively ground "on demand" and work on a proof tree. Our results show that the size of these proof trees should stay polynomial. However, there is not much work on top–down methods in ASP, and actually none of the competitive systems currently uses them. We refer to Section 6.3 for a summary of top–down methods in ASP.

In this section, we take a slightly different path, and indicate how one can use the fact that the size of the proof trees are polynomially bounded in order to write a sort of meta-interpretation program that reduces the input nondeterministically to a polynomial substrate and uses the approach of [14] to define the answer sets on top of this substrate. The resulting program should produce a polynomially sized grounding using most grounders, and in this way allows for re-using existing ASP solvers. Therefore, this technique can be seen as a polynomial reduction from reasoning over programs with bounded arities to propositional programs. In fact, our complexity results imply that such a reduction is feasible for sublanguages up to disjunctive HCF programs, but is probably infeasible for positive disjunctive non-HCF programs and richer languages.

For simplicity, we consider first the case of the sublanguage allowing definite Horn programs. As a further simplification, let us assume that the program contains only one predicate (with fixed arity). It is clear that any program can be transformed into this format (without changing the intended meaning of the program) by creating a new predicate symbol, which incorporates the previous predicate symbols as an argument, and by filling unused arguments with a new constant.

Example 8 Recall the program of Example 1 for $k = 5$, with a graph as in Fig. 1 where $n = 2$.

$$p_k(X_1, X_5) \text{ :- } e(X_1, X_2), e(X_2, X_3), e(X_3, X_4), e(X_4, X_5).$$
$$\mathit{reachable}(X, Y) \text{ :- } p_k(X, Y).$$
$$\mathit{reachable}(X, Y) \text{ :- } \mathit{reachable}(X, Z), p_k(Z, Y).$$
$$e(v_1, x_1). \quad e(v_1, y_1). \quad e(x_1, v_2). \quad e(y_1, v_2).$$
$$e(v_2, x_2). \quad e(v_2, y_2). \quad e(x_2, v_3). \quad e(y_2, v_3).$$

This program can be converted into a program with only one predicate symbol with fixed arity. Note that in this case all predicates of the original program have the same arity, so no "arity-filling" has to be done.

$$p(p_k, X_1, Y_k) \text{ :- } p(e, X_1, X_2), p(e, X_2, X_3), p(e, X_3, X_4), p(e, X_4, X_5).$$
$$p(reachable, X, Y) \text{ :- } p(p_k, X, Y).$$
$$p(reachable, X, Y) \text{ :- } p(reachable, X, Z), p(p_k, Z, Y).$$
$$p(e, v_1, x_1). \quad p(e, v_1, y_1). \quad p(e, x_1, v_2). \quad p(e, y_1, v_2).$$
$$p(e, v_2, x_2). \quad p(e, v_2, y_2). \quad p(e, x_2, v_3). \quad p(e, y_2, v_3).$$

In the program of Example 7, there is one predicate r with arity 0, while all other predicates have arity 1, so in this case a new constant (c_{new}) is needed. The converted program is

$$p(e, 0).\ p(e, 1).$$
$$p(p, X) \text{ :- } p(e, X), \text{not } p(q, X).$$
$$p(q, X) \text{ :- } p(e, X), \text{not } p(p, X).$$
$$p(r, c_{\text{new}}) \text{ :- } p(p, X_1), \ldots, p(p, X_k).$$

The meta-interpretation program is structured into Program Table, Instance Selection, and Meta-Interpreter. All three parts are modules in the sense of [12].

Program table The Program Table is simply a representation of rules as facts. For example, the rule

$$p(a, X) \text{ :- } p(X, Y), p(Y, Z).$$

will be represented as

$$tabH(r, a, \text{`}X\text{`}). \quad tabB(r, \text{`}X\text{`}, \text{`}Y\text{`}). \quad tabB(r, \text{`}Y\text{`}, \text{`}Z\text{`}).$$

where r is a new constant for identifying the rule, and 'X', 'Y', 'Z' are new constants identifying the variables.

Example 9 The program of Example 8 will be represented as

$$tabH(r1, p_k, varx1, varx5). \quad tabB(r1, e, varx1, varx2). \quad tabB(r1, e, varx2, varx3).$$
$$tabB(r1, e, varx3, varx4). \quad tabB(r1, e, varx4, varx5).$$
$$tabH(r2, reachable, varx, vary). \quad tabB(r2, p_k, varx, vary).$$
$$tabH(r3, reachable, varx, vary). \quad tabB(r3, reachable, varx, varz).$$
$$tabB(r3, p_k, varz, vary).$$
$$tabH(r4, e, v_1, x_1). \quad tabH(r5, e, v_1, y_1). \quad tabH(r6, e, x_1, v_2). \quad tabH(r7, e, y_1, v_2).$$
$$tabH(r8, e, v_2, x_2). \quad tabH(r9, e, v_2, y_2). \quad tabH(r10, e, x_2, v_3). \quad tabH(r11, e, y_2, v_3).$$

where $varx, vary, varz, varx1, varx2, varx3, varx4, varx5$ are new constants representing variables, and $r1, \ldots, r11$ are new constants representing rules.

Instance selection The Instance Selection now encodes the fact that it suffices to choose a polynomial amount of ground rules for deriving the truth of an atom. We

know that it is sufficient to use n^k rule instances (n being the number of constants, bounded by the program size, and k being the bound on predicate arities). We therefore create n^k new constants which act as labels. For simplicity, we assume that we have predicates *label* defining the n^k labels for each rule, *rule* defining the rule identifiers, *rulevar* defining the variable identifiers occurring in each rule, and *const* for defining the universe of the original program.

Example 10 For the program of Example 8, the following facts would be added.

$$
\begin{aligned}
&rule(r1).\quad rule(r2).\quad rule(r3).\quad rule(r4).\\
&rule(r5).\quad rule(r6).\quad rule(r7).\quad rule(r8).\\
&rule(r9).\quad rule(r10).\quad rule(r11).\\
&rulevar(r1, varx1).\quad rulevar(r1, varx2).\quad rulevar(r1, varx3).\\
&rulevar(r1, varx4).\quad rulevar(r1, varx5).\\
&rulevar(r2, varx).\quad rulevar(r2, vary).\\
&rulevar(r3, varx).\quad rulevar(r3, vary).\quad rulevar(r3, varz).\\
&const(v_1).\quad const(v_2).\quad const(v_3).\\
&const(x_1).\quad const(x_2).\quad const(y_1).\quad const(y_2).\\
&const(p_k).\quad const(e).\quad const(reachable).\\
&label(l1).\ \ldots\ label(l1000).
\end{aligned}
$$

Note that there are 10 constants in the program and the arity of the unique predicate is 3, so 10^3 labels are needed. In this small example, this seems quite excessive, but for growing input the behavior is definitely better than the native grounding. In this family of examples, break-even is reached at about a value of 20 for n (or 41 for k).

The Instance Selection program itself then looks as follows:

$$
\begin{aligned}
&sel(L, R) \vee nsel(L, R) \text{ :- } label(L), rule(R).\\
&val(L, R, V, C) \text{ :- } sel(L, R), rulevar(R, V), const(C), \text{not } nval(L, R, V, C).\\
&nval(L, R, V, C) \vee nval(L, R, V, C') \text{ :- } sel(L, R), rulevar(R, V),\\
&\qquad\qquad const(C), const(C'), C <> C'.
\end{aligned}
$$

The basic idea is to select a polynomially bounded number of rule instances by means of the first rule, and nondeterministically determine the variable valuations of the selected rules by means of the second and third rule. In this way, all relevant rule instantiations will be enumerated by the answer sets of this program. This will take exponential time, but polynomial space, which contrasts the exponential space bound of simply grounding the original program.

Example 11 Putting together the programs of Examples 9 and 10 and grounding them will yield more than one million ground rules, but the important property is that this number is in the order of n^3 rather than 2^n for growing program size. There are some obvious and simple optimizations, which can drastically reduce the grounding size, such as not including facts in the instance selection process, or not including constants representing predicate names (such as e, p_k, and *reachable* in our example) as possible variable valuations. Just these two simple improvements would already reduce the grounding size by about one half in our example.

Meta-Interpreter Finally, the Meta-Interpreter module can be taken over basically as is from [14]. In order to provide the programs obtained from the Instance Selector module directly as an input to the meta-interpreter (which works on propositional programs) it would be convenient to write for each rule something like

$$\begin{aligned} head(N, f(V_1, \ldots, V_k)) \text{ :- } & tabH(r, \text{`}X_1\text{`}, \ldots, \text{`}X_k\text{`}), sel(N, r), \\ & val(N, r, \text{`}X_1\text{`}, V1), \ldots, val(N, r, \text{`}X_k\text{`}, V_k). \\ pbl(N, f(V_1, \ldots, V_k)) \text{ :- } & tabB(r, \text{`}Y_1\text{`}, \ldots, \text{`}Y_k\text{`}), sel(N, r), \\ & val(N, r, \text{`}Y_1\text{`}, V_1), \ldots, val(N, r, \text{`}Y_k\text{`}, V_k). \end{aligned}$$

where $p(X_1, \ldots, X_k)$ was the head and $p(Y_1, \ldots, Y_k)$ was a body atom of the rule r in the original program, writing constant arguments directly where they occurred. However, doing this involves either the availability of function symbols or value invention (in that case, $f(V_1, \ldots, V_k)$ would be a new constant), which are not available in all ASP systems.[4] Alternatively, one can slightly alter the meta-interpreter program in order to work on a vector of k constants, which in our setting identify a propositional atom, as there is only one predicate symbol.

Example 12 Continuing from Example 11, we would add the following interface rules between the instance selector and the meta-interpreter if function symbols were available, f being a fresh function symbol.

$head(N, f(p_k, V_2, V_3))$:- $sel(N, r1), val(N, r1, varx1, V_2), val(N, r1, varx5, V_3)$.
$pbl(N, f(e, V_2, V_3))$:- $sel(N, r1), val(N, r1, varx1, V_2), val(N, r1, varx2, V_3)$.
$pbl(N, f(e, V_2, V_3))$:- $sel(N, r1), val(N, r1, varx2, V_2), val(N, r1, varx3, V_3)$.
$pbl(N, f(e, V_2, V_3))$:- $sel(N, r1), val(N, r1, varx3, V_2), val(N, r1, varx4, V_3)$.
$pbl(N, f(e, V_2, V_3))$:- $sel(N, r1), val(N, r1, varx4, V_2), val(N, r1, varx5, V_3)$.
$head(N, f(reachable, V_2, V_3))$:- $sel(N, r2), val(N, r2, varx, V_2), val(N, r2, vary, V_3)$.
$pbl(N, f(p_k, V_2, V_3))$:- $sel(N, r2), val(N, r2, varx, V_2), val(N, r2, vary, V_3)$.
$head(N, f(reachable, V_2, V_3))$:- $sel(N, r3), val(N, r3, varx, V_2), val(N, r3, vary, V_3)$.
$pbl(N, f(reachable, V_2, V_3))$:- $sel(N, r3), val(N, r3, varx, V_2), val(N, r3, varz, V_3)$.
$pbl(N, f(p_k, V_2, V_3))$:- $sel(N, r3), val(N, r3, varz, V_2), val(N, r3, vary, V_3)$.
$head(N, f(e, v_1, x_1))$:- $sel(N, r4)$.
$head(N, f(e, v_1, y_1))$:- $sel(N, r5)$.
$head(N, f(e, x_1, v_2))$:- $sel(N, r6)$.
$head(N, f(e, y_1, v_2))$:- $sel(N, r7)$.
$head(N, f(e, v_2, x_2))$:- $sel(N, r8)$.
$head(N, f(e, v_2, y_2))$:- $sel(N, r9)$.
$head(N, f(e, x_2, v_3))$:- $sel(N, r10)$.
$head(N, f(e, y_2, v_3))$:- $sel(N, r11)$.

Looking at the last rules it is evident that instance selection for facts is superfluous, as already argued in Example 11.

[4] Note however, (the relevant part of) $f(V_1, \ldots, V_k)$ can suitably be stored in a (polynomial size) table.

Together with the meta-interpreter program described in [14] and the Program Table and Instance Selector module, each answer set of the obtained program will include the representation of an answer set of the original program. Importantly, the grounding of this program stays polynomial also for larger programs of this type.

If function symbols are not available, one can just omit f, making *head* and *pbl* predicates of arity four instead of two, and modifying the meta-interpreter accordingly.

Note that Program Table, Instance Selection, and Meta-Interpreter form modules as defined in [12], and therefore the answer sets of the combined program can be obtained by combining the answer sets of the modules in an incremental way, as described in [12]. Furthermore, looking at the example, we can see that the program can be grounded using polynomial space in the size of the original program (since arities and the number of variables in rules are bounded). Using the resulting meta-interpretation program, one can transform brave (respectively cautious) reasoning over the original program to brave (respectively cautious) reasoning over the meta-interpretation program, using the predicate *in_AS* of the meta-interpreter, which holds for atoms in an answer set.

It should be noted that there is no correspondence between the answer sets of the original program $\mathcal{P}$ and answer sets of the meta-interpretation program using this simple approach. In order to establish such a correspondence, a check is needed whether $\mathcal{P}$ is closed under the set of facts described by the meta-interpreter in the predicate *in_AS*. Checking this property is co-NP-complete, and can be formulated in ASP by a small program using similar techniques as above. That program, receiving *in_AS* as input, has an answer set iff $\mathcal{P}$ is not closed under the described set of facts. The number of variables in each rule of this program is furthermore bounded by a constant (i.e., it corresponds to a propositional program of polynomial size). It guesses a rule instance (i.e., a rule name and values for its variables) and checks whether the body is not unsatisfied (thus satisfied), and whether the head is not satisfied. The checking program can then be integrated into the meta-interpretation program by exploiting techniques described in [11], or evaluated separately.

Constraints in a Horn program incur a similar need for a co-NP check under brave reasoning, as well as strong negation (checking that no constraint is violated respectively that no inconsistent pair of literals can be derived). An analogous situation arises when negation in rule bodies is allowed, for all of Answer Set Existence, Brave Reasoning, and Cautious Reasoning. Here, each answer set of $\mathcal{P}$ is again given by the answer set of a polynomial portion of the grounding of $\mathcal{P}$, which can be guessed and evaluated using the meta-interpretation program, and checked for closedness in a similar way.

6.3 Further discussion

As shown in previous sections, for many cases the complexity of the reasoning tasks for bounded predicate arities is one level higher up in the polynomial hierarchy than in the propositional case, which is intuitively due to the intractability of checking whether the body of a non-ground rule can fire in a given interpretation. Under suitable restrictions, the latter problem is tractable, and the complexity is not higher than in the propositional case; e.g., if the associated hypergraph of the rule is acyclic,

whose nodes are the variables in the rule body, and where each atom A in the body provides a hyperedge containing all variables in A; tractability then follows from well-known results in databases [1, 30]. Trivially, this is the case for monadic programs, i.e., if all predicates are unary.

We have also outlined, that in some cases the reasoning tasks may be polynomially transformed to propositional ASP, avoiding the grounding bottleneck. Top–down algorithms appear to be good candidates for methods which work in polynomial space, but so far there is little work on this topic. In [6] a resolution method for cautious reasoning with DL[not] programs has been presented. A top–down method for the answer set semantics (DL[not]) is mentioned as further work in [63], building on the method for the well-founded semantics developed in that paper. Several approaches to top–down derivation for DL[$\vee$] programs have been proposed, see e.g. [40, 69] and references therein. Recently, a method for top–down cautious query answering for DL[not$_s$, $\vee$] programs has been described [33]. Unfortunately, it is not clear whether the space complexities of these approaches stay within polynomial space if the predicate arities are bounded.

Another approach to overcome exponential space requirements could be to perform a focused grounding using the query, in principle "emulating" a top–down derivation. In [21, 32] generalizations of the magic sets technique to DL[$\vee$], in a more restricted form to DL[not], and implicitly by means of combination also to DL[not, $\vee$] have been described. However, in general these techniques will not be able to avoid exponential space consumption in all cases. In particular, the main feature of magic sets is that it allows for exploiting information from partially bound queries. As a side effect, this method can avoid parts of the program which are not relevant to the query, and it can restrict the relations of the remaining programs by exploiting the occurrence of constants in queries and rule bodies. However, this is somewhat orthogonal to the features that bounded arities allow to exploit. In particular, for programs and queries that do not contain constants, magic sets can do little apart from avoiding the grounding of program parts which are completely unrelated to the query. In particular, the program in Example 7 together with the query r cannot profit from magic sets, as all predicates are relevant to r and no partial bindings can be exploited.

In the analysis, we assumed (as implicit in [15]) that in presence of weak constraints the weight of a given answer set of a program is computable in polynomial time; this holds, e.g., if the number of ground instances of weak constraints is bounded by a polynomial in the number of constants. In the general case, computing the weight is #P-complete,[5] and remains #P-hard even under bounded predicate arities.[6] Thus, answer set checking, becomes hard for the class PP (since the decision problem can be solved resorting to relative preference rather than solving the function problem of computing the actual weight). According to current

[5]Membership in #P is under the proviso that for binary number representation, the highest level index is polynomial in the problem representation size. Recall that #P is the class of function problems "compute $f(x)$," where f is the number of accepting paths of an NP machine. The class PP, instead, contains all decision problems solvable by such a machine where at least 1/2 of the computation paths accept iff the answer is "yes".

[6]One can easily express counting the number of satisfying assignments for a monotone 2CNF $\phi = \bigwedge_{i=1}^{m}(x_{i_1} \vee x_{i_2})$ as computing the weight of the single answer set of the program consisting of the facts $c(1, 0)$, $c(0, 1)$, $c(1, 1)$ and the single weak constraint $:\sim c(X_{1_1}, X_{1_2}), \ldots, c(X_{m_1}, X_{m_2})$. [1 : 1].

beliefs in complexity theory PP is incomparable to the polynomial hierarchy. As a consequence, the reasoning tasks are no longer in the polynomial hierarchy, but still feasible in polynomial space (a more detailed account is that they are feasible with a PP oracle suitable replacing an NP oracle; a precise characterization remains for future work).

Finally, we mention that compared to strong equivalence, the problem of *uniform*[7] equivalence [9, 59] is in general undecidable for non-ground programs with negation-as-failure, see [16]. As already observed in [18], this undecidability result prevails under bounded predicate arities since the closely related problem of datalog equivalence in databases also remains undecidable in this setting [64].

7 Related work and conclusions

In this paper, we provided complexity results for answer set programming with non-ground programs where the predicate arities are small (more precisely, bounded by a constant), under various syntactic restrictions. We have considered the three major reasoning tasks (answer set existence, brave reasoning, and cautious reasoning) as well as deciding strong equivalence. Under bounded predicate arities, all these tasks are decidable in polynomial space, and thus have far lower complexity than in the unrestricted case where the problems considered range from EXP to $\text{EXP}^{\Sigma_2^P}$.

There has been previous work on the computational complexity of queries on relational databases where the *number of variables* in the query language is bounded by a constant [67]. This setting is orthogonal to ours, since bounded predicate arity still allows for arbitrarily many variables in each rule of a program, and conversely a bounded number of variables does not restrict the arity of predicates up front, since any variable may occur in the same atom multiple times. Our results on bounded arities therefore complement this previous work, and they consider more classes of programs.

Strong equivalence recently has become an important decision problem in ASP due to its connection with program replacement (cf., e.g., [17] and references there) and corresponding aspects of modularization and optimization. The complexity of strong equivalence testing has been studied for the most significant program classes in the propositional case [18, 19, 38] and the general non-ground case [16, 18, 38]. Checking strong equivalence under bounded predicate arities has first and very briefly been discussed in [18], where a detailed study has been left for further work.

The problem of strong equivalence together with preference criteria between answer sets has been analyzed in context with preference semantics which rely on orderings between rules [20] or where the preference is defined via a designated connective ("ordered disjunction") [22]. A slightly different line of research includes characterizations for further extensions of logic programs like weight-constraints or monotone constraints, see e.g., [39, 66]. To the best of our knowledge, however, strong equivalence in combination with weak constraints has not been considered so far.

[7]The problem of uniform equivalence between programs $\mathcal{P}_1, \mathcal{P}_2$ is to decide whether, for each set F of (non-disjunctive) facts, $\mathcal{AS}(\mathcal{P}_1 \cup F) = \mathcal{AS}(\mathcal{P}_2 \cup F)$ holds.

Our results in Tables 4, 5 and 6 in fact show that in the problem setting considered here, the reasoning tasks range from NP to Δ_4^P, and are thus even within the polynomial hierarchy. The classical approach of ASP, which computes in a first step the grounding of the program, and then works on a propositional program—which is employed by virtually all current competitive ASP systems—cannot guarantee polynomial space bounds, since even under bounded predicate arities and optimizations, the ground program may have exponential size in the worst case. Thus, the classic ASP approach suffers from a grounding bottleneck in this case. As we have outlined, in some cases the reasoning tasks may be polynomially transformed to propositional ASP, avoiding the grounding bottleneck. Since bounding arities is a natural restriction, this result is of high practical relevance.

There are several directions for future work. One direction is to extend and complete the results in this paper, to obtain a picture of the complexity for other reasoning tasks (such as answer set checking), for other notions of equivalence, and for further classes of programs. Furthermore, it would be interesting to analyze the expressiveness of ASP under the restrictions considered here, as a database query language (cf. [12, 60]) and a multi-valued function specification, respectively, cf. [41]. Finally, similar results may be obtained for other major non-monotonic formalisms, such as default logic, autoepistemic logic, or circumscription, since they are closely related to ASP.

Acknowledgements This work was partially supported by the Austrian Science Fund (FWF) under projects Z29-N04, P15068-INF, and P18019-N04, by M.I.U.R. under projects "Potenziamento e Applicazioni della Programmazione Logica Disgiuntiva" and "Sistemi basati sulla logica per la rappresentazione di conoscenza: estensioni e tecniche di ottimizzazione," as well as by the European Commission under projects IST-2002-33570 INFOMIX, IST-2001-32429 ICONS, IST-2001-37004 WASP, and the IST-2001-33123 CologNeT Network of Excellence. We thank the anonymous reviewers for useful comments.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 89–148. Morgan Kaufmann Publishers, Inc. (1988)
3. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2002)
4. Ben-Eliyahu, R., Dechter, R.: Propositional semantics for disjunctive logic programs. Ann. Math. Artif. Intell. **12**, 53–87 (1994)
5. Ben-Eliyahu-Zohary, R., Palopoli, L.: Reasoning with minimal models: efficient algorithms and applications. Artif. Intell. **96**, 421–449 (1997)
6. Bonatti, P.A.: Resolution for skeptical stable model semantics. J. Autom. Reason. **27**(4), 391–421 (2001)
7. Buccafurri, F., Leone, N., Rullo, P.: Enhancing disjunctive datalog by constraints. IEEE Trans. Knowl. Data Eng. **12**(5), 845–860 (2000)
8. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. **33**(3), 374–425 (2001)
9. Eiter, T., Fink, M.: Uniform equivalence of logic programs under the stable model semantics. In: Palamidessi, C. (ed.) Proceedings of the 19th International Conference on Logic Programming (ICLP'03). LNCS, vol. 2916, pp. 224–238. Springer (2003)
10. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: propositional case. Ann. Math. Artif. Intell. **15**(3/4), 289–323 (1995)

11. Eiter, T., Polleres, A.: Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. Theory Pract. Log. Program. **6**(1–2), 23–60 (2006)
12. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive datalog. ACM Trans. Database Syst. **22**(3), 364–418 (1997)
13. Eiter, T., Leone, N., Saccà, D.: Expressive power and complexity of partial models for disjunctive deductive databases. Theor. Comp. Sci. **206**(1–2), 181–218 (1998)
14. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: Computing preferred answer sets by meta-interpretation in answer set programming. Theory Pract. Log. Program. **3**, 463–498 (2003)
15. Eiter, T., Faber, W., Fink, M., Pfeifer, G., Woltran, S.: Complexity of answer set checking and bounded predicate arities for non-ground answer set programming. In: Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR'04), pp. 377–387. AAAI Press (2004)
16. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Strong and uniform equivalence in answer-set programming: characterizations and complexity results for the non-ground case. In: Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05), pp. 695–700. AAAI Press (2005)
17. Eiter, T., Fink, M., Tompits, H., Traxler, P., Woltran, S.: Replacements in non-ground answer-set programming. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06), pp. 340–351. AAAI Press (2006)
18. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Complexity results for checking equivalence of stratified logic programs. In: Veloso, M.M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), pp. 330–335. AAAI Press (2007)
19. Eiter, T., Fink, M., Woltran, S.: Semantical characterizations and complexity of equivalences in answer set programming. ACM Trans. Comput. Log. **8**(3) (2007)
20. Faber, W., Konczak, K.: Strong order equivalence. Ann. Math. Artif. Intell. **47**(1–2), 43–78 (2006)
21. Faber, W., Greco, G., Leone, N.: Magic sets and their application to data integration. J. Comput. Syst. Sci. **73**(4), 584–609 (2007). doi:10.1016/j.jcss.2006.10.012
22. Faber, W., Tompits, H., Woltran, S.: Characterizing notions of strong equivalence for logic programs with ordered disjunctions. In: Delgrande, J., Kießling, W. (eds.) Proceedings of the 3rd Multidisciplinary Workshop on Advances in Preference Handling (M-PREF'07) (2007)
23. Fernández, J., Minker, J.: Bottom-up computation of perfect models for disjunctive theories. J. Log. Program. **25**(1), 33–51 (1995)
24. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczynski, M.: The first answer set programming system competition. In: Baral, C., Brewka, G., Schlipf, J. (eds.) Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07). LNCS, vol. 4483, pp. 3–17. Springer (2007)
25. Gebser, M., Schaub, T., Thiele, S.: GrinGo: a new grounder for answer set programming. In: Baral, C., Brewka, G., Schlipf, J. (eds.) Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07). LNCS, vol. 4483, pp. 266–271. Springer (2007)
26. Gelfond, M.: Representing knowledge in A-Prolog. In: Kakas, A., Sadri, F. (eds.) Computational Logic: From Logic Programming into the Future. LNCS/LNAI, vol. 2408, pp. 413–451. Springer (2002)
27. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Logic Programming: Proceedings Fifth Intl Conference and Symposium, pp. 1070–1080. MIT Press, Cambridge, MA (1988)
28. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**, 365–385 (1991)
29. Gottlob, G., Leone, N., Veith, H.: Succinctness as a source of expression complexity. Ann. Pure Appl. Log. **97**(1–3), 231–260 (1999)
30. Gottlob, G., Leone, N., Scarcello, F.: The complexity of acyclic conjunctive queries. J. ACM **48**(3), 431–498 (2001)
31. Greco, S.: Optimization of disjunction queries. In: De Schreye, D. (ed.) Proceedings of the 16th International Conference on Logic Programming (ICLP'99), pp. 441–455. The MIT Press, Las Cruces, NM, USA (1999)
32. Greco, S.: Binding propagation techniques for the optimization of bound disjunctive queries. IEEE Trans. Knowl. Data Eng. **15**(2), 368–385 (2003) (extended Abstract appeared as [31])

33. Johnson, C.A.: Computing only minimal answers in disjunctive deductive databases. Tech. Rep. cs.LO/0305007, arXiv.org (2003)
34. Johnson, D.S.: A catalog of complexity classes. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Chap 2 (1990)
35. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3), 499–562 (2006)
36. Lifschitz, V.: Answer set programming and plan generation. Artif. Intell. **138**, 39–54 (2002)
37. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. **2**(4), 526–541 (2001)
38. Lin, F.: Reducing strong equivalence of logic programs to entailment in classical propositional logic. In: Fensel, D., Giunchiglia, F., McGuiness, D., Williams, M.A. (eds.) Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR'02), pp. 170–176. Morgan Kaufmann (2002)
39. Liu, L., Truszczynski, M.: Properties of programs with monotone and convex constraints. In: Veloso, M., Kambhampati, S. (eds.) Proceedings of the 20th National Conference on Artificial Intelligence (AAAI'05), pp. 701–706. AAAI Press (2005)
40. Lobo, J., Minker, J., Rajasekar, A.: Foundations of Disjunctive Logic Programming. The MIT Press (1992)
41. Marek, V.W., Remmel, J.B.: On the expressibility of stable logic programming. Theory Pract. Log. Program. **3**, 551–567 (2003)
42. Marek, V.W., Truszczyński, M.: Autoepistemic logic. J. Assoc. Comput. Mach. **38**(3), 588–619 (1991)
43. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: Apt, K., Marek, V.W., Truszczyński, M., Warren, D.S. (eds.) The Logic Programming Paradigm—A 25-Year Perspective, pp. 375–398. Springer (1999)
44. Minker, J.: On indefinite data bases and the closed world assumption. In: Loveland, D. (ed.) Proceedings of the 6th Conference on Automated Deduction (CADE '82). LNCS, vol. 138, pp. 292–308. Springer (1982)
45. Minker, J.: Overview of disjunctive logic programming. Ann. Math. Artif. Intell. **12**, 1–24 (1994)
46. Minker, J.: Logic and databases: a 20 year retrospective. In: Proceedings of the International Workshop on Logic in Databases (LID'96). LNCS, vol. 1154, pp. 3–57. Springer (1996)
47. Minker, J., Rajasekar, A.: A fixpoint semantics for disjunctive logic programs. J. Log. Program. **9**(1), 45–74 (1990)
48. Minker, J., Ruiz, C.: Semantics for disjunctive logic programs with explicit and default negation. Fundam. Inform. **20**(1/2/3), 145–192 (1994)
49. Minker, J., Seipel, D.: Disjunctive logic programming: a survey and assessment. In: Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I. LNCS, vol. 2407, pp. 472–511. Springer (2002)
50. Minker, J., Zanon, G.: An extension to linear resolution with selection function. Inf. Process. Lett. **14**(3), 191–194 (1982)
51. Niemelä, I.: Logic programming with stable model semantics as constraint programming paradigm. Ann. Math. Artif. Intell. **25**(3–4), 241–273 (1999)
52. Niemelä, I. (ed.): Language Extensions and Software Engineering for ASP. Tech. Rep. WP3, Working Group on Answer Set Programming (WASP), IST-FET-2001-37004 (2005). Available at http://www.tcs.hut.fi/Research/Logic/wasp/wp3/wasp-wp3-web/
53. Oikarinen, E., Janhunen, T.: Modular equivalence for normal logic programs. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06), pp. 412–416. IOS Press (2006)
54. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
55. Pearce, D., Tompits, H., Woltran, S.: Encodings for equilibrium logic and logic programs with nested expressions. In: Brazdil, P., Jorge, A. (eds.) Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA'01). LNCS, vol. 2258, pp. 306–320. Springer (2001)
56. Provetti, A., Son, T.C. (eds.): Proceedings AAAI 2001 Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning, AAAI Press, Stanford, CA (2001)
57. Przymusinski, T.C.: On the declarative semantics of deductive databases and logic programs. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 193–216. Morgan Kaufmann Publishers, Inc. (1988)
58. Reiter, R.: On closed world data bases. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 55–76. Plenum Press, New York (1978)

59. Sagiv, Y.: Optimising DATALOG programs. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 659–698. Morgan Kaufmann (1988)
60. Schlipf, J.S.: The expressive powers of logic programming semantics. J. Comput. Syst. Sci. **51**(1), 64–86 (1995)
61. Seipel, D., Minker, J., Ruiz, C.: A characterization of the partial stable models for disjunctive databases. In: Proceedings of the International Logic Programming Symposium, pp. 245–259. MIT Press (1997)
62. Seipel, D., Minker, J., Ruiz, C.: Model generation and state generation for disjunctive logic programs. J. Log. Program. **32**(1), 49–69 (1997)
63. Shen, Y.D., Yuan, L.Y., You, J.H.: SLT-resolution for the well-founded semantics. J. Autom. Reas. **28**(1), 53–97 (2002)
64. Shmueli, O.: Decidability and expressiveness aspects of logic queries. In: Proceedings of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'87), pp. 237–249. ACM Press (1987)
65. Syrjänen, T.: Lparse 1.0 User's Manual (2002) http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz
66. Turner, H.: Strong equivalence made easy: nested expressions and weight constraints. Theory Pract. Log. Program. **3**(4–5), 602–622 (2003)
67. Vardi, M.: On the Complexity of bounded-variable queries. In: Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'95), pp. 266–276. ACM Press (1995)
68. Woltran, S. (ed.): Answer set programming: model applications and proofs-of-concept. Tech. Rep. WP5, Working Group on Answer Set Programming (WASP), IST-FET-2001-37004 (2005). Available at http://www.kr.tuwien.ac.at/projects/WASP/report.html
69. Yahya, A.H.: Duality for goal-driven query processing in disjunctive deductive databases. J. Autom. Reas. **28**(1), 1–34 (2002)