Semantical Characterizations and Complexity of Equivalences in Answer Set Programming

THOMAS EITER, MICHAEL FINK, and STEFAN WOLTRAN
Technische Universität Wien

In recent research on nonmonotonic logic programming, repeatedly strong equivalence of logic programs P and Q has been considered, which holds if the programs $P \cup R$ and $Q \cup R$ have the same answer sets for any other program R. This property strengthens the equivalence of P and Q with respect to answer sets (which is the particular case for $R = \emptyset$), and has its applications in program optimization, verification, and modular logic programming. In this article, we consider more liberal notions of strong equivalence, in which the actual form of R may be syntactically restricted. On the one hand, we consider uniform equivalence where R is a set of facts, rather than a set of rules. This notion, which is well-known in the area of deductive databases, is particularly useful for assessing whether programs P and Q are equivalent as components of a logic program which is modularly structured. On the other hand, we consider relativized notions of equivalence where R ranges over rules over a fixed alphabet, and thus generalize our results to relativized notions of strong and uniform equivalence. For all these notions, we consider disjunctive logic programs in the propositional (ground) case as well as some restricted classes, providing semantical characterizations and analyzing the computational complexity. Our results, which naturally extend to answer set semantics for programs with strong negation, complement the results on strong equivalence of logic programs and pave the way for optimizations in answer set solvers as a tool for input-based problem solving.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Logic programming, nonmonotonic reasoning and belief revision*

General Terms: Theory

This article extends previous work that appeared in the *2003 Proceedings of the 19th International Conference on Logic Programming (ICLP)*, Lecture Notes in Computer Science, vol. 2916, Springer, pp. 224–238, and in the 2004 *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA)*, Lecture Notes in Computer Science, vol. 3229, Springer, pp. 161–173.
This work was supported by the Austrian Science Fund (FWF) under Project P18019-N04, and by the European Commission under Projects FET-2001-37004 WASP, IST-2001-33570 COLOGNET, and IST-2001-33570 INFOMIX.
Authors' addresses: T. Eiter, M. Fink, S. Woltran, Technische Universitat Wien, Institute of Information Systems, Knowledge-Based Systems Group, TU Vienna, Favoritenstraße 9-11, A-1040 Vienna, Austria; email: {eiter,michael,stefan}@kr.tuwien.ac.at.

Additional Key Words and Phrases: Answer set semantics, computational complexity, program optimization, stable models, strong equivalence, uniform equivalence

ACM Reference Format:

Eiter, T., Fink, M., and Woltran, S. 2007. Semantical characterizations and complexity of equivalences in answer set programming. ACM Trans. Comput. Logic 8, 3, Article 17 (July 2007), 53 pages. DOI = 10.1145/1243996.1244000 http://doi.acm.org/ 10.1145/1243996.1244000

1. INTRODUCTION

In the last decade, the approach to reduce finding solutions of a problem to finding "models" of a logical theory has gained increasing importance as a declarative problem solving method. The idea is that a problem at-hand is encoded to a logical theory such that the models of this theory correspond to solutions of the problem in a way such that from an arbitrary model of the theory, the corresponding solution can be extracted efficiently. Given that the mappings can be computed in polynomial time, this facilitates polynomial-time problem solving modulo the computation of a model of the constructed logical theory for which an efficient solver may be used. An example of a fruitful application of this approach is Kautz and Selman [1992], which showed that planning problems can be competitively solved by encodings to the classical propositional satisfiability problem (SAT) and running efficient SAT solvers. Encodings of planning problems to nonclassical logics, in particular to nonmonotonic logic programs, were later given in Subrahmanian and Zaniolo [1995], Dimopoulos et al. [1997], Lifschitz [1999], and Eiter et al. [2004]. Because of the features of nonmonotonic negation, such programs allow for a more natural and succinct encoding of planning problems than classical logic, and thus are attractive from a declarative point of view.

Given this potential, encoding problems to nonmonotonic logic programs under the answer set semantics [Gelfond and Lifschitz 1991, 1988], now known as *answer set programming (ASP)* [Provetti and Son 2001], has been considered in recent years for a broad range of other applications, including (amongst others) knowledge-base updates [Zhang and Foo 1998; Inoue and Sakama 1999; Alferes et al. 2000; Eiter et al. 2001], linguistics [Erdem et al. 2003], security requirements engineering [Giorgini et al. 2004], and symbolic model checking [Heljanko and Niemelä 2001]. Many of these applications are realized via dedicated languages (see, e.g., Eiter et al. [2003]) using ASP solvers as backends in which a specified reasoning task is translated into a corresponding logic program. Thus, an ever-growing number of programs is *automatically generated*, leaving the burden of optimizations to the underlying ASP system.

Despite the high sophistication of current ASP solvers like Simons et al. [2002], Leone et al. [2002], Lin and Zhao [2002], and Anger et al. [2001], their current support for optimizing the programs is restricted in the sense that optimizations are mainly geared towards on-the-fly model generation. In an ad hoc manner, program optimization aims at simplifying an input program such that the resulting program has the same answer sets. This is heavily exploited in the systems Smodels [Simons et al. 2002] and DLV [Leone et al. 2002], for instance, when variables are eliminated from programs via grounding.

However, such optimization can only be applied to the entire program. Local simplifications in parts of the program may not be correct at the global level, since by the nonmonotonicity of answer set semantics, adding the same rules to equivalent programs may lead to programs with different models. This particularly hampers offline optimization of programs to which at runtime further rules are added, which is important in different respects. Regarding code reuse, for instance, a program may be used as a "subprogram" or "expanded macro" within the context of another program (e.g., to nondeterministically choose an element from a set), and be thusly utilized in many applications. On the other hand, a problem encoding in ASP usually consists of two parts: a generic problem specification and instance-specific input (e.g., 3-colorability of a graph in general and a particular graph); here, an offline simplification of the generic part is desirable, regardless of the concrete input at runtime.

As widely understood and pointed out by several authors [Lifschitz et al. 2001; Eiter and Fink 2003a; Osorio et al. 2001], this calls for stronger notions of equivalence; compare with, for example, Lifschitz et al. [1999], which already uses a strong notion of equivalence for studying modular translations. As discussed next, there are different ways to access this problem, depending on the actual context of application and optimization. Accordingly, different notions of equivalence may serve as a theoretical basis for optimization procedures. In this article, we present the first systematic and thorough exploration of different notions of equivalence for answer set semantics with respect to semantical characterizations and computational complexity. It provides a theoretical underpinning for advanced methods of program optimization and for enhanced ASP application development, as well as a potential basis for the development of ASP debugging tools. In the following, we recall some notions of equivalence that have been considered for answer set semantics, illustrated with some examples.

Notions of Equivalence. A notion of equivalence which is feasible for the issues discussed previously is *strong equivalence* [Lifschitz et al. 2001; Turner 2001]: Two logic programs P_1 and P_2 are strongly equivalent if by adding any set of rules R to both P_1 and P_2, the resulting programs $P_1 \cup R$ and $P_2 \cup R$ are equivalent under the answer set semantics, that is, have the same answer sets. Thus, if a program P contains a subprogram Q which is strongly equivalent to a program Q', then we may replace Q by Q', in particular, if the resulting program is simpler to evaluate than the original.

Example 1.1. The programs $P_1 = \{a \vee b\}$ and $Q_1 = \{a \vee b;\ a \leftarrow \mathit{not}\ b\}$ are strongly equivalent. Intuitively, the rule $a \leftarrow \mathit{not}\ b$ in Q is redundant, since under answer set semantics a will be derived from the disjunction $a \vee b$ if b is false. On the other hand, the programs $P_2 = \{a \vee b\}$ and $Q_2 = \{a \leftarrow \mathit{not}\ b;\ b \leftarrow \mathit{not}\ a\}$ are not strongly equivalent: $P_2 \cup \{a \leftarrow b;\ b \leftarrow a\}$ has the answer set $\{a, b\}$, which is not an answer set of $Q_2 \cup \{a \leftarrow b;\ b \leftarrow a\}$. Indeed, it is well-known that no disjunction-free program is strongly equivalent to P_2 [Turner 2003].

Note that strong equivalence is, in general, suitable as a theoretical basis for local optimization. However, it is a very restrictive concept. There are two

fundamental options to weaken it and obtain less restrictive notions. On the one hand, one can restrict the syntax of possible program extensions R, or, on the other, one can restrict the set of atoms occurring in R.

The first approach leads us to to the well-known notion of *uniform equivalence* [Sagiv 1988; Maher 1988]. Two logic programs P_1 and P_2 are uniformly equivalent if by adding any set of *facts* F to both P_1 and P_2, the resulting programs $P_1 \cup F$ and $P_2 \cup F$ have the same set of answer sets. That strong equivalence and uniform equivalence are different concepts is illustrated by the following simple example.

Example 1.2. It can be checked that the programs P_2 and Q_2 from Example 1.1, while not strongly equivalent, are uniformly equivalent. We note that by adding the constraint $\leftarrow a, b$ to them, the resulting programs $P_3 = \{a \vee b;\ \leftarrow a, b\}$ and $Q_3 = \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ \leftarrow a, b\}$, which both express exclusive disjunction of a and b, are strongly equivalent (and hence also uniformly equivalent).

This example may suggest that disjunction is an essential feature to make a difference between strong and uniform equivalence. In fact, this is not the case, as shown by the following example.

Example 1.3. Let $P_4 = \{a \leftarrow not\ b;\ a \leftarrow b\}$ and $Q_4 = \{a \leftarrow not\ c;\ a \leftarrow c\}$. Then, it is easily verified that P_4 and Q_4 are uniformly equivalent. However, they are not strongly equivalent: For $P_4 \cup \{b \leftarrow a\}$ and $Q_4 \cup \{b \leftarrow a\}$, we have that $S = \{a, b\}$ is an answer set of $Q_4 \cup \{b \leftarrow a\}$, but not of $P_4 \cup \{b \leftarrow a\}$.

As for program optimization, compared to strong equivalence, uniform equivalence is more sensitive to a modular structure of logic programs, which naturally emerges by splitting them into layered *components* that receive input from lower layers by facts and, in turn, may output facts to a higher layer [Lifschitz and Turner 1994; Eiter et al. 1997]. In particular, it applies to the typical ASP setting outlined earlier, in which a generic problem specification component receives problem-specific input as a set of facts.

However, as mentioned before, a different way to obtain weaker equivalence notions than strong equivalence is to restrict the alphabet of possible program extensions. This is of particular interest whenever one wants to *exclude* dedicated atoms from program extensions. Such atoms may play the role of internal atoms in program components and are considered not to appear anywhere else in the complete program P. This notion of equivalence was originally suggested by Lin [2002], but not further investigated. We will formally define *strong equivalence relative to a given set of atoms* A of two programs P and Q as the test of whether, for all sets of rules S over a given set of atoms A, $P \cup S$ and $Q \cup S$ have the same answer sets.

Finally, we introduce the notion of *uniform equivalence relative to a given set of atoms* A as the property that for two programs P and Q and for all sets $F \subseteq A$ of facts, $P \cup F$ and $Q \cup F$ have the same answer sets. Note that relativized uniform equivalence generalizes the notion of equivalence of DATALOG programs in deductive databases [Shmueli 1993]. There, DATALOG programs are called equivalent if it holds that they compute the same outputs on any

set of external atoms (i.e., atoms that do not occur in any rule head) given as input. The next example illustrates that relativization weakens corresponding notions of equivalence.

Example 1.4. Let $P_5 = \{a \vee b\}$ and $Q_5 = \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ \leftarrow c\}$. The programs P_5 and Q_5 have the same answer set, but are neither uniformly nor strongly equivalent. In particular, it is sufficient to add the fact c. Then, $P_5 \cup \{c\}$ has answer sets $\{a, c\},\ \{b, c\}$, while $Q_5 \cup \{c\}$ has no answer set. However, if we exclude c from the alphabet of possible program extensions, uniform equivalence holds. More specifically, P_5 and Q_5 are uniformly equivalent relative to any set of atoms A such that $c \notin A$. Moreover, P_5 and Q_5 are not strongly equivalent relative to any A which includes both a and b, or c, as before. The reason is that adding $a \leftarrow b$ and $b \leftarrow a$ leads to different answer sets (compare with Example 1.1).

Main Contributions. In this article, we study semantical and complexity properties of the aforementioned notions of equivalence, where we focus on the propositional case (to which first-order logic programs reduce by instantiation). Our main contributions are briefly summarized as follows.

—*We provide characterizations of uniform equivalence of logic programs*. To this aim, we build on the concept of *strong-equivalence models (SE-models)* which have been introduced for characterizing strong equivalence [Turner 2003, 2001] in logic programming terms, resembling an earlier characterization of strong equivalence in terms of equilibrium logic that builds on the intuitionistic logic of here-and-there [Lifschitz et al. 2001]. A strong equivalence model of a program P is a pair (X, Y) of (Herbrand) interpretations such that $X \subseteq Y$, Y is a classical model of P, and X is a model of the Gelfond-Lifschitz reduct P^Y of P with respect to Y [Gelfond and Lifschitz 1991, 1988]. Our characterizations of uniform equivalence will elucidate the differences between strong and uniform equivalence, as illustrated in the previous examples, such that they immediately become apparent.

—*For the finitary case, we provide a mathematical simple and appealing characterization of a logic program with respect to uniform equivalence in terms of its uniform equivalence models (UE-models), which is a special class of SE-models*. Informally, these SE-models (X, Y) of a program P are UE-models such that X either equals Y or is a maximal proper subset of Y. On the other hand, we show that the uniform equivalence of infinite programs cannot be captured by any class of SE-models in general. Furthermore, the notion of logical consequence from UE-models, namely, $P \models_u Q$, turns out to be interesting, since programs P and Q are uniformly equivalent if and only if $P \models_u Q$ and $Q \models_u P$ holds. Therefore, logical consequence (relative to UE-models) can be fruitfully used to determine redundancies under uniform equivalence.

—*By suitably generalizing the characterizations of strong and uniform equivalence, and particularly of SE-models and UE-models, we also provide suitable semantical characterizations for both relativized strong and uniform equivalence.* Our new characterizations thus capture all considered notions of equivalence (including ordinary equivalence) in a uniform way. Moreover, we show

that relativized strong equivalence shares an important property with strong equivalence: Constraining possible program extensions to sets of rules of the form $A \leftarrow B$, where A and B are atoms, does not lead to a different concept (Corollary 4.3). The observation of Pearce and Valverde [2004b] that uniform and strong equivalence are essentially the only concepts of equivalence obtained by varying the *logical* form of program extensions therefore generalizes to relative equivalence.

—*Besides the general case, we consider various major syntactic subclasses of programs, particularly Horn programs, positive programs, disjunction-free programs, and head-cycle-free programs [Ben-Eliyahu and Dechter 1994], and consider how these notions of equivalence relate among each other.* For instance, we establish that for positive programs, all these notions coincide, and therefore only classical models of the programs have to be taken into account for equivalence testing. Interestingly, for head-cycle-free programs, eliminating disjunctions by shifting atoms from rule heads to the respective rule bodies preserves (relativized) uniform equivalence, while affecting (relativized) strong equivalence in general.

—*We thoroughly analyze the computational complexity of deciding (relativized) uniform equivalence and relativized strong equivalence, as well as the complexity of model checking for the corresponding model-theoretic characterizations.* We show that deciding the uniform equivalence of programs P and Q is Π_2^P-complete in the general propositional case, and thus harder than deciding strong equivalence of P and Q, which is coNP-complete [Pearce et al. 2001; Lin 2002; Turner 2003]. Relativized notions of equivalence have the same complexity as uniform equivalence in general (Π_2^P-completeness). These results reflect the intuitive complexity of equivalence checking using the characterizations we provide. Furthermore, we consider the problems for subclasses and establish coNP-completeness results for important fragments, including positive and head-cycle-free programs, thus obtaining a complete picture of the complexity landscape, which is summarized in Table II. Some of the results obtained are surprising; for example, checking relativized uniform equivalence of head-cycle-free programs is *easier* than deciding relativized strong equivalence. For an overview and discussion of the complexity results, we refer to Section 6.

—*Finally, we address extensions of our results with respect to modifications in the language of propositional programs, namely, the addition of strong negation or nested expressions, as well as disallowing constraints.* Moreover, we briefly discuss the general DATALOG case.

Our results extend recent results on strong equivalence of logic programs, and pave the way for optimization of logic programs under answer set semantics by exploiting strong equivalence, uniform equivalence, or relativized notions thereof.

Related Work. While strong equivalence of logic programs under answer set semantics has been considered in a number of papers [Cabalar 2002; de Jongh and Hendriks 2003; Lin 2002; Lifschitz et al. 2001; Osorio et al. 2001; Pearce et al. 2001; Turner 2003, 2001; Pearce 2004; Pearce and Valverde 2004a, investigations on uniform equivalence just started with preliminary parts of this

work [Eiter and Fink 2003a]. Recent papers on program transformations [Eiter et al. 2004a, 2004b] already take both notions into account. In the case of DATALOG, uniform equivalence is a well-known concept, however. Sagiv [1988], who coined the name, has studied the property in the context of definite Horn DATALOG programs where he showed decidability of uniform equivalence testing, which contrasts the undecidability of equivalence testing for DATALOG programs [Shmueli 1993]. Also, Maher [1988] considered uniform equivalence for definite general Horn programs (with function symbols) and reported undecidability. Moreover, both Sagiv [1988] and Maher [1988] showed that uniform equivalence coincides, for the respective programs, with Herbrand logical equivalence. Maher also pointed out that for DATALOG programs, this result has been independently established by Cosmadakis and Kanellakis [1986]. Finally, a general notion of equivalence has also been introduced by Inoue and Sakama [2004]. In their framework, called *update equivalence*, one can exactly specify a set of arbitrary rules which may be added to the programs under consideration and, furthermore, a set of rules which may be deleted. However, for such an explicit enumeration of rules for program extension, or respectively, modification, it seems to be much more complicated to obtain simple semantical characterizations.

The mentioned papers on strong equivalence mostly concern logical characterizations. In particular, the seminal work by Lifschitz et al. [2001] showed that strong equivalence corresponds to equivalence in the nonclassical logic of here-and-there. De Jongh and Hendriks [2003] generalized this result by showing that strong equivalence is characterized by equivalence in all intermediate logics lying between here-and-there (upper bound) and the logic KC of weak excluded middle [Kowalski 1968] (lower bound), which is axiomatized by intuitionistic logic together with the schema $\neg\varphi \vee \neg\neg\varphi$. In addition, Cabalar [2002] presents another multivalued logic known as L_3, which can be employed to decide strong equivalence in the same manner. However, the most popular semantical characterization was introduced by Turner [2003, 2001]. He abstracts from the Kripke semantics as used in the logic of here-and-there, resulting in the aforementioned SE-models. Approaches to implement strong equivalence can be found in Eiter et al. [2004b], Janhunen and Oikarinen [2004], and Pearce et al. [2001]. Complexity characterizations of strong equivalence were given by several authors [Pearce et al. 2001; Lin 2002; Turner 2003]. Our work refines and generalizes this work by considering (relativized) strong equivalence also for syntactic fragments, which previous work did not pay much attention to. In addition, we present a new syntactical criterion to retain strong equivalence when transforming head-cycle-free programs to disjunction-free ones, complementing work on program transformations [Eiter et al. 2004a, 2004b; Osorio et al. 2001; Pearce and Valverde 2004b]. Recent work by Pearce and Valverde [2004b] addresses strong equivalence of programs over disjoint alphabets which are synonymous under structurally defined mappings.

Structure of the Article. The remainder of this article is organized as follows. The next section recalls important concepts and fixes notation. After that, in Section 3, we present our characterizations of uniform equivalence.

We also introduce the notions of UE-model and UE-consequence and relate the latter to other notions of consequence. Then, Section 4 introduces relativized notions of equivalence, and we present our generalized characterizations in model-theoretic terms. Section 5 considers two important classes of programs, in particular, positive and head-cycle-free logic programs, which include Horn and normal logic programs, respectively. Section 6 is devoted to a detailed analysis of complexity issues, while Section 7 considers possible extensions of our results to nested logic programs and answer set semantics for programs with strong negation (also allowing for inconsistent answer sets), as well as to DATALOG programs. Section 8 concludes the article and outlines issues for further research.

2. PRELIMINARIES

We deal with disjunctive logic programs which allow use of the default negation *not* in rules. A rule r is a triple $\langle H(r), B^+(r), B^-(r)\rangle$, where $H(r) = \{A_1, \ldots, A_l\}$, $B^+(r) = \{A_{l+1}, \ldots, A_m\}$, $B^-(r) = \{A_{m+1}, \ldots, A_n\}$, where $0 \leq l \leq m \leq n$ and A_i, $1 \leq i \leq n$ are atoms from a first-order language. Throughout, we use the traditional representation of a rule as an expression of the form

$$A_1 \vee \ldots \vee A_l \leftarrow A_{l+1}, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n.$$

We call $H(r)$ the *head* of r, and $B(r) = \{A_{l+1}, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n\}$ the *body* of r. If $H(r) = \emptyset$, then r is a *constraint*. As usual, r is a *disjunctive fact* if $B(r) = \emptyset$, and a nondisjunctive *fact* if $B(r) = \emptyset$ and $l = 1$, both also represented by $H(r)$ if it is nonempty, and by $\bot$ (falsity) otherwise. A rule r is *normal* (or nondisjunctive) if $l \leq 1$; *definite* if $l = 1$; and *positive* if $n = m$. A rule is *Horn* if it is normal and positive. A definite Horn rule is called *unary* iff its body contains at most one atom.

A *disjunctive logic program* (DLP) P is a (possibly infinite) set of rules. A program P is a *normal logic program* (NLP) (respectively, definite, positive, Horn, or unary) if all rules in P are normal (respectively, definite, positive, Horn, unary). Furthermore, a program P is *head-cycle-free* (*HCF*) [Ben-Eliyahu and Dechter 1994] if each each $r \in P$ is head-cycle-free (in P), that is, if the dependency graph of P (defined as usual) where literals of form *not* A are disregarded, has no directed cycle that contains two atoms belonging to $H(r)$.

In the rest of this article, we focus on propositional programs over a set of atoms $\mathcal{A}$, programs with variables reduce to their ground (propositional) versions, as usual. The set of all atoms occurring in a program P is denoted by $Atm(P)$.

We shall deal with further variations of the syntax, where either *strong* negation is available or constraints are disallowed, in Section 7. There we shall also briefly discuss how to apply our results to programs with nested expressions [Lifschitz et al. 1999] or to nonground programs directly.

We recall the answer set semantics for DLPs [Gelfond and Lifschitz 1991] which generalizes the answer set semantics for NLPs [Gelfond and Lifschitz 1988]. An *interpretation* I, viewed as subset of $\mathcal{A}$, models the head of a rule r, denoted $I \models H(r)$, iff $A \in I$ for some $A \in H(r)$. It models $B(r)$, that is, $I \models B(r)$

iff: (i) each $A \in B^+(r)$ is true in I, that is, $A \in I$, and (ii) each $A \in B^-(r)$ is false in I, that is, $A \notin I$. Furthermore, I models rule r, that is, $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$, and I is a model of a program P, denoted $I \models P$, iff $I \models r$ for all $r \in P$. If $I \models P$ (respectively $I \models r$), I is called a *model* of P (respectively r).

The *Gelfond-Lifschitz reduct* of a program P *relative to* a set of atoms I, denoted P^I, is defined as $P^I = \{H(r) \leftarrow B^+(r) \mid r \in P, I \cap B^-(r) = \emptyset\}$. An interpretation I is an *answer set* (or *stable model* [Przymusinski 1991]) of a program P iff I is a minimal model (under inclusion $\subseteq$) of P^I. By $\mathcal{AS}(P)$ we denote the set of all answer sets of P.

Several notions for equivalence of logic programs have been considered; see Lifschitz et al. [2001], Maher [1988], and Sagiv [1988]. In answer set programming, two DLPs P and Q are regarded as equivalent, denoted $P \equiv Q$, iff $\mathcal{AS}(P) = \mathcal{AS}(Q)$.

The more restrictive form of strong equivalence [Lifschitz et al. 2001] is as follows.

Definition 2.1. Let P and Q be two DLPs. Then, P and Q are *strongly equivalent*, denoted $P \equiv_s Q$, iff for any program R, the programs $P \cup R$ and $Q \cup R$ are equivalent, that is, $P \cup R \equiv Q \cup R$.

One of the main results of Lifschitz et al. [2001] is a semantical characterization of strong equivalence in terms of the nonclassical logic HT. For characterizing strong equivalence in logic programming terms, Turner [2001, 2003] introduced the following notion of SE-models:

Definition 2.2. Let P be a DLP, and let X, Y be sets of atoms such that $X \subseteq Y$. The pair (X, Y) is an *SE-model* of P if $Y \models P$ and $X \models P^Y$. By $SE(P)$ we denote the set of all SE-models of P. For a single rule r, we write $SE(r)$ instead of $SE(\{r\})$.

Strong equivalence can be characterized as follows.

PROPOSITION 2.3 [TURNER 2001, 2003]. *For every two DLPs P and Q, $P \equiv_s Q$ iff $SE(P) = SE(Q)$.*

To check the strong equivalence of two programs P and Q, it is obviously sufficient to consider SE-interpretations (X, Y) over $Atm(P \cup Q)$, that is, with $X \subseteq Y \subseteq Atm(P \cup Q)$. We implicitly make use of this simplification when convenient.

Example 2.4. Reconsider the examples from the introduction. First take programs $P = \{a \vee b\}$ and $Q = \{a \leftarrow not\ b;\ b \leftarrow not\ a\}$. We have[1]

$$SE(P) = \{(a, a);\ (b, b);\ (a, ab);\ (b, ab);\ (ab, ab)\};$$
$$SE(Q) = \{(\emptyset, ab);\ (a, a);\ (b, b);\ (a, ab);\ (b, ab);\ (ab, ab)\}.$$

Thus, $(\emptyset, ab)$ is an SE-model of Q but not of P. This is due to the fact that $P^{\{a,b\}} = \{a \vee b\}$ and $Q^{\{a,b\}}$ is the empty program. The latter is modeled by the empty interpretation, while the former is not. Hence we derive $P \not\equiv_s Q$.

[1]To ease notation, we write abc instead of $\{a, b, c\}$, a instead of $\{a\}$, etc.

Example 2.5. For the second example $P = \{a \leftarrow not\ b;\ a \leftarrow b\}$ and $Q = \{a \leftarrow not\ c;\ a \leftarrow c\}$, we also get $P \not\equiv_s Q$. In this case, we have

$$SE(P) = \{(\emptyset, ab);\ (\emptyset, abc);\ (c, abc)\} \cup S;$$
$$SE(Q) = \{(\emptyset, ac);\ (\emptyset, abc);\ (b, abc)\} \cup S;$$

with $S = \{(X, Y) \mid \{a\} \subseteq X \subseteq Y \subseteq \{a, b, c\}\}$. This shows that $P \not\equiv_s Q$.

Note that from the proofs of the results in Lifschitz et al. [2001] and Turner [2003], it appears that for strong equivalence, only the addition of unary rules is crucial. In other words, constraining the rules in the set R in the definition of strong equivalence to normal rules having at most one positive atom in the body does not lead to a different concept. This is encountered by restriction to facts (i.e., empty rule bodies), however.

Moreover, the answer sets of a program can be characterized via its SE-models as follows.

PROPOSITION 2.6. *For any DLP P, $Y \in \mathcal{AS}(P)$ iff $(Y, Y) \in SE(P)$ and $(X, Y) \in SE(P)$ implies $X = Y$, for any X.*

Finally, we define a consequence relation associated to SE-models.

Definition 2.7. Let P be a DLP and r a rule. Then, r is an *SE-consequence* of P, denoted $P \models_s r$, iff for each $(X, Y) \in SE(P)$, it holds that $(X, Y) \in SE(r)$. Furthermore, we write $P \models_s Q$ iff $P \models_s r$, for every $r \in Q$.

PROPOSITION 2.8. *For any DLP P and Q, $P \equiv_s Q$ iff $P \models_s Q$ and $Q \models_s P$.*

Thus, the notion of SE-consequence captures the strong equivalence of logic programs.

3. UNIFORM EQUIVALENCE

Having presented the preliminary definitions, we now turn to the issue of uniform equivalence of logic programs. We follow the definitions of uniform equivalence in Sagiv [1988] and Maher [1988].

Definition 3.1. Let P and Q be two DLPs. Then, P and Q are *uniformly equivalent*, denoted $P \equiv_u Q$, iff for any set of (nondisjunctive) facts F, the programs $P \cup F$ and $Q \cup F$ are equivalent, that is, $P \cup F \equiv Q \cup F$.

3.1 A Characterization for Uniform Equivalence

We proceed by characterizing the uniform equivalence of logic programs in model-theoretic terms. As restated before, strong equivalence can be captured by the notion of an SE-model (equivalently, HT-model [Lifschitz et al. 2001]) for a logic program. The weaker notion of uniform equivalence can also be characterized in terms of SE-models by imposing further conditions.

We start with a seminal lemma which allows us to derive simple characterizations of uniform equivalence.

LEMMA 3.2. *Two DLPs P and Q are uniformly equivalent, that is, $P \equiv_u Q$, iff for every SE-model (X, Y) such that (X, Y) is an SE-model of exactly one of*

the programs P and Q, it holds that: (i) $Y \models P \cup Q$, *and* (ii) *there exists an SE-model* (X', Y) $X \subset X' \subset Y$ *of the other program.*

PROOF. For the only-if direction, suppose $P \equiv_u Q$. If Y neither models P nor Q, then (X, Y) is not an SE-model of either of the programs P and Q. Without loss of generality, assume that $Y \models P$ and $Y \not\models Q$. Then, since in this case $Y \models P^Y$ and no strict subset of Y models $P \cup Y$, $Y \in \mathcal{AS}(P \cup Y)$, while $Y \notin \mathcal{AS}(Q \cup Y)$. This contradicts our assumption $P \equiv_u Q$. Hence (i) must hold.

To show (ii), assume first that (X, Y) is an SE-model of P but not of Q. In view of (i), it is clear that $X \subset Y$ must hold. Suppose now that for every set X', $X \subset X' \subset Y$, it holds that (X', Y) is not an SE-model of Q. Then, since no subset of X models $Q^Y \cup X$, (Y, Y) is the only SE-model of $Q \cup X$ of form $(\cdot, Y)$. Thus $Y \in \mathcal{AS}(Q \cup X)$ in this case, while $Y \notin \mathcal{AS}(P \cup X)$ ($X \models P^Y$ implies that $X \models (P \cup X)^Y$, so (X, Y) is an SE-model of $P \cup X$). However, this contradicts $P \equiv_u Q$. Thus, it follows that for some X' such that $X \subset X' \subset Y$, (X, Y) is an SE-model of Q. The argument in the case where (X, Y) is an SE-model of Q, but not of P, is analogous. This proves (ii).

For the if direction, assume that (i) and (ii) hold for every SE-model (X, Y) which is an SE-model of exactly one of P and Q. Suppose that there exist sets of atoms F and X such that without loss of generality $X \in \mathcal{AS}(P \cup F) \setminus \mathcal{AS}(Q \cup F)$. Since $X \in \mathcal{AS}(P \cup F)$, we have that $F \subseteq X$ and moreover $X \models P$. Consequently, (X, X) is an SE-model of P. Since $X \notin \mathcal{AS}(Q \cup F)$, either $X \not\models (Q \cup F)^X$ or there exists $Z \subset X$ such that $Z \models (Q \cup F)^X$.

Let us first assume that $X \not\models (Q \cup F)^X$. Then, since $(Q \cup F)^X = Q^X \cup F$ and $F \subseteq X$, it follows that $X \not\models Q^X$. This implies $X \not\models Q$ and hence, (X, X) is not an SE-model of Q. Thus (X, X) is an SE-model of exactly one program P, but (X, X) violates (i), since $X \not\models Q$; this is a contradiction.

It follows that $X \models (Q \cup F)^X$ must hold, and that there must exist $Z \subset X$ such that $Z \models (Q \cup F)^X = Q^X \cup F$. So we can conclude $X \models Q$ and that (Z, X) is an SE-model of Q but not of P. To see the latter, note that $F \subseteq Z$ must hold. So if (Z, X) were an SE-model of P, then it would also be an SE-model of $P \cup F$, contradicting the assumption that $X \in \mathcal{AS}(P \cup F)$. Again we get an SE-model (Z, X) of exactly one of the programs, Q in this case. Hence, according to (ii), there exists an SE-model (X', X) of P, $Z \subset X' \subset X$. However, because of $F \subset Z$, it follows that (X', X) is also an SE-model of $P \cup F$, contradicting our assumption that $X \in \mathcal{AS}(P \cup F)$.

This proves that, given (i) and (ii) for every SE-model (X, Y) such that (X, Y) is an SE-model of exactly one of P and Q, no sets of atoms F and Z exist such that Z is an answer set of exactly one of $P \cup F$ and $Q \cup F$. In other words, $P \equiv_u Q$ holds. □

From Lemma 3.2 we immediately obtain the following characterization of uniform equivalence of logic programs.

THEOREM 3.3. *Two DLPs P and Q are uniformly equivalent, that is, $P \equiv_u Q$, iff for interpretations X, Y*:

(i) (X, X) *is an SE-model of* P *iff it is an SE-model of* Q*; and*
(ii) (X, Y)*, where* $X \subset Y$*, is an SE-model of* P (*respectively* Q) *iff there exists a set* X' *such that* $X \subseteq X' \subset Y$*, and* (X', Y) *is an SE-model of* Q (*respectively* P).

Example 3.4. Reconsider the programs $P = \{a \vee b\}$ and $Q = \{a \leftarrow not\ b;\ b \leftarrow not\ a\}$. By Theorem 3.3, we can easily verify that P and Q are uniformly equivalent: Their SE-models differ only in $(\emptyset, ab)$, which is an SE-model of Q but not of P. Thus, items (i) and (ii) clearly hold for all other SE-models. Moreover, (a, ab) is an SE-model of P, and thus item (ii) also holds for $(\emptyset, ab)$.

Recall that P and Q are strongly equivalent after adding the constraint $\leftarrow a, b$, which enforces exclusive disjunction (see Example 1.2). Uniform equivalence does not require such an addition.

From Theorem 3.3 we can derive the following characterization of uniform equivalence.

THEOREM 3.5. *Two DLPs* P *and* Q*, such that at least one of them is finite, are uniformly equivalent, that is,* $P \equiv_u Q$*, iff the following conditions hold:*

(i) *For every* X*,* (X, X) *is an SE-model of* P *iff it is an SE-model of* Q*; and*
(ii) *for every SE-model* $(X, Y) \in SE(P) \cup SE(Q)$ *such that* $X \subset Y$*, there exists an SE-model* $(X', Y) \in SE(P) \cap SE(Q)$ $(= SE(P \cup Q))$ *such that* $X \subseteq X' \subset Y$.

PROOF. Since (i) holds by virtue of Theorem 3.3, we only need to show (ii). Assume that (X, Y), where $X \subset Y$, is in $SE(P) \cup SE(Q)$.

If $(X, Y) \in SE(P) \cap SE(Q)$, then the statement holds. Otherwise, by virtue of Theorem 3.3 there exists (X_1, Y), $X \subseteq X_1 \subset Y$, such that (X_1, Y) is in $SE(P) \cup SE(Q)$. By repeating this argument, we obtain a chain of SE-models $(X, Y) = (X_0, Y), (X_1, Y), \ldots, (X_i, Y), \ldots$ such that $(X_i, Y) \in SE(P) \cup SE(Q)$ and $X_i \subseteq X_{i+1}$, for all $i \geq 0$. Furthermore, we may choose X_1 such that X_1 coincides with Y on all atoms which do not occur in $P \cup Q$ (and hence all X_i, $i \geq 1$, do so). Since one of P and Q is finite, it follows that $X_i = X_{i+1}$ must hold for some $i \geq 0$ and hence $(X_i, Y) \in SE(P) \cap SE(Q)$ must hold. This proves the result. □

3.2 Introducing UE-Models

In light of this result, we can capture the uniform equivalence of finite programs by the notion of UE-models, defined as follows.

Definition 3.6 [*UE-Model*]. Let P be a DLP. Then $(X, Y) \in SE(P)$ is a *uniform equivalence* (UE) model of P if for every $(X', Y) \in SE(P)$, it holds that $X \subset X'$ implies $X' = Y$. By $UE(P)$ we denote the set of all UE-models of P.

In other words, UE-models comprise all total SE-models (Y, Y) of a DLP plus all its *maximal* nontotal SE-models (X, Y), with $X \subset Y$. Formally,

$$UE(P) = \{(Y, Y) \in SE(P)\} \ \cup \ max_{\geq}\{(X, Y) \in SE(P) \mid X \subset Y\},$$

where $(X', Y') \geq (X, Y)$ iff jointly $Y' = Y$ and $X \subseteq X'$.

By means of UE-models, we then can characterize the uniform equivalence of finite logic programs by the following simple condition.

THEOREM 3.7. *Let P and Q be DLPs. Then*:

(a) $P \equiv_u Q$ *implies* $UE(P) = UE(Q)$*; and*
(b) $UE(P) = UE(Q)$ *implies* $P \equiv_u Q$*, whenever at least one of the programs* P*,* Q *is finite.*

PROOF. For proving (a), let $P \equiv_u Q$. Then, by Theorem 3.3 (i), $UE(P)$ and $UE(Q)$ coincide on models (X, X). Assume without loss of generality that (X, Y), $X \subset Y$, is in $UE(P)$, but not in $UE(Q)$. By Theorem 3.3 (ii), there exists (X', Y), $X \subseteq X' \subset Y$, which is an SE-model of Q, and by a further application, the existence of (X'', Y), $X' \subseteq X'' \subset Y$, which is an SE-model of P, follows. Since $X \subset X''$ contradicts $(X, Y) \in UE(P)$, let $X'' = X' = X$, that is, (X, Y) is an SE-model of Q as well, but is not in $UE(Q)$. Hence there exists $(Z, Y) \in SE(Q)$, $X \subset Z \subset Y$ and, again by Theorem 3.3 (ii), there exists (Z', Y), $Z \subseteq Z' \subset Y$, which is an SE-model of P. This again contradicts $(X, Y) \in UE(P)$. Hence $UE(P) = UE(Q)$ must hold.

For (*b*), assume that $UE(P) = UE(Q)$, and without loss of generality let P be finite. Since $UE(P) = UE(Q)$ implies Theorem 3.3 (i), towards a contradiction, suppose that Theorem 3.3 (ii) is not satisfied, that is, there exists $X \subset Y$, such that either: (1) $(X, Y) \in SE(P)$ and there does not exist $X \subseteq X' \subset Y$, $(X', Y) \in SE(Q)$, or vice versa (2) $(X, Y) \in SE(Q)$ and not exists $X \subseteq X' \subset Y$, $(X', Y) \in SE(P)$.

Case 1. We show the existence of a set Z, $X \subseteq Z \subset Y$, such that $(Z, Y) \in UE(P)$. If $(X, Y) \in UE(P)$, or Y is finite, this is trivial. So let $(X, Y) \notin UE(P)$ and Y be infinite. Then $Y_P = Y \cap Atm(P)$ and $X_P = X \cap Atm(P)$ are finite, $(X_P, Y_P) \in SE(P)$, and $X_P \subset Y_P$ (to see the latter, observe that otherwise, we end up in a contradiction by the fact that then $X_P \models P$, hence $X \models P$, and thus $(X, X) \in UE(P) = UE(Q)$, which implies $(X, Y) \in SE(Q)$, since $(Y, Y) \in UE(Q) = UE(P)$ holds). Since Y_P is finite, there exists a set Z_P, $X_P \subseteq Z_P \subset Y_P$, such that $(Z_P, Y_P) \in UE(P)$. Now, let $Z = Z_P \cup (Y \setminus Y_P)$. Then $X \subseteq Z \subset Y$ holds by construction. Furthermore $(Z, Y) \in UE(P)$, since $Y \setminus Z = Y_P \setminus Z_P$, $P^Y = P^{Y_P}$, and $(Z_P, Y_P) \in UE(P)$. By our assumption $(Z, Y) \in UE(Q)$ follows. Contradiction.

Case 2. We show the existence of a set Z, $X \subseteq Z \subset Y$, such that $(Z, Y) \in UE(Q)$. If $(X, Y) \in UE(Q)$, or Y is finite, this is trivial. So let $(X, Y) \notin UE(Q)$, and Y be infinite. Futhermore, $Y \setminus X \subseteq Atm(P)$ must hold (to see the latter, observe that otherwise we end up in a contradiction by taking any atom $a \in Y \setminus X$, such that $a \notin Atm(P)$, and considering $Z = Y \setminus \{a\}$. Then $X \subseteq Z \subset Y$ holds by construction and since $(Y, Y) \in UE(P) = UE(Q)$, $Y \models P$ and so does Z, that is, $(Z, Y) \in SE(P)$, a contradiction). However, since $Atm(P)$ is finite, this means that $Y \setminus X$ is finite, that is, there cannot exist an infinite chain of SE-models $(X, Y) = (X_0, Y), (X_1, Y), \ldots, (X_i, Y), \ldots$, such that $X_i \subset X_j \subset Y$, for $i < j$, and $(X_i, Y) \in SE(Q)$. Thus, there exists a maximal model $(Z, Y) \in UE(Q)$. By our assumption, $(Z, Y) \in UE(P)$ follows. Contradiction. Thus Theorem 3.3 (ii) holds as well, proving that $P \equiv_u Q$ in Case (b). □

This result shows that UE-models capture the notion of uniform equivalence for finite logic program in the same manner as SE-models capture strong equivalence. Specifically, the essence of a program P with respect to uniform equivalence is expressed by a semantic condition on P alone.

COROLLARY 3.8. *Two finite DLPs P and Q are uniformly equivalent, that is, $P \equiv_u Q$, if and only if $UE(P) = UE(Q)$.*

Example 3.9. Each SE-model of the program $P = \{a \vee b\}$ satisfies the condition of a UE-model, and thus $UE(P) = SE(P)$. The program $Q = \{a \leftarrow not\ b;\ b \leftarrow not\ a\}$ has the additional SE-model $(\emptyset, ab)$, and all of its SE-models except this one are UE-models of Q. Thus,

$$UE(P) = UE(Q) = \{(a,a);\ (b,b);\ (a,ab);\ (b,ab);\ (ab,ab)\}.$$

Note that the strong equivalence of P and Q fails because $(\emptyset, ab)$ is not an SE-model of P. This SE-model is enforced by the intersection property ((X_1, Y) and (X_2, Y) in $SE(P)$ implies $(X_1 \cap X_2, Y) \in SE(P)$). This intersection property is satisfied by the Horn program Q^Y, but violated by the disjunctive program P^Y (=P). The maximality condition of UE-models eliminates this intersection property.

Example 3.10. Reconsider $P = \{a \leftarrow not\ b;\ a \leftarrow b\}$, which has classical models (over $\{a,b,c\}$) of form $\{a\} \subseteq Y \subseteq \{a,b,c\}$. Its UE-models are (X,Y) where $X \in \{Y, Y \setminus \{b\}, Y \setminus \{c\}\}$. Note that the atoms b and c have symmetric roles in $UE(P)$. Consequently, the program obtained by exchanging the roles of b and c, $Q = \{a \leftarrow not\ c;\ a \leftarrow c\}$ has the same UE-models. Hence, P and Q are uniformly equivalent.

The following example shows why the characterization via UE-models fails if both compared programs are infinite. The crucial issue here is the expression of an "infinite chain" resulting in an infinite number of nontotal SE-models. In this case, the concept of maximal nontotal SE-models does not capture the general characterization from Theorem 3.3.

Example 3.11. Consider the programs P and Q over $\mathcal{A} = \{a_i \mid i \geq 1\}$, defined by

$$P = \{a_i \leftarrow \mid i \geq 1\}, \quad \text{and} \quad Q = \{a_i \leftarrow not\ a_i,\ a_i \leftarrow a_{i+1} \mid i \geq 1\}.$$

Both P and Q have the single classical model $\mathcal{A} = \{a_i \mid i \geq 1\}$. Furthermore, P has no "incomplete" SE-model $(X, \mathcal{A})$ such that $X \subset \mathcal{A}$, while Q has the incomplete SE-models $(X_i, \mathcal{A})$, where $X_i = \{a_1, \ldots, a_i\}$ for $i \geq 0$. Both P and Q have the same maximal incomplete SE-models (namely none), and hence they have the same UE-models.

However, $P \not\equiv_u Q$, since, for example, P has an answer set while Q obviously has not. Note that this is caught by our Theorem 3.3, item (ii): For $(X_0, \mathcal{A})$, which is an SE-model of Q but not of P, we cannot find an SE-model $(X, \mathcal{A})$ of P between $(X_0, \mathcal{A})$ and $(\mathcal{A}, \mathcal{A})$.

In fact, the uniform equivalence of infinite programs P and Q cannot be captured by a selection of SE-models.

THEOREM 3.12. *Let P and Q be infinite programs. There is no selection of SE-models $\sigma(SE(\cdot))$ such that $P \equiv_u Q$, if and only if $\sigma(SE(P)) = \sigma(SE(Q))$.*

PROOF. Consider programs over $\mathcal{A} = \{a_i \mid i \geq 1\}$ as follows. The program $P = \{a_i \leftarrow \mid i \geq 1\}$ in Example 3.11, as well as

$$\begin{aligned} Q &= \{a_i \leftarrow not\ a_i,\ a_i \leftarrow a_{i+1},\ a_{2i} \leftarrow a_{2i-1} \mid i \geq 1\}, \\ R &= \{a_i \leftarrow not\ a_i,\ a_i \leftarrow a_{i+1},\ a_{2i+1} \leftarrow a_{2i},\ a_1 \leftarrow \mid i \geq 1\}, \text{ and} \\ S &= \{a_i \leftarrow ,\ \leftarrow a_1 \mid i \geq 1\}. \end{aligned}$$

Considering the corresponding SE-models, it is easily verified that $SE(P) = \{(\mathcal{A}, \mathcal{A})\}$, $SE(S) = \emptyset$, as well as

$$\begin{aligned} SE(Q) &= \{(\emptyset, \mathcal{A}), (a_1a_2, \mathcal{A}), \ldots, (a_1a_2 \cdots a_{2i}, \mathcal{A}), \ldots, (\mathcal{A}, \mathcal{A}) \mid i \geq 0\}, \text{ and} \\ SE(R) &= \{(a_1, \mathcal{A}), (a_1a_2a_3, \mathcal{A}), \ldots, (a_1a_2 \cdots a_{2i+1}, \mathcal{A}), \ldots, (\mathcal{A}, \mathcal{A}) \mid i \geq 0\}. \end{aligned}$$

Hence, we have that $SE(Q) \cap SE(R) = \{(\mathcal{A}, \mathcal{A})\}$. Observe also that $Q \cup X$ and $R \cup X$ do not have an answer set for any proper subset $X \subset \mathcal{A}$, while $\mathcal{A}$ is (the only) answer set for both $Q \cup \mathcal{A}$ and $R \cup \mathcal{A}$. Thus $Q \equiv_u R$. However, $S \cup \mathcal{A}$ does not have an answer set and we get $Q \not\equiv_u S$ and $R \not\equiv_u S$. Since P has the answer set $\mathcal{A}$, we finally conclude that $P \not\equiv_u Q$, $P \not\equiv_u R$, and $P \not\equiv_u S$.

Towards a contradiction, let us assume that there exists a selection function $\sigma(SE(\cdot))$, such that $P_i \equiv_u P_j$ iff $\sigma(SE(P_i)) = \sigma(SE(P_j))$, for $P_i, P_j \in \{P, Q, R, S\}$. Then $\sigma(SE(S)) = \emptyset$ and, since $P \not\equiv_u S$, $\sigma(SE(P)) = \{(\mathcal{A}, \mathcal{A})\}$. Furthermore $Q \equiv_u R$ implies $\sigma(SE(Q)) = \sigma(SE(R))$ and by $SE(Q) \cap SE(R) = \{(\mathcal{A}, \mathcal{A})\}$ we conclude either $\sigma(SE(Q)) = \sigma(SE(R)) = \emptyset$, or $\sigma(SE(Q)) = \sigma(SE(R)) = \{(\mathcal{A}, \mathcal{A})\}$. From $P \not\equiv_u Q$, the former follows, that is, $\sigma(SE(Q)) = \sigma(SE(R)) = \emptyset$. However, then $\sigma(SE(Q)) = \sigma(SE(S))$ while $Q \not\equiv_u S$, which is a contradiction. □

3.3 Consequence under Uniform Equivalence

Based on UE-models, we define an associated notion of consequence under uniform equivalence.

Definition 3.13 [*UE-Consequence*]. A rule r is an *UE-consequence* of a program P, denoted $P \models_u r$, if $(X, Y) \in SE(r)$, for all $(X, Y) \in UE(P)$.

Clearly, $P \models_u r$ for all $r \in P$, and $\emptyset \models_u r$ iff r is a classical tautology. The next result shows that the UE-models of a program remain invariant under addition of UE-consequences.

PROPOSITION 3.14. *For any program P and rule r, if $P \models_u r$, then $UE(P) = UE(P \cup \{r\})$.*

As usual, we write $P \models_u R$ for any set of rules R if $P \models_u r$ for all $r \in R$. As a corollary, taking Theorem 3.7 (b) into account, we get the following.

COROLLARY 3.15. *For any finite program P and set of rules R, if $P \models_u R$ then $P \cup R \equiv_u P$.*

From this proposition, we also obtain an alternative characterization of uniform equivalence in terms of UE-consequence.

THEOREM 3.16. *Let P and Q be DLPs. Then*:

(a) *$P \equiv_u Q$ implies $P \models_u Q$ and $Q \models_u P$; and*
(b) *$P \models_u Q$ and $Q \models_u P$ implies $P \equiv_u Q$, whenever at least one of the programs P, Q is finite.*

PROOF. In Case (a), we have $UE(P) = UE(Q)$ if $P \equiv_u Q$ by Theorem 3.7 (a), and thus P and Q have the same UE-consequences. Since $(X, Y) \models P$ (respectively $(X, Y) \models Q$), for all $(X, Y) \in UE(P)$ (respectively $(X, Y) \in UE(Q)$), it follows that $Q \models_u P$ and $P \models_u Q$. For (b), we apply Proposition 3.14 repeatedly and obtain $UE(P) = UE(P \cup Q) = UE(Q)$. By Theorem 3.7 (b), $P \equiv_u Q$. □

Rewriting this result in terms of SE- and UE-models gives the following characterization (which has also been derived for finite programs in Eiter et al. [2004a, Prop. 5]).

PROPOSITION 3.17. *Let P and Q be DLPs. Then*:

(a) *$P \equiv_u Q$ implies $UE(P) \subseteq SE(Q)$ and $UE(Q) \subseteq SE(P)$; and*
(b) *$UE(P) \subseteq SE(Q)$ and $UE(Q) \subseteq SE(P)$ implies $P \equiv_u Q$, whenever at least one of the programs P, Q is finite.*

We note that with respect to uniform equivalence, every program P has a canonical normal form P^* given by its UE-consequences, that is, $P^* = \{r \mid P \models_u r\}$. Clearly, $P \subseteq P^*$ holds for every program P, and P^* has exponential size. Applying optimization methods built on UE-consequence, P (respectively P^*) may be transformed into smaller uniformally equivalent programs; we leave this for further study.

As for the relationship of UE-consequence to classical consequence and cautious consequence under answer set semantics, we note the following hierarchy. Let $\models_c$ denote consequence from the answer sets, namely, $P \models_c r$ iff $M \models r$ for every $M \in \mathcal{AS}(P)$.

PROPOSITION 3.18. *For any finite program P and rule r:* (i) *$P \models_u r$ implies $P \cup F \models_c r$, for each set of facts F;* (ii) *$P \cup F \models_c r$, for each set of facts F, implies $P \models_c r$; and* (iii) *$P \models_c r$ implies $P \models r$.*

This hierarchy is strict, namely, none of the implications holds in the converse direction. (For (i), note that $\{a \leftarrow not\ a\} \models_c a$ but $\{a \leftarrow not\ a\} \not\models_u a$, since the UE-model $(\emptyset, \{a\})$ violates a.)

We next present a semantic characterization in terms of UE-models, under which UE- and classical consequence, and thus all four notions of consequence, coincide.

LEMMA 3.19. *Let P be a DLP. Suppose that $(X, Y) \in UE(P)$ implies $X \models P$ (that is, X is a model of P). Then, $P \models r$ implies $P \models_u r$, for every rule r.*

THEOREM 3.20. *Let P be any DLP. Then the following conditions are equivalent:*

(i) *$P \models_u r$ iff $P \models r$, for every rule r; and*
(ii) *for every $(X, Y) \in UE(P)$, it holds that $X \models P$.*

PROOF. (ii) $\Rightarrow$ (i). Suppose (ii) holds. The only-if direction in (i) holds immediatly by Lemma 3.19. The if direction in (i) holds in gerenal, since $P \models_u r$ iff $UE(P) \subseteq SE(r)$. The latter clearly implies that each total SE-model of P is a total SE-model of r. Consequently, $P \models r$.

(i) $\Rightarrow$ (ii). Suppose $P \models_u r$ iff $P \models r$ for every rule r, but there exists some UE-model (X, Y) of P such that $X \not\models P$. Hence $X \not\models r$ for some rule $r \in P$. Let r' be the rule which results from r by shifting the negative literals to the head, that is, $H(r') = H(r) \cup B^-(r)$, $B^+(r') = B^+(r)$, and $B^-(r') = \emptyset$. Then $X \not\models r'$. On the other hand, $r \in P$ implies $(X, Y) \models r$. Hence $Y \models r$ and thus $Y \models r'$. Moreover, $B^-(r') = \emptyset$ implies that $r' \in P^Y$, and hence $X \models r'$. This is a contradiction. It follows that $X \models P$ for each UE-model (X, Y) of P. □

An immediate corollary to this result is that for finite *positive* programs, the notion of UE-consequence collapses with classical consequence, and hence the uniform equivalence of finite positive programs amounts to classical equivalence. We shall obtain these results as corollaries of more general results in Section 5.1, though.

4. RELATIVIZED NOTIONS OF STRONG AND UNIFORM EQUIVALENCE

In what follows, we formally introduce the notions of relativized strong equivalence (RSE) and relativized uniform equivalence (RUE).

Definition 4.1. Let P and Q be programs and let A be a set of atoms. Then:

(i) P and Q are *strongly equivalent relative to* A, denoted $P \equiv_s^A Q$, iff $P \cup R \equiv Q \cup R$, for all programs R over A; and

(ii) P and Q are *uniformly equivalent relative to* A, denoted $P \equiv_u^A Q$, iff $P \cup F \equiv Q \cup F$, for all (nondisjunctive) facts $F \subseteq A$.

Observe that the range of applicability of these notions covers ordinary equivalence (by setting $A = \emptyset$) of two programs P, Q, and *general* strong (respectively uniform) equivalence (whenever $Atm(P \cup Q) \subseteq A$). Also the following relation holds: For any set A of atoms, let $A' = A \cap Atm(P \cup Q)$. Then, $P \equiv_e^A Q$ holds, iff $P \equiv_e^{A'} Q$ holds, for $e \in \{s, u\}$.

Our first main result lists some properties for relativized strong equivalence. Among them, we show that RSE shares an important property with general strong equivalence: In particular, from the proofs of the results in Lifschitz et al. [2001] and Turner [2003], it appears that for strong equivalence, only the addition of unary rules is crucial. In other words, constraining the rules in the set R in Definition 4.1 to be unary does not lead to a different concept.

LEMMA 4.2. *For programs P, Q, and a set of atoms A, the following statements are equivalent:*

(1) *There exists a program R over A such that $\mathcal{AS}(P \cup R) \not\subseteq \mathcal{AS}(Q \cup R)$;*

(2) *there exists a unary program U over A such that $\mathcal{AS}(P \cup U) \not\subseteq \mathcal{AS}(Q \cup U)$; and*

(3) *there exists an interpretation Y such that and* (a) $Y \models P$; (b) *for each $Y' \subset Y$ with $(Y' \cap A) = (Y \cap A)$, $Y' \not\models P^Y$ holds; and* (c) $Y \models Q$ *implies existence of*

an $X \subset Y$, such that $X \models Q^Y$ and, for each $X' \subset Y$ with $(X' \cap A) = (X \cap A)$, $X' \not\models P^Y$ holds.

PROOF. (1) $\Rightarrow$ (3): Suppose an interpretation Y and a set R of rules over A such that $Y \in \mathcal{AS}(P \cup R)$ and $Y \notin \mathcal{AS}(Q \cup R)$. From $Y \in \mathcal{AS}(P \cup R)$, we get $Y \models P \cup R$ and, for each $Z \subset Y$, $Z \not\models P^Y \cup R^Y$. Thus (a) holds, and since $Y' \models R^Y$ holds, for each Y' with $(Y' \cap A) = (Y \cap A)$, (b) holds as well. From $Y \notin \mathcal{AS}(Q \cup R)$, we get that either $Y \not\models Q \cup R$ or there exists an interpretation $X \subset Y$, such that $X \models Q^Y \cup R^Y$. Note that $Y \not\models Q \cup R$ implies $Y \not\models Q$, since from the preceding, we have $Y \models R$. Thus, in the case of $Y \not\models Q \cup R$, (c) holds; otherwise we get that $X \models Q^Y$. Now since $X \models R^Y$, we know that, for each $X' \subset Y$ with $(X' \cap A) = (X \cap A)$, $X' \not\models P^Y$ has to hold, otherwise $Y \notin \mathcal{AS}(P \cup R)$. Hence (c) is satisfied.

(3) $\Rightarrow$ (2): Suppose an interpretation Y such that conditions (a–c) hold. We have two cases: First, if $Y \not\models Q$, consider the unary program $U = (Y \cap A)$. By conditions (a) and (b), it is easily seen that $Y \in \mathcal{AS}(P \cup U)$, and from $Y \not\models Q$, it follows that $Y \notin \mathcal{AS}(Q \cup U)$. So suppose $Y \models Q$. By (c), there exists an $X \subset Y$, such that $X \models Q^Y$. Consider the program $U = (X \cap A) \cup \{p \leftarrow q \mid p, q \in (Y \setminus X) \cap A\}$. Again, U is unary over A. Clearly, $Y \models Q \cup U$ and $X \models Q^Y \cup U$. Thus $Y \notin \mathcal{AS}(Q \cup U)$. It remains to show that $Y \in \mathcal{AS}(P \cup U)$. We have $Y \models P \cup U$. Towards a contradiction, suppose a $Z \subset Y$, such that $Z \models P^Y \cup U$. By definition of U, $Z \supseteq (X \cap A)$. If $(Z \cap A) = (X \cap A)$, condition (c) is violated; if $(Z \cap A) = (Y \cap A)$, condition (b) is violated. Thus $(X \cap A) \subset (Z \cap A) \subset (Y \cap A)$. But then $Z \not\models U$, since there exists at least one rule $p \leftarrow q$ in U, such that $q \in Z$ and $p \notin Z$. Contradiction.

(2) $\Rightarrow$ (1) is obvious. □

The next result is an immediate consequence of the fact that Propositions (1) and (2) from the previous result are equivalent.

COROLLARY 4.3. *For programs P, Q, and a set of atoms A, $P \equiv_s^A Q$ holds iff, for each unary program U over A, $P \cup U \equiv Q \cup U$ holds.*

We emphasize that therefore, also for relativized equivalences it holds that, by restricting the syntax of the added rules, RSE and RUE are the only concepts which differ. Note that this generalizes an observation reported in Pearce and Valverde [2004b] (see Theorem 2 and subsequent remarks therein) to relativized notions of equivalence, namely that uniform and strong equivalence are the only forms of equivalence obtained by varying the logical form of expressions, that is, the structure of the rules, in the extension.

4.1 A Characterization for Relativized Strong Equivalence

In this section, we provide a semantical characterization of RSE by generalizing the notion of SE-models. Hence, our aim is to capture the problem $P \equiv_s^A Q$ in model-like terms. We emphasize that the forthcoming results are also applicable to infinite programs. Moreover, having found a suitable notion of *relativized SE-models*, we expect that a corresponding pendant for relativized uniform equivalence can be derived in the same manner as general UE-models are

defined over general SE-models. As in the case of UE-models, we need some restrictions concerning the infinite case, that is, if infinite programs are considered.

We introduce the following notion.

Definition 4.4. Let A be a set of atoms. A pair of interpretations (X, Y) is a *(relativized) A-SE-interpretation* iff either $X = Y$ or $X \subset (Y \cap A)$. The former are called total and the latter nontotal A-SE-interpretations.

Moreover, an A-SE-interpretation (X, Y) is a (relativized) *A-SE-model* of a program P iff:

(i) $Y \models P$;
(ii) for all $Y' \subset Y$ with $(Y' \cap A) = (Y \cap A)$, $Y' \not\models P^Y$; and
(iii) $X \subset Y$ implies existence of a $X' \subseteq Y$ with $(X' \cap A) = X$, such that $X' \models P^Y$ holds.

The set of A-SE-models of P is given by $SE^A(P)$.

Compared to SE-models, this definition is more involved. This is due to the fact that we have to take care of two different effects when relativizing strong equivalence. The first is as follows: Suppose a program P has among its SE-models the pairs (Y, Y) and (Y', Y) with $(Y' \cap A) = (Y \cap A)$ and $Y' \subset Y$. Then, Y never becomes an answer set of a program $P \cup R$, regardless of the rules R over A that we add to P. This is due to the fact that either $Y' \models (P \cup R)^Y$ still holds for some $Y' \subset Y$, or $Y \not\models (P \cup R)^Y$ (the latter is a consequence of finding an R such that $Y' \not\models (P \cup R)^Y$, for $(Y' \cap A) = (Y \cap A)$, $Y' \subset Y$ modeling P). In other words, for the construction of a program R over A such that $\mathcal{AS}(P \cup R) \neq \mathcal{AS}(Q \cup R)$, it is not worthwhile to pay attention to any original SE-model of P of the form $(\cdot, Y)$, whenever there exists a $(Y', Y) \in SE(P)$ with $(Y' \cap A) = (Y \cap A)$. This motivates condition (ii). Condition (iii) deals with a different effect: Suppose P has SE-models (X, Y) and (X', Y), with $(X \cap A) = (X' \cap A) \subset (Y \cap A)$. Here, it is not possible to eliminate just one of these two SE-models by adding rules over A. Such SE-models that do not differ with respect to A are collected into a single A-SE-model $((X \cap A), Y)$.

The different roles of these two independent conditions become even more apparent in the following cases. On the one hand, setting $A = \emptyset$, the A-SE-models of a program P collapse with the answer sets of P. More precisely, all such $\emptyset$-SE-models have to be of the form (Y, Y), and it holds that (Y, Y) is an $\emptyset$-SE-model of a DLP P iff Y is an answer set of P. This is easily seen by the fact that under $A = \emptyset$, conditions (i) and (ii) in Definition 4.4 exactly coincide with the characterization of answer sets, following Proposition 2.6. Therefore, A-SE-model-checking for DLPs is not possible in polynomial time in the general case; otherwise we get that checking whether a DLP has some answer set is NP-complete, which is in contradiction to known results [Eiter and Gottlob 1995], provided that the polynomial hierarchy does not collapse. On the other hand, if each atom from P is contained in A, then the A-SE-models of P coincide with the SE-models (over A) of P. The conditions in Definition 4.4 are hereby instantiated as follows: A pair (X, Y) is an A-SE-interpretation iff

$X \subseteq Y$, and by (i) we get $Y \models P$, (ii) is trivially satisfied, and (iii) states that $X \models P^Y$.

The central result is as follows. In particular, we show that A-SE-models capture the notion of $\equiv_s^A$ in the same manner as SE-models capture $\equiv_s$.

THEOREM 4.5. *For programs P, Q and a set of atoms A, $P \equiv_s^A Q$ holds iff $SE^A(P) = SE^A(Q)$.*

PROOF. First suppose $P \not\equiv_s^A Q$ and without loss of generality consider for some R over A, $\mathcal{AS}(P \cup R) \not\subseteq \mathcal{AS}(Q \cup R)$. By Lemma 4.2, there exists an interpretation Y such that: (a) $Y \models P$; (b) for each $Y' \subset Y$ with $(Y' \cap A) = (Y \cap A)$, $Y' \not\models P^Y$; and (c) $Y \not\models Q$ or there exists an interpretation $X \subset Y$ such that $X \models Q^Y$ and, for each $X' \subset Y$ with $(X' \cap A) = (X \cap A)$, $X' \not\models P^Y$. First suppose $Y \not\models Q$, or $Y \models Q$ and $(X \cap A) = (Y \cap A)$. Then (Y, Y) is an A-SE-model of P but not of Q. Otherwise, $Y \models Q$ and $(X \cap A) \subset (Y \cap A)$, $((X \cap A), Y)$ is an A-SE-model of Q. But, by condition (c), $((X \cap A), Y)$ is not an A-SE-model of P.

For the converse direction of the theorem, suppose a pair (Z, Y) such that without loss of generality (Z, Y) is an A-SE-model of P but not of Q. First let $Z = Y$. We show that $\mathcal{AS}(P \cup R) \not\subseteq \mathcal{AS}(Q \cup R)$ for some program R over A. Since (Y, Y) is an A-SE-model of P, we get from Definition 4.4 that $Y \models P$ and, for each $Y' \subset Y$ with $(Y \cap A) = (Y' \cap A)$, $Y' \not\models P^Y$. Thus, conditions (a) and (b) in part (3) of Lemma 4.2 are satisfied for P by Y. On the other hand, (Y, Y) is not an A-SE-model of Q. By Definition 4.4, either $Y \not\models Q$ or there exists a $Y' \subset Y$, with $(Y' \cap A) = (Y \cap A)$, such that $Y' \models Q^Y$. Therefore, condition (c) from Lemma 4.2 is satisfied by either $Y \not\models Q$ or, if $Y \models Q$, by setting $X = Y'$. We apply Lemma 4.2 and get the desired result. Consequently, $P \not\equiv_s^A Q$. So suppose $Z \neq Y$. We show that then $\mathcal{AS}(Q \cup R) \not\subseteq \mathcal{AS}(P \cup R)$ holds for some program R over A. First, observe that whenever (Z, Y) is an A-SE-model of P, then also (Y, Y) is an A-SE-model of P. Hence, the case where (Y, Y) is not an A-SE-model of Q is already shown. So, suppose (Y, Y) is an A-SE-model of Q. We have $Y \models Q$ and, for each $Y' \subset Y$ with $(Y' \cap A) = (Y \cap A)$, $Y' \not\models Q^Y$. This satisfies conditions (a) and (b) in Lemma 4.2 for Q. However, since (Z, Y) is not an A-SE-model of Q, for each $X' \subset Y$ with $(X' \cap A) = Z$, $X' \not\models Q^Y$ holds. Since (Z, Y) in turn is an A-SE-model of P, there exists an $X \subset Y$ with $(X \cap A) = Z$ such that $X \models P^Y$. These observations imply that (c) holds in Lemma 4.2. We apply the lemma and finally get $P \not\equiv_s^A Q$. □

Although A-SE-models are more involved than SE-models, they share some fundamental properties with general SE-models. However, some properties do not generalize to A-SE-models. We shall discuss these issues in detail in Section 4.3. For the moment, we list some observations concerning the relation between SE-models and A-SE-models, in order to present some examples.

LEMMA 4.6. *Let P be a program and A a set of atoms. We have the following relations between A-SE-models and SE-models*:

(i) *If $(Y, Y) \in SE^A(P)$, then $(Y, Y) \in SE(P)$.*
(ii) *If $(X, Y) \in SE^A(P)$, then $(X', Y) \in SE(P)$, for some $X' \subseteq Y$ with $(X' \cap A) = X$.*

Table I. Comparing the A-SE-Models for Example Programs Q and Q'

A	A-SE-Models of Q	A-SE-Models of Q'
$\{a,b,c\}$	$(abc,abc),(a,abc),(b,abc)$	$(abc,abc),(a,abc),(b,abc),(\emptyset,abc)$
$\{a,b\}$	$(abc,abc),(a,abc),(b,abc)$	$(abc,abc),(a,abc),(b,abc),(\emptyset,abc)$
$\{a,c\}$	$(abc,abc),(a,abc),(\emptyset,abc)$	$(abc,abc),(a,abc),(\emptyset,abc)$
$\{b,c\}$	$(abc,abc),(\emptyset,abc),(b,abc)$	$(abc,abc),(b,abc),(\emptyset,abc)$
$\{a\}$	—	—
$\{b\}$	—	—
$\{c\}$	$(abc,abc),(\emptyset,abc)$	$(abc,abc),(\emptyset,abc)$
$\emptyset$	—	—

Example 4.7. Consider the programs

$$Q = \{a \vee b \leftarrow;\ a \leftarrow c;\ b \leftarrow c;\ \leftarrow not\ c;\ c \leftarrow a,b\};$$
$$Q' = \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ a \leftarrow c;\ b \leftarrow c;\ \leftarrow not\ c;\ c \leftarrow a,b\}.$$

Thus, Q' results from Q by replacing the disjunctive rule $a \vee b \leftarrow$ by the two rules $a \leftarrow not\ b$; $b \leftarrow not\ a$.

Table I lists for each $A \subseteq \{a,b,c\}$ the A-SE-models of Q and Q', respectively. The first row of the table gives the SE-models (over $\{a,b,c\}$) for Q and Q'. From this row, we can, by Definition 4.4 and Lemma 4.6, obtain the other rows quite easily. Observe that we have $Q \not\equiv_s Q'$. The second row shows that for $A = \{a,b\}$, $Q \not\equiv_s^A Q'$, as well. Indeed, adding $R = \{a \leftarrow b;\ b \leftarrow a\}$ yields $\{a,b,c\}$ as the answer set of $Q \cup R$, whereas $Q' \cup R$ has no answer set. For all other $A \subset \{a,b,c\}$, the A-SE-models of Q and Q' coincide. Basically, there are two different reasons. First, for $A = \{a,c\}$, $A = \{b,c\}$, or $A = \{c\}$, condition (iii) from Definition 4.4 comes into play. In these cases, at least one of the SE-interpretations (a,abc) or (b,abc) is "switched" to $(\emptyset,abc)$, and thus the original difference between the SE-models disappears when considering A-SE-models. In the remaining cases, namely, $A \subset \{a,b\}$, condition (ii) prevents any $(\cdot,abc)$ from being an A-SE-model of Q or Q'. Then, neither Q nor Q' possesses any A-SE-model.

4.2 A Characterization for Relativized Uniform Equivalence

In what follows, we consider the problem of checking relativized uniform equivalence. Therefore, we shall make use of the newly introduced A-SE-models in the same manner as Section 3 provided characterizations for uniform equivalence using SE-models.[2]

We start with a generalization of Lemma 3.2. The proof is similar to the proof of Lemma 3.2 and thus relegated to the online Appendix, which can be accessed at the ACM Digital Library.

LEMMA 4.8. *Two DLPs P and Q are uniformly equivalent with respect to to a set of atoms A, that is, $P \equiv_u^A Q$, iff for every A-SE-model (X,Y) such that (X,Y) is an A-SE-model of exactly one of the programs P and Q, it holds that* (i) *$(Y,Y) \in SE^A(P) \cap SE^A(Q)$, and* (ii) *there exists an A-SE-model (X',Y), $X \subset X' \subset Y$, of the other program.*

[2]For a slightly different way to prove the main results on RUE, we refer to Woltran [2004].

From Lemma 4.8 we immediately obtain the following characterization of relativized uniform equivalence.

THEOREM 4.9. *Two programs P and Q are uniformly equivalent with respect to to a set of atoms A, $P \equiv_u^A Q$, iff:*

(i) *For each Y, $(Y,Y) \in SE^A(P)$ iff $(Y,Y) \in SE^A(Q)$, that is, the total A-SE-models of P and Q coincide; and*

(ii) *for each (X,Y), where $X \subset Y$, (X,Y) is an A-SE-model of P (respectively Q) iff there exists a set X' such that $X \subseteq X' \subset Y$, and (X',Y) is an A-SE-model of Q (respectively P).*

In contrast to uniform equivalence, we can obtain further characterizations for $\equiv_u^A$ also for infinite programs, provided that A is finite.

THEOREM 4.10. *Let P and Q be programs and A a set of atoms such that P, Q, or A is finite. Then $P \equiv_u^A Q$, iff the following conditions hold:*

(i) *For each Y, $(Y,Y) \in SE^A(P)$ iff $(Y,Y) \in SE^A(Q)$, that is, the total A-SE-models of P and Q coincide; and*

(ii) *for each $(X,Y) \in SE^A(P) \cup SE^A(Q)$ such that $X \subset Y$, there exists an $(X',Y) \in SE^A(P) \cap SE^A(Q)$ such that $X \subseteq X' \subset Y$.*

The result is proved by the same arguments as used in the proof of Theorem 3.5. The only additional argumentation is needed for the cases where P and Q are both infinite, but A is finite. Recall that in this case there is also only a finite number of nontotal A-SE-interpretations (X,Y) for fixed Y, since $X \subseteq A$ holds by the definition of A-SE-interpretation. Therefore, any chain (as used in the proof of Theorem 3.5) of different A-SE-models (X,Y) with fixed Y is finite.

As mentioned before, we aim at defining relativized A-UE-models over A-SE-models in the same manner as general UE-models are defined over general SE-models, following Definition 3.6.

Definition 4.11. Let A be a set of atoms and P be a program. A pair (X,Y) is a *(relativized) A-UE-model* of P iff it is an A-SE-model of P and, for every A-SE-model (X',Y) of P, $X \subset X'$ implies $X' = Y$. The set of A-UE-models of P is given by $UE^A(P)$.

An alternative characterization of A-UE-models, which will be useful later, is immediately obtained from Definitions 4.4 and 4.11 as follows.

PROPOSITION 4.12. *An A-SE-interpretation (X,Y) is an A-UE-model of a program P iff:*

(i) *$Y \models P$;*

(ii) *for each $X'' \subset Y$ with either $(X \cap A) \subset (X'' \cap A)$ or $(X'' \cap A) = (Y \cap A)$, $X'' \not\models P^Y$; and*

(iii) *if $X \subset Y$, there exists a $X' \subseteq Y$ with $(X' \cap A) = (X \cap A)$ such that $X' \models P^Y$.*

Next, we derive the desired characterization for relativized uniform equivalence, generalizing the results in Theorem 3.7. The proof of the theorem is given in the online Appendix.

THEOREM 4.13. *Let P and Q be DLPs, and A a set of atoms. Then:*

(a) $P \equiv_u^A Q$ *implies* $UE^A(P) = UE^A(Q)$*; and*
(b) $UE^A(P) = UE^A(Q)$ *implies* $P \equiv_u^A Q$ *whenever at least one of* P*,* Q*, or* A *is finite.*

Example 4.14. Recall our example programs Q and Q' from before. Via the first row in the table (that is, for $A = \{a, b, c\}$, yielding the respective SE-models), it is easily checked by Proposition 3.7 that Q and Q' are uniformly equivalent. In fact, the SE-model $(\emptyset, abc)$ of Q' is not a UE-model of Q' due to the presence of the SE-model (a, abc), or alternatively because of (b, abc). Note that $Q \equiv_u Q'$ implies that $Q \equiv_u^A Q'$ for any A. Inspecting the remaining rows in the table, it can be seen that for any A, the sets of A-UE-models of Q and Q' are equal, as expected.

We conclude this section by remarking that we do not have a directly corresponding result to Theorem 3.16 for relativized uniform equivalence (see also the next subsection). A generalization of Proposition 3.17 is possible, however. The proof is in the online Appendix.

THEOREM 4.15. *Let P and Q be DLPs, and A a set of atoms. Then,*

(a) $P \equiv_u^A Q$ *implies* $UE^A(P) \subseteq SE^A(Q)$ *and* $UE^A(Q) \subseteq SE^A(P)$*;*
(b) $UE^A(P) \subseteq SE^A(Q)$ *and* $UE^A(Q) \subseteq SE^A(P)$ *implies* $P \equiv_u^A Q$*, whenever at least one of* P*,* Q*, or* A *is finite.*

4.3 Properties of Relativized Equivalences

This section collects a number of properties of A-SE-models and A-UE-models, respectively. Note that there are situations where A-SE-models and A-UE-models are the same concepts. In what follows, we shall use $card(A)$ to denote the cardinality of a set A.

PROPOSITION 4.16. *For any program P, and a set of atoms A with $card(A) < 2$, it holds that $SE^A(P) = UE^A(P)$.*

COROLLARY 4.17. *For programs P, Q and a set of atoms A with $card(A) < 2$, $P \equiv_s^A Q$ iff $P \equiv_u^A Q$.*

The following results are only given in terms of A-SE-models; the impact of the results on properties of A-UE-models is in most cases obvious, and thus not explicitly mentioned.

First, we are able to generalize Proposition 2.6 to relativized SE-models.

LEMMA 4.18. *An interpretation Y is an answer set of a program P iff $(Y, Y) \in SE^A(P)$ and, for each $X \subset Y$, $(X, Y) \notin SE^A(P)$.*

One drawback of A-SE-models is that they are not closed under program composition. Formally, $SE^A(P \cup Q) = SE^A(P) \cap SE^A(Q)$ does not hold in general; however, it holds whenever A contains all atoms occurring in P or Q. The fact that, in general, $SE^A(P \cup Q) \neq SE^A(P) \cap SE^A(Q)$ is not a surprise, since for

$A = \emptyset$, A-SE-models capture answer sets; and if this closure property would hold, answer set semantics would be monotonic.

PROPOSITION 4.19. *For programs P, Q, and a set of atoms A, we have the following relations:*

(i) $(Y, Y) \in SE^A(P) \cap SE^A(Q)$ *implies* $(Y, Y) \in SE^A(P \cup Q)$*;*
(ii) *for* $X \subset Y$, $(X, Y) \in SE^A(P \cup Q)$ *implies* $(X, Y) \in SE^A(R)$*, whenever* $(Y, Y) \in SE^A(R)$*, for* $R \in \{P, Q\}$*; and*
(iii) *the converse directions of* (i) *and* (ii) *do not hold in general.*

PROOF. *ad* (i): Suppose $(Y, Y) \notin SE^A(P \cup Q)$; then either: (a) $Y \not\models P \cup Q$; or (b) there exists a $Y' \subset Y$, with $(Y' \cap A) = (Y \cap A)$ such that $Y' \models (P \cup Q)^Y$. If $Y \not\models P \cup Q$, then either $Y \not\models P$ or $Y \not\models Q$. Consequently, $(Y, Y) \notin SE^A(P)$ or $(Y, Y) \notin SE^A(Q)$. So, suppose $Y \models P \cup Q$ and that (b) holds. Then, neither $(Y, Y) \in SE^A(P)$ nor $(Y, Y) \in SE^A(Q)$.

ad (ii): Let $R \in \{P, Q\}$. Suppose $(Y, Y) \in SE^A(R)$ and $(X, Y) \notin SE^A(R)$. The latter implies that no $X' \subset Y$, with $(X' \cap A) = (X \cap A)$, satisfies $X' \models P^Y$. Consequently, no such X' satisfies $X' \models (P \cup Q)^Y$, and thus $(X, Y) \notin SE^A(P \cup Q)$.

ad (iii): Take the following example programs. Consider programs over $V = \{a, b, c\}$ containing rules $R = \{ \leftarrow not\ a; \quad \leftarrow not\ b; \quad \leftarrow not\ c\}$. Note that $SE(R) = \{(X, V) \mid X \subseteq V\}$. Let

$$\begin{aligned} P_a &= R \cup \{a \leftarrow;\ b \leftarrow c;\ c \leftarrow b\}; \\ P_b &= R \cup \{b \leftarrow;\ a \leftarrow c;\ c \leftarrow a\}; \\ P_c &= R \cup \{c \leftarrow;\ a \leftarrow b;\ b \leftarrow a\}. \end{aligned}$$

Then, the SE-models of P_v are given by (v, abc) and (abc, abc), for $v \in V$.

Set now, for instance, $A = \{c\}$. Then, we have $SE^A(P_a) = SE^A(P_b) = \{(\emptyset, abc), (abc, abc)\}$, while $SE^A(P_c) = \emptyset$. However, $SE^A(P_a \cup P_b) = SE^A(P_a \cup P_c) = SE^A(P_b \cup P_c) = \{(abc, abc)\}$. This shows that for both (i) and (ii) in Proposition 4.19, the converse direction does not hold. □

The aforementioned result crucially influences the behavior of relativized consequence operators, that is, generalizations of $\models_e$, with $e \in \{s, u\}$, as introduced in Definitions 2.7 and 3.13, respectively, to the relativized notions of equivalence.

To check rule redundancy in the context of relativized strong equivalence, we give the following result.

Definition 4.20. A rule r is an *A-relativized SE-consequence* of a program P, denoted $P \models_s^A r$, if $(X, Y) \in SE^A(\{r\})$, for all $(X, Y) \in SE^A(P)$.

LEMMA 4.21. *For any set of atoms A, program P, and rule r with $(B^+(r) \cup H(r)) \subseteq A$, it holds that if $P \models_s^A r$, then $P \cup \{r\} \equiv_s^A P$.*

This result similarly applies to the notion of UE-consequence relative to A, that is, the restriction $(H(r) \cup B^+(r)) \subseteq A$ is also necessary in that case. However

(as in Proposition 3.14), the result has to be slightly rephrased for A-UE-models in order to properly handle the case of infinite programs.

In general, checking rule-redundancy with respect to relativized equivalences is a more involved task; we leave it for further study.

5. RESTRICTED CLASSES OF PROGRAMS

So far, we discussed several forms of equivalence for propositional programs in general. This section is devoted to two prominent subclasses of disjunctive logic programs, namely, positive and head-cycle-free programs. Notice that these classes include Horn and disjunction-free logic programs, respectively.

5.1 Positive Programs

While for programs with negation, strong and uniform equivalence are different, these notions coincide for positive programs, also in the relativized cases. We start with some technical results.

LEMMA 5.1. *Let P be a program, and A, $X \subset Y$ be sets of atoms. We have the following relations:*

(1) *If $(Y, Y) \in SE^A(P)$ and $(X, X) \in SE^A(P)$, then $((X \cap A), Y) \in SE^A(P)$.*
(2) *If $(X, Y) \in SE^A(P)$, then $(Y, Y) \in SE^A(P)$ and, whenever P is positive, there exists an $X' \subseteq Y$, with $(X' \cap A) = X$ such that $(X', X') \in SE^A(P)$.*

In other words, the set of all A-SE-models of a positive program P is determined by its total A-SE-models. An important consequence of this result is the following.

PROPOSITION 5.2. *Let P, Q be programs, P be positive, and suppose that the total A-SE-models of P and Q coincide. Then, $SE^A(P) \subseteq SE^A(Q)$.*

From this result, we get that deciding relativized strong and uniform equivalence of positive programs collapses to checking whether total A-SE-models coincide.

THEOREM 5.3. *Let P and Q be positive DLPs, and A a set of atoms. The following propositions are equivalent*:

(i) $P \equiv_s^A Q$;
(ii) $P \equiv_u^A Q$; *and*
(iii) $(Y, Y) \in SE^A(P)$ *iff* $(Y, Y) \in SE^A(Q)$, *for each interpretation* Y.

PROOF. (i) implies (ii) by definition; (ii) implies (iii) by Theorem 4.13. We show that (iii) implies (i). Applying Proposition 5.2 in the case of two positive programs immediately yields that (iii) implies $SE^A(P) = SE^A(Q)$. Hence $P \equiv_s^A Q$. □

Therefore, RSE and RUE are the same concepts for positive programs; we thus sometimes write generically $\equiv_e$ for $\equiv_s$ and $\equiv_u$.

An important consequence of this result is that A-UE-models (and thus UE-models) are capable of dealing with infinite programs as well, provided that they are positive.

COROLLARY 5.4. *Let A be a (possibly infinite) set of atoms, and P, Q (possibly infinite) positive program. Then $P \equiv_u^A Q$ holds iff $UE^A(P) = UE^A(Q)$.*

PROOF. The only-if direction has already been obtained in Theorem 4.13. For the if direction, note that $UE^A(P) = UE^A(Q)$ implies (iii) from Theorem 5.3, and since P and Q are positive, we derive $P \equiv_u^A Q$ immediately from that theorem. □

Concerning strong equivalence and uniform equivalence, Lemma 5.1 generalizes some well-known observations for positive programs.

PROPOSITION 5.5. *For any positive program P, and sets of atoms $X \subseteq Y$, $(X, Y) \in SE(P)$ iff $(X, X) \in SE(P)$ and $(Y, Y) \in SE(P)$.*

In other words, the set of all SE-models of a program P is determined by its total SE-models (that is, by the classical models of P). As known and easy to see from main results [Lifschitz et al. 2001; Turner 2003, 2001], on the class of positive programs classical and strong equivalence coincide. Using Theorem 5.3, we can extend this result.

THEOREM 5.6. *For positive programs P, Q, $P \equiv_e Q$ ($e \in \{s, u\}$) iff P and Q have the same classical models.*

Note that Sagiv [1988] showed that the uniform equivalence of DATALOG programs Π and Π' coincides with the equivalence of Π' and Π over Herbrand models; this implies the previous result for definite Horn programs. Maher [1988] showed a generalization of Sagiv's result for definite Horn logic programs with function symbols. Furthermore, Maher also pointed out that for DATALOG programs, this result has been independently established by Cosmadakis and Kanellakis [1986].

Example 5.7. Consider the positive programs $P = \{a \vee b \leftarrow a;\ b \leftarrow a\}$ and $Q = \{b \leftarrow a\}$. Clearly, $P \models Q$, since $Q \subset P$, but also $Q \models P$ holds (note that $b \leftarrow a$ is a subclause of $a \vee b \leftarrow a$). Hence, P and Q are uniformly equivalent, and even strongly equivalent (which is also easily verified).

Example 5.8. Consider the positive programs $P = \{a \vee b;\ c \leftarrow a;\ c \leftarrow b\}$ and $Q = \{a \vee b;\ c\}$. Their classical models are $\{a, c\}$, $\{b, c\}$, and $\{a, b, c\}$. Hence, P and Q are uniformly equivalent, and even strongly equivalent (due to Theorem 5.3).

Concerning the relativized notions, a result corresponding directly to Theorem 5.6 is not achievable. However, this is not surprising, otherwise we would have that in the case of empty A, $P \equiv_s^A Q$ (or $P \equiv_u^A Q$) collapses to classical equivalence. This of course cannot be the case, since for positive programs, $P \equiv Q$ denotes the equivalence of the *minimal* classical models of P and Q, rather than classical equivalence.

Thus, while for strong and uniform equivalence total models (Y, Y) for a positive program P coincide with the classical models Y of P, the relativized variants capture a more specific relation, namely, minimal models. We therefore define as follows.

Definition 5.9. An *A-minimal model* of a program P is a classical model Y of P such that for each $Y' \subset Y$ with $(Y' \cap A) = (Y \cap A)$, Y' is not a classical model of P.

Then, we can generalize Theorem 5.6 in the following manner.

THEOREM 5.10. *Let P and Q be positive DLPs, and A a set of atoms. Then $P \equiv_e^A Q$ ($e \in \{s, u\}$) iff P and Q have the same A-minimal models.*

PROOF. By Theorem 5.3, it is sufficient to show that the total A-SE-models of a program P equal its A-minimal models. This relation holds for positive programs, since $P^Y = P$ for any positive program P and any interpretation Y. In this case, the conditions for $(Y, Y) \in SE^A(P)$ are the same as for Y being A-minimal for P. □

Note that for $A = \emptyset$, the theorem states that $P \equiv_e^A Q$ iff the minimal classical models of P and Q coincide, reflecting the minimal model semantics of positive programs. By contrast, for $A = U$, the theorem states that $P \equiv_e^A Q$ iff all classical models of P and Q coincide, as stated before.

5.2 Head-Cycle-Free Programs

The class of head-cycle-free programs generalizes the class of normal logic programs by permitting a restricted form of disjunction. Still, it is capable of expressing nondeterminism such as, for example, a guess for the value of an atom a, which does not occur in the head of any other rule. For a definition of head-cycle-freeness, we refer to Section 2. As shown by Ben-Eliyahu and Dechter [1994], each head-cycle-free program can be rewritten to an ordinary equivalent normal program, which is obtained by shifting atoms from the head to the body.

More formally, let us define the following notations.

Definition 5.11. For any rule r, let

$$r^{\rightarrow} = \begin{cases} \{a \leftarrow B^+(r), not\ (B^-(r) \cup (H(r) \setminus \{a\})) \mid a \in H(r)\} & \text{if } H(r) \neq \emptyset, \\ \{r\} & \text{otherwise.} \end{cases}$$

For any DLP P, let $P_r^{\rightarrow} = (P \setminus \{r\}) \cup r^{\rightarrow}$; and $P^{\rightarrow} = \bigcup_{r \in P} r^{\rightarrow}$.

It is well-known that for any head-cycle-free program P, it holds that $P \equiv P^{\rightarrow}$ (see Ben-Eliyahu and Dechter [1994]). This result can be strengthened to uniform equivalence as well as to its relativized forms.

THEOREM 5.12. *For any head-cycle-free program P, and any set of atoms A, it holds that $P \equiv_u^A P^{\rightarrow}$.*

PROOF. For any set of facts $F \subseteq A$, it holds that $(P \cup F)^{\rightarrow} = P^{\rightarrow} \cup F$ and that this program is head-cycle-free iff P is head-cycle-free. Thus $P \cup F \equiv (P \cup F)^{\rightarrow} \equiv P^{\rightarrow} \cup F$. Hence $P \equiv_u^A P^{\rightarrow}$. □

We emphasize that a similar result for strong equivalence fails, as shown by the canonical counterexample in Example 1.1. Recall that the program $P = \{a \vee b \leftarrow .\}$ is not strongly equivalent to any NLP. Thus, we can not conclude without further consideration that a simple disjunctive "guessing clause" like the one in P (such that a and b do not occur in other rule heads) can be replaced in a more complex program by the unstratified clauses $a \leftarrow not\ b$ and $b \leftarrow not\ a$ (the addition of a further constraint $\leftarrow a, b$ is required). However, we can conclude this under uniform equivalence, taking standard program splitting results into account [Lifschitz and Turner 1994; Eiter et al. 1997]. Indeed, if we can split a program Q into programs Q_1 and Q_2, and further, if programs Q'_1, Q'_2 are uniform equivalent to Q_1 and Q_2, respectively, then, as easily seen, $Q \equiv_u Q'_1 \cup Q'_2$. Since P can be split into $P_1 = \{a \vee b \leftarrow .\}$ and $P_2 = P \setminus P_1$, the claim follows for uniform equivalence from Theorem 5.12. In fact, this result can be refined to *local shifting* of a rule [Eiter et al. 2004b].

The following result provides a characterization of arbitrary programs which are relativized strongly equivalent to their shift variant. A more detailed discussion of eliminating disjunction under different notions of equivalences was recently published in Eiter et al. [2004a].

First, we state a simple technical result.

LEMMA 5.13. *For any rule r, $SE(r) \subseteq SE(r^{\rightarrow})$.*

PROOF. Indirect. Suppose $(X, Y) \in SE(r)$ and $(X, Y) \notin SE(r^{\rightarrow})$. Then, $Y \models r$ and either $Y \cap B^-(r) \neq \emptyset$, $X \not\models B^+(r)$, or $X \cap H(r) \neq \emptyset$. By classical logic, $Y \models r$ iff $Y \models r^{\rightarrow}$. By assumption $(X, Y) \notin SE(r^{\rightarrow})$, there exists a rule in $r^{\rightarrow}$, with a as the only atom in its head, such that $a \notin X$, $Y \cap B^-(r) = \emptyset$, $X \models B^+(r)$, and $Y \cap (H(r) \setminus \{a\}) = \emptyset$. Hence, from the aforementioned conditions for $(X, Y) \in SE(r)$, only $X \cap H(r) \neq \emptyset$ applies. Then, some b from $H(r)$ is contained in X. If $a = b$, we get a contradiction to $a \notin X$; otherwise we get a contradiction to $Y \cap (H(r) \setminus \{a\}) = \emptyset$, since $Y \supseteq X$ and thus $b \in Y$. □

Next, we define the following set which characterizes the exact difference between r and $r^{\rightarrow}$ in terms of SE-models.

Definition 5.14. For any rule r, define

$$S_r = \{(X, Y) \mid X \subseteq Y,\ X \models B^+(r),\ Y \cap B^-(r) = \emptyset,\ card(H(r) \cap Y) \geq 2,\ H(r) \cap X = \emptyset\}.$$

PROPOSITION 5.15. *For any disjunctive rule r, $SE(r^{\rightarrow}) \setminus SE(r) = S_r$.*

A proof for this result can be found in Eiter et al. [2004a]. Hence, together with Lemma 5.13, we get that for any disjunctive rule r, S_r characterizes exactly the difference between r and $r^{\rightarrow}$ in terms of SE-models.

THEOREM 5.16. *Let P be a program, and $r \in P$. Then $P \equiv_s^A P_r^{\rightarrow}$ iff for each SE-model $(X, Y) \in SE(P_r^{\rightarrow}) \cap S_r$, there exists an $X' \subset Y$, with $X' \neq X$ and $(X' \cap A) = (X \cap A)$ such that $(X', Y) \in SE(P)$.*

As an immediate consequence of this result, we obtain the following characterization for general strong equivalence.

COROLLARY 5.17. *Let P be any DLP. Then, $P \equiv_s P^{\rightarrow}$ if and only if for every disjunctive rule $r \in P$, it holds that $P^{\rightarrow}$ has no SE-model $(X, Y) \in S_r$ (that is, $SE(P^{\rightarrow}) \cap S_r = \emptyset$).*

Example 5.18. Reconsider $P = \{a \vee b \leftarrow\}$. Then $P^{\rightarrow} = \{a \leftarrow not\ b, b \leftarrow not\ a\}$ has the SE-model $(\emptyset, ab)$ which satisfies the conditions for $S_{a \vee b \leftarrow}$. Note that also the extended program $P' = \{a \vee b \leftarrow, a \leftarrow b, b \leftarrow a\}$ is not strongly equivalent to its shifted program $P'^{\rightarrow}$. Indeed, $(\emptyset, ab)$ is also an SE-model of $P'^{\rightarrow}$. Furthermore, P' is also not uniformly equivalent to $P'^{\rightarrow}$, since $(\emptyset, ab)$ is moreover a UE-model of $P'^{\rightarrow}$, but P' has the single SE-model (and thus UE-model) (ab, ab).

We already have seen that shifting is possible if the disjunction is made exclusive with an additional constraint (see also Example 1.2).

Example 5.19. Let P be a program containing the two rules $r = a \vee b \leftarrow$ and $r' = \leftarrow a, b$. The rule r' guarantees that no SE-model (X, Y) of P or of $P_r^{\rightarrow}$ with $\{a, b\} \subseteq Y$ exists. But then S_r does not contain an element from $SE(P_r^{\rightarrow})$, and we get by Corollary 5.17 that $P \equiv_s P_r^{\rightarrow}$.

So far, we have presented a general semantic criterion for deciding whether shifting is invariant under $\equiv_s^A$. We close this section with a syntactic criterion generalizing the concept of head-cycle-freeness.

Definition 5.20. For a set of atoms A, a rule r is *A-head-cycle-free* (A-HCF) in a program P, iff the dependency graph of P, augmented with the clique over A, does not contain a cycle going through two atoms from $H(r)$. A program is A-HCF iff all of its rules are A-HCF.

In other words, the considered augmented graph of P as used in the definition is given by the pair $(A \cup Atm(P), E)$, with

$$E = \bigcup_{r \in P} \{(p, q) \mid p \in B^+(r), q \in H(r), p \neq q\} \cup \{(p, q),\ (q, p) \mid p, q \in A,\ p \neq q\}$$

and obviously coincides with the (ordinary) dependency graph of the program $P \cup R$, where R is the set of all unary rules over A. Recall that following Corollary 4.3, unary rules characterize relativized strong equivalence sufficiently. From this observation, the forthcoming results follow in a straightforward manner.

THEOREM 5.21. *For any program P, $r \in P$, and set of atoms A, $P \equiv_s^A P_r^{\rightarrow}$ whenever r is A-HCF in P.*

Note that if r is A-HCF in P, then r is HCF in $P \cup R$, where R is the set of unary rules over A. In turn, r then is HCF in all programs $P \cup R'$, with $R' \subseteq R$. Thus $P \cup R' \equiv P_r^{\rightarrow} \cup R'$ holds for all R', by known results. Consequently $P \equiv_s^A P_r^{\rightarrow}$.

COROLLARY 5.22. *For any program P, and set of atoms A, $P \equiv_s^A P^{\rightarrow}$ holds whenever P is A-HCF.*

Table II. Complexity of Equivalence Checking in Terms of Completeness Results

$P \equiv_s^A Q$ / $P \equiv_u^A Q$ / $P \stackrel{k}{\equiv}_s^A Q$ / $P \stackrel{k}{\equiv}_u^A Q$ / $P \equiv_u Q$	DLP	Positive	HCF	Normal	Horn
Horn	Π_2^P	coNP	coNP	coNP	coNP
	coNP	coNP	coNP	coNP	P
normal	Π_2^P	Π_2^P	Π_2^P/coNP	coNP	
	coNP	coNP	coNP	coNP	
HCF	Π_2^P	Π_2^P	Π_2^P/coNP		
	coNP	coNP	coNP		
positive	Π_2^P	Π_2^P			
	coNP/Π_2^P/Π_2^P	coNP			
DLP	Π_2^P				
	coNP/Π_2^P/Π_2^P				

6. COMPUTATIONAL COMPLEXITY

In this section, we address the computational complexity of checking various notions of equivalence for logic programs. We start with uniform equivalence, also taking the associated consequence operator into account. Then, we generalize these results and consider the complexity of relativized equivalence. Finally, we consider *bounded* relativization, that is, the problem of deciding $P \equiv_e^A Q$ ($e \in \{s, u\}$) such that the number of atoms *missing* in A is bounded by a constant k, denoted $P \stackrel{k}{\equiv}_e^A Q$. For all three groups of problems we provide a fine-grained picture of their complexity by taking different classes of programs into account.

Recall that $\Pi_2^P = \text{coNP}^{\text{NP}}$ is the class of problems such that the complementary problem is nondeterministically decidable in polynomial time with the help of an NP oracle, that is, in $\Sigma_2^P = \text{NP}^{\text{NP}}$. Furthermore, the class D^P consists of all problems expressible as the conjunction of a problem in NP and a problem in coNP. Moreover, any problem in D^P can be solved with a fixed number of NP-oracle calls, and is thus intuitively easier than a problem which is complete for Δ_2^P.

Our results are summarized in Table II. More precisely, the table shows the complexity of the considered problems $P \equiv_s^A Q$ and $P \equiv_u^A Q$ in the general case, as well as in the bounded case ($P \stackrel{k}{\equiv}_s^A Q$ and $P \stackrel{k}{\equiv}_u^A Q$). Moreover, we explicitly list the problem of uniform equivalence $P \equiv_u Q$. Depending on the program classes to which P and Q belong, the corresponding entry shows the complexity (in terms of a completness result) for all five equivalence problems with respect to these classes. In fact, the table has to be read as follows. For instance, the complexity of equivalence checking for DLPs in general is given by the entry in the last line and first column of Table II. The entry's first line refers to the problems $P \equiv_s^A Q$ and $P \equiv_u^A Q$ (which are both Π_2^P-complete), and the entry's second line refers to the problems $P \stackrel{k}{\equiv}_s^A Q$, $P \stackrel{k}{\equiv}_u^A Q$, and $P \equiv_u Q$, respectively. The latter two show Π_2^P-completenes while $P \stackrel{k}{\equiv}_s^A Q$ is coNP-complete. As another example, the complexity of deciding the equivalence of a head-cycle-free program and a normal program is reported by the entry in the second line of the third column.

Table III. Complexity of Model Checking

	A-SE-Models				A-UE-Models		
	A Bounded	General	$card(A)=1$	$A=\emptyset$	General	A Bounded	UE-Models
DLP/positive	in P	D^P	D^P	coNP	D^P	coNP	coNP
HCF	in P	NP	P	P	P	P	P
normal/Horn	in P	P	P	P	P	P	P

We now highlight the most interesting entries of Table II.

—*(Unrelativized) uniform equivalence is* harder *than (unrelativized) strong equivalence, and this result carries over to the case of bounded relativization*. This difference in complexity is only obtained if both programs involved contain head cycles and at least one contains default negation.

—*For the case of relativization, uniform equivalence is in some cases* easier *to decide than relativized strong equivalence*. This effect occurs only if both programs are head-cycle-free whereby one of them may be normal (but not Horn).

—*Another interesting case arises if two Horn programs are involved*. Therein, relativized equivalence is harder than in the bounded case, but also harder than *ordinary* equivalence (see Theorem 6.18 in Section 6.2 to follow). In every other case, relativization is never harder than ordinary equivalence.

—*Finally, we list those cases where bounded relativizations decrease the complexity*: As already mentioned for both RSE and RUE, this holds for comparing Horn programs. Additionally, in the case of RSE, there is a proper decrease whenever one program is disjunctive and the other is not Horn, or when P contains negation as well as head cycles and Q is Horn. In the latter situation, we also observe a descrease in the case of RUE. Additionaly, such a decrease for RUE is present if P is normal or HCF and Q is disjunctive and contains head cycles, or if two positive DLPs containing head cycles are compared.

Some of these effects can be explained by inspecting the underlying decision problem of model checking. For a set of atoms A, the problem of A-SE-model checking (respectively A-UE-model checking) is defined as follows: Given sets of atoms X, Y, and a program P, decide whether $(X,Y) \in SE^A(P)$ (respectively $(X,Y) \in UE^A(P)$). We compactly summarize our results on A-SE-model checking, respectively, A-UE-model checking, in Table III. This table has to be read as follows. The lines determine the class of programs to be dealt with and the columns refer to model checking problems in different settings. From left-to-right we have: (i) bounded A-SE-model checking of a program P, that is, it is assumed that $Atm(P) \setminus A$ contains a fixed number of atoms; (ii) the general A-SE-model checking problem; (iii) the special case of $card(A) = 1$, where A-SE-model checking and A-UE-model checking coincide; (iv) the special case of $card(A) = 0$ where both A-SE-model checking and A-UE-model checking coincide with answer set checking; (v) the general A-UE-model checking problem; (vi) bounded A-UE-model checking (analogously to bounded A-SE-model checking); and finally, we explicitly list the results for (vii) UE-model checking. All

results from Table III are proven in detail in the subsequent sections, as well. All entries except those in the first column are completeness results. Some interesting observations which also intuitively explain the different results for $\equiv_s^A$ and $\equiv_u^A$ include: (1) A-SE-model checking is easier than A-UE-model checking in the case of DLPs and bounded A: Roughly speaking, in this case the additional test for maximality in A-UE-model checking is responsible for the higher complexity; (2) for the case of head-cycle-free programs, A-SE-model checking is harder than A-UE-model checking, namely, it is NP-complete. This result is a consequence of Theorem 5.16, which guarantees that in terms of uniform equivalence, shifted HCF (and thus normal) programs can be employed; recall that this simplification is not possible in the context of strong equivalence.

Towards showing all the results in detail, we introduce the following notions used throughout this section. We often reduce propositional formulas to logic programs using, for a set of propositional atoms V, an additional set of atoms $\bar{V} = \{\bar{v} \mid v \in V\}$ within the programs to refer to negative literals. Consequently, we associate to each interpretation $I \subseteq V$ an extended interpretation $\sigma_V(I) = I \cup \{\bar{v} \mid v \in V \setminus I\}$, usually dropping the subscript V if clear from the context. The classical models of a formula ϕ are denoted by M_ϕ. Furthermore, we have a mapping $(\cdot)^*$ defined as $v^* = v$, $(\neg v)^* = \bar{v}$, and $(\phi \circ \psi)^* = \phi^* \circ \psi^*$, with v an atom, ϕ and ψ formulas, and $\circ \in \{\vee, \wedge\}$. A further mapping $\overline{(\cdot)}$ is defined as $\overline{v} = \bar{v}$, $\overline{\neg v} = v$, $\overline{\phi \vee \psi} = \overline{\phi} \wedge \overline{\psi}$, and $\overline{(\phi \wedge \psi)} = \overline{\phi} \vee \overline{\psi}$. To use these mappings in logic programs, we denote rules also by $a_1 \vee \ldots \vee a_l \leftarrow a_{l+1} \wedge \ldots \wedge a_m \wedge \mathit{not}\ a_{m+1} \wedge \ldots \wedge \mathit{not}\ a_n$.

Finally, we define for a set of atoms $Y \subseteq U$ the following sets of Horn rules:

$$
\begin{aligned}
Y_{\subseteq}^U &= \{\leftarrow y \mid y \in U \setminus Y\} \\
Y_{\subset}^U &= Y_{\subseteq}^U \cup \{\leftarrow y_1, \ldots, y_n\} \\
Y_{=}^U &= Y_{\subseteq}^U \cup Y
\end{aligned}
$$

Sometimes we do not write the superscript U, which refers to the universe. We assume that unless stated otherwise, U refers all atoms occurring in the programs under consideration.

6.1 Complexity of Uniform Equivalence

In this section, we address the computational complexity of uniform equivalence. While our main interest is in the problem of deciding the uniform equivalence of two given programs, we also consider the related problems of UE-model checking and UE-consequence. Our complexity results for deciding uniform equivalence of two given programs are collected from Table II into Table IV for matters of presentation. The table has to be read as with Table II. Note that, in general, uniform equivalence is complete for class Π_2^P and therefore more complex than deciding strong equivalence, which is in coNP [Pearce et al. 2001; Lin 2002; Turner 2003]. Thus, the more liberal notion of uniform equivalence generally comes at higher computational cost. However, for important classes of programs it has the same complexity as strong equivalence.

In what follows, we prove all the results in Table IV. Towards these results, we start with the problem of UE-model checking. Let $\|\alpha\|$ denote the size of an object α.

Table IV. Complexity of Uniform Equivalence in Terms of Completeness Results

$P \equiv_u Q$	DLP	Positive	HCF	Normal	Horn
Horn	coNP	coNP	coNP	coNP	P
normal	coNP	coNP	coNP	coNP	
HCF	coNP	coNP	coNP		
positive	Π_2^P	coNP			
DLP	Π_2^P				

THEOREM 6.1. *Given a pair of sets* (X, Y) *and a program* P*, the problem of deciding whether* $(X, Y) \in UE(P)$ *is* (i) coNP*-complete in general, and* (ii) *feasible in polynomial time with respect to* $\|P\| + \|X\| + \|Y\|$ *if* P *is head-cycle-free. Hardness in case* (i) *holds, even for positive programs.*

PROOF. Testing $Y \models P$ and $X \models P^Y$, that is, $(X, Y) \in SE(P)$, for given interpretations X, Y, is possible in polynomial time. If $X \subset Y$ it remains to check, whether no X', $X' \models P^Y$, exists such that $X \subset X' \subset Y$. This can be done via checking

$$P^Y \cup X \cup Y_{\subset} \quad \models \quad X_{=}. \tag{1}$$

In fact, each model X' of $P^Y \cup X \cup Y_{\subset}$ gives a nontotal SE-model (X', Y) of P with $X \subseteq X' \subset Y$. By contrast, the only model of $X_{=}$ is X itself. Hence, Eq. (1) holds iff no X' with $X \subset X' \subset Y$ exists such that $(X', Y) \in SE(P)$, namely, iff $(X, Y) \in UE(P)$. In general, deciding (1) is in coNP, witnessed by the membership part of (i).

If P is normal, then the involved programs in (1) are Horn and, since classical consequence can be decided in polynomial time for Horn programs, the overall check proceeds in polynomial time. Finally, if P is head-cycle-free, then so is P^Y. Moreover, by Theorem 5.12 we have $P \equiv_u P^{\rightarrow}$. Hence, in this case, (1) holds iff $(P^{\rightarrow})^Y \cup X \cup Y_{\subset} \models X_{=}$. Since $P^{\rightarrow}$ is normal, the latter test can be done in polynomial time (with respect to $\|P\| + \|X\| + \|Y\|$). This shows (ii).

It remains to show the coNP-hardness of UE-model checking for positive programs. We show this by a reduction from tautology checking. Let $F = \bigvee_{k=1}^{m} D_k$ be a propositional formula in DNF containing literals over atoms $X = \{x_1, \ldots, x_n\}$, and consider the following program P:

$$\begin{aligned} P = \{ \;& x_i \vee \bar{x}_i \leftarrow x_j. \quad x_i \vee \bar{x}_i \leftarrow \bar{x}_j. && \mid \; 1 \le i \ne j \le n \} \cup \\ \{ \;& x_i \leftarrow x_j, \bar{x}_j. \quad \bar{x}_i \leftarrow x_j, \bar{x}_j. && \mid \; 1 \le i \ne j \le n \} \cup \\ \{ \;& x_i \leftarrow D_k^*. \quad \bar{x}_i \leftarrow D_k^*. && \mid \; 1 \le k \le m, 1 \le i \le n \}, \end{aligned}$$

where D_k^* results from D_k by replacing literals $\neg x_i$ by $\bar{x}_i$.

Since P is positive, the SE-models of P are determined by its classical models, which are given by $\emptyset$, $X \cup \bar{X}$, and $\sigma(I)$, for each interpretation $I \subseteq X$ making F false. Hence, $(\emptyset, X \cup \bar{X})$ is an SE-model of P and $(\emptyset, X \cup \bar{X}) \in UE(P)$ iff F is a tautology. This proves coNP-hardness. □

In fact, also those UE-model checking problems which are feasible in polynomial time are hard for the class P.

THEOREM 6.2. *Given a pair of sets* (X, Y) *and a head-cycle-free program* P*, the problem of deciding whether* $(X, Y) \in UE(P)$ *is* P*-complete. Hardness holds, even if* P *is definite Horn.*

PROOF. Membership has already been shown in Theorem 6.1. We show hardness via a reduction from the P-complete problem HORNSAT to UE-model checking for Horn programs. Hence, let $\phi = \phi_f \wedge \phi_r \wedge \phi_c$ be a Horn formula over atoms V, where $\phi_f = a_1 \wedge \cdots \wedge a_n$; $\phi_r = \bigwedge_{i=1}^{m}(b_{i,1} \wedge \cdots \wedge b_{i,k_i} \rightarrow b_i)$; and $\phi_c = \bigwedge_{i=1}^{l} \neg(c_{i,1} \wedge \cdots \wedge c_{i,k_i})$. Without loss of generality, suppose $n \geq 1$ (otherwise ϕ would be trivially satisfiable by the empty interpretation). Let u, w be new atoms, and take the program

$$\begin{aligned} P \;=\; & \{a_i \leftarrow u \mid 1 \leq i \leq n\} \cup \\ & \{b_i \leftarrow b_{i,1}, \ldots, b_{i,k_i} \mid 1 \leq i \leq m\} \cup \\ & \{w \leftarrow c_{i,1}, \ldots, c_{i,k_i} \mid 1 \leq i \leq l\} \cup \\ & \{u \leftarrow v;\; v \leftarrow w \mid v \in V\} \cup \{u \leftarrow w\}. \end{aligned}$$

We show that ϕ is unsatisfiable iff $(\emptyset, V \cup \{u, w\})$ is a UE-model of P. Note that both $\emptyset$ and $V \cup \{u, w\}$ are classical models of P for any ϕ. Since P is positive, it is sufficient to show that ϕ is satisfiable iff a model M of P exists such that $\emptyset \subset M \subset (V \cup \{u, w\})$.

Suppose ϕ is satisfiable, and M is a model of ϕ; then it is easily checked that $M \cup \{u\}$ is a model of P. So suppose ϕ is unsatisfiable, and towards a contradiction let some M with $\emptyset \subset M \subset (V \cup \{u, w\})$ be a model of P. From the rules $\{v \leftarrow w \mid v \in V\} \cup \{u \leftarrow w\}$, we get $w \notin M$. Hence, the constraints ϕ_c are true under M. Since M is not empty, either $u \in M$ or some $v \in V$ is in M. However, the latter implies that $u \in M$ as well (by rules $\{u \leftarrow v \mid v \in V\}$). Recall that ϕ_f is not empty by assumption, hence all a_i's from ϕ_f are in M. Then, it is easy to see that $M \setminus \{u\}$ satisfies ϕ, which contradicts our assumption that ϕ is unsatisfiable. □

We now consider the problem of our main interest, namely, deciding uniform equivalence. By the previous theorem, the following upper bound on the complexity of this problem is obtained.

LEMMA 6.3. *Given two DLPs* P *and* Q*, deciding whether* $P \equiv_u Q$ *is in the class* Π_2^P.

PROOF. To show that two DLPs P and Q are not uniformly equivalent, we can by Theorem 3.7 guess an SE-model (X, Y) such that (X, Y) is UE-model of exactly one of the programs P and Q. By Theorem 6.1, the guess for (X, Y) can be verified in polynomial time with the help of an NP oracle. This proves Π_2^P-membership of $P \equiv_u Q$. □

This upper bound has a complementary lower bound, proved in the following result.

THEOREM 6.4. *Given two DLPs* P *and* Q*, deciding whether* $P \equiv_u Q$ *is* Π_2^P*-complete. Hardness holds even if one of the programs is positive.*

PROOF. Membership in Π_2^P has already been established in Lemma 6.3. To show Π_2^P-hardness, we provide a polynomial reduction of evaluating a quantified Boolean formula (QBF) from a fragment which is known Π_2^P-complete to deciding the uniform equivalence of two DLPs P and Q.

Consider a $QBF_{2,\forall}$ of the form $F = \forall X \exists Y \phi$ with $\phi = \bigwedge_{i=1}^{m} C_i$, where each C_i is a disjunction of literals over the Boolean variables in $X \cup Y$. Deciding whether a given such F is true is well-known to be Π_2^P-complete.

For the moment, let us assume that $X = \emptyset$, that is, the QBF amounts to a SAT-instance F over Y. More precisely, in what follows we reduce the satisfiability problem of the quantifier-free formula ϕ to the problem of deciding the uniform equivalence of two programs P and Q. Afterwards, we take the entire QBF, that is, F, into account.

Let a and b be fresh atoms and define

$$P = \{y \vee \bar{y} \leftarrow \mid y \in Y\} \cup \tag{2}$$
$$\{b \leftarrow y, \bar{y};\ y \leftarrow b;\ \bar{y} \leftarrow b \mid y \in Y\} \cup \tag{3}$$
$$\{b \leftarrow \overline{C}_i \mid 1 \leq i \leq m\} \cup \tag{4}$$
$$\{a \leftarrow\}. \tag{5}$$

Note that P is positive. The second program is defined as follows:

$$Q = \{y \vee \bar{y} \leftarrow z \mid y \in Y;\ z \in Y \cup \bar{Y} \cup \{a\}\} \cup \tag{6}$$
$$\{b \leftarrow y, \bar{y};\ y \leftarrow b;\ \bar{y} \leftarrow b \mid y \in Y\} \cup \tag{7}$$
$$\{b \leftarrow \overline{C}_i \mid 1 \leq i \leq m\} \cup \tag{8}$$
$$\{a \leftarrow b;\ a \leftarrow not\ b;\ a \leftarrow not\ a\} \tag{9}$$

The only differences between the two programs P and Q are located in the rules (2) compared to (6) as well as (5) compared to (9). Note that (9) also contains default negation.

Let us first compute the SE-models of P. Since P is positive, it is sufficient to consider classical models. Let $\mathcal{A} = Y \cup \bar{Y} \cup \{a, b\}$. First, $\mathcal{A}$ is clearly a classical model of P, and so is $\sigma(I) \cup \{a\}$, for each classical model $I \in M_\phi$. In fact, these are the only models of P. This can be seen as follows. By rules (2), at least one y or $\bar{y}$ must be contained in a model for each $y \in Y$. By (3), if both y and $\bar{y}$ are contained in a candidate model for some $y \in Y$ or b is contained in the candidate, then the candidate is spoiled up to $Y \cup \bar{Y} \cup \{b\}$. Hence the classical models of (2)–(3) are given by $\{\sigma(I) \mid I \subseteq Y\}$ and $Y \cup \bar{Y} \cup \{b\}$. Now, (4) eliminates those candidates which make ϕ false by "lifting" them to $Y \cup \bar{Y} \cup \{b\}$. By (5) we finally have to add a to the remaining candidates.

Hence, the SE-models of P are given by

$$\{(\sigma(I) \cup \{a\}, \sigma(I) \cup \{a\}) \mid I \in M_\phi\} \cup \{(\sigma(I) \cup \{a\}, \mathcal{A}) \mid I \in M_\phi\} \cup (\mathcal{A}, \mathcal{A}).$$

Obviously, each SE-model of P is also a UE-model of P.

We now analyze Q. First observe that the classical models of P and Q coincide. This is due the fact that (5) is classically equivalent to (9) and thus classically derives a, making (6) and (2) do the same job in this context. However, since Q is not positive, we have to consider the respective reducts of Q to compute the

SE-models. We start with SE-models of the form $(X, \mathcal{A})$. In fact, $(X, \mathcal{A}) \in SE(Q)$ iff $X \in \{\emptyset, \mathcal{A}\} \cup \{\sigma(I) \mid I \in M_\phi\} \cup \{\sigma(I) \cup \{a\} \mid I \in M_\phi\}$. The remaining SE-models of Q are all total and, as for P, given by $\{(\sigma(I) \cup \{a\}, \sigma(I) \cup \{a\}) \mid I \in M_\phi\}$.

Hence, the set of all SE-models of Q is

$$\{(\sigma(I) \cup \{a\}, \sigma(I) \cup \{a\}) \mid I \in M_\phi\} \ \cup \ \{(\sigma(I) \cup \{a\}, \mathcal{A}) \mid I \in M_\phi\}$$
$$\cup \, (\mathcal{A}, \mathcal{A}) \ \cup \{(\sigma(I), \mathcal{A}) \mid I \in M_\phi\} \ \cup \ (\emptyset, \mathcal{A});$$

having additional SE-models compared to P, namely, $(\emptyset, \mathcal{A})$ and $\{(\sigma(I), \mathcal{A}) \mid I \in M_\phi\}$. Note, however, that the latter SE-models are never UE-models of Q, since clearly $\sigma(I) \subset (\sigma(I) \cup \{a\})$ for all $I \in M_\phi$.

Thus, if M_ϕ is not empty, the UE-models of P and Q coincide; otherwise there is a single nontotal UE-model of Q, namely $(\emptyset, \mathcal{A})$. Note that the latter is not a UE-model of Q in the case $M_\phi \neq \emptyset$, since, for each $I \in M_\phi$, $\sigma(I) \neq \emptyset$. Consequently, the UE-models of P and Q coincide iff M_ϕ is not empty, that is, iff ϕ is satisfiable.

So far we have shown how to construct programs P and Q such that uniform equivalence encodes SAT. To complete the reduction for the QBF, we now also take X into account.

We add in both P and Q the set of rules

$$\{x \vee \bar{x} \leftarrow; \ \leftarrow x, \bar{x} \mid x \in X\},$$

where the $\bar{x}$'s are fresh atoms. The set $\mathcal{A}$ remains as before, that is, without any atom of the form x or $\bar{x}$.

This has the following effects. First, the classical models of both P and Q are now given by $\sigma_{X \cup Y}(I) \cup \{a\}$, for each $I \in M_\phi$, and $(\sigma_{X \cup Y}(J) \cup \mathcal{A}) = (\sigma_X(J) \cup \mathcal{A})$, for each $J \subseteq X$. Therefore, the SE-models of P are given by

$$\{(\sigma_{X \cup Y}(I) \cup \{a\}, \sigma_{X \cup Y}(I) \cup \{a\}) \mid I \in M_\phi\} \cup \tag{10}$$
$$\{(\sigma_{X \cup Y}(I) \cup \{a\}, \sigma_X(I) \cup \mathcal{A}) \mid I \in M_\phi\} \cup \tag{11}$$
$$\{(\sigma_X(J) \cup \mathcal{A}, \sigma_X(J) \cup \mathcal{A}) \mid J \subseteq X\}. \tag{12}$$

Again, each SE-model of P is also a UE-model of P. For Q, the argumentation from before is used analogously. In particular, for each $J \subseteq X$, we get an additional SE-model $\{(\sigma_X(J), \sigma_X(J) \cup \mathcal{A})\}$ for Q. Thus, the UE-models of P and Q coincide iff none of these additional SE-models $\{(\sigma_X(J), \sigma_X(J) \cup \mathcal{A})\}$ of Q is an UE-model of Q, either. This is the case iff for each $J \subseteq X$ there exists a truth assignment to Y making ϕ true, namely, iff the QBF $\forall X \exists Y \phi$ is true.

Since P and Q are obviously constructible in polynomial time, our result follows. □

For the construction of P and Q, in the preceding proof we used—for matters of presentation—two additional atoms a and b. However, one can resign on b by replacing rules (3) and (4) in both programs by $\{y \leftarrow \overline{C}_i; \ \bar{y} \leftarrow \overline{C}_i \mid y \in Y; \ 1 \leq i \leq m\}$; and additionally, rules (9) in Q by $\{a \leftarrow \overline{C}_i \mid 1 \leq i \leq m\} \cup \{\leftarrow not\ a\}$. Hence, already a single occurrence of default negation in one of the compared programs makes the problem harder. Note that the equivalence of two positive disjunctive programs is among the coNP-problems discussed in the following.

THEOREM 6.5. *Let P and Q be positive DLPs. Then, deciding whether $P \equiv_u Q$ is* coNP- *complete, where* coNP-*hardness holds even if one of the programs is Horn.*

PROOF. By Theorem 5.3, uniform equivalence and strong equivalence are the same concepts for positive programs. Since strong equivalence is generally in coNP, the membership part of the theorem follows immediately.

We show coNP-hardness for a positive DLP P and a Horn program Q by a reduction from UNSAT. Given a propositional formula in CNF $F = \bigwedge_{i=1}^{m} C_i$ over atoms X, let

$$P = \{C_i^* \vee a \leftarrow \mid 1 \leq i \leq m\} \cup \{\leftarrow x, \bar{x} \mid x \in X\}; \quad \text{and}$$
$$Q = \{a \leftarrow\} \cup \{\leftarrow x, \bar{x} \mid x \in X\}.$$

By Theorem 5.6, $P \equiv_u Q$ iff P and Q have the same classical models. The latter holds iff each model of P contains the atom a. But then, F is unsatisfiable. □

We now turn to head-cycle-free programs.

THEOREM 6.6. *Let P and Q be DLPs, and P head-cycle-free. Then, deciding whether $P \equiv_u Q$ is* coNP- *complete where* coNP-*hardness holds even if P is normal and Q is Horn.*

PROOF. For the membership part, by Theorem 3.16, $P \equiv_u Q$ iff $P \models_u Q$ and $Q \models_u P$. Both tasks are in coNP (see Theorem 6.9 to follow). Since the class coNP is closed under conjunction, it follows that deciding whether $P \equiv_u Q$ is in coNP.

To show coNP-hardness, consider the programs from the proof of Theorem 6.5. Indeed, P is HCF and therefore $P \equiv_u P^{\rightarrow}$, by Theorem 5.12. Using the same argumentation as previously, yields $P^{\rightarrow} \equiv_u Q$ iff F is unsatisfiable. This shows the coNP-hardness result for comparing normal and Horn programs. □

Note that Sagiv [1988] showed that deciding $P \equiv_u Q$ for given definite Horn programs P and Q is polynomial, which easily follows from his result that the property of uniform containment (whether the least model of $P \cup R$ is always a subset of $Q \cup R$) can be decided in polynomial time. As pointed out by Maher [1988], Buntine [1988] has, like Sagiv, provided an algorithm for deciding uniform containment.

Sagiv's result clearly generalizes to arbitrary Horn programs, since by Theorem 5.6, deciding $P \equiv_u Q$ reduces to checking the classical equivalence of Horn theories, which is known to be P-complete.

COROLLARY 6.7. *Deciding the uniform equivalence of Horn programs is* P-*complete.*

This concludes our analysis on the complexity of checking uniform equivalence. Our results cover all possible combinations of the classes of programs considered, namely, DLPs, positive programs, normal programs, head-cycle-free programs, as well as Horn programs, as already highlighted in Table IV.

Finally, we complement the results on uniform equivalence and UE-model checking by addressing the complexity of UE-consequence. The proofs of these results can be found in the online Appendix.

Table V. Complexity of Relativized Equivalences in Terms of Completeness Results

$P \equiv_s^A Q$ / $P \equiv_u^A Q$	DLP	Positive	HCF	Normal	Horn
Horn	Π_2^P	coNP	coNP	coNP	coNP
normal	Π_2^P	Π_2^P	Π_2^P/coNP	coNP	
HCF	Π_2^P	Π_2^P	Π_2^P/coNP		
positive	Π_2^P	Π_2^P			
DLP	Π_2^P				

THEOREM 6.8. *Given a DLP P and a rule r, deciding $P \models_u r$ is:* (i) Π_2^P*-complete in general,* (ii) coNP*-complete if P is either positive or head-cycle-free, and* (iii) *polynomial if P is Horn.*

THEOREM 6.9. *Let P, Q be DLPs. Then, $P \models_u Q$ is* coNP*-complete whenever one of the programs is head-cycle-free.* coNP*-hardness holds even if P is normal and Q is Horn.*

6.2 Complexity of Relativized Equivalence

We now generalize the complexity results to relativized forms of equivalence. In particular, we investigate the complexity of A-SE/UE-model checking, as well as of the equivalence problems $\equiv_s^A$ and $\equiv_u^A$, respectively. Like in the previous section, we also consider different classes of programs. Our results are summarized at a glance in Table V for both RSE and RUE just by highlighting where the complexity differs. Note that the only differences between RSE and RUE stem from the entries Π_2^P/coNP in the column for head-cycle-free programs. Here we have that in the cases HCF/HCF and HCF/normal, checking whether $\equiv_s^A$ is in general harder for RSE than for RUE. Another issue to mention is that already for uniform equivalence, the concept of relativization make things more difficult. One need only compare the first two columns of Tables IV and V, respectively. This is even worse for strong equivalence, which is in coNP in its unrelativized version and now jumps up to Π_2^P-completeness in several cases. Finally, also the comparison of two Horn programs becomes intractable, namely coNP-complete, compared to the polynomial-time result in the cases of unrelativized strong and uniform equivalence.

To summarize, both RSE and RUE are: (i) harder to decide than in their unrelativized versions in several cases, and (ii) generally of the same complexity, except when head-cycle-free programs are involved. Note that observation (ii), on the one hand, contrasts the current view that notions of strong equivalence have milder complexity than notions like uniform equivalence. On the other hand, the intuition behind this gap becomes apparent if one takes into account that for HCF programs P, $P \equiv_u^A P^{\rightarrow}$ holds, whereas $P \equiv_s^A P^{\rightarrow}$ does not.

For an even more fine-grained picture, note that the problems associated with equivalence tests relative to an atom set A call for further distinctions between several cases concerning the concrete instance A. We identify the following:

—$card(A) = 0$: In this case, both A-SE- and A-UE-model checking collapse to answer set checking; correspondingly, RSE and RUE collapse to ordinary equivalence; and

—$card(A) < 2$: By Proposition 4.16 and Corollary 4.17, A-SE-models and A-UE-models coincide, and thus, RSE and RUE are the same concepts.

Our results for $\equiv_e^A$, $e \in \{s, u\}$, given in the following, consider arbitrary fixed A unless stated otherwise. Moreover, we consider that A contains only atoms which also occur in the programs under consideration. In some cases the hardness part of the complexity results is obtained only if $card(A) > k$ for some constant k. We shall make these cases explicit.

Another special case for A is to consider *bounded relativization*. This denotes the class of problems where the cardinality of $(V \setminus A)$ is less than or equal to a fixed constant k, with V being the atoms occurring in the two programs compared. Note that this concept contains strong and uniform equivalence, respectively, as special cases, that is, if $(V \setminus A) = \emptyset$. We deal with bounded relativization explicitly in the subsequent section.

Towards deriving the results from Table V, we first consider model checking problems. Formally, for a set of atoms A, the problem of A-SE-model checking (respectively A-UE-model checking) is defined as follows: Given sets of atoms X, Y, and a program P, decide whether $(X, Y) \in SE^A(P)$ (respectively $(X, Y) \in UE^A(P)$). We start with the following tractable cases.

THEOREM 6.10. *Given a pair of sets (X, Y), a set of atoms A, and a program P, the problem of deciding whether $(X, Y) \in SE^A(P)$ (respectively $(X, Y) \in UE^A(P)$) is feasible in polynomial time with respect to $\|P\|+\|X\|+\|Y\|$, whenever P is normal (respectively whenever P is HCF).*

PROOF. We start with the test for whether (X, Y) is an A-SE-model of a normal program P. Note that P^Y is Horn, and that Y is a model of P^Y iff Y is a model of P. Consider the following algorithm.

Algorithm

(1) Check whether Y is a model of P^Y.
(2) Check whether $P_Y = P^Y \cup (Y \cap A) \cup Y_\subset$ is unsatisfiable.
(3) If $X \subset Y$, check whether $P_X = P^Y \cup (X \cap A) \cup \{\leftarrow x \mid x \in (A \setminus X)\} \cup Y_\subseteq$ is satisfiable.

Note that each step is feasible in polynomial time, especially since both P_X and P_Y are Horn. Hence, it remains to proof that the preceding algorithm holds exactly if (X, Y) is A-SE-model of P. This is seen as follows: Each step exactly coincides with one of the conditions for checking whether (X, Y) is an A-SE-model, namely: (1) $Y \models P$; (2) for all $Y' \subset Y$ with $(Y' \cap A) = (Y \cap A)$, $Y' \not\models P^Y$; and (3) $X \subset Y$ implies the existence of a $X' \subseteq Y$ with $(X' \cap A) = X$ such that $X' \models P^Y$.

For the result on A-UE-model checking we use similar argumentation. First, suppose that P is normal and consider the algorithm from before, but replacing the second step by

(2a) Check whether $P^Y \cup (X \cap A) \cup Y_\subset \models (X \cap A) \cup \{\leftarrow x \mid x \in (A \setminus X)\}$.

The desired algorithm then corresponds to the respective conditions for A-UE-model checking following Proposition 4.12. To be more specific, the models

of $P^Y \cup (X \cap A) \cup Y_\subset$ are those X' with $(X \cap A) \subseteq X' \subset Y$ such that $X' \models P^Y$. The set of models of the righthand-side is given by $\{Z \mid (Z \cap A) = (X \cap A)\}$. Hence, the test in (2a) is violated iff there exists an X' with $(X \cap A) \subset (X' \cap A)$ and $X' \subset Y$ such that $X' \models P^Y$, that is, iff $(X, Y) \notin UE^A(P)$. Moreover, for HCF programs, $P^\rightarrow$ is A-UE-equivalent to P, following Theorem 5.12, namely, the A-UE-models for P and $P^\rightarrow$ coincide. Applying $P^\rightarrow$ to the presented procedure thus shows that A-UE-model checking is feasible in polynomial time also for HCF programs. □

Without a formal proof, we mention that these tractable model checking problems are complete for the class P. Indeed, one can reuse the argumentation from the proof of Theorem 6.2 and take, for instance, $A = \{u\}$. Then $(\emptyset, V \cup \{u, w\}) \in SE^A(P) = UE^A(P)$ iff the encoded Horn formula is satisfiable. Note that P-hardness holds also for answer set checking (that is, $A = \emptyset$) by the straightforward observation that a Horn program P has an answer set iff P is satisfiable.

Next, we consider the case of A-SE-model checking for head-cycle-free programs. Recall that for $card(A) < 2$, A-SE-model checking coincides with A-UE-model checking, and thus in these cases A-SE-model checking is feasible in polynomial time, as well. However, in general, A-SE-model checking is harder than A-UE-model checking for head-cycle-free programs.

THEOREM 6.11. *Let (X, Y) be a pair of interpretations, and P a head-cycle-free program. Deciding whether $(X, Y) \in SE^A(P)$ is* NP*-complete. Hardness holds for any fixed A with $card(A) \geq 2$.*

PROOF. For the membership result we argue as follows. First, we check whether $(Y, Y) \in SE^A(P)$. Note that $(Y, Y) \in SE^A(P)$ holds iff $(Y, Y) \in UE^A(P)$. By Theorem 6.10, the latter test is feasible in polynomial time. It remains to check whether there exists a $X' \subseteq Y$ with $(X' \cap A) = X$ such that $X' \models P^Y$. This task is in NP, and therefore the entire test is in NP.

For the corresponding NP-hardness, consider the problem of checking the satisfiability of a formula $\psi = \bigwedge_{j=1}^m C_j$ in CNF given over a set of atoms V. This problem is NP-complete. We reduce it to A-SE-model checking for an HCF program. Consider the following program with additional atoms a_1, a_2, $\bar{V} = \{\bar{v} \mid v \in V\}$, and let $A = \{a_1, a_2\}$.

$$P = \{v \vee \bar{v} \leftarrow \mid v \in V\} \tag{13}$$

$$\{v \leftarrow a_1;\ \bar{v} \leftarrow a_1 \mid v \in V\} \tag{14}$$

$$\{a_2 \leftarrow \overline{C}_j \mid 1 \leq j \leq m\} \tag{15}$$

$$\{a_2 \leftarrow v, \bar{v} \mid v \in V\} \tag{16}$$

Note that P is HCF. Let $Y = V \cup \bar{V} \cup A$. We show that $(\emptyset, Y) \in SE^A(P)$ iff ψ is satisfiable. It is clear that $Y \models P$ and no $Y' \subset Y$ with $(Y' \cap A) = (Y \cap A)$ satisfies $Y' \models P^Y = P$ due to rules (14). This shows that $(Y, Y) \in SE^A(P)$. Now, $(\emptyset, Y) \in SE^A(P)$ iff there exists a $X \subseteq (V \cup \bar{V})$ such that $X \models P^Y = P$. Suppose $X \models P$. Since $a_2 \notin X$, X must represent a consistent guess due to rules (13) and (16). Moreover, X has to represent a model of ψ due to rules (15). Finally,

$X \models (14)$ holds by trivial means, namely since $a_1 \notin X$. The converse direction is by analogous arguments. Hence $(\emptyset, Y) \in SE^A(P)$ iff there exists a model of ψ, that is, iff ψ is satisfiable.

This shows the hardness for $card(A) = 2$. To obtain coNP-hardness for any A with $k = card(A) > 2$ and such that all $a \in A$ are also occurring in the program, consider P as earlier augmented by rules $\{a_{i+1} \leftarrow a_i \mid 2 \leq i < k\}$ and $A = \{a_i \mid 1 \leq i \leq k\}$. By analogous arguments as before, one can show that then $(\emptyset, (V \cup \bar{V} \cup A)) \in SE^A(P)$ iff ψ is satisfiable. □

The next result concerns A-SE-model checking and A-UE-model checking of disjunctive logic programs in general and for positive DLPs. For $A = \emptyset$, these tasks coincide with answer set checking, which is known to be coNP-complete (see, e.g., Eiter and Gottlob [1995]). Already a single element in A yields a mild increase of complexity.

THEOREM 6.12. *Let (X, Y) be a pair of interpretations, and P a DLP. Deciding whether $(X, Y) \in SE^A(P)$ (respectively $(X, Y) \in UE^A(P)$) is D^P-complete. Hardness holds for any fixed A with $card(A) \geq 1$, even for positive programs.*

PROOF. We first show D^P-membership. By Definition 4.4, a pair of interpretations (X, Y) is an A-SE-model of P iff: (1) (X, Y) is a valid A-SE-interpretation; (2) $Y \models P$; (3) for all $Y' \subset Y$ with $(Y' \cap A) = (Y \cap A)$, $Y' \not\models P^Y$; and (4) $X \subset Y$ implies the existence of a $X' \subseteq Y$ with $(X' \cap A) = X$ such that $X' \models P^Y$ holds. Obviously, items (1) and (2) can be verified in polynomial time. The complementary problem of (3) can be verified by a guess for Y' and a derivability check. Furthermore, (4) can be verified by a guess for X' and a derivability check. Hence, (3) is in coNP and (4) is in NP, which shows D^P-membership. The similar holds in the case of A-UE-models: By Proposition 4.12, $(X, Y) \in UE^A(P)$ iff: (1) (X, Y) is a valid A-SE-interpretation; (2) $Y \models P$; (3) for each $X'' \subset Y$ with $(X \cap A) \subset (X'' \cap A)$ or $X'' = (Y \cap A)$, $X'' \not\models P^Y$ holds; and (4) $X \subset Y$ implies that there exists a $X' \subseteq Y$ with $(X' \cap A) = X$ such that $X' \models P^Y$. As before, one can verify that the first two conditions are feasible in polynomial time, whereas checking (3) is a coNP-test, and checking (4) an NP-test.

For the matching lower bound, we consider the case where $card(A) = 1$. Therefore, the D^P-hardness of both A-SE-model checking is A-UE-model checking is captured at once. We consider the problem of jointly checking whether:

(a) a formula $\phi = \bigvee_{i=1}^{n} D_i$ in DNF is a tautology; and
(b) a formula $\psi = \bigwedge_{j=1}^{m} C_j$ in CNF is satisfiable.

This problem is D^P-complete, even if both formulas are given over the same set of atoms V. Consider the following positive program:

$$P = \{v \vee \bar{v} \leftarrow \mid v \in V\} \tag{17}$$

$$\{v \leftarrow a_1, D_i^*;\ \bar{v} \leftarrow a_1, D_i^* \mid v \in V,\ 1 \leq i \leq n\} \tag{18}$$

$$\{a_1 \leftarrow \overline{C}_j \mid 1 \leq j \leq m\} \tag{19}$$

$$\{a_1 \leftarrow v, \bar{v} \mid v \in V\}; \tag{20}$$

where a_1 is a fresh atom. Let $Y = \{a_1\} \cup V \cup \bar{V}$ and $A = \{a_1\}$. We show

that $(\emptyset, Y)$ is an A-SE-model of P iff (a) and (b) jointly hold. Since P is positive, we can argue via classical models (over Y). Rules (17) have classical models $\{X \mid \sigma(I) \subseteq X \subseteq Y, I \subseteq V\}$. By (18), this set splits into $S = \{X \mid \sigma(I) \subseteq X \subseteq (Y \setminus \{a_1\})\}$ and $T = \{\sigma(I) \cup \{a_1\} \mid I \notin M_\phi\} \cup \{Y\}$. By (20), S reduces to $\{\sigma(I) \mid I \subseteq V\}$, and by (19) only those elements $\sigma(I)$ survive with $I \in M_\psi$. To summarize, the models of P are given by

$$\{\sigma(I) \mid I \subseteq V, I \in M_\psi\} \cup \{\sigma(I) \cup \{a_1\} \mid I \subseteq V, I \notin M_\phi\} \cup \{Y\}.$$

From this the A-SE-models are easily obtained. We want to check whether $(\emptyset, Y) \in SE^A(P)$. We have that $Y \models P$. Further, we have that no $Y' \subset Y$ with $a_1 \in Y'$ exists such that $Y' \models P = P^Y$ iff there exists no $I \subseteq V$ making ϕ false, namely, iff ϕ is a tautology. Finally, to show that $(\emptyset, Y) \in SE^A(P)$, there has to exist an $X \subseteq (V \cup \bar{V})$ such that $X \models P = P^Y$. This holds exactly if ψ is satisfiable. Since P is always polynomial in the size of ϕ plus ψ, we derive D^P-hardness.

This shows the claim for $card(A) = 1$. For $card(A) > 1$, we apply a similar technique as in the proof of Theorem 6.11. However, since we deal here with both A-SE-models and A-UE-models, we have to be a bit more strict. Let $k = card(A) > 1$. We add to P the following rules $\{a_{i+1} \leftarrow a_i;\ a_i \leftarrow a_{i+1} \mid 1 \leq i < k\}$ and set $A = \{a_i \mid 1 \leq i \leq k\}$. One can show that then, for $Y = A \cup V \cup \bar{V}$, $(\emptyset, Y) \in SE^A(P)$ iff $(\emptyset, Y) \in UE^A(P)$ iff (a) and (b) jointly hold. □

With these results for model checking at-hand, we obtain numerous complexity results for deciding relativized equivalence.

THEOREM 6.13. *For programs P, Q, a set of atoms A, and $e \in \{s, u\}$, $P \equiv_e^A Q$ is in Π_2^P.*

PROOF. We guess an A-SE-interpretation (X, Y). Then, by virtue of Theorem 6.12, we can verify that (X, Y) is A-SE-model (respectively A-UE-model) of exactly one of the programs P, Q in polynomial time with four calls to an NP oracle (since the two model-checking tasks are in D^P). Hence, the complementary problem of deciding relativized equivalence is in Σ_2^P. This shows Π_2^P-membership. □

THEOREM 6.14. *Let P, Q be DLPs, A a set of atoms, and $e \in \{s, u\}$. Then, $P \equiv_e^A Q$ is Π_2^P-complete. Π_2^P-hardness holds even if Q is Horn.*

PROOF. Membership is already shown in Theorem 6.13.

For the hardness part, we reduce the Σ_2^P-complete problem of deciding the truth of a QBF $\exists X \forall Y \phi$ with $\phi = \bigvee_{i=1}^n D_i$ a DNF to the complementary problem of $P \not\equiv_s^A Q$. We define

$$\begin{aligned} P \ = \ & \{x \vee \bar{x} \leftarrow;\ \leftarrow x, \bar{x} \mid x \in X\} \cup \\ & \{y \vee \bar{y} \leftarrow;\ y \leftarrow a;\ \bar{y} \leftarrow a;\ a \leftarrow y, \bar{y} \mid y \in Y\} \cup \\ & \{a \leftarrow D_i^* \mid 1 \leq i \leq n\} \cup \\ & \{\leftarrow \mathit{not}\ a\}; \end{aligned}$$

and take $Q = \{\bot\}$. Note that $\{\bot\}$ has no A-SE-model for any A. It thus remains to show that P has an A-SE-model iff the QBF $\exists X \forall Y \phi$ is true.

P has an answer set (that is, an $\emptyset$-SE-model) iff $\exists X \forall Y \phi$ is true (see the Σ_2^P-hardness proof for the program consistency problem in Eiter and Gottlob [1995]). From this we get that ordinary equivalence is Π_2^P-hard. This shows the claim for $card(A) = 0$. For A of arbitrary cardinality k, it is sufficient to add "dummy" rules $a_i \leftarrow a_i$, for each $1 \leq i \leq k$, to P. These rules do not have any effect on our argument. Whence, for any fixed A, $\equiv_s^A$ and $\equiv_u^A$ are Π_2^P-hard as well. □

A slight modification (see the online Appendix for details) of this proof gives us the following result.

THEOREM 6.15. *Let P be a positive program, A a set of atoms, and $e \in \{s, u\}$. Then, deciding whether $P \equiv_e^A Q$ is Π_2^P-complete, where Π_2^P-hardness holds even if Q is either positive or normal.*

For head-cycle-free programs, RSE and RUE have different complexities. We first consider RSE.

THEOREM 6.16. *Let P and Q be head-cycle-free programs, and A be a set of atoms. Then, deciding whether $P \equiv_s^A Q$ is Π_2^P-complete, where Π_2^P-hardness holds even if Q is normal, and with fixed A with $card(A) \geq 2$.*

This concludes the collection of problems which are located at the second level of the polynomial hierarchy. Let us remark that in the hardness part of the proof of Theorem 6.16 (see the online Appendix for details), we used at least two elements in A. In fact, for HCF programs and $card(A) \leq 1$ the complexity is different. Since for $card(A) \leq 1$, $\equiv_s^A$ and $\equiv_u^A$ are the same concepts, this special case is implicitly considered in the next theorem. Another issue is to decide whether $P \equiv_s^A Q$ if both P and Q are A-HCF, as introduced in Definition 5.20. In this case, we can employ $P^{\rightarrow} \equiv_s^A Q^{\rightarrow}$, and thus the complexity coincides with the complexity of $\equiv_s^A$ for normal programs. This is also part of the next theorem.

THEOREM 6.17. *Deciding $P \equiv_e^A Q$ is* coNP*-complete in the following settings*:

(i) *$e \in \{s, u\}$, P positive, Q Horn;*
(ii) *$e = s$, P head-cycle-free and Q Horn;*
(iii) *$e \in \{s, u\}$, P and Q normal; and*
(iv) *$e = u$, P and Q head-cycle-free.*

The coNP*-hardness of $P \equiv_e^A Q$ ($e \in \{s, u\}$) holds, even if P is normal or positive and Q is Horn.*

PROOF. We start with the coNP-membership results. Cases (iii) and (iv) follow immediately from Theorem 6.10, since A-SE/UE-model checking for the programs involved is feasible in polynomial time. The more complicated cases (i) and (ii) are addressed in the online Appendix.

It remains to show the coNP-hardness part of the theorem. We use UNSAT of a formula $F = \bigwedge_{i=1}^n C_i$ in CNF over atoms X. Take

$$P = \{x \vee \bar{x} \leftarrow;\ \leftarrow x, \bar{x} \mid x \in X\} \cup \{\leftarrow \overline{C}_i \mid 1 \leq i \leq n\}.$$

Note that this program is positive and HCF. The program has a classical model iff F is satisfiable, that is, iff it is not equivalent to the Horn program $Q = \{\bot\}$. In other words, $SE^A(P) \neq \emptyset$ (or, respectively $UE^A(P) \neq \emptyset$) iff ϕ is satisfiable. Note that A can thus be of any form. Since the rules $\leftarrow x, \bar{x}$ are present in P, we have $P \equiv_s^A P^{\rightarrow}$. This proves coNP-hardness also for the case where one program is normal and the other is Horn. □

A final case remains open, namely, that of checking the relativized equivalence of Horn programs. Unfortunately, this task is coNP-complete. However, whenever the cardinality of A is fixed by a constant, the problem gets tractable. This is in contrast to the hardness results proved so far, which even hold in the case where $card(A)$ is fixed. The proof of the theorem is given in the online Appendix.

THEOREM 6.18. *Deciding $P \equiv_e^A Q$, for $e \in \{s, u\}$, is* coNP*-complete for Horn programs P, Q. Hardness holds whenever $card(A)$ is not fixed by a constant, and even for definite Horn programs.*

Whenever the cardinality of A is bounded, we can decide this problem in polynomial time.

THEOREM 6.19. *Let P, Q be Horn programs and A be a set of atoms such that $card(A) \leq k$ with a fixed constant k. Then, deciding $P \equiv_e^A Q$ is feasible in polynomial time with respect to $\|P\| + \|Q\| + k$.*

PROOF. It is sufficient to show the claim for $e = u$. By explicitly checking whether $(P \cup S) \equiv (Q \cup S)$ holds for any $S \subseteq A$, we obtain a polynomial-time algorithm, since checking the ordinary equivalence of Horn programs is polynomial and we need at most 2^k such checks. □

6.3 Complexity of Bounded Relativization

In this section, we pay attention to the special case of tests $\equiv_s^A$ and $\equiv_u^A$ where the number of atoms from the considered programs *missing* in A is bounded by some constant k (we sometimes abbreviate these tests by $P \equiv_s^{A\,k} Q$ and $P \equiv_u^{A\,k} Q$, respectively). Hence, the respective problem classes apply to programs P, Q, only if $card(Atm(P \cup Q) \setminus A) \leq k$. Apparently, this class of problems contains strong and uniform equivalence in its unrelativized versions ($k = 0$). The complexity results are summarized in Table VI. In particular, we get that in the case of RSE, all entries (except Horn/Horn) reduce to coNP-completeness. This generalizes results on strong equivalence. Previous work reported some of these results, but not in form of this exhaustive list.

In what follows, we first give the respective results for model checking, and then we prove the entries in Table VI.

LEMMA 6.20. *For a program P and a set of atoms A such that $card(Atm(P) \setminus A) \leq k$, with k a fixed constant, A-SE-model checking the is feasible in polynomial time with respect to $\|P\| + k$.*

PROOF. By the conditions in Definition 4.4, deciding $(X, Y) \in SE^A(P)$ can be done as follows: (i) checking $Y \models P$; (ii) checking whether for all $Y' \subset Y$ with

Table VI. Complexity of Equivalences with Bounded Relativization in Terms of Completeness Results

$P \equiv_s^{A,k} Q$ / $P \equiv_u^{A,k} Q$	DLP	Positive	HCF	Normal	Horn
Horn	coNP	coNP	coNP	coNP	P
normal	coNP	coNP	coNP	coNP	
HCF	coNP	coNP	coNP		
positive	coNP/Π_2^P	coNP			
DLP	coNP/Π_2^P				

$(Y' \cap A) = (Y \cap A)$, $Y \not\models P^Y$ holds; and (iii) if $X \subset Y$, checking the existence of a $X' \subseteq Y$ with $(X' \cap A) = X$ such that $X' \models P^Y$ holds. Test (i) can be done in polynomial time; test (ii) is a conjunction of at most $2^k - 1$ independent polynomial tests (for each such Y'), while (iii) is a disjunction of at most 2^k polynomial tests (for each X'). Since we have fixed k, the entire test is feasible in polynomial time. □

Compared to the model checking problems discussed so far, the polynomial-time decidable problems of A-SE-model checking in the bounded case do not belong to the class of P-complete problems, but are easier. This is best illustrated by SE-model checking, which obviously reduces to two (ordinary) independent model-checking tests which, in turn, are in ALOGTIME [Buss 1987] (see also Barrington et al. [1990] and Immerman [1999]). For bounded A-SE-model checking the situation is basically the same, since it is sufficient to employ a fixed number of independent model-checking tests.

Concerning UE-model checking, we already established some P-hardness results in Theorem 6.2 which generalize to the relativized case for arbitrary bound A. In general, for A-UE-model checking the decrease of complexity is in certain cases only moderate compared to the corresponding decrease in the case of A-SE-model checking.

LEMMA 6.21. *For a program P and a set of atoms A such that $card(Atm(P) \setminus A) \leq k$, with k a fixed constant, A-UE-model checking is* coNP*-complete. Hardness holds even for positive programs.*

PROOF. We show NP-membership for the complementary problem, that is, for checking whether a given pair (X, Y) is not in $UE^A(P)$. We first check whether (X, Y) is an A-SE-model of P. This can be done in polynomial time, by Lemma 6.20. If this is not the case, we are done; otherwise, we guess an X' with $X \subset X' \subset (Y \cap A)$ and check whether (X', Y) is an A-SE-model of P. This guess for (X', Y) can be verified in polynomial time using an NP oracle. Therefore, the entire problem is in NP. The correctness of the procedure is given by its direct reflection of Definition 4.11. This yields coNP-membership for bounded A-UE-model checking.

Hardness is obtained via the case $card(Atm(P) \setminus A) = 0$, that is, ordinary UE-model checking and the respective result in Theorem 6.1. □

THEOREM 6.22. *For programs P, Q and a set of atoms A such that $card(Atm(P \cup Q) \setminus A) \leq k$ with k a fixed constant, $P \equiv_s^A Q$ is* coNP*-complete. Hardness holds, provided that P and Q are not Horn.*

PROOF. By Lemma 6.20, A-SE-model checking is feasible in polynomial time in the bounded case. Hence, coNP-membership for $P \equiv_s^A Q$ is an immediate consequence. The hardness result is easily obtained by the hardness part from Theorem 6.17. □

For RUE, some cases remain on the second level, however. This is not a surprise, since as we have seen in Theorem 6.4, (unrelativized) uniform equivalence is Π_2^P-complete in general.

THEOREM 6.23. *For programs P, Q and a set of atoms A such that $card(Atm(P \cup Q) \setminus A) \leq k$ with k a fixed constant, $P \equiv_u^A Q$ is Π_2^P-complete. Π_2^P-hardness holds even if one of the programs is positive.*

PROOF. Membership is obtained by the fact that A-UE-model checking with A bounded is coNP-complete (see Lemma 6.21). Hardness comes from the Π_2^P-hardness of uniform equivalence. □

For all other cases, RUE for bounded A is in coNP. The proofs of the final two theorems are relegated to the online Appendix.

THEOREM 6.24. *For programs P, Q and a set of atoms A such that $card(Atm(P \cup Q) \setminus A) \leq k$ with k a fixed constant, $P \equiv_u^A Q$ is* coNP*-complete if either:* (i) *both programs are positive; or* (ii) *at least one program is head-cycle-free. Hardness holds even if P is normal or positive, and Q is Horn.*

One final case remains to be considered.

THEOREM 6.25. *Let P and Q be Horn programs and let A be a set of atoms such that $card(Atm(P \cup Q) \setminus A) \leq k$ with a fixed constant k. Then, deciding $P \equiv_e^A Q$ is feasible in polynomial time with respect to $\|P\| + \|Q\| + k$.*

7. LANGUAGE VARIATIONS

In this section, we briefly address how our results apply to variations of the language of logic programs. First, we consider modifications within the case of propositional programs, and then discuss the general DATALOG case.

7.1 Extensions in the Propositional Case

Adding Classical Negation. Our results easily carry over to extended logic programs, that is, programs where classical (also called strong) negation is allowed as well. If the inconsistent answer set is disregarded, that is, an inconsistent program has no models, then, as usual, the extension can be semantically captured by representing strongly negated atoms $\neg A$ by a positive atom A' and adding the constraints $\leftarrow A, A'$, for every atom A, to any program.

However, if in the extended setting the inconsistent answer set is taken into account, then the given definitions have to be slightly modified such that the characterizations of uniform equivalence capture the extended case properly. The same holds true for the characterization of strong equivalence by SE-models, as illustrated by the following example. Note that the redefinition of $\equiv_u$ and $\equiv_s$ is straightforward.

Let $Lit_{\mathcal{A}} = \{A, \neg A \mid A \in \mathcal{A}\}$ denote the (inconsistent) set of all literals using strong negation over $\mathcal{A}$. Note that an extended DLP P has an inconsistent answer set iff $Lit_{\mathcal{A}}$ is an answer set of it; moreover, it is in the latter case the only answer set of P. Call any DLP P *contradiction-free* if $Lit_{\mathcal{A}}$ is not an answer set of it, and *contradictory* otherwise.

Example 7.1. Consider the extended logic programs $P = \{a \vee b \leftarrow ;\ \neg a \leftarrow a;\ \neg b \leftarrow b\}$ and $Q = \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ \neg a \leftarrow a;\ \neg b \leftarrow b\}$. They both have no SE-model; hence, by the criterion of Propostion 2.3, $P \equiv_s Q$ would hold, which implies that $P \equiv_u Q$ and $P \equiv Q$. However, P has the inconsistent answer set $Lit_{\mathcal{A}}$, while Q has no answer set. Thus formally, P and Q are not equivalent, even if $Lit_{\mathcal{A}}$ is admitted as an answer set.

Since Turner [2003, 2001] and Lifschitz et al. [2001] made no distinction between no answer set and an inconsistent answer set, in Eiter and Fink [2003b] we adapted the definition of SE-models accordingly and got more general characterizations in terms of so-called SEE-models for extended programs. Many results easily carry over to the extended case: For example, for positive programs, uniform and strong equivalence coincide also in this case and, as a consequence of previous complexity results, checking $P \equiv_u Q$ (respectively $P \equiv_s Q$) for extended logic programs P and Q is Π_2^P-hard (respectively coNP-hard).

However, not all properties do carry over. As Example 7.1 reveals, in generally a head-cycle-free extended DLP P is no longer equivalent, hence not uniformly equivalent, to its shift variant $P^{\leftarrow}$ (see Eiter and Fink [2003b] for a characterization of head-cycle-and contradiction-free programs for which this equivalence holds).

We expect a similar picture for the relativized equivalences of extended logic programs, but adapting corresponding proofs is still the subject of future work.

Disallowing Constraints. Sometimes, it is desirable to consider constraints just as abbreviations so as to have core programs which are definite, that is, without constraints. The most direct approach is to replace each constraint $\leftarrow B$ by $w \leftarrow B, not\ w$, where w is a designated atom not occurring in the original program. Obviously, this does not influence ordinary equivalence tests, but for notions such as uniform and strong equivalence some more care is required. Take the strongly equivalent programs $P = \{a \leftarrow not\ a\}$ and $Q = \{\leftarrow not\ a\}$. By the preceding rewriting Q becomes $Q' = \{w \leftarrow not\ a, not\ w\}$. Then, $(\cdot, w) \notin SE(P)$ but $(\cdot, w) \in SE(Q')$. Hence, this rewriting is not sensitive under strong equivalence. However, if we disallow w to appear in possible extensions, namely, employing $\equiv_s^A$ instead of $\equiv_s$, we can circumvent this problem. Simply take $A = U \setminus \{w\}$, where U is the universe of atoms. Observe that this employs bounded relativization, and in light of Theorem 6.22, this strategy does not result in a more complex problem. For uniform equivalence, the methodology can be applied in the same manner.

However, this approach requires (unstratified) negation. If we want to get rid of constraints for comparing positive programs, an alternative method is to use a designated (spoiled) answer set to indicate that the original program had no answer set. The idea is to replace each constraint $\leftarrow B$ by $w \leftarrow B$ where

w is a designated atom as described earlier; additionally, we add the collection of rules $v \leftarrow w$ for each atom v of the universe to both programs (even if no constraint is present). This rewriting retains any equivalence notion, even if w is allowed to occur in the extensions.

The problem of comparing, say, a positive program P (with constraints) and a normal program Q is more subtle if we require to replace the constraints in P by the positive rules themselves. We leave this for further study, but refer to some results in Eiter et al. [2004b], which suggest that these settings may not be solved in an easy manner. To wit, Eiter et al. [2004b] report that the complexity for some problems of the form "Given a program P from class C; does there exist a program Q from class C' such that $P \equiv_e Q$?" differs with respect to allowing constraints.

Using Nested Expressions. Programs with nested expressions [Lifschitz et al. 1999] (also called nested logic programs) extend DLPs in such a way that arbitrarily nested formulas, formed from literals using negation as failure, conjunction, and disjunction, constitute the heads and bodies of rules. Our characterizations for uniform equivalence are well-suited for this class. This is as follows from Pearce and Valverde [2004b], which extends results from Eiter and Fink [2003a] to propositional theories, that is, to equilibrium logic. Since the proofs of our main results are generic in the use of reducts, we expect that all results (including relativized notions of equivalence) can be carried over to nested logic programs without any problems. Note, however, that the concrete definitions for subclasses (positive, normal, etc.) have to be extended in the context of nested logic programs (see Linke et al. [2004] for such an extension of head-cycle-free programs). It remains for further work to apply our results to such classes.

7.2 DATALOG Programs

The results in the previous sections on propositional logic programs provide an extensive basis for studying the equivalences of DATALOG programs if, as usual, their semantics is given in terms of propositional programs. Basic notions and concepts for strong and uniform equivalence, such as SE-models, UE-models, and the respective notions of consequence, generalize naturally to this setting using Herbrand interpretations over a relational alphabet and a set of constants in the usual way (Eiter et al. [2005]). Furthermore, fundamental results can be applied to DATALOG programs by reduction to the propositional case. In particular, the elementary characterizations $P \equiv_e Q$ iff $M_e(P) = M_e(Q)$ iff $P \models_e Q$ and $Q \models_e P$ carry over to the DATALOG setting for $e \in \{s, u\}$ and $M_s(\cdot) = SE(\cdot)$, respectively, $M_u(\cdot) = UE(\cdot)$ (see also Eiter et al. [2005]). However, a detailed analysis of the DATALOG case including relativized notions of equivalence is a subject of ongoing work.

Nevertheless, let us conclude this section with some remarks on the complexity of programs with variables. For such programs, in the case of a given *finite* Herbrand universe the complexity of equivalence checking (respectively model checking) increases by an exponential. Intuitively, this is explained by the exponential size of a Herbrand interpretation, namely, the ground instance of a

program over the universe. Note that Lin [2002] reported (without proof) that checking strong equivalence for programs in this setting is in coNP, and thus would have the same complexity as in the propositional case. However, coNP-membership holds only in case of a constant upper bound on the number of variables occurring in the rules of the program, but checking strong equivalence is exponentially harder (i.e., co-NEXPTIME-complete) in the general case. Unsurprisingly, over *infinite* domains, in light of the results in Shmueli [1993] and Halevy et al. [2001], the decidability of equivalence and inference problems for DATALOG programs is no longer guaranteed. While strong equivalence and SE-inference remain decidable (more precisely, complete for co-NEXPTIME), this is not the case for uniform equivalence (respectively inference) in general. For positive programs, however, the two notions coincide and are decidable (more precisely, complete for co-NEXPTIME); see Eiter et al. [2005] for details. It remains as an issue for future work to explore the decidability versus undecidability frontier for classes of DATALOG programs, possibly under restrictions as in Halevy et al. [2001] and Chaudhuri and Vardi [1992].

8. CONCLUSION AND FURTHER WORK

In this article, we have extended the research about equivalence of nonmonotonic logic programs under answer set semantics in order to simplify parts (or modules) of a program without analyzing the entire program. Such local simplifications call for alternative notions of equivalence, since a simple comparison of the answer sets does not provide information as to whether a program part can be replaced by its simplification. To wit, by the nonmonotonicity of the answer set semantics, two (ordinary) equivalent (parts of) programs may lead to different answer sets if they are used in the same global program R. Alternative notions of equivalence thus require that the answer sets of the two programs coincide under different R: strong equivalence [Lifschitz et al. 2001], for instance, requires that the compared programs are equivalent under any extension R.

In this article, we have considered further notions of equivalence in which the actual form of R is syntactically constrained:

—*We have addressed uniform equivalence of logic programs, which had been considered earlier for DATALOG and general Horn logic programs [Sagiv 1988; Maher 1988].* Under answer set semantics, the uniform equivalence can be exploited for the optimization of components in a logic program which is modularly structured.

—*Relativized notions of both uniform and strong equivalence restrict the alphabet of the extensions.* This allows to specify which atoms may occur in the extensions and which not. This notion of equivalence for answer set semantics was originally suggested by Lin [2002] but not further investigated. In practice, relativization is a natural concept, since it allows to specify internal atoms which only occur in the compared program parts, but guarantees that they do not occur anywhere else.

We have provided semantical characterizations of all these notions of equivalence by adopting the concept of SE-models [Turner 2001] (equivalently,

HT-models [Lifschitz et al. 2001]), which capture the essence of a program with respect to strong equivalence. Furthermore, we have thoroughly analyzed the complexity of equivalence checking and related problems for the general case and several important fragments. This collection of results gives a valuable theoretical underpinning for advanced methods of program optimization and for enhanced ASP application development, as well as a potential basis for the development of ASP debugging tools.

Several issues remain for further work. One issue is a characterization of uniform equivalence in terms of "models" for arbitrary programs in the infinite case; as we have shown, no subset of SE-models serves this purpose. In particular, a notion of models corresponding to UE-models in the case where the latter capture uniform equivalence would be interesting.

We focused here on the propositional case, to which general programs with variables reduce, and briefly mentioned a possible extension to a DATALOG setting [Eiter et al. 2005]. Here, the undecidability of uniform equivalence arises if negation may be present in programs. A thorough study of cases under which uniform and other notions of equivalence are decidable is needed, along with complexity characterizations. Given that in addition to the syntactic conditions on propositional programs considered here, further conditions involving predicates might be taken into account (compare with Chaudhuri and Vardi [1992] and Halevy et al. [2001]), quite a number of different cases remains to be analyzed.

Finally, an important issue is to explore the usage of uniform equivalence and relativized equivalence in program replacement and rewriting, and to develop optimization methods and tools for answer set programming. A first step in this direction, picking up some of the results of this article, has been made in Eiter et al. [2004b]. However, much more remains to be done.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

The authors would like to thank David Pearce for interesting discussions and comments about this work and pointers to related literature, as well as Katsumi Inoue and Chiaki Sakama for their valuable remarks on relativizing equivalence. We are also grateful to the anonymous reviewers of *ICLP* 2003 and *JELIA* 2004 for their comments on submissions of extended abstracts preliminary to this article, as well as to the anonymous reviewers of this article for their valuable suggestions for improvements.

REFERENCES

ALFERES, J. J., LEITE, J. A., PEREIRA, L. M., PRZYMUSINSKA, H., AND PRZYMUSINSKI, T. C. 2000. Dynamic updates of non-monotonic knowledge bases. *J. Logic Program. 45*, 1-3, 43–70.

ANGER, C., KONCZAK, K., AND LINKE, T. 2001. NoMoRe: A system for non-monotonic reasoning. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, T. Eiter et al., Eds. vol. 2173. Springer Verlag, 406–410.

BARRINGTON, D., IMMERMAN, N., AND STRAUBING, H. 1990. On uniformity within NC^1. *J. Comput. Syst. Sci. 41*, 274–306.

BEN-ELIYAHU, R. AND DECHTER, R. 1994. Propositional semantics for disjunctive logic programs. *Annals Math. Artif. Intell. 12*, 53–87.

BUNTINE, W. 1988. Generalised subsumption and its applications to induction and redundancy. *Artif. Intell. 36*, 2, 149–176.

BUSS, S. 1987. The Boolean formula value problem is in ALOGTIME. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*. 123–131.

CABALAR, P. 2002. A three-valued characterization for strong equivalence of logic programs. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*. AAAI Press/MIT Press, Cambridge, MA, 106–111.

CADOLI, M. AND LENZERINI, M. 1994. The complexity of propositional closed world reasoning and circumscription. *J. Comput. Syst. Sci. 48*, 2, 255–310.

CHAUDHURI, S. AND VARDI, M. Y. 1992. On the equivalence of recursive and nonrecursive datalog programs. In *Proceedings of the 11th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, 55–66.

COSMADAKIS, S. AND KANELLAKIS, P. 1986. Parallel evaluation of recursive rule queries. In *Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*. ACM Press, New York, 280–293.

DE JONGH, D. AND HENDRIKS, L. 2003. Characterizations of strongly equivalent logic programs in intermediate logics. *Theory Pract. Logic Program. 3*, 3, 259–270.

DIMOPOULOS, Y., NEBEL, B., AND KOEHLER, J. 1997. Encoding planning problems in nonmonotonic logic programs. In *Proceedings of the European Conference on Planning (ECP)*, S. Steel and R. Alami, Eds. Lecture Notes in Computer Science, vol. 1348. Springer Verlag, 169–181.

EITER, T., FABER, W., LEONE, N., PFEIFER, G., AND POLLERES, A. 2003. A logic programming approach to knowledge-state planning, II: The DLV$^{\mathcal{K}}$ system. *Artif. Intell. 144*, 1-2, 157–211.

EITER, T., FABER, W., LEONE, N., PFEIFER, G., AND POLLERES, A. 2004. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Trans. Comput. Logic 5*, 2, 206–263.

EITER, T. AND FINK, M. 2003a. Uniform equivalence of logic programs under the stable model semantics. In *Proceedings of the 19th International Conference on Logic Programming (ICLP)*, C. Palamidessi, Ed. Lecture Notes in Computer Science, vol. 2916. Springer Verlag, 224–238.

EITER, T. AND FINK, M. 2003b. Uniform equivalence of logic programs under the stable model semantics. Tech. Rep. INFSYS RR-1843-03-08, Institut für Informationssysteme, Technische Universität Wien, Austria.

EITER, T., FINK, M., SABBATINI, G., AND TOMPITS, H. 2001. A framework for declarative update specifications in logic programs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, B. Nebel, Ed. Morgan Kaufmann, San Fransisco, CA; 649–654.

EITER, T., FINK, M., TOMPITS, H., AND WOLTRAN, S. 2004a. On eliminating disjunctions in stable logic programming. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, D. Dubois et al., Eds. AAAI Press, 447–458.

EITER, T., FINK, M., TOMPITS, H., AND WOLTRAN, S. 2004b. Simplifying logic programs under uniform and strong equivalence. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, V. Lifschitz and I. Niemelä, Eds. Lecture Notes in Computer Science, vol. 2923. Springer Verlag, 87–99.

EITER, T., FINK, M., TOMPITS, H., AND WOLTRAN, S. 2005. Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. Accepted for publication in the *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*.

EITER, T. AND GOTTLOB, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Annals Math. Artif. Intell. 15*, 3/4, 289–323.

EITER, T., GOTTLOB, G., AND MANNILA, H. 1997. Disjunctive datalog. *ACM Trans. Database Syst. 22*, 3, 364–418.

ERDEM, E., LIFSCHITZ, V., NAKHLEH, L., AND RINGE, D. 2003. Reconstructing the evolutionary history of Indo-European languages using answer set programming. In *Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages (PADL)*, V. Dahl and P. Wadler, Eds. Lecture Notes in Computer Science, vol. 2562. Springer Verlag, 160–176.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming (ICLP)*, R. Kowalski and K. Bowen, Eds. MIT Press, Cambridge, MA, 1070–1080.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Gen. Comput. 9*, 365–385.

GIORGINI, P., MASSACCI, F., MYLOPOULOS, J., AND ZANNONE, N. 2004. Requirements engineering meets trust management: Model, methodology, and reasoning. In *Proceedings of the 2nd International Conference on Trust Management (iTrust)*. Lecture Notes in Computer Science, vol. 2995. Springer Verlag, 176–190.

HALEVY, A. Y., MUMICK, I. S., SAGIV, Y., AND SHMUELI, O. 2001. Static analysis in datalog extensions. *J. ACM 48*, 5, 971–1012.

HELJANKO, K. AND NIEMELÄ, I. 2001. Bounded LTL model checking with stable models. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, T. Eiter et al., Eds. Lecture Notes in Computer Science, vol. 2173. Springer Verlag, 200–212.

IMMERMAN, N. 1999. *Descriptive Complexity*. Springer Verlag.

INOUE, K. AND SAKAMA, C. 1999. Updating extended logic programs through abduction. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, M. Gelfond et al., Eds. Lecture Notes in Computer Science, vol. 1730. Springer Verlag, 147–161.

INOUE, K. AND SAKAMA, C. 2004. Equivalence of logic programs under updates. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA)*, J. J. Alferes and J. A. Leite, Eds. Lecture Notes in Computer Science, vol. 3229. Springer Verlag, 174–186.

JANHUNEN, T. AND OIKARINEN, E. 2004. LPEQ and DLPEQ-Translators for automated equivalence testing of logic programs. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, V. Lifschitz and I. Niemelä, Eds. Lecture Notes in Computer Science, vol. 2923. Springer Verlag, 336–340.

KAUTZ, H. AND SELMAN, B. 1992. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*, B. Neumann, Ed. Wiley, 359–363.

KOWALSKI, V. 1968. The calculus of the weak "Law of Excluded Middle". *Math. USSR 8*, 648–658.

LEONE, N., PFEIFER, G., FABER, W., EITER, T., GOTTLOB, G., PERRI, S., AND SCARCELLO, F. 2002. The DLV system for knowledge representation and reasoning. Tech. Rep. cs.AI/0211004, arXiv.org. November. To appear in *ACM Trans. Comput. Logic*.

LIFSCHITZ, V. 1999. Action languages, answer sets and planning. In *The Logic Programming Paradigm – A 25-Year Perspective*, K. Apt et al., Eds. Springer Verlag, 357–373.

LIFSCHITZ, V., PEARCE, D., AND VALVERDE, A. 2001. Strongly equivalent logic programs. *ACM Trans. Comput. Logic 2*, 4, 526–541.

LIFSCHITZ, V., TANG, L., AND TURNER, H. 1999. Nested expressions in logic programs. *Annals Math. Artif. Intell. 25*, 3-4, 369–389.

LIFSCHITZ, V. AND TURNER, H. 1994. Splitting a logic program. In *Proceedings of the 11th International Conference on Logic Programming (ICLP)*, P. Van Hentenryck, Ed. MIT-Press, Cambridge, MA, 23–38.

LIN, F. 2002. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, D. Fensel et al., Eds. Morgan Kaufmann, San Fransisco, CA, 170–176.

LIN, F. AND ZHAO, Y. 2002. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*. AAAI Press / MIT Press, Cambridge, MA, 112–117.

LINKE, T., TOMPITS, H., AND WOLTRAN, S. 2004. On acyclic and head-cycle free nested logic programs. In *Proceedings of the 20th International Conference on Logic Programming (ICLP)*, B. Demoen and V. Lifschitz, Eds. Lecture Notes in Computer Science, vol. 3132. Springer Verlag, 225–239.

MAHER, M. J. 1988. Equivalences of logic programs. In *Foundations of Deductive Databases and Logic Programming*, J. Minker. Morgan, Kauffman, San Fransico, CA, 627–658.

MINKER, J., 1988. *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, San Fransico, CA.

OSORIO, M., NAVARRO, J., AND ARRAZOLA, J. 2001. Equivalence in answer set programming. In *Proceedings of the 11th International Workshop on Logic Based Program Synthesis and Transformation (LOPSTR)*, A. Pettorossi, Ed. Lecture Notes in Computer Science, vol. 2372. Springer Verlag, 57–75.

PEARCE, D. 2004. Simplifying logic programs under answer set semantics. In *Proceedings of the 20th International Conference on Logic Programming (ICLP)*, B. Demoen and V. Lifschitz, Eds. Lecture Notes in Computer Science, vol. 3132. Springer Verlag.

PEARCE, D., TOMPITS, H., AND WOLTRAN., S. 2001. Encodings for equilibrium logic and logic programs with nested expressions. In *Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA)*, P. Brazdil and A. Jorge, Eds. Lecture Notes in Computer Science, vol. 2258. Springer Verlag, 306–320.

PEARCE, D. AND VALVERDE, A. 2004a. Synonymous theories in answer set programming and equilibrium logic. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, R. L. de Mántaras and L. Saitta, Eds. IOS Press, Amsterdam, the Netherland, 388–392.

PEARCE, D. AND VALVERDE, A. 2004b. Uniform equivalence for equilibrium logic and logic programs. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, V. Lifschitz and I. Niemelä, Eds. Lecture Notes in Computer Science, vol. 2923. Springer Verlag, 194–206.

PROVETTI, A. AND SON, T. C. 2001. *Proceedings of the Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*. AAAI Press.

PRZYMUSINSKI, T. 1991. Stable semantics for disjunctive programs. *New Gen. Comput. 9*, 401–424.

SAGIV, Y. 1988. Optimizing datalog programs. In *Foundations of Deductive Databases and Logic Programming*, J. Minker. Morgan, Kauffman, San Fransico, CA, 659–698.

SHMUELI, O. 1993. Equivalence of datalog queries is undecidable. *J. Logic Program. 15*, 3, 231–242.

SIMONS, P., NIEMELÄ, I., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artif. Intell. 138*, 1-2, 181–234.

SUBRAHMANIAN, V. S. AND ZANIOLO, C. 1995. Relating stable models and AI planning domains. In *Proceedings of the 12th International Conference on Logic Programming (ICLP)*, L. Sterling, Ed. MIT Press, Cambridge, MA, 233–247.

TURNER, H. 2003. Strong equivalence made easy: Nested expressions and weight constraints. *Theory Pract. Logic Programm. 3*, 4-5, 602–622.

TURNER, H. 2001. Strong equivalence for logic programs and default theories (made easy). In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, T. Eiter et al., Eds. Lecture Notes in Computer Science, vol. 2173. Springer Verlag, 81–92.

WOLTRAN, S. 2004. Characterizations for relativized notions of equivalence in answer set programming. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA)*, J. J. Alferes and J. A. Leite, Eds. Lecture Notes in Computer Science, vol. 3229. Springer Verlag, 161–173.

ZHANG, Y. AND FOO, N. Y. 1998. Updating logic programs. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI)*, H. Prade, Ed. Wiley, 403–407.

Received February 2005; revised July 2005; accepted July 2005