Project Number:	IST-2001-33570
Project Acronym:	INFOMIX
Project Full Name:	Infomix: Boosting the Information Integration
Deliverable Number:	D5.3
Title:	**Methods and Techniques for Query Optimization**
Workpackage:	WP6
Document Type:	Deliverable
Distribution:	Infomix Consortium
Status:	Final
Document file:	WP5 T3 D5.3 0100.pdf
Version:	1.5
Number of pages:	84
Due Date:	March 31, 2004
Delivery Date:	June 12, 2004
Partners contributed:	TUWIEN,UNICAL,UNIROME1
Partners owning:	TUWIEN
Man Months:	7

Short Description:

In this document we address the important aspect of scalability. We first analyze the INFOMIX architecture in order to identify the points where optimizations are applicable. We then give an overview of methods which can be employed for this task. Some of the methods are subsequently discussed in detail. In particular, we deal with equivalences of Datalog programs in full depth, we present focussing techniques for repair programs, efficient recombination of split repairs, and query plan optimizations. Also relevant in this respect are magic set optimizations, which have been dealt with in Deliverable D4.3 in detail. Yet other optimization techniques can be taken over from traditional database research with minor efforts, so we also do not go into detail for such approaches. For some of the techniques, which are analysed in detail in this deliverable, we also provide an experimental study by means of a prototype implementation, which back the theoretical results empirically.

Document Change Record		
Version	**Date**	**Reason for Change**
v.1.0	December 21, 2003	First draft
v.1.1	March 5, 2004	Refined draft
v.1.2	March 15, 2004	Incorporate Focussing and Recombination
v.1.3	March 17, 2004	Incorporate Propositional Equivalences
v.1.4	March 25, 2004	Introduction and Overview
v.1.5	March 28, 2004	Major Streamlining
v.1.6	March 30, 2004	Add Nonpropositional Equivalences
v.1.7	April 1, 2004	Document Completion
v.1.7	April 9, 2004	Checked Document
v.1.8	June 3, 2004	Update (new results)

Contents

1 Introduction

The INFOMIX project is aimed at exploiting advanced reasoning capabilities in order to provide a set of techniques and tools for next-generation information integration. To this end, nonmonotonic logic programs are used to resolve conflicts and inconsistencies in the data encountered in the integration process, which originate from the autonomy of the single sources, and to compute the query answer. The formal framework of data integration in INFOMIX has been described in previous deliverables, namely D2.1 (Functional Specification of the INFOMIX System), and D3.1 (Formalisms for the Information Integration Model (IIM) Specification), and query answering and rewriting to logic programs in D5.1 (Methods and Techniques for Query Answering) and D5.2 (Methods and Techniques for Query Rewriting), respectively.

Objectives. The purpose of this document is to present and discuss techniques and methods for optimizing the performance of the INFOMIX data integration prototype to be built, and to assess their effectiveness with respect to scalability.

Query optimization techniques are very important, because performance and scalability of a data integration approach to large amounts of data are two main criteria for its practical relevance. It appears that a range of possibilities for query optimization exist, given the model of query answering which has been developed in the previous deliverables: Roughly speaking, a user query q formulated in the INFOMIX Query Language (IQL, described in Deliverable D3.2) against the formal description of a (fixed) data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, comprising a global schema $\mathcal{G}$, source schema $\mathcal{S}$, and a mapping $\mathcal{M}$ between them, over a database $\mathcal{D}$ for the sources in $\mathcal{S}$, is mapped into a logic program $\Pi_q(\mathcal{D})$, which comprises a set of rules (involving negation as failure) and a set of database facts, and this program is evaluated under the answer set semantics of non-monotonic logic programs (in the cautious reasoning mode) in order to determine the query answer (see Deliverables D5.1, D5.2). Both static query optimization, by reasoning over queries and integrity constraints, as well as the intensional part of the query program, and dynamic optimization, combining query optimization with query execution, can be envisaged, where well-known concepts and techniques from query optimization in distributed databases might be used in order to increase the performance of access to the sources.

Given this range of possibilities, this report aims at focussing on query optimization techniques which are relevant to the particular approach of INFOMIX in which logic programs are used as a computational back-end for data integration, and addresses other optimization methods to a lesser extent. In particular, the combination of logic programs and standard relational database technology in order to increase performance and scalability is an important issue here.

The remainder of the document is structured as follows. The next section contains some preliminaries, recalling the query evaluation in INFOMIX and introducing some basic concepts and notation for logic programs and databases. Section 3 gives an overview of different types of optimization methods that can be applied with respect the logic program $\Pi_q(\mathcal{D})$, and briefly discusses how they can be exploited for INFOMIX. After that, in Section 4 optimization of logic programs is considered, in particular with respect to the intensional part.

Section 5 addresses focussing in query answering using logic programs, and presents genuine techniques for pruning the logic program into which query answering is compiled. This part is most relevant for the concerns of keeping the set of program rules and the set of facts on which they have to be evaluated small. To this end, a decomposition of query evaluation is devised into an intersection of evaluations.

In Section 6, we consider the issue of recombining the pieces of the decomposition efficiently, by exploiting standard database technology. The intent there is to make use of the highly-optimized query evaluation engines of available database systems.

Section 7 addresses then briefly the issue of generating query plans for accessing the source databases.

Most of the material in this paper has been disseminated in publications and presentations at workshops and conferences already [19, 20, 25, 22, 24, 23, 16, 17, 18, 21, 10].

Relations to other documents. This report is directly related to the documents D3.2 (Query Language), D5.2 (Methods and Techniques For Query Rewriting), D4.2 (Algorithms and Implementation Techniques for the Internal Integration Formalism) and D4.3 (Optimization Techniques for the Internal Integration Formalism).

It is furthermore relevant for the implementation of the INFOMIX prototype, in particular for the Information Integration Management Layer prototype (Deliverable 7.1). We note here that not necessarily all of the optimizations described in this document have to be implemented in the INFOMIX prototype.

Intended audience and usage guidelines. This report is accessible to the public.

2 Preliminaries

In this section, we give an overview of query evaluation within the INFOMIX system and provide a brief review of relevant formalisms, which will be used later on in this deliverable.

2.1 Datalog

A *Datalog*$^{\vee,\neg}$ *rule* r is a clause $a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, \neg b_{k+1}, \ldots, \neg b_{k+m}$ where $n > 0$, $k, m \geq 0$ and $a_1, \ldots, a_n, b_1, \ldots, b_{k+m}$ are function-free atoms. The *head* of r is the set $H(r) = \{a_1, \ldots, a_n\}$, and the *body* of r is $B(r) = \{b_1, \ldots, b_k, \neg b_{k+1}, \ldots, \neg b_{k+m}\}$. We also define $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_{k+m}\}$. Moreover, for a set of atoms $A = \{a_1, \ldots, a_l\}$, $\neg A$ denotes the set $\{\neg a_1, \ldots, \neg a_l\}$. If $n = 1$ then r is a *Datalog*$^{\neg}$ *rule* (or a *normal rule*), and if also $k + m = 0$ the rule is called *fact*. If $m = 0$ and $n = 1$, then r is a *plain Datalog* (or simply, a Datalog) *rule*. A *Datalog*$^{\vee,\neg}$ *program* $\mathcal{P}$ is a finite set of Datalog$^{\vee,\neg}$ rules; it is *plain* (or simply, a Datalog program), if all its rules are plain. A Datalog$^{\neg}$ program with stratified negation is denoted by *Datalog*$^{\neg_s}$.

The model theoretic semantics assigns to any program $\mathcal{P}$ the set of its (disjunctive) *stable models* [34, 63]. A stable model of a program $\mathcal{P}$ is a set $X \subseteq B_{\mathcal{P}}$ such that X is a minimal model of $ground(\mathcal{P}, HU_{\mathcal{P}})^X$, where $ground(\mathcal{P}, U)$ denotes the grounding of $\mathcal{P}$ on the set of constants U, $HU_{\mathcal{P}}$ is the Herbrand universe of program $\mathcal{P}$, and $B_{\mathcal{P}}$ is the set of all ground literals constructible from the predicate symbols appearing in $\mathcal{P}$ and the constants of $HU_{\mathcal{P}}$. For short, $ground(\mathcal{P})$ denotes $ground(\mathcal{P}, HU_{\mathcal{P}})$.

The set of all stable models for a program $\mathcal{P}$ is denoted by $\mathrm{SM}(\mathcal{P})$. For positive $\mathcal{P}$, stable model and minimal model semantics coincide, i.e. $\mathrm{SM}(\mathcal{P}) = \mathrm{MM}(\mathcal{P})$. It is well-known that $|\mathrm{SM}(\mathcal{P})| = 1$ for any plain and Datalog$^{\neg_s}$ program $\mathcal{P}$. This definition applies to (finite) Datalog$^{\vee,\neg}$ programs and generalizes, without any change, to infinite programs (which consist of an infinite set of rules) as well.

Given a set of facts $\mathcal{D}$, the program $\mathcal{P}$ on input $\mathcal{D}$, denoted by $\mathcal{P}[\mathcal{D}]$, is the union $\mathcal{P} \cup \mathcal{D}$. A ground atom g is true w.r.t. $\mathcal{P}[\mathcal{D}]$, under *brave* semantics, if there exists a stable model M of $\mathcal{P}[\mathcal{D}]$ such that $g \in M$. Analogously, g is true, under *cautious* semantics, if g is true in every stable model of $\mathcal{P}[\mathcal{D}]$.

In the following we are mainly interested in the atoms that are true under the cautious semantics, denoted by $cautious(\mathcal{P}[\mathcal{D}])$.

Disjunctive logic programs are the core DLV language, see [49] and INFOMIX Deliverable D4.2 (Algorithms and Implementation Techniques for the Internal Integration Formalism), which has been accommodated to compute some generalizations of the stable model semantics.

The stable model semantics has been generalized to various classes of programs, including programs with both strong ("classical") and weak ("default") negation ("extended logic programs" [34]), to nested logic programs [52], or to programs with aggregates (See Deliverable D4.2). For programs with both classical and default negation, where the use of classical negation is limited to atoms, the stable models are (consistent) sets of ground literals and named *answer sets* [34].[1] They can be viewed as three-valued interpretations, in which the value of an atom can be either true, false, or unknown. Technically, extended logic programs can be reduced to logic programs with only default negation by a simple standard transformation, in which for each predicate p a further predicate of the same arity $\bar{p}$ is introduced, which intuitively encodes the complement of p. Then, in any extended logic program P, each occurrence of a classically negated literal $-p(t_1, \ldots, t_n)$ is replaced by the literals $\bar{p}(t_1, \ldots, t_n)$, and for each predicate p of arity n the constraint $\leftarrow p(X_1, \ldots, X_n), \bar{p}(X_1, \ldots, X_n)$ is added. Then, the stable models of the transformed program correspond to the answer sets of the original program P. This form of transformation is basically the one applied by DLV. In this work, as far as the DLV language is concerned, we shall focus on optimization methods for the core of the DLV language, which are disjunctive programs as discussed above. We shall address also in brief extensions to the core in Section 4.9.

For further background, we refer to [33, 34, 63, 28, 49].

2.2 Relational Databases

Databases We assume a fixed, countable database domain $\mathcal{U}$ whose elements are referenced by constants under the *unique name assumption*, i.e., different constants c and c'

[1]In the seminal paper [34], also inconsistent answer sets are considered, which are however often discarded in answer set programming since they are of little interest in practice.

denote different objects. We will consider finite and infinite $\mathcal{U}$, though we will elaborate more on finite domains.
A *relational schema* (or simply *schema*) $\mathcal{RS}$ is a pair $\langle \Psi, \Sigma \rangle$, where:

- Ψ is a set of relation (predicate) symbols, each with an associated arity that indicates the number of its attributes;
- Σ is a set of *integrity constraints* (ICs), i.e., a set of assertions that are intended to be satisfied by each database instance. We assume to deal with universally quantified constraints [1], i.e., first order formulas of the form:

$$\forall(\vec{\mathbf{x}}) \bigwedge_{i=1}^{l} A_i \supset \bigvee_{j=1}^{m} B_j \vee \bigvee_{k=1}^{n} \phi_k, \qquad (1)$$

  where $l + m > 0$, $n \geq 0$, $\vec{\mathbf{x}}$ is a list of distinct variables, $A_1, \ldots, A_l$ and $B_1, \ldots, B_m$ are positive literals, and $\phi_1, \ldots, \phi_n$ are built-in literals.

Notice that many classical constraints issued on a relational schema can be expressed in this format, as key and functional dependencies, exclusion dependencies, or inclusion dependencies of the form $\forall(\vec{\mathbf{x}}) p_1(\vec{\mathbf{x}}) \supset p_2(\vec{\mathbf{x}})$.

Example 2.1 *Let the global schema $\mathcal{G}_0$ be the database schema $\langle \Psi_0, \Sigma_0 \rangle$, where Ψ_0 consists of the ternary relation symbols*

$$\begin{array}{l} \mathit{player}(\mathit{Pcode}, \mathit{Pname}, \mathit{Pteam}), \\ \mathit{team}(\mathit{Tcode}, \mathit{Tname}, \mathit{Tleader}), \text{ and} \\ \mathit{coach}(\mathit{Ccode}, \mathit{Cname}, \mathit{Cteam}). \end{array}$$

The associated constraints Σ_0 can be formally defined as follows (quantifiers are omitted):

$$\begin{array}{r} \mathit{player}(X, Y, Z) \wedge \mathit{player}(X, Y1, Z1) \supset Y{=}Y1 \\ \mathit{player}(X, Y, Z) \wedge \mathit{player}(X, Y1, Z1) \supset Z{=}Z1 \\ \mathit{team}(X, Y, Z) \wedge \mathit{team}(X, Y1, Z1) \supset Y{=}Y1 \\ \mathit{team}(X, Y, Z) \wedge \mathit{team}(X, Y1, Z1) \supset Z{=}Z1 \\ \mathit{coach}(X, Y, Z) \wedge \mathit{coach}(X, Y1, Z1) \supset Y{=}Y1 \\ \mathit{coach}(X, Y, Z) \wedge \mathit{coach}(X, Y1, Z1) \supset Z{=}Z1 \\ \mathit{coach}(X, Y, Z) \wedge \mathit{player}(X1, Y1, Z1) \supset X{\neq}X1 \\ \mathit{coach}(X, Y, Z) \wedge \mathit{team}(X1, Y1, Z1) \supset X{\neq}Z1 \end{array}$$

The first six rows encode the key dependencies, namely that the keys of player, team, and coach, are the attributes Pcode, Tcode, and Ccode, respectively, whereas the last two rows model the two constraints stating that a coach cannot be a player or a team leader. □

A *(finite) database instance* (or simply *database*) $\mathcal{DB}$ for a schema $\mathcal{RS}$ is a (finite) set of facts of the form $r(t)$ where r is a relation of arity n in Ψ and t is an n-tuple of values from $\mathcal{U}$.

Given a constraint $\sigma \in \Sigma$ of the form $\forall(\vec{\mathbf{x}})\alpha(\vec{\mathbf{x}})$, we denote by $ground(\sigma)$ the set of all ground formulas $\alpha(\vec{\mathbf{x}})\theta$, also called *ground constraints*, where θ is any substitution of values in $\mathcal{U}$ for $\vec{\mathbf{x}}$; furthermore, $ground(\Sigma) = \bigcup_{\sigma\in\Sigma} ground(\sigma)$.

Given a schema $\mathcal{RS} = \langle\Psi, \Sigma\rangle$ and a database instance $\mathcal{DB}$ for $\mathcal{RS}$, we say that $\sigma^g \in ground(\Sigma)$ is *satisfied* (resp., *violated*) in $\mathcal{DB}$, if σ^g evaluates to true (resp., false) in $\mathcal{DB}$. Moreover, $\mathcal{DB}$ is *consistent* with Σ if every $\sigma \in ground(\Sigma)$ is satisfied in $\mathcal{DB}$.

2.3 A Logic Framework for Query Answering

In this section, we present an abstract framework for modeling query answering in data integration systems using logic programs, recapitulating Deliverables D3.1, D3.3 and D5.1.

Formally, a data integration system $\mathcal{I}$ is a triple $\langle\mathcal{G}, \mathcal{S}, \mathcal{M}\rangle$, where:

1. $\mathcal{G}$ is the *global schema*. We assume that $\mathcal{G}$ is a relational schema, i.e., $\mathcal{G} = \langle\Psi, \Sigma\rangle$.

2. $\mathcal{S}$ is the *source schema*, constituted by the schemas of the various sources that are part of the data integration system. We assume that $\mathcal{S}$ is a relational schema of the form $\mathcal{S} = \langle\Psi', \emptyset\rangle$, in particular there are no integrity constraints on the sources. Note that in the INFOMIX architecture sources are defined using ISDF (Internal Source Data Format), a subset of XML. However, as demonstrated in deliverable D6.2, this format allows for a one-to-one translation to relational format, so in this report we can safely assume that the sources are given in relational format.

3. $\mathcal{M}$ is the *mapping between $\mathcal{G}$ and $\mathcal{S}$*. In this deliverable, we assume the mapping to be given by the GAV (Global As View) approach, i.e., each global relation in Ψ is associated with a *view*, i.e., a query, over the sources. We assume that the language used to express queries in the mapping is Datalog$^{\neg_s}$.

Example 2.2 *As a running example, we consider a data integration system $\mathcal{I}_0 = \langle\mathcal{G}_0$, $\mathcal{S}_0$, $\mathcal{M}_0\rangle$, referring to the context of football teams. The global schema $\mathcal{G}_0$ is the same as in Example 2.1. The source schema $\mathcal{S}_0$ comprises the relations s_1, s_2, s_3, and s_4.*

Finally, the mapping $\mathcal{M}_0$ is defined by the datalog program:

$$\begin{array}{l} player(X,Y,Z) \leftarrow s_1(X,Y,Z,W) \\ team(X,Y,Z) \leftarrow s_2(X,Y,Z) \\ team(X,Y,Z) \leftarrow s_3(X,Y,Z) \\ coach(X,Y,Z) \leftarrow s_4(X,Y,Z) \end{array}$$

□

We call any database $\mathcal{D}$ for the source schema $\mathcal{S}$ a *source database* for $\mathcal{I}$. Based on $\mathcal{D}$, it is possible to compute database instances for $\mathcal{G}$, called *global databases* for $\mathcal{I}$, according to the mapping specification. Given a data integration system $\mathcal{I} = \langle\mathcal{G}, \mathcal{S}, \mathcal{M}\rangle$ and a source database $\mathcal{D}$, the *retrieved global database*, $ret(\mathcal{I}, \mathcal{D})$, is the global database obtained by evaluating each view of the mapping $\mathcal{M}$ over $\mathcal{D}$.

Notice that $ret(\mathcal{I},\mathcal{D})$ might be inconsistent with respect to Σ, since data stored in local and autonomous sources need in general not satisfy constraints expressed on the global schema. Hence, in case of constraint violations, we cannot conclude that $ret(\mathcal{I},\mathcal{D})$ is a "legal" global database for $\mathcal{I}$ [48].

s_1 :

10	*Totti*	*RM*	27
9	*Beckham*	*MU*	28

s_2 :

RM	*Roma*	10

s_3 :

MU	*Man. Utd.*	8
RM	*Real Madrid*	10

s_4 :

7	*Ferguson*	*MU*

Figure 1: Source database $\mathcal{D}_0$.

$player^{\mathcal{R}}$:

10	*Totti*	*RM*
9	*Beckham*	*MU*

$team^{\mathcal{R}}$:

RM	*Roma*	10
MU	*Man. Utd.*	8
RM	*Real Madrid*	10

$coach^{\mathcal{R}}$:

7	*Ferguson*	*MU*

Figure 2: The retrieved global database $ret(\mathcal{I}_0, D_0)$.

Example 2.3 *Assuming that the information content of the sources is given by the database $\mathcal{D}_0$ depicted in Figure 1, the retrieved global database $ret(\mathcal{I}_0,\mathcal{D}_0)$ is given in Figure 2. It violates the key constraint on team, witnessed by the facts $team(RM, Roma, 10)$ and $team(RM,\ Real\ Madrid, 10)$, which coincide on Tcode but differ on Tname.* □

Since we have seen that $ret(\mathcal{I},\mathcal{D})$ may be inconsistent w.r.t. global constraints, we define the semantics of a data integration system $\mathcal{I}$ in terms of *repairs* of the database $ret(\mathcal{I},\mathcal{D})$.

Repairs Let us first consider the setting of a single database. Let $\mathcal{RS} = \langle\Psi,\Sigma\rangle$ be a relational schema, $\mathcal{DB}$ a (possibly inconsistent) database for $\mathcal{RS}$, and $\mathcal{R}_1$ and $\mathcal{R}_2$ two databases for $\mathcal{RS}$ consistent with Σ. We say that $\mathcal{R}_1 \leq_{\mathcal{D}} \mathcal{R}_2$ if $\triangle(\mathcal{R}_1,\mathcal{D}) \subseteq \triangle(\mathcal{R}_2,\mathcal{D})$, where $\triangle(X,Y)$ denotes the symmetric difference between sets X and Y. Furthermore, $\mathcal{R}_1 <_{\mathcal{D}} \mathcal{R}_2$ stands for $\mathcal{R}_1 \leq_{\mathcal{D}} \mathcal{R}_2 \wedge \mathcal{R}_2 \not\leq_{\mathcal{D}} \mathcal{R}_1$. Then, a database $\mathcal{R}$ is a *repair for $\mathcal{DB}$ w.r.t.* Σ, if $\mathcal{R}$ is a database for $\mathcal{RS}$ consistent with Σ and $\mathcal{R}$ is *minimal w.r.t.* $\leq_{\mathcal{D}}$, i.e., there exists no database $\mathcal{R}'$ for $\mathcal{RS}$ consistent with Σ such that $\mathcal{R}' <_{\mathcal{D}} \mathcal{R}$. We refer to the set of all such repairs as $rep(\mathcal{DB})$ *wrt* Σ; when clear from the context, Σ is omitted.

Definition 2.4 Let $\mathcal{I} = \langle\mathcal{G},\mathcal{S},\mathcal{M}\rangle$ be a data integration system where $\mathcal{G} = \langle\Psi,\Sigma\rangle$, and let $\mathcal{D}$ be a source database for $\mathcal{I}$. A global database $\mathcal{R}$ for $\mathcal{I}$ is a *repair for $\mathcal{I}$ w.r.t. $\mathcal{D}$* if $\mathcal{R}$ is a repair for $ret(\mathcal{I},\mathcal{D})$ w.r.t. Σ. The set of all repairs for $\mathcal{I}$ w.r.t. $\mathcal{D}$ is denoted by $rep_{\mathcal{I}}(\mathcal{D})$. □

$player^{\mathcal{R}_1}$:

10	*Totti*	*RM*
9	*Beckham*	*MU*

$team^{\mathcal{R}_1}$:

RM	*Roma*	10
MU	*Man. Utd.*	8

$coach^{\mathcal{R}_1}$:

7	*Ferguson*	*MU*

$player^{\mathcal{R}_2}$:

10	*Totti*	*RM*
9	*Beckham*	*MU*

$team^{\mathcal{R}_2}$:

MU	*Man. Utd.*	8
RM	*Real Madrid*	10

$coach^{\mathcal{R}_2}$:

7	*Ferguson*	*MU*

Figure 3: Repairs $\mathcal{R}_1$ and $\mathcal{R}_2$ of $\mathcal{I}_0$ w.r.t. $\mathcal{D}_0$.

Intuitively, each repair is obtained by properly adding and deleting facts from $ret_{\mathcal{I}}(\mathcal{D})$ in order to satisfy constraints in Σ, as long as we "minimize" such additions and deletions.

Note that, in the above definition we have considered the mapping $\mathcal{M}$ as *exact*, i.e., we have assumed that the data retrieved from the sources by the views of the mapping are exactly the data that satisfy the global schema, provided suitable repairing operations. Other different assumptions can be adopted on the mapping (e.g., *soundness* or *completeness* assumptions, cf. deliverables D3.3 and D5.1). Roughly speaking, such assumptions impose some restrictions or preferences on the possibility of adding or removing facts from $ret(\mathcal{I}, \mathcal{D})$ to repair constraint violations giving rise to different notions of minimality.

We stress that dealing only with exact mappings is not an actual limitation for the techniques presented here; in fact, in many practical cases, the computation of the repairs under other mapping assumptions can be modelled by means of a logic program similar to the computation of repairs under the exactness assumption.

Query A *query* over the global schema $\mathcal{G}$ is a non-recursive Datalog$^\neg$ program that is intended to extract a set of tuples over $\mathcal{U}$; note that in real integration applications, typically a language subsumed by non-recursive Datalog is adopted. For any query $q(X_1, \ldots, X_n)$, we call the set $ans(q, \mathcal{I}, \mathcal{D}) = \{q(c_1, \ldots, c_n) \mid q(c_1, \ldots, c_n) \in \mathrm{SM}(q[\mathcal{R}])$ for each $\mathcal{R} \in rep_{\mathcal{I}}(\mathcal{D})\ \}$ the *consistent answers* to q.

Example 2.5 *Recall that in our scenario, the retrieved global database $ret(\mathcal{I}_0, \mathcal{D}_0)$ shown in Figure 2 violates the key constraint on team, witnessed by $team(RM, Roma, 10)$ and $team(RM,\ Real\ Madrid, 10)$. A repair results by removing exactly one of these facts; hence, $rep_{\mathcal{I}_0}(\mathcal{D}_0) = \{\mathcal{R}_1, \mathcal{R}_2\}$, where $\mathcal{R}_1$ and $\mathcal{R}_2$ are as shown in Figure 3.*

For the query

$$\begin{array}{lcl} q(X) & \leftarrow & player(X, Y, Z); \\ q(X) & \leftarrow & team(V, W, X), \end{array}$$

which asks for the codes of all players and coaches, we thus obtain that $ans(q, \mathcal{I}_0, \mathcal{D}_0) = \{q(8), q(9), q(10)\}$. If we consider the query $q'(Y) \leftarrow team(X, Y, Z)$, we have that

$$ans(q', \mathcal{I}_0, \mathcal{D}_0) = \{q'(Man.\ Utd.)\},$$

while considering $q''(X, Z) \leftarrow team(X, Y, Z)$, we have that

$$ans(q'', \mathcal{I}_0, \mathcal{D}_0) = \{q''(RM, 10), q''(MU, 8)\}.$$

$\Box$

In the framework of the INFOMIX system we provide answers to user queries by encoding the mapping assertions in $\mathcal{M}$ and the constraints in Σ in a Datalog program enriched with unstratified negation or disjunction, in such a way that the stable models of this program map to the repairs of the retrieved global database.

Definition 2.6 Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, $\mathcal{D}$ is a source database for $\mathcal{I}$, and q is a non-recursive Datalog$^{\neg}$ query over $\mathcal{G}$. Then, a *logic specification for querying* $\mathcal{I}$ is a Datalog$^{\vee,\neg}$ program $\Pi_{\mathcal{I}}(q) = \Pi_{\mathcal{M}} \cup \Pi_{\Sigma} \cup \Pi_q$ such that

1. $ret(\mathcal{I}, \mathcal{D}) \rightleftharpoons \mathrm{SM}(\Pi_{\mathcal{M}}[D])$, where $\Pi_{\mathcal{M}}$ is a Datalog$^{\neg_s}$ program,
2. $rep_{\mathcal{I}}(\mathcal{D})) \rightleftharpoons \mathrm{SM}(\Pi_{\Sigma}[ret(\mathcal{I}, \mathcal{D})])$, and
3. $ans(q, \mathcal{I}, \mathcal{D}) \rightleftharpoons \{q(t) \mid q(t) \in M \text{ for each } M \in \mathrm{SM}((\Pi_q \cup \Pi_{\Sigma})[ret(\mathcal{I}, \mathcal{D})])\}$, where Π_q is a non-recursive Datalog$^{\neg}$ program,

where $\rightleftharpoons$ denotes a polynomial-time computable correspondence between two sets. $\Box$

This definition establishes a connection between the semantics of $\Pi_{\mathcal{I}}(q)$ and the consistent answers to a query posed to $\mathcal{I}$ (Item 3) provided some syntactic transformations, which typically are simple encodings such that $\rightleftharpoons$ is a linear-time computable bijection. In particular, $\Pi_{\mathcal{I}}(q)$ is composed by three modules that can be hierarchically evaluated, i.e., $\Pi_{\mathcal{M}} \triangleright \Pi_{\Sigma} \triangleright \Pi_q$ [28], using Splitting Sets [53]:

- $\Pi_{\mathcal{M}}$ is used for retrieving data from the sources: the retrieved global database can be derived from its unique stable model (Item 1);
- Π_{Σ} is used for enforcing the constraints on the retrieved global database, whose repairs can be derived from the stable models of $\Pi_{\Sigma}[ret(\mathcal{I}, \mathcal{D})]$ (Item 2);
- finally, Π_q is used for encoding the user query q.

2.4 Query Evaluation in Infomix

Query evaluation has been defined in the Run-Time architecture in the INFOMIX system architecture (Deliverable D2.1), which is reproduced in Figure 4.

Analyzing only the data-flow in the system, which is relevant to queries, we obtain a reduced and refined diagram depicted in Figure 5. Since we are concerned with the run-time phase, metadata such as wrappers, source schemata, and mappings are already available, as they have been defined during the design-time phase before.

Whenever the system user now poses a query over the global schema, it first reaches the *Query Reformulator and Optimizer* module. In this module the query is analyzed using the mapping metadata. From this data, a *Query Plan* is produced, which basically specifies which sources have to be made available for answering the user query. This *Query Plan* (together with metadata about wrappers) is then used to retrieve *Source Data* and store it in the *Internal Data Store*.

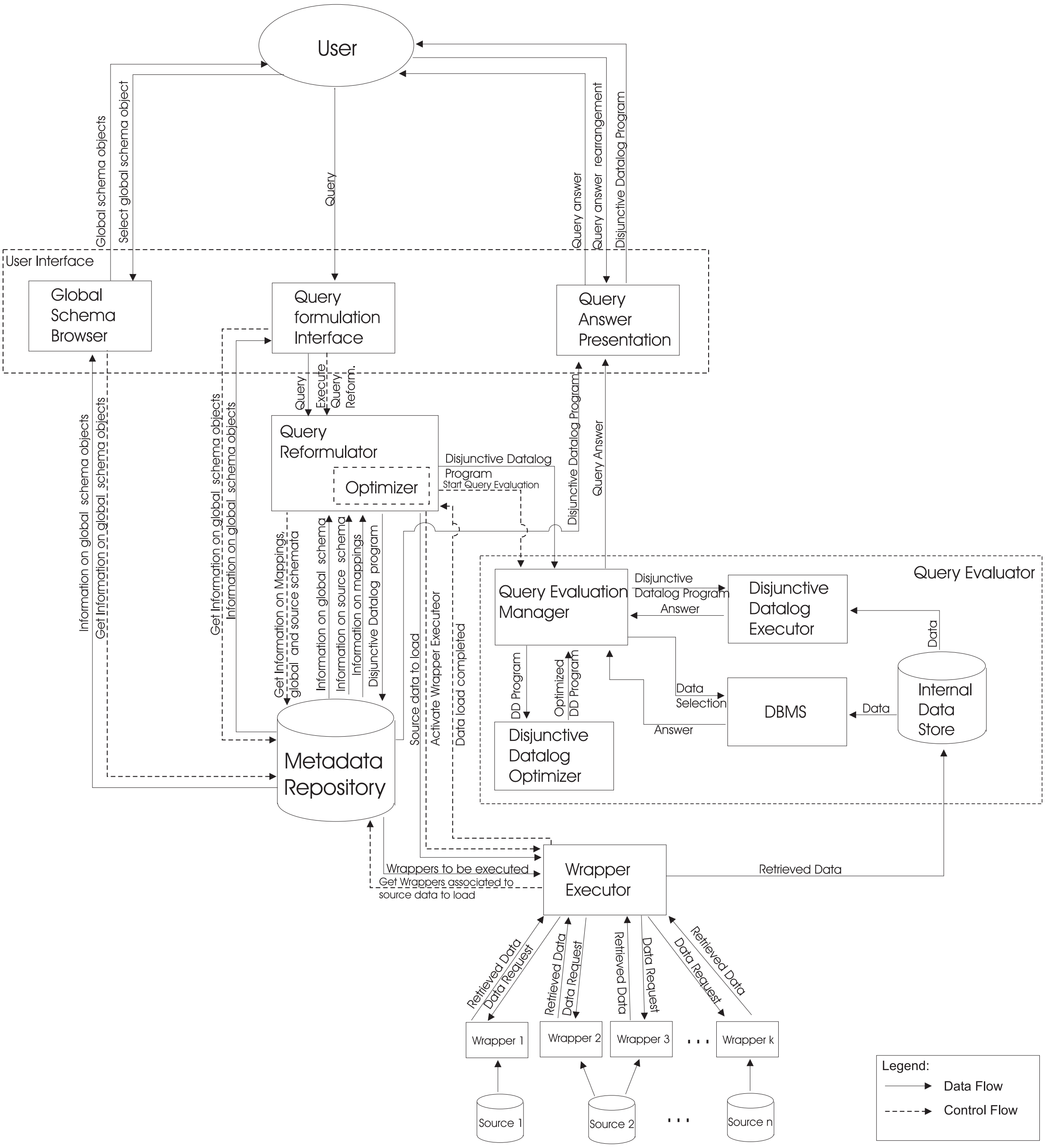


Figure 4: Run-Time Architecture of the INFOMIX System

In addition, the *Query Reformulator and Optimizer* produces a *Datalog Query*, the main part of which consists of the so-called *repair program*, which applies the mapping to the source data and produces repair scenarios, in which the constraints defined over the global schema hold. The actual query is then applied over all such repairs.

This Datalog Query is then processed by the *Disjunctive Datalog Optimizer*, which tries to rewrite the Datalog Query in order to produce an equivalent (with respect to the

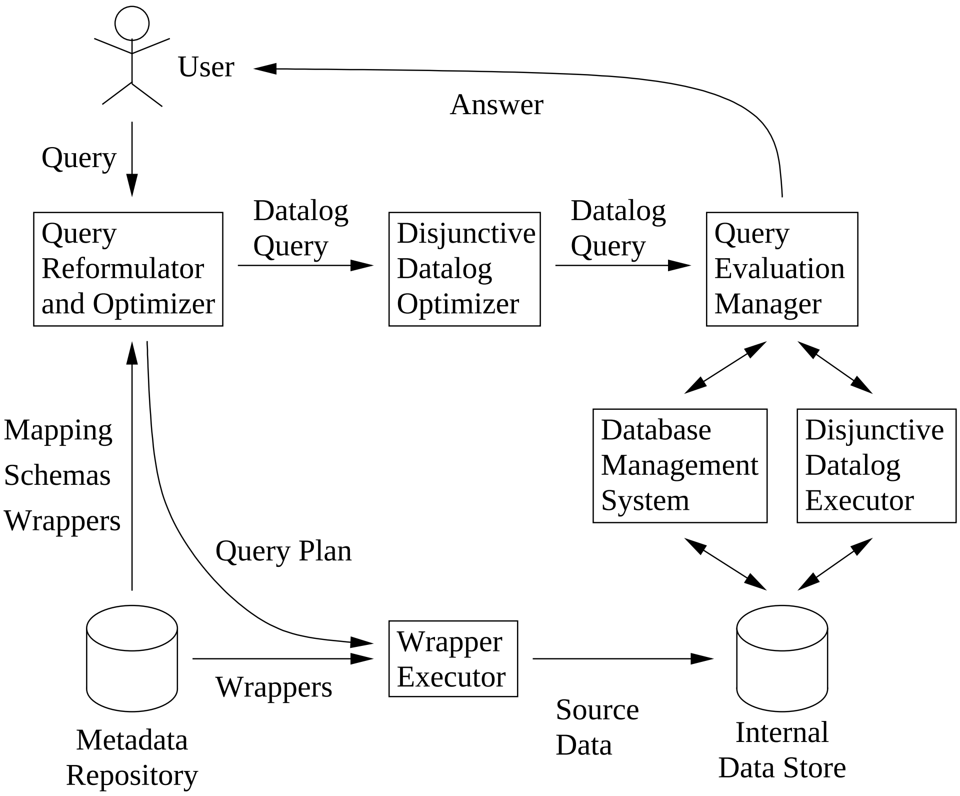


Figure 5: Query Answering in the INFOMIX System

query), but more efficient Datalog Query.

Eventually, answers to the optimized Datalog Query are produced by the *Query Evaluation Manager*. This module makes use of a Database Management System (to solve parts which are "easy" to compute) and a Disjunctive Datalog Executor (to solve "hard" parts).

3 Optimization Methods

Examining Figure 5 in Section 2.4, we can identify the following areas, where optimizations are applicable:

1. Query Reformulator and Optimizer;
2. Disjunctive Datalog Optimizer;
3. Query Evaluation Manager.

Concerning item 1, its main task concerning optimization is that of producing a suitable Query Plan, which minimizes wrapper deployment, and exploits wrapper specialization via parameter, whenever the respective wrapper supports it, in order to minimize the source data to be retrieved.

The Disjunctive Datalog Optimizer (item 2) is fully dedicated to optimization, and its task is to transform the Datalog Query to an equivalent one, which can be evaluated more efficiently.

Finally, item 3, the Query Evaluation Manager, tries to split its task into an "easy" and a "hard" part. The rationale is that we expect a large portion of the data to be smoothly integratable without requiring repairs (the "easy" part), while only a small portion of the data needs repair (the "hard" part). Deviding the data into these parts, making the "hard" part as small as possible, is the optimization task of the Query Evaluation Manager.

In the present report, we will focus on items 2 and 3, as they are more at the heart of the INFOMIX project objectives. Nevertheless, we will also discuss item 1 in section 7, but in less detail.

3.1 Types of Optimizations

Next, we will develop a simple taxonomy for classifying opimization methods, which we will use in the sequel to classify applicable optimization methods.

Considering the discussion above, we can observe that many of the optimizations in the Query Reformulator and Optimizer and in the Query Evaluation Manager are rather specific w.r.t. the INFOMIX system architecture, while the optimizations in the Disjunctive Datalog Optimizer and some part of the Query Reformulator and Optimizer are of a more general type, which does not depend on the INFOMIX system architecture.

Therefore, our main classification will consist of

1. general optimizations, and
2. INFOMIX-specific optimizations.

We can further divide the *general optimizations* into

1.1. Datalog optimizations, and

1.2. Data Integration optimizations,

and the *INFOMIX-specific optimizations* into

2.1. Repair-oriented optimizations, and

2.2. Query-plan-oriented optimizations.

3.2 Overview of Optimization Methods

We will now summarize several possible optimizations methods for the INFOMIX system along the taxonomy outlined in the previous section. We will subsequently discuss some of them in this report, while for some other we just give an overview and references to complementary literature.

3.2.1 General Datalog Optimizations

Magic Sets DLV, the Disjunctive Datalog Executor to be employed in the INFOMIX system, essentially uses bottom-up techniques for query answering. A main drawback of this approach is that a large number of derivations could be performed, which are not related to the query at all, and are therefore redundant with respect to the query.

For simpler program classes, such as positive, nondisjunctive Datalog, this has been realized already in the 1980s, and led to the development of various transformation techniques, which aim at making the bottom-up process more query-oriented. Probably the most succesful of these techniques is the *Magic Sets Transformation* [2, 3, 41, 57, 46].

The first attempt to generalize the magic sets transformation to positive disjunctive Datalog has been made in [39, 40] and refined in [12].

These techniques and further generalizations to programs containing negation and aggregates have been discussed at length in Deliverable D4.3, and will not be repeated here.

Equivalences of Datalog Programs In general, one is interested whether one Datalog program is equivalent to the other. However, this is a question which is not easy to answer: In [67] it has been shown that this question is undecidable even for positive, nondisjunctive Datalog, and therefore it is also not decidable for more general Datalog classes.

Immediately after this negative result, in [64, 55], a restricted version of equivalence, called *uniform equivalence* has been proposed and shown to be decidable. There is an appealing motivation behind this notion: If one is considering *modules* instead of programs, then uniform equivalence is a good test for deciding whether two modules are equivalent to each other.

More than ten years later, a similar, but even more restricted notion, called *strong equivalence*, has been proposed in [52] for propositional Datalog programs. The intuition behind this notion is that not modules but rather arbitrary program parts (which can have complex dependencies with the remainder of programs) are tested for equivalence.

In [20] the notion of uniform equivalence has been transferred to a propositional setting with disjunctive logic programs.

In Section 4 these notions and useful optimization techniques following from them are discussed in detail. Most of this discussion assumes propositional programs or Datalog programs with finite domains. For the more general setting of Datalog programs with infinite domains the situation is somehow more involved and not much research has been done in this direction so far. We address this issue here for both uniform and strong equivalence in the general setting.

As shown (and non-surprising), deciding equivalence of logic programs (under the various forms), even if decidable, will be intractable in general. This, however, thus not mean that optimization can not be fruitfully applied. Indeed, for *static optimization*, which is performed upon the creation of the initial program, more efforts might be spent then for *dynamic optimization* at runtime. In the INFOMIX context, static programs optimization is of importance for stored user queries, which are predefined queries which are repeatedly invoked (usually, with different frequencies). Investment into query optimization at query design time might pay off greatly at run-time, and the time consumed might be not the most important criterion.

DLV^{DB} Yet another direction for optimization is to couple Disjunctive Datalog Executors more tightly to traditional Database Management Systems (DBMS). Doing this would allow for delegating parts of the computation to a DBMS, thus exploiting sophisticated techniques implemented in DBMSs.

DLV^{DB} is the first attempt to do this for a Disjunctive Datalog engine. Its features and design have been discussed in Deliverable D4.2, and will not be repeated here.

3.2.2 Data Integration Optimizations

Pushing Constants Concerning the Query Reformulator and Optimizer, one can try to minimize the source data used for integration. In particular, data which cannot contribute to answering the query should be disregarded.

Importantly, one can try to push constants (or more general: selections) from the query to the source level by exploiting the mapping. Given that we are focussed on the GAV setting, this boils down to performing query composition, as the mapping is also a query in this case. Such a technique in the context of data integration is outlined in [10], for example. In a more classical database setting, [36] describes an efficient optimization method for this task. Some complementary material can be found for example in [1].

Query Tree Optimizations Apart from pushing selections, also more advanced optimizations on query trees have been defined. One can apply them to the composed query described above. Several such techniques have been studied for data integration systems, cf. [43]. Moreover, sometimes standard database optimization techniques can be directly adapted, see e.g. [1, 44] for a comprehensive overview.

3.2.3 Repair-oriented optimizations

Focussing Another line of optimizations is concerned with the repair program (Datalog Query) itself, taking into account the constraints imposed on the global schema.

In particular, this technique aims at identifying (as precisely as possible) the part of the source data which will give rise to constraint violations on the global schema level. The rationale is that only this part of the source data needs to be treated by the repair program, while the rest of the source data can be integrated without repair, as it will not give rise to constraint violations.

Given that repairing is the most complex task in integration, focussing is a very important technique and will be dealt with in detail in Section 5.

Repair Compilation Somehow orthogonal to the previous optimization is the task of answering queries over repairs. The semantics of query answering in INFOMIX is defined as answers which hold in any repair (cf. Section 2.3).

Now even if focussing has been applied, the number of repairs may be large, and the question arises how these repairs should be stored for answering queries. The simplest method — storing all repairs — is rather inefficient as it consumes a lot of space.

In Section 6 we will present more sophisticated methods, which employ a DBMS for storing information about the query in repairs. The benefits are that in this way the space consumption can be cut considerably and that the efficiency of DBMSs can be exploited.

Ad-hoc Program Rewriting The previous two methods were aimed at automatically separating and reuniting programs. Another, technically less involved method, which may of course be combined with the previous techniques, is to simply consider variants of repair programs and study their resource consumption w.r.t. the employed computational engine.

One may employ program transformation rules in order to (semi-) automatically generate candidates for such program variants. One can also use some techniques described in Section 4 for checking for errors in these variants.

Since there is a very high degree of freedom in such a process, and since it is clear how it can be done, we do not further discuss this optimization methods in this report.

3.2.4 Query-plan-oriented Optimizations

The optimizations in this category extend those described in Section 3.2.2 to the wrapper level and are in this sense closely related. It turns ut that the query plan is the key datastructure in this respect.

Selective Wrapper Deployment If it can be determined that some source need not be considered for answering the query, then the respective source data should not only be disregarded, but should also not be produced by the associated wrapper in the first place.

One can therefore devise the query plan in a way that the respective wrapper will not be deployed during the process of answering the query.

Pushing Queries into Wrappers While the previous technique works on a rather coarse level (causing wrappers to be deployed or not), in some cases one can go one step further and modify the inner structure of the wrappers whose deployment is needed.

In particular, this is possible with *Query Wrappers* (as defined in Deliverable D6.2) and to some extent also with *Visual Wrappers*. The rationale is that optimizations such as the pushing of selections (described in Section 3.2.2) do not stop at the source data level, but are pushed inside the associated wrappers and hence to the level of the external data sources. It is clear that this works only if the respective wrapper type allows for such a modification (or specialization) of data access.

Taking this approach to its limits, one could even think about transferring more complex data access to the external source level, such as computing (semi-) joins.

These techniques will be described in more detail in Section 7.

4 Equivalences of Logic Programs

In this section, we discuss program simplifications based on different notions of equivalence. We first analyze in Sections 4.1–4.7 propositional programs in detail, and afterwards, in Section 4.8, we present some corresponding results for the first-order case.

Following custom, we refer to propositional Datalog$^{\vee,\neg}$ programs as *disjunctive logic programs* (DLPs) simpliciter. As well, we call a program P a *normal logic program* (NLP) (resp., definite, positive, or Horn), if all rules in P are normal (resp., definite, positive, or Horn). Furthermore, a program P is *head-cycle free* (*HCF*) [4], if its dependency graph (which is defined as usual) has no directed cycle that contains two atoms belonging to the head of the same rule.

For convenience, we define the *reduct* of a rule r *relative to* a set of atoms I, denoted r^I, as the positive rule r' such that $H(r') = H(r)$ and $B^+(r') = B^+(r)$ if $I \cap B^-(r) = \emptyset$, and is void otherwise.

The following property will be required later on.

Proposition 4.1 *Let P be a DLP and $X, Y \subseteq Z$ interpretations. Then, $X \models P^Y$ implies $X \models P^Z$.*

The result is seen by the observation that $Y \subseteq Z$ implies $P^Z \subseteq P^Y$. Thus, $X \models P^Y$ implies $X \models P^Z$. In particular, for $X = Y$, $X \models P^X$ iff $X \models P$, and thus $X \models P$ implies $X \models P^Z$, for any $X \subseteq Z$.

4.1 Notions of Equivalence

4.1.1 Strong and Uniform Equivalence

Several notions of equivalence between logic programs have been considered in the literature (cf., e.g., [52, 55, 64]). Under stable semantics, two DLPs P and Q are regarded as equivalent, denoted $P \equiv Q$, iff $\mathrm{SM}(P) = \mathrm{SM}(Q)$. The more restrictive forms of *strong equivalence* and *uniform equivalence* are as follows:

Definition 4.2 *Let P and Q be two DLPs. Then,*

(*i*) *P and Q are* strongly equivalent, *or s-*equivalent, *denoted $P \equiv_s Q$, iff, for any set R of rules, the programs $P \cup R$ and $Q \cup R$ are equivalent, i.e., $P \cup R \equiv Q \cup R$.*

(*ii*) *P and Q are* uniformly equivalent, *or u-*equivalent, *denoted $P \equiv_u Q$, iff, for any set F of non-disjunctive facts, $P \cup F$ and $Q \cup F$ are equivalent, i.e., $P \cup F \equiv Q \cup F$.*

Obviously, $P \equiv_s Q$ implies $P \equiv_u Q$ but not vice versa. Both notions of equivalence, however, enjoy interesting semantical characterizations. As shown in [52], strong equivalence is closely related to the non-classical logic of here-and-there, which was adapted to logic-programming terms by Turner [70, 71]:

Definition 4.3 *Let P be a DLP, and let $X, Y \subseteq At$ such that $X \subseteq Y$. The pair (X, Y) is an* SE-model *of P (over At), if $Y \models P$ and $X \models P^Y$. By $M_s^{At}(P)$ we denote the set of all SE-models over At of P.*

In what follows, we usually leave the set At implicit and write $M_s(P)$ *simpliciter* instead of $M_s^{At}(P)$.

Proposition 4.4 ([70, 71]) *For any DLP P and Q, $P \equiv_s Q$ iff $M_s(P) = M_s(Q)$.*

SE-models can also be used to determine the stable models of a program [59, 52].

Proposition 4.5 *Let P be a DLP. Then, $Y \in \mathrm{SM}(P)$ iff $(Y,Y) \in M_s(P)$ and, for each $X \subset Y$, $(X,Y) \notin M_s(P)$.*

4.1.2 General Characterizations

In analogy to Propostion 4.4, we now show that uniform equivalence can be characterized in terms of SE-models as well, by imposing further conditions.

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

Lemma 4.6 *Two DLPs P and Q are uniformly equivalent, i.e. $P \equiv_u Q$, 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 (M,Y), $X \subset M \subset Y$, 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 any of the programs P and Q. Without loss of generality, assume $Y \models P$ and $Y \not\models Q$. Then, since in this case $Y \models P^Y$ and no strict subset of Y models $P \cup Y$, $Y \in \mathrm{SM}(P \cup Y)$, while $Y \notin \mathrm{SM}(Q \cup Y)$. This contradicts our assumption $P \equiv_u Q$. Hence, item (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 M, $X \subset M \subset Y$, it holds that (M,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, $Y \in \mathrm{SM}(Q \cup X)$ in this case, while $Y \notin \mathrm{SM}(P \cup X)$ ($X \models P^Y$ implies $X \models (P \cup X)^Y$, so (X,Y) is an SE-model of $P \cup X$). However, this contradicts $P \equiv_u Q$. Thus, it follows that for some M such that $X \subset M \subset Y$, (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 item (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 A and X, such that w.l.o.g., $X \in \mathrm{SM}(P \cup A) - \mathrm{SM}(Q \cup A)$. Since $X \in \mathrm{SM}(P \cup A)$, we have that $A \subseteq X$, and, moreover, $X \models P$. Consequently, (X,X) is an SE-model of P. Since $X \notin \mathrm{SM}(Q \cup A)$, either $X \not\models (Q \cup A)^X$, or there exists $X' \subset X$ such that $X' \models (Q \cup A)^X$.

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

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 A and X exists such that X is a stable model of exactly one of $P \cup A$ and $Q \cup A$. That is, $P \equiv_u Q$ holds. □

From Lemma 4.6 we immediately obtain the following characterization of uniform equivalence between logic programs.

Theorem 4.7 *Two DLPs, P and Q, are uniformly equivalent, $P \equiv_u Q$, iff*

(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 M, such that $X \subseteq M \subset Y$, and (M, Y) is an SE-model of Q (respectively P).*

Example 4.8 *Consider the programs $P = \{a \vee b\}$ and $Q = \{a \leftarrow \neg b;\ b \leftarrow \neg a\}$. It is easily checked P and Q are not strongly equivalent. Indeed, $P \cup \{a \leftarrow b;\ b \leftarrow a\}$ has the stable model $\{a, b\}$, which is not a stable model of $Q \cup \{a \leftarrow b;\ b \leftarrow a\}$. By Theorem 4.7, we can easily verify that P and Q are uniformly equivalent: Their SE-models differ only in $(\emptyset, \{a, b\})$, which is an SE-model of Q but not of P. Thus, items (i) and (ii) clearly hold for all other SE-models. Moreover, $(\{a\}, \{a, b\})$ is an SE-model of P, and thus item (ii) also holds for $(\emptyset, \{a, b\})$.*

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

Example 4.9 *Let P and Q be as in the previous example. Since $M_s(R \cup S) = M_s(R) \cap M_s(S)$ for any programs P and S, the pair $(\emptyset, \{a, b\})$ is no longer an SE-model of $Q \cup \{c : \ \leftarrow a, b\}$ (because $\{a, b\} \not\models c$). Hence, $P \cup \{c\} \equiv_s Q \cup \{c\}$.*

For finite programs, we can derive from Theorem 4.7 the following characterization of uniform equivalence.

Theorem 4.10 *Two finite DLPs P and Q are uniformly equivalent, i.e., $P \equiv_u Q$, iff the following conditions hold:*

(i) *(X, X) is an SE-model of P iff it is an SE-model of Q for every X, and*

(ii) *for every SE-model $(X, Y) \in M_s(P) \cup M_s(Q)$ such that $X \subset Y$, there exists an SE-model $(M, Y) \in M_s(P) \cap M_s(Q)$ $(=M_s(P \cup Q))$ such that $X \subseteq M \subset Y$.*

Proof. Since (i) holds by virtue of Theorem 4.7, we only need to show (ii). Assume (X, Y), where $X \subset Y$, is in $M_s(P) \cup M_s(Q)$.

If $(X, Y) \in M_s(P) \cap M_s(Q)$, then the statement holds. Otherwise, by virtue of Theorem 4.7, there exists (M_1, Y), $X \subseteq M_1 \subset Y$, such that (M_1, Y) is in $M_s(P) \cup M_s(Q)$. By repeating this argument, we obtain a chain of SE-models $(X, Y) = (M_0, Y)$, (M_1, Y),

$\ldots$, (M_i, Y), $\ldots$ such that $(M_i, Y) \in \mathcal{M}_s(P) \cup \mathcal{M}_s(Q)$ and $M_i \subseteq M_{i+1}$, for all $i \geq 0$. Furthermore, we may choose M_1 such that M_1 coincides with Y on all atoms which do not occur in $P \cup Q$ (and hence all M_i, $i \geq 1$, do so). Since P and Q are finite, it follows that $M_i = M_{i+1}$ must hold for some $i \geq 0$ and hence $(M_i, Y) \in \mathcal{M}_s(P) \cap \mathcal{M}_s(Q)$ must hold. This proves the result. □

Note that the previous theorem remains valid even if only one of P and Q is finite.

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

Definition 4.11 (UE-model) *Let P be a DLP. Then, any $(X, Y) \in \mathcal{M}_s(P)$ is a* uniform equivalence *(UE) model of P, if for every $(X', Y) \in \mathcal{M}_s(P)$ it holds that $X \subset X'$ implies $X' = Y$. By $\mathcal{M}_u(P)$ we denote the set of all UE-models of P.*

That is, the UE-models comprise all SE-models of a DLP which correspond to classical models of P (for $Y = X$), plus all its maximal "non-classical" SE-models, i.e., $\mathcal{M}_u(P) = \{(X, X) \in \mathcal{M}_s(P)\} \cup \max_{\geq}\{(X, Y) \in \mathcal{M}_s(P) \mid X \subset Y\}$, where $(X', Y') \geq (X, Y) \Leftrightarrow Y' = Y \wedge X \subseteq X'$.

In the sequel, we shall write $\mathcal{M}_e(r)$ instead of $\mathcal{M}_e(\{r\})$, for a rule r and $\alpha \in \{s, u\}$.

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

Theorem 4.12 *Two finite DLPs P and Q are uniformly equivalent, i.e., $P \equiv_u Q$, if and only if $\mathcal{M}_u(P) = \mathcal{M}_u(Q)$.*

Proof. By Theorem 4.10 we have to show that Conditions (i) $(X, X) \models P \Leftrightarrow (X, X) \models Q$ and (ii) $(X, Y) \models P \wedge X \subset Y \Rightarrow \exists M,\ X \subseteq M \subset Y\ :\ (M, Y) \models P \cup Q$ hold iff $\mathcal{M}_u(P) = \mathcal{M}_u(Q)$.

For the if direction assume $\mathcal{M}_u(P) = \mathcal{M}_u(Q)$. Then (i) holds by definition of UE-models. Now let (X, Y) be an SE-model of P, such that $X \subset Y$. There are two possibilities: If (X, Y) is maximal, then $(X, Y) \in \mathcal{M}_u(Q)$ as well and thus (ii) holds $(M = X)$; otherwise, (X, Y) is not maximal, which means that there exists some $(X', Y) \in \mathcal{M}_u(P)$ such that $X \subset X' \subset Y$, and since $\mathcal{M}_u(P) = \mathcal{M}_u(Q)$ Condition (ii) holds again $(M = X')$.

For the only-if direction let $P \equiv_u Q$. Then by Condition (i) $\mathcal{M}_u(P)$ and $\mathcal{M}_u(Q)$ coincide on models (X, X). Assume w.l.o.g. that (X, Y), $X \subset Y$, is in $\mathcal{M}_u(P)$, but not in $\mathcal{M}_u(Q)$. By (ii) there exists (M, Y), $X \subseteq M \subset Y$, which is an SE-model of both P and Q. Since $X \subset M$ contradicts $(X, Y) \in \mathcal{M}_u(P)$, let $M = X$, i.e., (X, Y) is an SE-model of Q as well, but it is not in $\mathcal{M}_u(Q)$. Hence, there exists $(X', Y) \in \mathcal{M}_u(Q)$, $X \subset X' \subset Y$ and by (ii) there exists (M', Y), $X' \subseteq M \subset Y$, which is an SE-model of P. This again contradicts $(X, Y) \in \mathcal{M}_u(P)$. Hence, $\mathcal{M}_u(P) = \mathcal{M}_u(Q)$. □

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

Example 4.13 *Each SE-model of the program $P = \{a \vee b\}$ satisfies the condition of an UE-model, and thus $\mathcal{M}_u(P) = \mathcal{M}_s(P)$. The program $Q = \{a \leftarrow \neg b;\ b \leftarrow \neg a\}$ has* *the additional SE-model* $(\{\}, \{a, b\})$*, and all of its SE-models except this one are UE-models of* Q*. Thus,* $M_u(P) = M_u(Q) = \{(\{a\}, \{a\}), (\{b\}, \{b\}), (\{a\}, \{a, b\}), (\{b\}, \{a, b\}), (\{a, b\}, \{a, b\})\}$*.*

Note that strong equivalence between P *and* Q *fails because* $(\emptyset, \{a, b\})$ *is not an SE-model of* P*. This SE-model is enforced by the intersection property* ((X_1, Y) *and* (X_2, Y) *in* $M_s(P)$ *implies* $(X_1 \cap X_2, Y) \in M_s(P)$) *which the Horn program* Q^Y *enjoys, which however is not satisfied by the disjunctive program* $P^Y (= P)$*. The maximality condition of UE-models eliminates this intersection property.*

Example 4.14 *Reconsider* $P = \{a \leftarrow \neg b;\ a \leftarrow b\}$*, which has classical models* $\{a\} \cup Y$*,* $Y \subseteq \{b, c\}$*. Its UE-models are of form* $(\{a\} \cup X, \{a\} \cup Y)$ *where* $X \in \{Y, Y - \{b\}, Y - \{c\}\}$*. Note that the atoms* b *and* c *have symmetric roles in* $M_u(P)$*. Consequently, the program obtained by exchanging the roles of* b *and* c*,* $Q = \{a \leftarrow \neg c;\ a \leftarrow c\}$ *has the same UE models. Hence,* P *and* Q *are uniformly equivalent.*

Like Theorem 4.10, also Theorem 4.12 remains valid if only one of P and Q is finite. However, the following example shows that it fails if both P and Q are infinite.

Example 4.15 *Consider the programs* P *and* Q *over* $At = \{a\} \cup \{b_i \mid i \geq 1\}$*, defined by*

$$P = \{a \leftarrow\ ,\ \ b_i \leftarrow \mid i \geq 1\} \quad \textit{and} \quad Q = \{a \leftarrow \neg a,\ b_i \leftarrow b_{i+1},\ b_i \leftarrow a \mid i \geq 1\}.$$

Both P *and* Q *have the single classical model* $M = \{a, b_i \mid i \geq 1\}$*. Furthermore,* P *has no "incomplete" SE-model* (X, Y) *such that* $X \subset Y$*, while* Q *has the incomplete SE-models* (X_i, M)*, where* $X_i = \{b_1, \ldots, b_i\}$ *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, e.g.,* P *has a stable model while* Q *has obviously not. Note that this is caught by our Theorem 4.7, item (ii): for* (X_0, M)*, which is an SE-model of* Q *but not of* P*, we cannot find an SE-model* (X, M) *of* P *between* (X_0, M) *and* (M, M)*.*

The test of Theorem 4.12 can be reformulated as follows.

Proposition 4.16 *For finite DLPs* P *and* Q*,* $P \equiv_u Q$ *iff* $M_u(P) \subseteq M_s(Q)$ *and* $M_u(Q) \subseteq M_s(P)$*.*

Proof. From Theorem 4.12, $P \equiv_u Q$ iff both $M_u(P) \subseteq M_u(Q)$ and $M_u(Q) \subseteq M_u(P)$ hold. Clearly, $M_u(R) \subseteq M_s(R)$ holds for any DLP R, which immediately gives the only-if direction. For the converse, suppose $P \not\equiv_u Q$. Hence, there exists an SE-interpretation (X, Y) such that either (i) $(X, Y) \in M_u(P)$ and $(X, Y) \notin M_u(Q)$, or (ii) $(X, Y) \in M_u(Q)$ and $(X, Y) \notin M_u(P)$. We only deal with Case (i); the second case proceeds analogously. Assume therefore that (i) holds. Then, by the definition of UE-models, there are two subcases to consider. First, $(X, Y) \notin M_s(Q)$. But then, $M_u(P) \subseteq M_s(Q)$ cannot hold. Second, there exists a set X' with $X \subset X' \subset Y$ and such that $(X', Y) \in M_u(Q)$. But $(X', Y) \notin M_s(P)$ since $(X, Y) \in M_u(P)$, hence $M_u(Q) \subseteq M_s(P)$ cannot hold. □

Based on SE- and UE-models, we define an associated notion of consequence under *strong* and *uniform equivalence*, respectively. Recall that (X, Y) models a rule r iff $Y \models r$ and $X \models r^Y$.

Definition 4.17 *Let P be a DLP, r a rule, and $e \in \{s, u\}$. Then, $P \models_e r$ iff, for each $(X, Y) \in M_e(P)$, $(X, Y) \models_e r$, i.e., $Y \models r$ and $X \models \{r\}^Y$. Furthermore, we write $P \models_e Q$ iff $P \models_e r$, for every $r \in Q$.*

Proposition 4.18 *For any DLP P and Q, $P \equiv_e Q$ iff $P \models_e Q$ and $Q \models_e P$, $e \in \{s, u\}$.*

As for the relationship of UE-consequence to classical consequence and cautious consequence under stable semantics, we note the following hierarchy. Let $\models_c$ denote consequence from the stable models, i.e., $P \models_c r$ iff $M \models r$ for every $M \in \mathrm{SM}(P)$.

Proposition 4.19 *For any finite program P and rule r,*

(i) *$P \models_u r$ implies $P \cup A \models_c r$, for each set of facts A;*

(ii) *$P \cup A \models_c r$, for each set of facts A, implies $P \models_c r$; and*

(iii) *$P \models_c r$ implies $P \models r$.*

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

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

Theorem 4.20 *Let P be a finite DLP and $e \in \{s, u\}$. Then, the following conditions are equivalent:*

(i) *$P \models_e r$ iff $P \models r$, for every rule r.*

(ii) *For every $(X, Y) \in M_e(P)$, it holds that $X \models P$.*

Corollary 4.21 *For any finite positive DLP P and any rule r, $P \models_e r$ iff $P \models r$, $e \in \{s, u\}$.*

In other words, for finite positive programs, the notions of strong, uniform, and classical consequence collapse. We shall obtain these results as corollaries of more general results in the next section, though.

As a final result here, which will be required later on, we characterize the set of SE-models of a disjunctive rule.

Proposition 4.22 *Let r be a disjunctive rule and (X, Y) an SE-interpretation. Then, $(X, Y) \in M_s(r)$ iff one of the following conditions is satisfied: (i) $X \models H(r)$; (ii) $Y \not\models B(r)$; or (iii) $X \not\models B^+(r)$ and $Y \models H(r)$.*

Proof. By definition, $(X,Y) \in M_s(r)$ iff $Y \models r$ and $X \models r^Y$. The former holds iff either of $Y \models H(r)$, $Y \not\models B^+(r)$, or $Y \cap B^-(r) \neq \emptyset$ is satisfied. $X \models r^Y$ holds iff either of $X \models H(r)$, $X \not\models B^+(r)$, or $Y \cap B^-(r) \neq \emptyset$ holds. Hence, $(X,Y) \in M_s(r)$ iff $Y \cap B^-(r) \neq \emptyset$, or

$$Y \models H(r) \text{ or } Y \not\models B^+(r), \quad \text{and} \tag{2}$$
$$X \models H(r) \text{ or } X \not\models B^+(r) \tag{3}$$

jointly hold. Clearly, $Y \not\models B^+(r)$ implies $X \not\models B^+(r)$, and $X \models H(r)$ implies $Y \models H(r)$. From this, it is easily verified that (X,Y) satisfies (2) and (3) iff either $Y \not\models B^+(r)$ and $X \models H(r)$, or jointly $X \not\models B^+(r)$ and $Y \models H(r)$ holds. Hence, we have that $(X,Y) \in M_s(P)$ iff either $Y \cap B^-(r) \neq \emptyset$, $Y \not\models B^+(r)$, (i), or (iii) holds. Finally, $Y \cap B^-(r) \neq \emptyset$ or $Y \not\models B^+(r)$ holds exactly if $Y \not\models B(r)$ (i.e., (ii)) holds. □

4.1.3 Equivalence for Restricted Classes of Programs

After discussing uniform equivalence of general propositional programs, let us now consider two prominent subclasses of programs, namely positive and head-cycle free programs.

Positive programs While for programs with negation, strong equivalence and uniform equivalence are different, the notions coincide for positive programs, as shown next.

Proposition 4.23 *Let P and Q be positive DLPs. Then $P \equiv_u Q$ iff $P \equiv_s Q$.*

Proof. The if-direction is immediate as $P \equiv_s Q$ implies $P \equiv_u Q$.

For the only-if-direction, we show that if P and Q are not strongly equivalent then P and Q are not uniformly equivalent. To start with, observe that $P^X = P$ holds for any positive program P and any set of literals X.

W.l.o.g., let (X,Y) be an SE-model of P but not of Q. By definition of SE-model we have $X \models P^Y$, i.e. $X \models P$. On the other hand, since (X,Y) is not an SE-model of Q, either (i) $X \not\models Q^Y$, i.e., $X \not\models Q$, or (ii) $Y \not\models Q$.

(i) Consider the programs $P_X = P \cup X$ and $Q_X = Q \cup X$. Clearly, $X \models P_X$ and for each $X' \subset X$, $X' \not\models P_X = P_X^X$. Hence, X is a stable model of P_X. On the other hand, $X \not\models Q$ and thus $X \not\models Q_X$. Hence, X cannot be a stable model of Q_X.

(ii) Consider the programs $P_Y = P \cup Y$ and $Q_Y = Q \cup Y$. Clearly, $Y \models P_Y$ and for each $Y' \subset Y$, $Y' \not\models P_Y = P_Y^Y$. Hence, Y is a stable model of P_Y. On the other hand, $Y \not\models Q$ and thus $Y \not\models Q_Y$. Hence, Y cannot be a stable model of Q_Y.

In any case we must conclude that P and Q are not uniformly equivalent. □

As known and easy to see from the main results of [52, 70, 71], on the class of positive programs classical equivalence and strong equivalence coincide. By combining this and the previous result, we obtain:

Theorem 4.24 *Let P and Q be positive DLPs. Then, $P \equiv_u Q$ if and only if $P \models Q$ and $Q \models P$, i.e., P and Q have the same set of classical models.*

Note that Sagiv [64] showed that uniform equivalence of datalog programs Π and Π' is equivalent to equivalence of Π' and Π over Herbrand models; this implies the above result for definite Horn programs. Maher [55] showed a generalization of Sagiv's result for definite Horn logic programs with function symbols.

Example 4.25 *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 Proposition 4.23*).

Head-cycle free programs The class of head-cycle free programs generalizes the class of NLPs by permitting a restricted form of disjunction. Still, it is capable of expressing nondeterminism such as a guess for the value of an atom a, which does not occur in the head of any other rule.

As shown by Ben-Eliyahu and Dechter, each head-cycle free program can be rewritten to an NLP, obtained by shifting atoms from the head to the body, which has the same stable models. More formally, let us define the following notation:

Definition 4.26 *For any rule* r, *let* $r^{\rightarrow} = \{r' \mid H(r') = \{a\}, a \in H(r), B^+(r') = B^+(r), B^-(r') = B^-(r) \cup H(r) - \{a\}\}$ *if* $H(r) \neq \emptyset$ *and* $r^{\rightarrow} = \{r\}$ *otherwise. For any DLP* P, *let* $P^{\rightarrow} = \bigcup_{r \in P} r^{\rightarrow}$.

It is well-known that for any head-cycle free program P, it holds that $P \equiv P^{\rightarrow}$ (cf. [4]). This result can be strengthened to uniform equivalence.

Theorem 4.27 *For any head-cycle free program* P, *it holds that* $P \equiv_u P^{\rightarrow}$.

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

We emphasize that a similar result for strong equivalence fails, as shown by the canonical counterexample in Example 4.8. Moreover, 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 $a \leftarrow \neg b$ and $b \leftarrow \neg a$; 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 [53, 28]. We discuss the question of eliminating disjunctions from programs in detail in Section 4.5.

We close this section with the following result, which provides a characterization of arbitrary programs which are strongly equivalent to their shift variant.

Theorem 4.28 *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) *such that* (*i*) $|H(r) \cap Y| \geq 2$ *and* (*ii*) $X \cap H(r) = \emptyset$ *and* $X \models B^+(r)$, *i.e.,* X *violates the reduced rule* r^Y.

Algorithm *Simplify*$(P, \equiv_e)$

Input: A DLP P and a notion of equivalence $\equiv_e$.
Output: A simplified DLP P' such that $P' \equiv_e P$.

var P' : DLP, I : changeInfo ;
$P' := P$; $I := all$;
while $I \neq \emptyset$ **do**
  $P' := \mathit{Simplify\_Syntactic}(P', \equiv_e, I)$;
  $P' := \mathit{Remove\_Redundant\_Rule}(P', \equiv_e, I)$;
  if $(I = \emptyset)$ **then** $P' := \mathit{Remove\_Redundant\_Literal}(P', \equiv_e, I)$; **fi**
od
return P';

Figure 6: Generic simplification algorithm.

Example 4.29 *Reconsider* $P = \{a \vee b \leftarrow\}$. *Then* $P^{\rightarrow} = \{a \leftarrow \neg b, b \leftarrow \neg a\}$ *has the SE-model* $(\emptyset, \{a, b\})$, *which satisfies the conditions* (i) *and* (ii) *for* $r : a \vee b \leftarrow$. *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,* $(\emptyset, \{a, b\})$ *is also an SE-model of* $P'^{\rightarrow}$. *Furthermore,* P *is also not uniformly equivalent to* $P'^{\rightarrow}$, *since* $(\emptyset, \{a, b\})$ *is moreover a UE-model of* $P'^{\rightarrow}$, *but* P *has the single SE-model (and thus UE-model)* $(\{a, b\}, \{a, b\})$.

4.2 Local Program Simplifications

We now deal with the issue of simplifying programs under different notions of equivalence. More specifically, given a logic program P and a notion of equivalence $\equiv_e$ $(e \in \{s, u\})$, we aim at a procedure to systematically simplify P to a program P' which is e-equivalent to P. As a starting point, we consider *local modifications*, i.e., iterative modifications of P on a rule-by-rule basis. Later on, in Section 4.4, we present results for replacing—under certain conditions—the *entire program* by a simpler, e-equivalent program.

Figure 6 gives the general structure of such a simplification algorithm, using different kinds of simplifications: A first simplification can be obtained by analyzing the syntax of the program and by applying appropriate syntactic transformation rules. After this "static" analysis, the notion of consequence induced by the given type of equivalence can be used for further semantic simplifications by means of consequence tests (see also next section). Systematic consequence testing can be applied to simplify rules by identifying redundant literals for removal, as well as for program simplification by identifying redundant rules.

The algorithm depicted in Figure 6 interleaves syntactic with semantic simplifications, giving preference to syntactic ones since they are easier to accomplish. Furthermore, it uses a data structure for recording latest changes (changeInfo I) in order to restrict the set of applicable transformation rules as well as to restrict their range of application: By means of this information, in each iteration, only a particular set of syntactic transformation rules has to be tested for a specific rule. Of course, this basic algorithm can be

Table 1: Syntactic transformation rules (r_1, r_2 are different rules).

Name	Condition	Transformation
TAUT	$H(r) \cap B^+(r) \neq \emptyset$	$P' = P - \{r\}$
RED$^+$	$a \in B^-(r_1)$, $\nexists r_2 \in P:\ a \in H(r_2)$	$P' = P - \{r_1\} \cup \{r'\}^\dagger$
RED$^-$	$H(r_2) \subseteq B^-(r_1)$, $B(r_2) = \emptyset$	$P' = P - \{r_1\}$
NONMIN	$H(r_2) \subseteq H(r_1)$, $B(r_2) \subseteq B(r_1)$	$P' = P - \{r_1\}$
GPPE	$a \in B^+(r_1)$, $G_a \neq \emptyset$, for $G_a = \{r_2 \in P \mid a \in H(r_2)\}$	$P' = P - \{r_1\} \cup G_a'^{\ddagger}$
WGPPE	same condition as for GPPE	$P' = P \cup G_a'^{\ddagger}$
CONTRA	$B^+(r) \cap B^-(r) \neq \emptyset$	$P' = P - \{r\}$
S-IMP	$r_1, r_2 \in P$, $r_2 \lhd r_1$	$P' = P - \{r_1\}$
LSH	$r \in P$, $\lvert H(r)\rvert > 1$, r head-cycle free in P	$P' = P - \{r\} \cup r^{\rightarrow}$

† $r' : H(r_1) \ \leftarrow\ B^+(r_1) \cup \neg(B^-(r_1) - \{a\})$.
‡ $G'_a = \{H(r_1) \cup (H(r_2) - \{a\}) \ \leftarrow\ (B^+(r_1) - \{a\}) \cup \neg B^-(r_1) \cup B(r_2) \mid r_2 \in G_a\}$.

further optimized to fit particular needs. Finally, we remark that *Simplify* is sound and that it is an anytime algorithm, since we use rewriting rules for a transformation-based calculus, where $P = P_0$ is transformed by applying rules T_i iteratively: $P = P_0 \xrightarrow{T_{i_1}} P_1 \xrightarrow{T_{i_2}} P_2 \cdots P_{k-1} \xrightarrow{T_{i_k}} P_k$, such that $P_j \equiv_e P_{j+1}$, $0 \leq j < k$.

4.2.1 Syntactic Transformation Rules

Basic transformation rules A set of basic syntactic transformation rules has been introduced and initially studied by Brass and Dix [6]. Table 1 briefly summarizes these rules, called *elimination of tautologies* (TAUT), *positive-* and *negative reduction* (RED$^+$ and RED$^-$, respectively), *partial evaluation* (GPPE), *weak partial evaluation* (WGPPE), *elimination of non-minimal rules* (NONMIN), and *elimination of contradictions* (CONTRA).

Osorio et al. [58] reported results for TAUT, RED$^+$, RED$^-$, GPPE, and NONMIN under strong equivalence. The positive results for TAUT, RED$^-$, and NONMIN directly carry over to uniform equivalence. The programs $P = \{a \leftarrow \neg b\}$ and $Q = \{a \leftarrow b;\, b \leftarrow c\}$ show that RED$^+$ and GPPE also fail to preserve uniform equivalence. Moreover, it is easily verified that CONTRA preserves both notions of equivalence.

While GPPE does neither preserve s-equivalence nor u-equivalence, both are preserved, however, by the weak variant WGPPE.

Proposition 4.30 *Let P be a DLP and consider $r_1, r_2 \in P$ such that $a \in B^+(r_1)$ and $a \in H(r_2)$, for some atom a. Then, $P \cup \{r'\} \equiv_s P$, where r' is given by $H(r_1) \cup (H(r_2) - \{a\}) \ \leftarrow\ (B^+(r_1) - \{a\}) \cup \neg B^-(r_1) \cup B(r_2)$.*

Note that WGPPE creates new rules and may make other transformations applicable. Furthermore, it may be applied exponentially often during iteration steps, creating further (even minimal) rules. Hence, to avoid an exponential explosion in the simplification algorithm, the number of applications of WGPPE may be polynomially bounded.

Table 2: Syntactic transformations preserving e-equivalence ($e \in \{s, u\}$).

Eq.	TAUT	RED$^+$	RED$^-$	NONMIN	GPPE	WGPPE	CONTRA	S-IMP	LSH
$\equiv_s$	yes*	no*	yes*	yes*	no*	yes	yes	yes	no
$\equiv_u$	yes	no	yes	yes	no	yes	yes	yes	yes

*Results due to Osorio et al. [58].

S-implication (S-IMP) Recently, Wang and Zhou [72] introduced the notion of s-implication:

Definition 4.31 *A rule r' is an* s-implication *of a rule $r \neq r'$, symbolically $r \lhd r'$, iff there exists a set $A \subseteq B^-(r')$ such that (i) $H(r) \subseteq H(r') \cup A$, (ii) $B^-(r) \subseteq B^-(r') - A$, and (iii) $B^+(r) \subseteq B^+(r')$.*

For example, if $r = a \vee b$ and $r' = a \leftarrow \neg b$, then $r \lhd r'$ (choose $A = \{b\}$).

Proposition 4.32 *Let P be a DLP, and $r, r' \in P$ such that $r \lhd r'$. Then, $P - \{r'\} \equiv_s P$.*

Example 4.33 *Consider the program $P = \{r_1 : a \leftarrow \neg b;\ r_2 : a \vee b \leftarrow\}$. Since $r_2 \lhd r_1$, P can be simplified to the strongly equivalent program $P' = \{a \vee b \leftarrow\}$.*

Local shifting (LSH) Finally, we present a new transformation rule, called *local shifting*. It relies on the concept of head-cycle freeness and turns out to preserve uniform equivalence, but not strong equivalence.

For any rule r, define $r^{\rightarrow} = \{a \leftarrow B(r), \neg(H(r) - \{a\}) \mid a \in H(r)\}$ if $H(r) \neq \emptyset$, and $r^{\rightarrow} = \{r\}$ otherwise. Furthermore, for any DLP P, let $P^{\rightarrow} = \bigcup_{r \in P} r^{\rightarrow}$.

Definition 4.34 *The rule* local shifting *(LSH) allows replacing a rule r in a DLP P by $r^{\rightarrow}$, providing r is head-cycle free in P.*

Theorem 4.35 *Let P be a DLP and $r \in P$ HCF in P. Then, $P \equiv_u (P - \{r\}) \cup r^{\rightarrow}$.*

Example 4.36 *Consider $P = \{a \vee b \leftarrow;\ c \vee d \leftarrow b;\ c \leftarrow a, d;\ d \leftarrow b, c\}$. Here, $a \vee b \leftarrow$ is HCF in P, whilst $c \vee d \leftarrow b$ is not. Hence, under $\equiv_u$, program P can be simplified to $P' = (P - \{a \vee b \leftarrow\}) \cup \{a \leftarrow \neg b; b \leftarrow \neg a\}$.*

Table 2 summarizes the results for syntactic transformation rules under both uniform and strong equivalence.

4.2.2 Semantic Transformation Rules

More powerful than the above syntactic transformation rules for program simplification are semantic ones based on $\models_e$, in the light of Propositions 4.4–4.18. However, checking the applicability of these rules is intractable in general (cf. Section 4.7.3 below), while checking the applicability of the above syntactic rules is tractable.

Brass and Dix [6] considered *supraclassicality* (SUPRA): If $P \models a$, then $P \equiv P \cup \{a \leftarrow\}$. However, SUPRA fails under $\equiv_u$ (and thus also under under $\equiv_s$). A simple counterexample is $P = \{a \leftarrow \neg b;\ a \leftarrow b;\ a \leftarrow b, c;\ b \leftarrow a, c\}$. Clearly, $P \models a$, but P has a UE-model $(\{c\}, \{a, b, c\})$ which is not a UE-model of $P \cup \{a \leftarrow\}$.

Rule simplification One way is minimizing individual rules r, based on property NONMIN: replace r by a subsumer r' (with $H(r') \subseteq H(r)$ and $B(r') \subseteq B(r)$) such that $(P - \{r\}) \cup \{r'\} \equiv_e P$. Rule simplification (or *condensation*) thus amounts to the systematic search for such r', which must satisfy $P \models_e r'$. The easiest way is literalwise removal, i.e., where r' is r with some literal in r removed.

Rule redundancy In addition, we can simplify P by removing redundant rules. Proposition 4.18 implies that $r \in P$ is redundant in P iff $P - \{r\} \models_e r$.

Notice that rule simplification might create redundant rules, as shown by the following example, which is well-known in the context of computing a minimal cover of a set of functional dependencies in databases.

Example 4.37 *Consider the program* $P = \{c \leftarrow a, b;\ a \leftarrow b, c;\ b \leftarrow a;\ b \leftarrow c\}$*. None of the rules in* P *is redundant, neither under strong nor under uniform consequence. However,* b *can be removed in both* $c \leftarrow a, b$ *and* $a \leftarrow b, c$*, yielding* $P' = \{c \leftarrow a;\ a \leftarrow c;\ b \leftarrow a;\ b \leftarrow c\}$*. Here,* $b \leftarrow c$ *or* $b \leftarrow a$ *can be eliminated.*

We remark that some of the syntactic transformation rules, such as TAUT, RED$^-$, NONMIN, and CONTRA, can be regarded as special cases of semantic transformation rules. However, others such as LSH, cannot be viewed this way.

We may bypass e-consequence and stay with relation $\models$ in some cases. This is true, e.g., for positive programs, in view of Corollary 4.21. Thus, for every positive program, a UE- and SE-canonical form is given by any irredundant prime CNF, written in positive rule form. Hence, minimization of any program which is semantically equivalent to a positive one is possible in polynomial time with an NP-oracle; detecting this property is considered in Section 4.4.

4.2.3 Computational Cost of *Simplify* and Incomplete Simplification

As pointed out above, testing the applicability and execution of each syntactic rule in Section 4.2.1 valid for $\equiv_e$ is clearly polynomial. Since all transformation rules except WGPPE shrink or remove program rules, *Simplify*$(P, \equiv_e)$ is polynomial with an oracle for $\equiv_e$ if WGPPE is applied at most polynomially often.

As discussed in Section 4.7.3, following from Corollary 4.21, deciding rule and literal redundancy is intractable in general. Interestingly, the following extension of Corollary 4.21 yields sound inference in terms of $\models$ for arbitrary DLPs.

Proposition 4.38 *For any DLP* P *and rule* r*,* $P^+ \models r$ *implies* $P \models_e r$*, for* $e \in \{s, u\}$*.*

However, the condition is not necessary; e.g., for $P = \{a \leftarrow b;\ a \leftarrow \neg b\}$ and $r = a \leftarrow \neg b$, we have $P \models_s r$ but $P^+ \not\models r$. Note that $P \models r$ for positive r does not imply $P \models_e r$, as shown by $P = \{a \leftarrow \neg a\}$ and $r = a \leftarrow$.

By exploiting Proposition 4.38, we can simplify any DLP P using its positive part P^+, without guaranteeing completeness. Notice that if P^+ is Horn (e.g., if P is a NLP), this sound redundancy test is polynomial, and is in coNP for general P^+. Thus, for P^+ being Horn it is cheaper than the sound and complete test for NLPs (which is coNP-complete for both $\models_s$ and $\models_u$), and for arbitrary P^+ for $\models_u$ (which is Π^P_2-complete; cf. Section 4.7.3).

4.3 Consequence Testing

In order to enable the systematic tests from the previous section by means of answer-set solvers, we provide here efficient translations of checking SE- and UE-consequence into (disjunctive) logic programs under the stable-model semantics.

We employ the following notation. For any set I of atoms, we define $I' = \{v' \mid v \in I\}$ and $\bar{I} = \{\bar{v} \mid v \in I\}$, where v' and $\bar{v}$ are globally new atoms, for each atom v. Furthermore, r' (resp., $\bar{r}$) denotes the result of replacing in r each occurrence of an atom v by v' (resp., $\bar{v}$). Informally, primed atoms will be used to characterize X in a UE- or SE-model (X, Y); whilst atoms of form $\bar{v}$ will be used to encode the negation of v.

4.3.1 Encoding SE-consequence

Based on results in [54, 71], a SAT encoding of $\models_s$ is easily obtained. In what follows, we describe an analogous encoding in terms of normal logic programs.

Definition 4.39 *For any DLP P and atom set $V = \{v_1, \ldots, v_n\}$, define $S_{P,V}$ as:*

$$
\begin{array}{lll}
(1) & v_i \leftarrow \neg\bar{v}_i; \quad \bar{v}_i \leftarrow \neg v_i, & 1 \leq i \leq n, \\
(2) & v_i' \leftarrow v_i, \neg\bar{v}_i'; \quad \bar{v}_i' \leftarrow \neg v_i', & 1 \leq i \leq n, \\
(3) & \leftarrow B^+(r), \neg B^-(r), \neg H(r), & r \in P, \\
(4) & \leftarrow B^+(r'), \neg B^-(r), \neg H(r'), & r \in P.
\end{array}
$$

Intuitively, (1) and (2) guess sets $X, Y \subseteq V$ such that $X \subseteq Y$, where atoms in X are represented by primed variables; (3) checks $Y \models P$; and (4) $X \models P^Y$.

The next result describes the desired encoding of $\models_s$.

Theorem 4.40 *Let P, Q be DLPs, V the set of atoms occurring in $P \cup Q$, and w a fresh atom. Then, $P \models_s Q$ iff the following NLP has no stable model:*

$$
\begin{array}{lll}
S_{P,V} \cup \{\leftarrow \neg w\} & \cup & \{w \leftarrow B^+(r), \neg B^-(r), \neg H(r); \\
& & \quad w \leftarrow B^+(r'), \neg B^-(r), \neg H(r') \mid r \in Q\}.
\end{array}
$$

4.3.2 Encoding UE-consequence

Definition 4.41 *Given a DLP P, a set of atoms $V = \{v_1, \ldots, v_n\}$, and sets of new atoms $V_i = \{v_{i,1}, \ldots, v_{i,n}\}$, for $i \in \{1, \ldots, n\}$, $U_{P,V}$ is defined as follows:*

$$
\begin{array}{lll}
(1) & S_{P,V}, & \\
(2) & v_{i,j} \leftarrow v_i, \bar{v}_i', v_j'; \quad v_{i,i} \leftarrow v_i, \bar{v}_i', & 1 \leq i, j \leq n, \\
(3) & \leftarrow v_i, \bar{v}_i', v_j, \bar{v}_j', \neg v_{i,j}; \quad \leftarrow v_i, \bar{v}_i', \bar{v}_j, v_{i,j}, & 1 \leq i, j \leq n, \\
(4) & H(r_i) \leftarrow v_i, \bar{v}_i', B^+(r_i), \neg B^-(r), & r \in P, 1 \leq i \leq n,
\end{array}
$$

where rule r_i is the result of replacing in r each atom v_j by $v_{i,j}$.

Intuitively, $U_{P,V}$ works as follows. First, $S_{P,V}$ yields all SE-models (X, Y) of P. Then, (2)–(4) check if no Z with $X \subset Z \subset Y$ satisfies $Z \models P^Y$. This is done as follows: For each $v_i \in Y - X$, (2) initializes sets $X_i = X \cup \{v_i\}$ via atoms V_i. Then, (4) yields, for each $v_i \in Y - X$, minimal sets $Z_i \supseteq X_i$ satisfying $Z_i \models P^Y$. Finally, constraints (3) apply for each such $Z_i \neq Y$. Hence, we get a stable model iff, for each i such that $v_i \in Y - X$, the minimal set Z_i matches Y. But then, $(X, Y) \in M_u(P)$. Formally, we have the following result:

Lemma 4.42 *Let P be a DLP over atoms $V = \{v_1, \ldots, v_n\}$. Then,* $\mathrm{SM}(U_{P,V})$ *is given by* $\{X' \cup (\bar{V}' - \bar{X}') \cup Y \cup (\bar{V} - \bar{Y}) \cup J_{X,Y} \mid X, Y \subseteq V, (X, Y) \in M_u(P)\}$*, where* $J_{X,Y} = \{v_{i,j} \mid v_i \in Y - X, v_j \in Y\}$.

Theorem 4.43 *Let P,Q be DLPs, V the set of atoms in $P \cup Q$, and u, w, s new atoms. Then, $P \models_u Q$ iff the program $C^U_{P,Q}$, given as follows, has no stable model:*

$$
\begin{array}{lll}
(1) & U_{P,V}, & \\
(2) & u \leftarrow B^+(r), \neg B^-(r), \neg H(r), & r \in Q, \\
(3) & u \leftarrow B^+(r'), \neg B^-(r), \neg H(r'), & r \in Q, \\
(4) & v''_i \leftarrow v'_i; \quad v''_i \leftarrow v_i, \neg \bar{v}''_i; \quad \bar{v}''_i \leftarrow \neg v''_i, & 1 \leq i \leq n, \\
(5) & w \leftarrow \neg v''_i, v_i; \quad s \leftarrow v''_i, \neg v'_i, & 1 \leq i \leq n, \\
(6) & \leftarrow \neg w, \neg u; \quad \leftarrow \neg s, \neg u, & \\
(7) & \leftarrow B^+(r''), \neg B^-(r), \neg H(r''), \neg u, & r \in Q.
\end{array}
$$

The intuition behind $C^U_{P,Q}$ is as follows. (1) computes all UE-models (X, Y) of P via atoms V' and V (characterizing X' resp. Y). We must check that each of them is also a UE-model of Q, and if so, the program should have no stable model. Therefore, (2) checks $Y \models Q$ and (3) $X \models Q^Y$. If this fails, $(X, Y) \notin M_s(Q)$, and thus $(X, Y) \notin M_u(Q)$. Whenever this is detected, we derive u, which is used to disable the constraints (6)–(7). Finally, we must check whether no Z exists such that $X \subset Z \subset Y$ and $(Z, Y) \in M_s(Q)$. To this end, (4) guesses a set Z such that $X \subseteq Z \subseteq Y$, and (5) derives w if some $p \in Y - Z$ exists, and s if some $q \in Z - X$ exists. Using (6), we remain with those Z which are in between X and Y. Finally, (7) checks $(Z, Y) \notin M_s(Q)$.

Now let I be any stable model of $C^U_{P,Q}$. If $u \in I$, then the members of V,V' in I give an UE-model of P which is not an SE-model of Q; and if $u \notin I$, then V, V'' give an SE-model (Z, Y) of Q where $(X, Y) \in M_u(P)$ with $X \subset Z$.

Notice that both $U_{P,V}$ and $C^U_{P,Q}$ are normal (resp., HCF) if P is normal (resp., HCF). Hence, the test for $P \models_u Q$ yields a normal (resp., HCF) program if P is normal (resp., HCF), which is in coNP compared to Π^P_2-completeness in general (cf. Section 4.7.1). $U_{P,V}$ is thus appealing from this point of view.

4.4 Global Program Simplifications

In this section, we consider the following issue: Given a DLP P from some class $\mathcal{C}$, under what conditions is there a program Q from a class $\mathcal{C}'$ such that $P \equiv_e Q$? This question is crucial for program transformation and simplification from a more general perspective than local transformations. We first deal with the case where $\mathcal{C}$ is the class of DLPs and $\mathcal{C}'$ is the class of NLPs, i.e., we are interested in eliminating disjunctions from programs.

4.5 Eliminating Disjunctions

In what follows, we provide general characterizations for eliminating disjunctions from programs. We first deal with a criterion for eliminating disjunctions under strong equivalence, and afterwards we show that such an elimination is always possible under uniform and ordinary equivalence.

4.5.1 Strong Equivalence

We start our analysis with some informal discussion. Consider the following logic programs, each of them having $r = a \vee b \leftarrow$ as its only disjunctive rule:

$$
\begin{aligned}
P_1 &= \{a \vee b \leftarrow\};\\
P_2 &= \{a \vee b \leftarrow; a \leftarrow\};\\
P_3 &= \{a \vee b \leftarrow; a \leftarrow b\};\\
P_4 &= \{a \vee b \leftarrow; a \leftarrow; \leftarrow \neg b\};\\
P_5 &= \{a \vee b \leftarrow; a \leftarrow b; \leftarrow \neg b\};\\
P_6 &= \{a \vee b \leftarrow; a \leftarrow; b \leftarrow\};\\
P_7 &= \{a \vee b \leftarrow; a \leftarrow b; b \leftarrow a\};\\
P_8 &= \{a \vee b \leftarrow; \leftarrow a, b\};\\
P_9 &= \{a \vee b \leftarrow; \leftarrow \neg a; \leftarrow \neg b\}.
\end{aligned}
$$

Let us first compute the SE-models (over $At = \{a, b\}$) of these programs:[2]

$$
\begin{aligned}
M_s(P_1) &= \{\ (ab, ab),\ (a, ab),\ (b, ab),\ (a, a),\ (b, b)\ \};\\
M_s(P_2) &= M_s(P_3) = \{\ (ab, ab),\ (a, ab),\ (a, a)\ \};\\
M_s(P_4) &= M_s(P_5) = \{\ (ab, ab),\ (a, ab)\ \};\\
M_s(P_6) &= M_s(P_7) = \{\ (ab, ab)\ \};\\
M_s(P_8) &= \{\ (a, a),\ (b, b)\ \};\\
M_s(P_9) &= \{\ (ab, ab)\ (a, ab),\ (b, ab)\ \}.
\end{aligned}
$$

A good approximation for obtaining strongly equivalent normal logic programs is to replace $a \vee b \leftarrow$ by the two rules $a \leftarrow \neg b$ and $b \leftarrow \neg a$, i.e., by applying the usual shifting technique [35, 15]. It is easy to see that this replacement works for P_2, P_4, P_6, and P_8, but not for P_1, P_3, P_5, P_7, and P_9. As a matter of fact, for the latter programs, this replacement yields an additional SE-model, $(\emptyset, ab)$.

In general, we have the following relation between the SE-models of a disjunctive rule and its corresponding shifted rules.

Proposition 4.44 *For a disjunctive rule, r, define*

$$
\begin{aligned}
r^{\rightarrow} &= \{p \leftarrow B(r), \neg(H(r) - \{p\}) \mid p \in H(r)\};\ \textit{and}\\
S_r &= \{(X, Y) \in INT_{At} \mid X \not\models H(r),\ X \models B^{+}(r),\\
&\qquad Y \cap B^{-}(r) = \emptyset,\ |Y \cap H(r)| > 1\}.
\end{aligned}
$$

[2] We write ab instead of $\{a, b\}$, and a instead of $\{a\}$, etc.

Then, $M_s(r^{\rightarrow}) = M_s(r) \cup S_r$.

Proof. Let r_p denote that rule in $r^{\rightarrow}$ with $H(r_p) = p$.

We first show that $S_r \subseteq M_s(r^{\rightarrow})$ and $M_s(r) \subseteq M_s(r^{\rightarrow})$. The former relation is easily seen, for each $r_p \in r^{\rightarrow}$: If $(X, Y) \in S_r$, then $|Y \cap H(r)| > 1$. Thus, $Y \cap B^-(r_p) \neq \emptyset$, yielding $(X, Y) \in M_s(r^{\rightarrow})$. For the latter relation, let $(X, Y) \in M_s(r)$. If $Y \not\models B(r)$ then $Y \not\models B(r_p)$, for each $r_p \in r^{\rightarrow}$. Hence, $(X, Y) \in M_s(r^{\rightarrow})$. So assume $Y \models B(r)$, and thus $Y \models H(r)$. Then, there exists some $p \in Y$ such that $p \in H(r)$. Hence, $Y \models r_p$. $(X, Y) \in M_s(r_p)$ is easily seen by similar arguments. Moreover, both $X \models (r^{\rightarrow})^Y$ and $Y \models r^{\rightarrow}$ hold, since for each $r_q \in r^{\rightarrow}$ with $p \neq q$, $Y \cap B^-(r_q) \neq \emptyset$ by definition.

It remains to show that $M_s(r^{\rightarrow}) \subseteq M_s(r) \cup S_r$. Therefore, consider some $(X, Y) \in M_s(r^{\rightarrow})$, and suppose that $(X, Y) \notin M_s(r)$. We show that $(X, Y) \in S_r$. Towards a contradiction, suppose that $(X, Y) \notin S_r$. Since $(X, Y) \notin M_s(r)$, we get by Proposition 4.22 that $X \not\models H(r)$, $Y \models B(r)$, and either $X \models B^+(r)$ or $Y \not\models H(r)$.

We consider two cases. First, if $Y \not\models H(r)$, i.e., $|Y \cap H(r)| = 0$, then we have a contradiction to the assumption that $(X, Y) \in M_s(r^{\rightarrow})$. Otherwise, if $Y \models H(r)$, we have $X \models B^+(r)$, and we get $|Y \cap H(r)| = 1$, otherwise $(X, Y) \in S_r$. Let $Y \cap H(r) = \{p\}$, and consider the rule r_p. Obviously, $Y \cap B^-(r_p) = \emptyset$. Moreover, we have $X \models B^+(r_p) = B^+(r)$. But then, $X \not\models H(r)$ yields $p \notin X$, which in turn contradicts $(X, Y) \in M_s(r^{\rightarrow})$. □

Observe that $M_s(r)$ and S_r are disjoint, in view of Proposition 4.22. Therefore, $S_r \cap M_s(P - \{r\})$ is the (possibly empty) set of additional SE-models generated by the shifting process. In the above examples, we have $r = a \vee b \leftarrow$, $r^{\rightarrow} = \{a \leftarrow \neg b;\ b \leftarrow \neg a\}$, and $S_r = \{(\emptyset, ab)\}$.

The question is how to eliminate these additional SE-models. A possible method is to add suitable rules to the programs resulting from replacing disjunctive rules by shifted ones. For the above examples, it can be shown that adding $a \leftarrow$ to P_3, P_5, and P_7 does the job, since $(\emptyset, ab)$ is not an SE-model of the rule $a \leftarrow$, and, for each $(X, Y) \in M_s(P_i)$, $(X, Y) \in M_s(a \leftarrow)$, for $i \in \{3, 5, 7\}$. However, for P_1 and P_9, this does not work. The problem here is that both (a, ab) and (b, ab) are contained in $M_s(P_1)$ and $M_s(P_9)$, and it is not possible to "delete" the undesired interpretation $(\emptyset, ab)$ without "deleting" one of the necessary models (a, ab) or (b, ab) as well. Indeed, there is no normal logic program which is strongly equivalent to P_1 or to P_9.

Now, what is the distinguishing feature in these examples? In all of the above programs, except for P_1 or P_9, we have the property that with (X, Y) and (X', Y) being SE-models, $(X \cap X', Y)$ is an SE-model too. To wit, in the case of programs P_1 and P_9, the pair of SE-models (a, ab) and (b, ab) violates this property, since $(\emptyset, ab)$ is not contained in $M_s(P_1)$, resp. $M_s(P_9)$. This basic observation motivates the following definition:

Definition 4.45 *Let P be a DLP over At. We say that P is* closed under here-intersection[3] *iff, for any pair (X, Y) and (X', Y) of SE-models of P, $(X \cap X', Y)$ is also an SE-model of P. We call $(X \cap X', Y)$ the* here-intersection of *(X, Y) and (X', Y), for any pair $(X, Y), (X', Y) \in INT_{At}$.*

[3] The term "here-intersection" derives from the logical underpinning of strong equivalence [52], given by the logic of here-and-there [59], in which the first component of an SE-interpretation refers to the world "here".

Hence, P_2–P_8 are closed under here-intersection, whilst P_1 and P_9 are not.

Lemma 4.46 *Each normal logic program is closed under here-intersection.*

Proof. Since P is normal, P^Y is Horn. Then, $X \models P^Y$ and $X' \models P^Y$ immediately implies $X \cap X' \models P^Y$, since each Horn program P' satisfies the following intersection property: if $X \models P'$ and $X' \models P'$, then $X \cap X' \models P'$. □

Hence, we can state the following result:

Theorem 4.47 *Let P be a DLP. If there exists a normal logic program Q such that P and Q are strongly equivalent, then P is closed under here-intersection.*

As discussed next, it turns out that closure under here-intersection is also a *sufficient* condition for the existence of a strongly equivalent normal program, given an arbitrary DLP. To this end, we introduce the following objects:

Definition 4.48 *Let P be a DLP over At, let $r \in P$ be a disjunctive rule, and let $P_r^- = P - \{r\}$. Then,*

$$P_{r,s} = P_r^- \cup r^{\rightarrow} \cup \hat{r}_{P_s},$$

where

$$\begin{aligned}
\hat{r}_{P_s} &= \bigcup_{(X,Y,Z) \in S_r^{\uparrow}(P)} r_{X,Y,Z}; \\
r_{X,Y,Z} &= \{p \leftarrow X, \neg(At - Z) \mid p \in Y\}; \text{ and} \\
S_r^{\uparrow}(P) &= \{(X,Y,Z) \mid (X,Z) \in S_r \cap M_s(P_r^-), \\
&\qquad X \subseteq Y,\ (Y,Z) \in M_s(P), \\
&\qquad \forall Y' : X \subseteq Y' \subset Y \Rightarrow (Y',Z) \notin M_s(P)\}.
\end{aligned}$$

Intuitively, $S_r^{\uparrow}(P)$ collects, for each $(X,Z) \in S_r$ which is also an SE-model of P_r^-, the minimal SE-models (Y,Z) of P "above" X (with fixed Z). Note that, by definition of S_r, for any $(X,Z) \in INT_{At}$, $(X,X,Z) \notin S_r^{\uparrow}(P)$, but $(X,Z) \in S_r$ implies the existence of an interpretation Y with $X \subseteq Y \subseteq Z$ such that $(X,Y,Z) \in S_r^{\uparrow}(P)$, since, at least for $Y = Z$, $(Y,Z) \in M_s(P)$ holds (again by definition of S_r).

The rules $r_{X,Y,Z}$, added in accord to the elements of $S_r^{\uparrow}(P)$, behave as follows:

Proposition 4.49 *For any sets $X, Y, Z \subseteq At$, we have that $(X',Y') \in INT_{At}$ is an SE-model of $r_{X,Y,Z}$ iff one of the following conditions holds: (i) $Y \subseteq X'$; (ii) $X \not\subseteq Y'$; (iii) $Y' \not\subseteq Z$; or (iv) $X \not\subseteq X'$ and $Y \subseteq Y'$.*

Proof. For $p \in Y$, let r_p denote the corresponding rule in $r_{X,Y,Z}$ with $H(r_p) = p$. Note that all rules in $r_{X,Y,Z}$ have the same body, $B = (X, \neg(At - Z))$. Clearly, $(X',Y') \in M_s(r_{X,Y,Z})$ iff $(X',Y') \in M_s(r_p)$, for each $p \in Y$. Applying Proposition 4.22 to each r_p yields $(X',Y') \in M_s(r_{X,Y,Z})$ iff (1) $Y' \not\models B$, or (2) for each $p \in Y$, $X' \models H(r_p)$ or jointly $X' \not\models B^+(r_p)$ and $Y' \models H(r_p)$ holds. Observe that (1) holds exactly if $X \not\subseteq Y'$ (i.e., Condition (ii)), or $Y' \cap (At - Z) \neq \emptyset$ (i.e., Condition (iii)) holds. It suffices to show that (2) is satisfied iff (i) or (iv) holds. Since $B^+(r_p) = X$, for each $p \in Y$, we proceed as follows. Assume $X' \not\models X$, i.e., $X \not\subseteq X'$. Then, either $X' \models H(r_p)$ or $Y' \models H(r_p)$, for each $p \in Y$. However, $Y' \models H(r_p)$ holds whenever $X' \models H(r_p)$. Hence, in this case, $Y' \models H(r_p)$ holds for all $p \in Y$. We thus arrive at $X \not\subseteq X'$ and $Y \subseteq Y'$, i.e., the properties of (iv). Otherwise, if $X \subseteq X'$, $X' \models H(r_p)$ must hold, for all $p \in Y$. Thus, $Y \subseteq X'$, i.e., Condition (i) holds. $\square$

Note that $r_{X,Y,Z}$ may contain redundant rules, e.g., if we have $X \subseteq Y$. It can be shown that then $r_{X,Y,Z} \equiv_s r_{X,Y-X,Z}$, which reduces the number of rules. However, for technical reasons we subsequently do not pay attention to this potential optimization.

Lemma 4.50 *If P is a DLP closed under here-intersection, then $M_s(P) = M_s(P_{r,s})$, for any disjunctive rule $r \in P$.*

Proof. First observe that, by Proposition 4.44, we have

$$\begin{aligned} M_s(P_{r,s}) &= M_s(P_r^- \cup r^{\rightarrow} \cup \hat{r}_{P_s}) = \\ &= M_s(P_r^-) \cap M_s(r^{\rightarrow}) \cap M_s(\hat{r}_{P_s}) \\ &= M_s(P_r^-) \cap (M_s(r) \cup S_r) \cap M_s(\hat{r}_{P_s}) \\ &= (M_s(P_r^-) \cap M_s(r) \cap M_s(\hat{r}_{P_s})) \cup \\ &\qquad (M_s(P_r^-) \cap S_r \cap M_s(\hat{r}_{P_s})) \\ &= (M_s(P) \cap M_s(\hat{r}_{P_s})) \cup \\ &\qquad (M_s(P_r^-) \cap S_r \cap M_s(\hat{r}_{P_s})). \end{aligned}$$

The strategy for the remainder of the proof is as follows. We first show that $T = M_s(P_r^-) \cap S_r \cap M_s(\hat{r}_{P_s}) = \emptyset$. This leads us to $M_s(P_{r,s}) = M_s(P) \cap M_s(\hat{r}_{P_s})$, and thus it remains to show that $M_s(P) = M_s(P) \cap M_s(\hat{r}_{P_s})$, i.e., that $M_s(P) \subseteq M_s(\hat{r}_{P_s})$ holds.

We show $T = \emptyset$. Let $(X, Z) \in S_r$. If $(X, Z) \notin M_s(P_r^-)$, we have $(X, Z) \notin T$. So suppose $(X, Z) \in M_s(P_r^-)$. Then, there exists a triple $(X, Y, Z) \in S_r^{\uparrow}(P)$. Hence, we can assume $r_{X,Y,Z} \subseteq \hat{r}_{P_s}$. Moreover, we have $X \subset Y \subseteq Z$. We now show that $(X, Z) \notin M_s(r_{X,Y,Z})$. Assume $(X, Z) \in M_s(r_{X,Y,Z})$. Then, by Proposition 4.49, one of the following conditions has to hold: (i) $Y \subseteq X$, (ii) $X \not\subseteq Z$, (iii) $Z \not\subseteq Z$, or (iv) $X \not\subseteq X$ and $Y \subseteq Z$. Condition (i) does not hold since we have $X \subset Y$; (ii) does not hold since we get $X \subset Z$ from $X \subset Y \subseteq Z$; and (iii) and (iv) do not hold trivially. We arrive at a contradiction, and we get $T = \emptyset$.

We now show that $M_s(P) \subseteq M_s(\hat{r}_{P_s})$ holds. Clearly, if $\hat{r}_{P_s}$ is empty, we are done, since then each SE-interpretation is also an SE-model of $\hat{r}_{P_s}$. So, suppose $\hat{r}_{P_s} \neq \emptyset$. We show $(X', Z') \in M_s(r_{X,Y,Z})$, for each $(X', Z') \in M_s(P)$ and each $r_{X,Y,Z} \subseteq \hat{r}_{P_s}$. Towards a contradiction, consider some $r_{X,Y,Z} \subseteq \hat{r}_{P_s}$ and some $(X', Z') \in M_s(P)$ such that $(X', Z') \notin M_s(r_{X,Y,Z})$. On the one hand, from $r_{X,Y,Z} \subseteq \hat{r}_{P_s}$, we have $(X, Y, Z) \in S_r^{\uparrow}(P)$, which implies that (a) $(X, Z) \in S_r \cap M_s(P_r^-)$; (b) $(Y, Z) \in M_s(P)$; and (c) $X \subset Y \subseteq Z$. On the other hand, by Proposition 4.49, we know from $(X', Z') \notin M_s(r_{X,Y,Z})$ that (1) $Y \not\subseteq X'$, (2) $X \subseteq Z'$, (3) $Z' \subseteq Z$, and (4) $X \subseteq X'$ or $Y \not\subseteq Z'$.

By assumption, $(X', Z') \in M_s(P)$. Hence, $X' \models P^{Z'}$ and $Z' \models P^{Z'}$. Moreover, by (3), $Z' \subseteq Z$ holds. By Proposition 4.1, we get $X' \models P^Z$ and $Z' \models P^Z$, and from (b) we get $Z \models P$. Therefore, $(X', Z) \in M_s(P)$ and $(Z', Z) \in M_s(P)$. Now, P is closed under here-intersection, yielding $(Y \cap X', Z) \in M_s(P)$ and $(Y \cap Z', Z) \in M_s(P)$. We use (4) to distinguish between the following two cases:

First, assume $X \subseteq X'$. By (c), $X \subset Y$, and thus $X \subseteq (Y \cap X')$. From (1), we have that $Y \not\subseteq X'$. This implies $(Y \cap X') \subset Y$. Hence, $X \subseteq (Y \cap X') \subset Y$ follows. Together with (a) and $(Y \cap X', Z) \in M_s(P)$, we obtain $(X, Y \cap X', Z) \in S_r^{\uparrow}(P)$, contradicting $(X, Y, Z) \in S_r^{\uparrow}(P)$.

Now assume $X \not\subseteq X'$. Similarly, from (4), we get $Y \not\subseteq Z'$, yielding $(Y \cap Z') \subset Y$. Moreover, by (2), $X \subseteq Z'$, and by (c), $X \subset Y$. Thus, $X \subseteq (Y \cap Z')$. Again, we have $X \subseteq (Y \cap Z') \subset Y$, and by (a) and $(Y \cap Z', Z) \in M_s(P)$, we arrive at a contradiction to $(X, Y, Z) \in S_r^{\uparrow}(P)$. □

Hence, by applying the above transformation successively for all disjunctive rules in a given DLP P, we eventually obtain a normal logic program strongly equivalent to P.

Theorem 4.51 *Let P be a DLP over At. If P is closed under here-intersection, then there exists a normal logic program Q over At such that P and Q are strongly equivalent.*

Therefore, in view of Theorem 4.47, a DLP P possesses a strongly equivalent NLP *precisely if* P is closed under here-intersection.

For illustration, let us apply the construction of $P_{r,s}$ to the examples P_2 and P_3 from the above, with $r = a \vee b \leftarrow$. Consider $S_r^{\uparrow}(P_2)$. Clearly, $S_r \cap M_s((P_2)_r^-) = \emptyset$, and r is just replaced by $r^{\rightarrow}$. But $S_r \cap M_s((P_3)_r^-) = \{(\emptyset, ab)\}$, and by the SE-models of P_3, we get $S_r^{\uparrow}(P_3) = \{(\emptyset, a, ab)\}$. Hence, we exchange r in P_3 by $r^{\rightarrow} \cup r_{\emptyset,a,ab}$, where $r_{\emptyset,a,ab} = \{a \leftarrow\}$ (under the assumption that $At = \{a, b\}$). For the other programs, the construction is similar.

4.5.2 Uniform Equivalence

If we change from strong to uniform equivalence, and retaining our overall strategy, the intuitive problems are very similar to those observed in the case of strong equivalence. But now an SE-model (X, Y) from S_r comes into play only if it is also a UE-model of the remaining program. Thus, if we want to eliminate such an SE-model, the problem of eliminating further SE-models, which should be retained, is less complicated compared to the case of strong equivalence. Roughly speaking, because of this difference, we are *always* able to construct a uniformly equivalent normal program. For instance, all our example programs P_1–P_9 except P_7 are uniformly equivalent to the program resulting from P_i by replacing $r = a \vee b \leftarrow$ by its shifting $r^{\rightarrow}$. For the program $P_7 = \{r; a \leftarrow b; b \leftarrow a\}$, however, we obtain $(\emptyset, ab)$ as additional SE- *and* UE-model of $(P_7 - \{a \vee b \leftarrow\}) \cup \{a \leftarrow \neg b; b \leftarrow \neg a\}$. Adding rules $a \leftarrow$ or $b \leftarrow$ (or both of them) circumvents this problem. Hence, in some cases, but in fewer than for strong equivalence, we again have to add further rules to achieve our goal. To wit, only rules $r_{X,Y,Z}$, for $(X, Y, Z) \in S_r^{\uparrow}(P)$, are used where $Y = Z$.

Definition 4.52 *Let P, r, P_r^-, $S_r^{\uparrow}(P)$, and $r_{X,Y,Z}$ be as in Definition 4.48, and define*

$$P_{r,u} = P_r^- \cup r^{\rightarrow} \cup \hat{r}_{P_u}, \ \textit{for} \ \hat{r}_{P_u} = \bigcup_{(X,Z,Z) \in S_r^{\uparrow}(P)} r_{X,Z,Z}.$$

In contrast to the case of strong equivalence, this transformation is always applicable in order to retain uniform equivalence.

Lemma 4.53 *Given a DLP P with $r \in P$ disjunctive, it holds that $M_u(P) = M_u(P_{r,u})$.*

Proof. First of all, observe that, analogously to the proof of Lemma 4.50, we get that $M_s(P_{r,u})$ is given by

$$(M_s(P) \cap M_s(\hat{r}_{P_u})) \cup (M_s(P_r^-) \cap S_r \cap M_s(\hat{r}_{P_u})). \tag{4}$$

We show $M_u(P) = M_u(P_{r,u})$. By Proposition 4.16, this holds iff both $M_u(P) \subseteq M_s(P_{r,u})$ and $M_u(P_{r,u}) \subseteq M_s(P)$ hold.

We first show $M_s(P) \subseteq M_s(P_{r,u})$, which clearly implies $M_u(P) \subseteq M_s(P_{r,u})$. Note that if $\hat{r}_{P_u}$ is empty, we are done, since then $M_s(P) \subseteq M_s(P) \cap M_s(\hat{r}_{P_u})$ holds trivially. So consider $\hat{r}_{P_u} \neq \emptyset$. We show $(X', Y') \in M_s(r_{X,Y,Y})$, for each $(X', Y') \in M_s(P)$ and each $r_{X,Y,Y} \subseteq \hat{r}_{P_u}$. Towards a contradiction, consider $r_{X,Y,Y} \subseteq \hat{r}_{P_u}$ and $(X', Y') \in M_s(P)$ such that $(X', Y') \notin M_s(r_{X,Y,Y})$. On the one hand, since $r_{X,Y,Y} \subseteq \hat{r}_{P_u}$, we have (a) $(X, Y) \in S_r \cap M_s(P_r^-)$, (b) $X \subset Y$, and (c) for each SE-interpretation (Z, Y) with $X \subseteq Z$, $(Z, Y) \in M_s(P)$ implies $Z = Y$. By Proposition 4.49, on the other hand, $(X', Y') \notin M_s(r_{X,Y,Y})$ yields (i) $Y \not\subseteq X'$, (ii) $X \subseteq Y'$, (iii) $Y' \subseteq Y$, and (iv) $X \subseteq X'$ or $Y \not\subseteq Y'$. We use (iv) for distinguishing between the following two cases:

First, assume $X \subseteq X'$. Clearly, $X' \subseteq Y'$. By (i), $X' \neq Y$, and by (iii), $Y' \subseteq Y$. We thus get $X \subseteq X' \subset Y$. Moreover, $X' \models P^{Y'}$ holds, since $(X', Y') \in M_s(P)$. By Proposition 4.1, we get $X' \models P^Y$. Furthermore, $Y \models P$ holds by (a). Hence $(X', Y) \in M_s(P)$, which clearly is in contradiction to (c).

Second, assume $X \not\subseteq X'$. By (iv), $Y \not\subseteq Y'$. Together with (iii), we thus have $Y' \subset Y$. Moreover, $X \subseteq Y'$ holds by (ii). Since $(X', Y') \in M_s(P)$, $Y' \models P^{Y'}$ holds. Proposition 4.1 yields $Y' \models P^Y$, and since $Y \models P$, we have $(Y', Y) \in M_s(P)$ with $X \subseteq Y' \subset Y$. Again, this is in violation to (c).

It remains to show that $M_u(P_{r,u}) \subseteq M_s(P)$. In particular, we show that $M_u(P_{r,u}) \cap T = \emptyset$ holds, where

$$T = M_s(P_r^-) \cap S_r \cap M_s(\hat{r}_{P_u}).$$

By inspecting (4), it can be seen that $M_u(P_{r,u}) \cap T = \emptyset$ implies $M_u(P_{r,u}) \subseteq M_s(P) \cap M_s(\hat{r}_{P_u})$, which proves the claim since $M_s(P) \cap M_s(\hat{r}_{P_u}) \subseteq M_s(P)$ holds trivially.

To derive $M_u(P_{r,u}) \cap T = \emptyset$, we show that for any $(X, Y) \in S_r \cap M_s(P_r^-)$, either $(X, Y) \notin M_u(P_{r,u})$ or $(X, Y) \notin M_s(\hat{r}_{P_u})$ holds. Fix a $(X, Y) \in S_r \cap M_s(P_r^-)$. We consider two cases.

Assume $(X, Y, Y) \notin S_r^{\uparrow}(P)$. Hence, there exists a set $X \subseteq X' \subset Y$ such that $(X', Y) \in M_s(P)$. We know that $(X, X, Y) \notin S_r^{\uparrow}(P)$. Thus, $X \subset X'$. We already have shown that $M_s(P) \subseteq M_s(P_{r,u})$, yielding $(X', Y) \in M_s(P_{r,u})$. But then, $(X, Y) \notin M_u(P_{r,u})$, since $X \subset X' \subset Y$.

So assume $(X, Y, Y) \in S_r^{\uparrow}(P)$, and thus $r_{X,Y,Y} \subseteq \hat{r}_{P_u}$. However, we have $(X, Y) \notin M_s(r_{X,Y,Y})$, since none of the following conditions, which hold by Proposition 4.49, is satisfied: (i) $Y \subseteq X$, (ii) $X \not\subseteq Y$, (iii) $Y \not\subseteq Y$, or (iv) $X \not\subseteq X$ and $Y \subseteq Y$. For (i) and (ii), this is seen by the fact that $(X, Y) \in S_r$, and thus $X \subset Y$; and (iii) and (iv) fail trivially. Hence, $(X, Y) \notin M_s(\hat{r}_{P_u})$. $\square$

Theorem 4.54 *For each DLP P, there exists a normal program Q such that $P \equiv_u Q$.*

As already discussed above, the only program from our examples P_1–P_9 which is not uniformly equivalent after replacing $r = a \vee b \leftarrow$ by $r^{\rightarrow}$ is P_7. However, since P_7 is closed under here-intersection, we already know how to derive a strongly (and thus uniformly) equivalent normal logic program. In fact, one can verify that $(P_7)_{r,s} = (P_7)_{r,u}$. For an example program P which is not closed under here-intersection and such that $\hat{r}_{P_u} \neq \emptyset$, with $r \in P$ disjunctive, consider $P = \{a \vee b \leftarrow;\ a \leftarrow c, b; b \leftarrow c, a\}$ over $At = \{a, b, c\}$. The SE-models of P are given as follows:

$$\begin{aligned} M_s(P) = \{&(abc, abc), (ab, abc), (ab, ab), (a, abc),\\ &(a, ab), (a, a), (b, abc), (b, ab), (b, b)\}. \end{aligned}$$

Indeed, P is not closed under here-intersection. S_r is given by $\{(\emptyset, ab), (\emptyset, abc), (c, abc)\}$, and $S_r \cap M_s(P_r^-) = S_r$ holds. Observe that in contrast to (c, abc), the remaining elements in S_r, i.e., $(\emptyset, ab)$ and $(\emptyset, abc)$, are not problematic, since adding them to $M_s(P)$ does not change the set of UE-models. $S_r^{\uparrow}(P)$ is given by the set

$$\{(\emptyset, a, ab), (\emptyset, b, ab), (\emptyset, a, abc), (\emptyset, b, abc), (c, abc, abc)\},$$

but only the last triple, (c, abc, abc), is applied in the construction of $\hat{r}_{P_u}$. In fact, we have to add

$$r_{c,abc,abc} \ = \ \{a \leftarrow c;\ b \leftarrow c;\ c \leftarrow c\}$$

to $P_r^- \cup r^{\rightarrow}$. For the resulting normal program $P_{r,u}$, we then have $M_s(P_{r,u}) = M_s(P) \cup \{(\emptyset, ab), (\emptyset, abc)\}$, but the "critical" SE-interpretation, (c, abc), has been eliminated. In fact, $M_u(P) = M_u(P_{r,u})$ holds, since neither $(\emptyset, ab)$ nor $(\emptyset, abc)$ is a UE-model of $P_{r,u}$.

4.5.3 Ordinary Equivalence

Finally, we discuss the case of ordinary equivalence. Since uniform equivalence implies ordinary equivalence, in view of Theorem 4.54, for any DLP P, there always exists an NLP Q such that P and Q are equivalent. Moreover, the normal program obtained by successive applications of transformation $P_{r,u}$ clearly does the job. Hence:

Theorem 4.55 *For each DLP P, there exists a normal program Q such that $P \equiv Q$.*

In fact, this result is also obtained by an enumeration of stable models.

Theorem 4.56 *Let P be a DLP and $r_{X,Y,Z}$ as in Definition 4.48.*
Then, $\mathrm{SM}(P) = \mathrm{SM}(\tilde{P})$, *with* $\tilde{P} = \bigcup_{Y \in \mathrm{SM}(P)} r_{\emptyset,Y,Y}$.

Proof. To begin with, we note the following property, which is a simple consequence of Proposition 4.49:

$(*)$ For any $Y \subseteq At$, $M_s(r_{\emptyset,Y,Y}) = \{(X', Y') \in \mathit{INT}_{At} \mid Y \subseteq X' \text{ or } Y' \not\subseteq Y\}$.

Consider $Y \in \mathrm{SM}(P)$. Then, for each $Y' \in \mathrm{SM}(P)$, either $Y \not\subseteq Y'$ or $Y = Y'$. By $(*)$, $(Y,Y) \in M_s(\tilde{P})$. Towards a contradiction, suppose $Y \notin \mathrm{SM}(\tilde{P})$. By Proposition 4.5, there exists a $X \subset Y$ such that $(X,Y) \in M_s(\tilde{P})$. In particular, we must have $(X,Y) \in M_s(r_{\emptyset,Y,Y})$. But by $(*)$, this is impossible in view of $X \subset Y$. Therefore, $Y \in \mathrm{SM}(\tilde{P})$. This proves $\mathrm{SM}(P) \subseteq \mathrm{SM}(\tilde{P})$.

Suppose there is some $Y \in \mathrm{SM}(\tilde{P})$ such that $Y \notin \mathrm{SM}(P)$. That is, $(X,Y) \in M_s(P)$ for some $X \subset Y$. By Proposition 4.1, we get $(X,Z) \in M_s(P)$ for each $Y \subseteq Z$. Thus, for each $Y' \in \mathrm{SM}(P)$, $Y \not\subseteq Y'$ holds. Hence, we have $(X,Y) \in M_s(r_{\emptyset,Y',Y'})$, for each $Y' \in \mathrm{SM}(P)$, by $(*)$. Thus, $(X,Y) \in M_s(\tilde{P})$ with $X \subset Y$ But this contradicts $Y \in \mathrm{SM}(\tilde{P})$, and thus $\mathrm{SM}(P) = \mathrm{SM}(\tilde{P})$ must hold. □

Finally, we also note the following transformation for ordinary equivalence, following the line of $P_{r,s}$ and $P_{r,u}$, but being more compact with respect to ordinary equivalence.

Lemma 4.57 *Let P, r, P_r^-, $S_r^{\uparrow}(P)$, and $r_{X,Y,Z}$ be as in Definition 4.48, and define*

$$P_{r,e} = P_r^- \cup r^{\rightarrow} \cup \hat{r}_{P_e},$$

where

$$\hat{r}_{P_e} = \bigcup_{\substack{(X,Z,Z) \in S_r^{\uparrow}(P),\\ Z \in \mathrm{SM}(P)}} r_{X,Z,Z}.$$

Then, $\mathrm{SM}(P) = \mathrm{SM}(P_{r,e})$.

Proof. We first show that,

$(*)$ for each $Y \subseteq At$, $Y \models P$ iff $Y \models P_{r,e}$.

Indeed, from Lemma 4.53, we have $M_u(P) = M_u(P_{r,u})$, and thus $Y \models P$ iff $Y \models P_{r,u}$. By definition, $P_{r,e} \subseteq P_{r,u}$, from which the only-if direction is an immediate consequence. For the proof of the if-direction, assume $Y \models P_{r,e}$. Then, $Y \models P_r^- \cup r^{\rightarrow}$. By classical logic, this clearly implies $Y \models P$.

We proceed with the proof of the lemma. First, fix some $Y \in \mathrm{SM}(P)$. Then, $Y \models P$, and by $(*)$, $Y \models P_{r,e}$. It remains to show that no $X \subset Y$ yields an SE-model (X,Y) of $P_{r,e}$. Towards a contradiction, suppose some $X \subset Y$ exists such that $(X,Y) \in M_s(P_{r,e})$. Clearly, $(X,Y) \in M_s(P_r^-)$, hence, since $Y \in \mathrm{SM}(P)$, $(X,Y) \notin M_s(r)$ must hold. Then, by Proposition 4.44, $(X,Y) \in S_r$. Therefore, $(X,Y) \in S_r \cap M_s(P_r^-)$ and $Y \in \mathrm{SM}(P)$, and we thus get $r_{X,Y,Y} \in P_{r,e}$ by construction. By Proposition 4.49, $(X,Y) \in M_s(r_{X,Y,Y})$ only if $X = Y$. Thus $(X,Y) \notin M_s(P_{r,e})$, which is a contradiction. Hence, $Y \in \mathrm{SM}(P_{r,e})$, which proves $\mathrm{SM}(P) \subseteq \mathrm{SM}(P_{r,e})$.

Now consider some $Y \in \mathrm{SM}(P_{r,e})$. By $(*)$, we get $Y \models P$. Assume $Y \notin \mathrm{SM}(P)$, i.e., there exists a set $X \subset Y$ such that $(X,Y) \in M_s(P)$. By Proposition 4.44, $(X,Y) \in M_s(P_r^- \cup r^{\rightarrow})$. Since $Y \models P$, and thus $Y \models P^Y$, we get by Proposition 4.1 that $Y \models P^{Y'}$, for each Y' with $Y \subseteq Y'$. Hence, $Y \not\subseteq Y'$, for each $Y' \in \mathrm{SM}(P)$. By Proposition 4.49, $(X,Y) \in M_s(r_{X,Y',Y'})$, for each Y' with $Y \not\subseteq Y'$. Hence, $(X,Y) \in M_s(P_{r,e})$. By Proposition 4.5, this contradicts $Y \in \mathrm{SM}(P_{r,e})$. Thus $Y \in \mathrm{SM}(P)$ must hold, which proves $\mathrm{SM}(P_{r,e}) \subseteq \mathrm{SM}(P)$. □

To summarize, given a DLP P with $r \in P$ disjunctive, we are able to construct (via a replacement of r by normal rules)

- a logic program $P_{r,e}$ which is ordinary equivalent to P;
- a program $P_{r,u}$ which is uniformly equivalent to P; and
- a program $P_{r,s}$ which is strongly equivalent to P, whenever P is closed under here-intersection.

All these programs are of the form

$$P_{r,\alpha} = P_r^- \cup r^\to \cup \hat{r}_{P_\alpha}, \text{ for } \alpha \in \{e, u, s\},$$

and satisfy

$$P_{r,e} \subseteq P_{r,u} \subseteq P_{r,s}.$$

Recall that successive applications of these rulewise transformations, for all disjunctive rules in a given program, leads to a normal logic program. Hence, our method can be seen as a uniform framework for obtaining normal logic programs from disjunctive logic programs with respect to all important notions of equivalence. Moreover, our results extend and generalize methods based on shifting techniques, since the outcome of these methods coincides with the rewriting $P_{r,\alpha}$, whenever $\hat{r}_{P_\alpha}$ is empty. In particular, concerning equivalence in terms of stable models, we presented a semantic criterion (in contrast to the syntactic criterion of head-cycle freeness as discussed in [4]) which allows for shifting.

We note that the size of the outcoming programs in our method is in general exponential in the size of the input program. However, as we discuss in Section 4.7.2, this exponential increase is, in a certain sense, unavoidable from a complexity-theoretic point of view.

4.6 More General Global Simplifications

After having discussed general conditions under which disjunctions can be eliminated from programs, we now deal with the question when a given DLP is equivalent to a positive or a Horn program.

Theorem 4.58 *Let P be a DLP. Then, there exists a positive program Q such that $P \equiv_e Q$ iff, for all $(X, Y) \in M_e(P)$, it holds that $X \models P$, where $e \in \{s, u\}$.*

Theorem 4.59 *For any DLP P, let $M_e^1(P) = \{X \mid (X, Y) \in M_e(P)\}$, for $e \in \{s, u\}$. Then, there exists some Horn program Q such that $P \equiv_e Q$ iff (i) $M_e^1(P) \models P$, and (ii) $M_e^1(P)$ is closed under intersection.*

Note that Theorem 4.58 implies that if P is e-equivalent to some positive program Q, then we can obtain Q by shifting $\neg$-literals to the head. We can exploit this observation for a simple test. Let, for any rule r, r^{ls} be the rule such that $H(r^{ls}) = H(r) \cup B^-(r)$, and $B(r^{ls}) = B^+(r)$ (i.e., the "left shift" of r).

Theorem 4.60 *Let P be any DLP and $e \in \{s, u\}$. Then, there exists a positive program Q such that $P \equiv_e Q$ iff $P \models_e r^{ls}$, for every $r \in P$ such that $B^-(r) \neq \emptyset$.*

To test for the existence of an e-equivalent Horn program Q, we just need to add a check whether the models of P are intersection closed (which is in coNP).

In the case of $\models_s$, we can even get rid of SE-consequence and come up with a test using $\models$ only. In the following, we characterize strong equivalence of any program P, possessing a classical model, to some positive program Q in terms of $\models$; if P has no classical model, then $P \equiv_s \{\bot\}$. Note that $\{\bot\}$ is positive.

Recall that $(X, Y) \in M_e$ requires $X \subseteq Y$, but the reduct P^Y of program P may have models X' such that $X' \not\subseteq Y$. To select exactly those X' such that $X' \subseteq Y$, let, for any DLP P and interpretation I, $P^I_\leq$ be the positive logic program $P^I \cup \{ \leftarrow a \mid I \not\models a\}$.

Proposition 4.61 *Let P be a DLP having a classical model, and let r be a positive rule. Then, $P \models_s r$ iff $P^M_\leq \models r$, for every maximal model M of P.*

Note that the proposition fails for non-positive r. Indeed, consider the program $P = \{a \leftarrow b;\ a \leftarrow \neg b\}$ and $r = a \leftarrow \neg b$. The unique maximal model of P is $At = \{a, b\}$, and $P^{At}_\leq = \{a \leftarrow b\}$, but $P^{At}_\leq \not\models a \leftarrow \neg b$.

Theorem 4.62 *Let P, Q be DLPs, where P is positive having a classical model. Then, $P \equiv_s Q$ iff (i) $P \models Q$, and (ii) $Q^M_\leq \models P$, for every maximal model M of P.*

We finally arrive at the desired characterization of semantic equivalence to some positive program, by exploiting that P^+ is a singular candidate for Q.

Theorem 4.63 *Let P be any DLP without positive constraints. Then, there exists some positive program Q such that $P \equiv_s Q$ iff $P^+ \models P - P^+$.*

For strong equivalence of P to some Horn theory Q, we have in the right-hand side the additional condition that the models of P^+ are closed under intersection.

We close this subsection with remarking that in parallel to our work, further interesting results on program optimization for logic programs under the answer set semantics have been obtained based on provability in intermediate logics [60], which might be incorporated into our optimization framework.

4.7 Complexity Issues

This section deals with complexity issues related to the results discussed above. We first analyse the computational complexity of uniform equivalence. While our main interest is with the problem of deciding uniform equivalence between two given programs, we also consider the related problems of UE-model checking and UE-consequence. Afterwards, in Section 4.7.2, we address the complexity of checking closure of a DLP under here-intersection. Moreover, we investigate the expressiveness of the class of DLPs closed under here-intersection. As we show, this class resides at the same level of the polynomial hierarchy (PH) as the class of arbitrary DLPs. We also show that the exponential increase of program size in the worst case of our general rewriting method of Section 4.5 is unavoidable, providing the PH does not collapse. We conclude our complexity analysis with results regarding the local simplification rules, as well as global simplifications involving positive, Horn, constraint-free, and definite programs.

4.7.1 Complexity of Uniform Equivalence

For UE-model checking, we have the following result. Let $\|\alpha\|$ denote the size of an object α.

Theorem 4.64 *Given a pair of sets (X, Y) and a program P, deciding whether $(X, Y) \in M_u(P)$ 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.*

Corollary 4.65 *UE-model checking for Horn programs is polynomial.*

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.

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 (i.e., in $\Sigma_2^P = \text{NP}^{\text{NP}}$).

Lemma 4.66 *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 4.12 guess an SE-model (X, Y) such that (X, Y) is an UE-model of exactly on of the programs P and Q. By Theorem 4.64, 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 4.67 *Given two DLPs P and Q, deciding $P \equiv_u Q$ is Π_2^P-complete.*

Proof. (Sketch) Membership in Π_2^P has already been established in Lemma 4.66. 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 uniform equivalence of two DLPs P and Q.

Consider a $QBF_{2,\exists}$ F of form $F = \exists X \forall Y \bigvee_{i=1}^{i=l} D_i$, where each D_i is a conjunct of at most three literals over the Boolean variables in $X \cup Y$, $X = \{x_i \mid 1 \leq i \leq n\}$ and $Y = \{y_i \mid 1 \leq i \leq m\}$. Deciding whether a given such F is true is well-known to be Σ_2^P-complete; thus deciding whether F is false is Π_2^P-complete.

W.l.o.g., we assume that each D_i contains some literal over Y. Now let P and Q be the following programs:

$$
\begin{array}{lllllll}
P = \{ & x_i \vee x_i' \leftarrow . & 1 \leq i \leq n; & & \\
& y_i \vee y_i' \leftarrow y_j. & 1 \leq i \neq j \leq m; & & \\
& y_i \vee y_i' \leftarrow y_j'. & 1 \leq i \neq j \leq m; & & \\
& w_0 \leftarrow x_i, x_i'. & 1 \leq i \leq n; & w_1 \leftarrow y_i, y_i'. & 1 \leq i \leq m; \\
& x_i \leftarrow w_0. & 1 \leq i \leq n; & w_1 \leftarrow D_i^*. & 1 \leq i \leq l; \\
& x_i' \leftarrow w_0. & 1 \leq i \leq n; & y_i \leftarrow w_1. & 1 \leq i \leq m;
\end{array}
$$

$$
\begin{array}{lllll}
y_i \;\leftarrow\; w_0. & 1 \leq i \leq m; & y_i' \;\leftarrow\; w_1. & 1 \leq i \leq m; & \\
y_i' \;\leftarrow\; w_0. & 1 \leq i \leq m; & w_1 \;\leftarrow\; \neg w_1. & & \\
w_1 \;\leftarrow\; w_0. & & & & \},
\end{array}
$$

and $\quad Q = P \cup \{y_1 \vee y_1' \;\leftarrow\; .\},$

where D_i^* results from D_i by replacing literals $\neg x_i$ and $\neg y_i$ by x_i' and y_i', respectively.

Informally, the disjunctive clauses with x_i and x_i' in the head resp. y_i and y_i' serve for selecting a truth assignment to the variable x_i (resp., y_i). The atom w_0 serves for handling a spoiled assignment to some x_i, which occurs if both x_i and x_i' are true, and enforces the maximal interpretation as the unique model of P, by the rules with w_0 in the body. Similarly, w_1 recognizes a spoiled assignment to some variable y_i or that some disjunct D_j of the QBF is true, and enforces for all atoms y_i, y_i' and w_1 the unique value true. However, for P the selection of a truth assignment is conditional to the truth of any atom y_i or y_i', while for Q it is mandatory by the additional rule $y_1 \vee y_1' \leftarrow$. This difference leads to a suite of candidate SE-models (A_χ, M_χ) of P which do not satisfy Q, where χ corresponds to a truth assignment to X and all atoms not in $X \cup X'$ are false, such that (A_χ, M_χ) violates uniform equivalence of P and Q via (*ii*) just if there is no way to make all D_j false by some assignment μ to Y. These candidate models for spoiling uniform equivalence are eliminated iff formula F evaluates to false. Since P and Q are obviously constructible in polynomial time, our result follows. □

The previous result shows that deciding uniform equivalence of DLPs P and Q is more complex than deciding strong equivalence, which is in coNP [71]. Thus, the more liberal notion of uniform equivalence comes at higher computational cost in general. However, for important classes of programs, it has the same complexity.

Theorem 4.68 *Let P and Q be DLPs without simultaneous negation and head-cycles (i.e., each program is either positive or head-cycle free). Then, deciding $P \equiv_u Q$ is* coNP-*complete, where* coNP-*hardness holds if P is either positive or a NLP, and Q is Horn.*

Note that Sagiv [64] showed that deciding $P \equiv_u Q$ for given definite Horn programs P and Q is polynomial. This clearly generalizes to arbitrary Horn programs. We further remark that for NLPs deciding strong equivalence is also coNP-hard.

Finally, we complement the results on uniform equivalence and UE-model checking with briefly addressing the complexity of UE-consequence.

Theorem 4.69 *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.*

Proof. (Sketch) The complementary problem, $P \not\models_u r$, is in Σ_2^P for general P and in NP for head-cycle free P, since a guess for a UE-model (X, Y) of P which violates r can, by Theorem 4.64 be verified with a call to a NP-oracle resp. in polynomial time. In case of a positive P, by Theorem 4.20, $P \models_u r$ iff $P \models r$, which is in coNP for general P and polynomial for Horn P. The hardness results can be obtained by adapting the constructions in hardness proofs of previous results. □

We remark that the reduction in the proof of Theorem 4.67 contains a single rule with negation ($w_1 \leftarrow \neg w_1$). This rule can be equivalently replaced, for our purposes, with the constraint $\leftarrow \neg w_1$. This shows that a positive disjunctive programs are a maximal class for which uniform equivalence is "easier" than in the general case. In other words, a single use of negation in combination with disjunction is sufficient to reach the full complexity of uniform equivalence.

4.7.2 Complexity of Disjunction Elimination

Checking closure under here-intersection We can express testing a DLP P for being closed under here-intersection via the following normal logic program, which is linear in the size of P. To this end, for any rule r, let r'_i denote the rule obtained from r by replacing each occurrence of an atom p_i in r by p'_i.

Definition 4.70 *Let P be a DLP over atoms V. For each atom $v \in V$, let $\bar{v}$, v'_1, $\bar{v}'_1$, v'_2, $\bar{v}'_2$, v'_3 be pairwise distinct new atoms, and let u be a new atom. Define P_Q as the program containing the following items:*

1. *for each $v \in V$ and each $i \in \{1, 2\}$,*

$$v \leftarrow \neg \bar{v}; \qquad \bar{v} \leftarrow \neg v; \tag{5}$$
$$v'_i \leftarrow v, \neg \bar{v}'_i; \quad \bar{v}'_i \leftarrow \neg v'_i; \tag{6}$$

2. *for each $r \in P$ and each $i \in \{1, 2\}$,*

$$\leftarrow B(r), \neg H(r); \tag{7}$$
$$\leftarrow B^+(r'_i), \neg B^-(r), \neg H(r'_i); \tag{8}$$

3. *for each $v \in V$,*

$$v'_3 \leftarrow v'_1, v'_2; \tag{9}$$

4. *for each $r \in P$,*

$$u \leftarrow B^+(r'_3), \neg B^-(r), \neg H(r'_3); \tag{10}$$

and

5. *the constraint*

$$\leftarrow \neg u. \tag{11}$$

Intuitively, the program P_Q works as follows. Rules of form (5) guess an interpretation Y of P, and rules of form (7) check that Y is a model of P. Similarly, rules of form (6) guess subsets X_1 and X_2 of Y such that both are models of P^Y, which is enforced by the constraints of form (8). Hence, the program consisting of all rules of form (5)–(8) "computes" all pairs of SE-models (X_1, Y) and (X_2, Y) of P.

Now, the rules of form (9) compute the intersection $X_1 \cap X_2$, and via the rules of form (10) the new atom u can be derived iff the intersection does not satisfy P^Y, i.e., iff $(X_1 \cap X_2, Y)$ is no SE-model of P. The constraint (11) eliminates all models of P_Q for which this is not the case, i.e., for which $(X_1 \cap X_2, Y)$ is an SE-model of P. Thus, items (9)–(11) ensure that P_Q has no stable model iff P is closed under here-intersection. Formally, we have:

Lemma 4.71 *A DLP P is closed under here-intersection iff* $\mathrm{SM}(P_Q) = \emptyset$.

Based on this, we derive the following complexity result.

Theorem 4.72 *Checking closure under here-intersection, for a given DLP, is* coNP-*complete.*

Proof. By Lemma 4.71 and the linear encoding from Definition 4.70, we get that closure under here-intersection is in coNP.

We show coNP-hardness by a reduction to the coNP-complete problem of deciding whether a given interpretation is the unique model of a positive DLP as follows:

Let P be a positive DLP over atoms $V = \{v_1, \ldots, v_n\}$, let q, q' be new atoms, and consider the program

$$Q = P \cup \{q \vee q' \leftarrow;\ q \leftarrow v_1, \ldots, v_n;\ q' \leftarrow v_1, \ldots, v_n\}.$$

We show that Q is closed under here-intersection iff V is the unique model of P.

First, if V is the unique model of P then, by construction, $V \cup \{q, q'\}$ is the unique model of Q, and since Q is positive—and hence constant under reduction—Q is trivially closed under here-intersection.

Second, for the only-if direction, assume that Q is closed under here-intersection. Towards a contradiction, assume that there is a model $M \subset V$ of P. Then, both $M \cup \{q\}$ and $M \cup \{q'\}$ are models of Q, and thus also both $(M \cup \{q\}, V \cup \{q, q'\})$ and $(M \cup \{q'\}, V \cup \{q, q'\})$ are SE-models of Q. However, $(M, V \cup \{q, q'\})$ is not an SE-model of Q, since $M \not\models q \vee q' \leftarrow$. This contradicts our assumption that Q is closed under here-intersection. Hence, V is the unique model of P. □

In view of Theorem 4.51, the above result immediately implies the following property:

Corollary 4.73 *Checking whether, for a given DLP P, there exists a normal program Q such that P and Q are strongly equivalent, is* coNP*-complete.*

Expressiveness of here-intersection closed DLPs We now consider the expressiveness of the class of DLPs closed under here-intersection. We show that this class of programs possesses the same worst-case complexity as arbitrary DLPs. More specifically, the relevant reasoning tasks in the context of DLPs are

- checking the existence of a stable model of a given DLP ("consistency problem");
- checking whether a given atom belongs to at least one stable model of a given DLP ("brave reasoning"); and

- checking whether a given atom belongs to all stable models of a given DLP ("cautious reasoning").

As shown in [26], for arbitrary disjunctive programs, the consistency problem and brave reasoning are Σ_2^P-complete, whilst cautious reasoning is Π_2^P-complete. Moreover, the respective hardness results for these problems hold even for a quite restricted class of DLPs. This class comprises DLPs where each disjunctive rule is a fact and where each stable model contains an *exact hitting set*[4] for the collection of disjunctive facts in P.

We define the following kinds of programs:

Definition 4.74 *A DLP P is called*

1. *a* disjunctive-fact program *(DFP) iff each disjunctive rule in P has an empty body, i.e., for each $r \in P$, $|H(r)| > 1$ implies $B(r) = \emptyset$; and*
2. *a* hitting-set program *iff, for each stable model $I \in \mathrm{SM}(P)$, $|H(r) \cap I| = 1$, for each disjunctive rule $r \in P$.*

DLPs satisfying both conditions are called hitting-set DFPs *(HDFPs).*

Observe that HDFPs are in general *not* closed under here-intersection, as seen by the program $\{a \vee b \leftarrow\}$. However, we construct a polynomial-time translation mapping each DFP P into a DLP Q (over an extended alphabet) such that (i) Q is closed under here-intersection and (ii) there is a one-to-one correspondence (over the original alphabet) between the stable models of P and the stable models of Q, whenever P is also a HDFP. From this, we derive the same lower complexity bounds for DLPs closed under here-intersection as for arbitrary DLPs.

We employ the following notation: For a given alphabet At, $\overline{At} = \{\overline{p} \mid p \in At\}$ is a set of globally new disjoint atoms. Accordingly, for a rule r, $\overline{r}$ is the rule resulting from r by replacing each atom p in r by $\overline{p}$. Finally, let $At^* = At \cup \overline{At}$.

Definition 4.75 *For a DLP P over At, let P^+ be the program resulting from P by adding $\{p \leftarrow \overline{p} \mid p \in At\}$ and, for each r with $|H(r)| > 1$, adding $(\overline{r})^{\rightarrow} \cup \{\leftarrow \overline{p}, \overline{q} \mid p, q \in H(r), p \neq q\}$.*

Lemma 4.76 *For a DFP P, we have that $M_s^{At^*}(P^+)$ is given by those $(X, Y) \in INT_{At^*}$ which satisfy the following conditions:*

1. *$(X, Y) \in M_s^{At^*}(P)$;*
2. *$X \cap \overline{At} \subseteq \overline{(X \cap At)}$, $Y \cap \overline{At} \subseteq \overline{(Y \cap At)}$; and*
3. *for each $r \in P$ with $|H(r)| > 1$,*

$$X \cap H(\overline{r}) = Y \cap H(\overline{r}) = \{\overline{p}\}.$$

[4]The *exact hitting set* problem is as follows. Given a collection C of subsets of a set S, decide whether there exists a subset $S' \subseteq S$ such that $|S' \cap C'| = 1$, for each $C' \in C$. This problem is known to be NP-complete [45].

Proof. First, $M_s^{At^*}(P \cup \{p \leftarrow \overline{p} \mid p \in At\})$ is clearly given by those $(X, Y) \in INT_{At^*}$ which satisfy Conditions 1 and 2. Furthermore, consider a disjunctive fact $r \in P$. Applying Proposition 4.44, we get $M_s^{At^*}((\overline{r})^{\rightarrow}) = M_s^{At^*}(\overline{r}) \cup S_{\overline{r}}$, which in turn is given by

$$\{(X, Y) \in INT_{At^*} \mid H(\overline{r}) \cap X \neq \emptyset \text{ or } H(\overline{r}) \cap X = \emptyset; |Y \cap H(\overline{r})| > 1\}.$$

In the construction of P^+, it remains to consider the rules $R = \{\leftarrow \overline{p}, \overline{q} \mid p, q \in H(r), p \neq q\}$ having as its SE-models (over At^*)

$$\{(X, Y) \in INT_{At^*} \mid |Y \cap H(\overline{r})| \leq 1\}.$$

Putting things together, for each disjunctive fact $r \in P$, we get as SE-models (over At^*) of rules $\overline{r}^{\rightarrow} \cup R$ exactly those $(X, Y) \in INT_{At^*}$ which satisfy Condition 3. □

For illustration, reconsider $P_1 = \{a \vee b \leftarrow\}$ over $At = \{a, b\}$. We get P_1^+ given by

$$\{a \vee b \leftarrow;\ a \leftarrow \overline{a};\ b \leftarrow \overline{b};\ \overline{a} \leftarrow \neg\overline{b};\ \overline{b} \leftarrow \neg\overline{a};\ \leftarrow \overline{a}, \overline{b}\}.$$

The set of SE-models (over $\{a, b, \overline{a}, \overline{b}\}$) of P_1^+ is then given by

$$\{(a\overline{a}, a\overline{a}),\ (b\overline{b}, b\overline{b}),\ (a\overline{a}, ab\overline{a}),\ (b\overline{b}, ab\overline{b}),\ (ab\overline{a}, ab\overline{a}),\ (ab\overline{b}, ab\overline{b})\}.$$

Observe that P_1^+ is closed under here-intersection, whilst P_1 is not. This example mirrors the following general property:

Lemma 4.77 *For any DFP P, P^+ is closed under here-intersection.*

Proof. If P is an NLP, then P^+ is clearly closed under here-intersection. So suppose that $P \in \text{DLP} - \text{NLP}$, and assume that P^+ is not closed under here-intersection. Then, there exist SE-models (X, Y) and (Z, Y) of P^+ such that $(X \cap Z, Y) \notin M_s(P^+)$. Let P_1 consist of all normal rules in P^+ and P_2 of all disjunctive rules in P^+. We know that P_1 is closed under here-intersection, i.e., $(X \cap Z, Y) \in M_s(P_1)$. Moreover, by Proposition 4.22, for all $r \in P_2$, we have that $X \models H(r)$ and $Z \models H(r)$, since each body $B(r)$ is empty. Fix such an r (observe that $P_2 \neq \emptyset$ by hypothesis and moreover $P_2 \subseteq P$ by definition). By Lemma 4.76, we get

$$X \cap H(\overline{r}) = Z \cap H(\overline{r}) = Y \cap H(\overline{r}) = \{\overline{p}\},$$

$X \cap \overline{At} \subseteq \overline{(X \cap At)}$, and $Z \cap \overline{At} \subseteq \overline{(Z \cap At)}$. By $p \in H(r)$, we obtain $p \in X \cap Z$, and thus $X \cap Z \models H(r)$. Consequently, $(X \cap Z, Y) \in M_s(r)$. Since this holds for all $r \in P_2$, we end up with $(X \cap Z, Y) \in M_s(P_2)$. We already know that $(X \cap Z, Y) \in M_s(P_1)$, and so $(X \cap Z, Y) \in M_s(P^+)$. This, however, is a contradiction to $(X \cap Z, Y) \notin M_s(P^+)$. Hence, P^+ must be closed under here-intersection. □

Lemma 4.78 *Let P be a HDFP over At. Then,*

1. *if $I \in \text{SM}(P)$, then there exists a $\overline{K} \subseteq \overline{At}$ such that $I \cup \overline{K}$ is a stable model of P^+; and*

2. if $I \in \mathrm{SM}(P^+)$, then $I \cap At$ is a stable model of P.

Proof. To show Part 1, let $I \in \mathrm{SM}(P)$ and let $\overline{K}$ be constructed by any $K \subseteq I$ satisfying $|K \cap H(r)| = 1$, for each disjunctive fact $r \in P$. Such $\overline{K}$ exists since P is an HDFP. Clearly, $I \cup \overline{K} \models P$ since no atoms from $\overline{At}$ occur in P and $I \models P$. Moreover,

$$I \cup \overline{K} \models \{p \leftarrow \overline{p} \mid p \in At\}$$

since $K \subseteq I$ by definition. Let R be the collection of the rules $(\overline{r})^{\rightarrow}$ and $\{\leftarrow \overline{p}, \overline{q} \mid p, q \in H(r), p \neq q\}$, for each disjunctive fact $r \in P$. Then, $I \cup \overline{K} \models R$, by the assumption that K satisfies $|K \cap H(r)| = 1$, for each $r \in P$ with $|H(r)| > 1$. Hence $I \cup \overline{K} \models P^+$.

It remains to show that no proper subset J of $I \cup \overline{K}$ is a model of $(P^+)^I$. Suppose $(J \cap At) \subset (I \cap At)$. Then $J \models (P^+)^I$ would imply that $J \models P^I$, contradicting $I \in \mathrm{SM}(P)$. Now suppose $(J \cap \overline{At}) \subset (I \cap \overline{At})$. Then, $J \not\models R^I$, since we would have at least one rule from $(\overline{r})^{\rightarrow}$ which is not satisfied by J.

Concerning Part 2, assume $I \in \mathrm{SM}(P^+)$. Since $P \subseteq P^+$ and no atoms from At occur in P, we have $I \cap At \models P$. Again, it remains to show that no $J \subset (I \cap At)$ exists, such that $J \models P^I$ holds. Towards a contradiction, let $J \subset (I \cap At)$ be a model of P^I. We show that then there exists a $\overline{K} \subseteq \overline{At}$ such that both $(J \cup \overline{K}) \subset I$ and $J \cup \overline{K} \models (P^+)^I$ holds, thus contradicting $I \in \mathrm{SM}(P^+)$. First, since $I \in \mathrm{SM}(P^+)$, we have $(I, I) \in M_s^{At^*}(P^+)$, and from Lemma 4.76 we get $I \cap \overline{At} \subseteq \overline{(I \cap At)}$. Now let $\overline{K} = \overline{J} \cap (I \cap \overline{At})$. Consequently, $(J \cup \overline{K}) \subset I$, and $J \cup \overline{K} \models \{p \leftarrow \overline{p} \mid p \in At\}^I$. Moreover, for each disjunctive fact $r \in P$, we have $|J \cap H(r)| \geq 1$ (otherwise $J \not\models P^I$) as well as $|I \cap H(\overline{r})| = 1$ (otherwise $I \not\models P^+$). Since $J \subset I$ and $I \cap \overline{At} \subseteq \overline{(I \cap At)}$ holds by $I \in \mathrm{SM}(P^+)$, we finally get $J \cup \overline{K} \models R^I$. Hence, we derive $J \cup \overline{K} \models (P^+)^I$. □

The relations from Lemmas 4.77 and 4.78 guarantee a faithful reduction (from arbitrary HDFPs to DLPs closed under here-intersection) of the relevant reasoning tasks in the context of logic programming. As already mentioned, the hardness results given in [26] carry over for HDFPs. We thus obtain our next result which shows that DLPs closed under here-intersection possess the same worst-case complexity as general DLPs.

Theorem 4.79 *Both the consistency problem and brave reasoning for DLPs closed under here-intersection is Σ_2^P-complete, and cautious reasoning for DLPs closed under here-intersection is Π_2^P-complete.*

Succinctness of DLPs Finally, we discuss the size of the rewriting of a given DLP P into an equivalent NLP Q (if it exists).

Theorem 4.80 *There is no rewriting f from the class of DLPs into the class of NLPs such that (i) $P \equiv_\alpha f(P)$, and (ii) $f(P)$ is polynomial in the size of P, for every $P \in DLP$, with $\alpha \in \{u, s\}$, unless the PH collapses.*

Proof. Assume that a polynomial-size rewriting f of the described kind exists. Consider the Π_2^P-hard problem of checking whether, for a given positive DLP P and a given atom a, $\neg a$ is a cautious consequence of P, i.e., whether a is not contained in any stable model of P [26].

Table 3: Complexity of deciding rule and literal redundancy.

$\equiv_s$ / $\equiv_u$	Horn LP	Normal LP	Positive LP	HCF LP	DLP
$r \in P$ redundant?	P	coNP	coNP	coNP	coNP/Π_2^P
ℓ in $r \in P$ redundant?	P	coNP	coNP	coNP	coNP/Π_2^P

Define $P_1 = P^+$ if $\alpha = s$, and $P_1 = P$ if $\alpha = u$. Then, $\neg A$ is a cautious consequence of P iff it is a cautious consequence of P_1. By the existence of f, we can guess an NLP P' in nondeterministic polynomial time such that $P' \equiv_\alpha P_1$ ($\alpha \in \{u, s\}$). Checking $P' \equiv_\alpha P_1$ is in coNP [20], and checking whether $\neg a$ is a cautious consequence of P' is in coNP (since P' is normal). Thus, the Π_2^P-hard problem of deciding whether $\neg a$ is a cautious consequence of P is in Σ_2^P, which is a contradiction unless the PH collapses. □

Also for rewritings under ordinary equivalence we cannot avoid an exponential blowup unless the PH collapses, as shown from results in [8, 9].

Proposition 4.81 *There exists no polynomial-size rewriting f : DLP $\rightarrow$ NLP such that $P \equiv f(P)$, for every $P \in$ DLP, unless the PH collapses.*

Clearly Theorem 4.80 is implied by Proposition 4.81, but the proof of the latter refers to non-uniform complexity classes, while ours is from first principles. In particular, a direct proof of Proposition 4.81 would show that a polynomial-size rewriting f : DLP $\rightarrow$ NLP such that $P \equiv f(P)$ implies coNP $\subseteq$ P/poly (P/poly is the class of problems decidable in polynomial time with polynomial advice), which in turn implies a collapse of the PH. Furthermore, Proposition 4.81 remains true for generalized rewritings f which admit projective extra variables, i.e., $P \equiv f(P)|_{At}$, where $f(P)$ is defined over atoms $At' \supseteq At$ and $f(P)|_{At}$ denotes the restriction of the stable models of $f(P)$ to the original atoms At. This is a consequence of combining results from [8] and the facts that (i) model checking for NLPs is polynomial and (ii) model checking for circumscription (which is hard for the non-uniform compilability variant of coNP) is a special case of model checking for DLPs (both rely on minimal model checking).

We remark that, in terms of [37], DLPs are, because of the exponential blow up, a stronger KR formalism than NLPs, unless the PH collapses, regardless of the notion of equivalence considered.

4.7.3 Complexity for Other Program Simplifications

The complexity of deciding redundant literals and rules is given in Table 3, for various classes of programs, where entries represent completeness for the respective complexity classes. The upper complexity bounds are easily obtained from the complexity of deciding $P \models_s r$ resp. $P \models_u r$ for the respective program class $\mathcal{C}$ in Table 3 (cf. Section 4.7.1), since $P \in \mathcal{C}$ implies $P - \{r\} \in \mathcal{C}$ in each case. The hardness parts for positive programs, HCF programs, and DLPs under $\equiv_s$ are easily obtained from the coNP-completeness of deciding $P \models v$ for an atom v and a positive HCF P. For Horn programs, the problem is polynomial since $P \models_s r$ and $P \models_u r$ coincide with $P \models r$, which is polynomial for Horn

Table 4: Complexity of deciding if for $P \in \mathcal{C}$ (row) some $Q \in \mathcal{C}'$ (column) exists with $P \equiv_e Q$.

$\equiv_s$ / $\equiv_u$	Positive LP	Horn LP	constraint-free positive LP	Definite Horn
NLP	coNP	coNP	P/coNP	P/coNP
HCF LP	coNP	coNP	coNP	coNP
DLP	coNP/Π_2^P	coNP/in Π_2^P	coNP/Π_2^P	coNP/in Π_2^P

P. The Π_2^P-hardness of DLPs under $\equiv_u$ can be shown by suitable adaptions of proofs from Section 4.7.1. However, redundancy checking is tractable in certain cases:

Theorem 4.82 *Let P be a DLP without positive constraints such that P^+ is normal. Then, checking redundancy of a positive rule $r \in P$ under $\equiv_s$ is decidable in polynomial time, and likewise checking redundancy of a literal ℓ in a positive rule $r \in P$.*

We next consider the complexity of deciding whether a given program is equivalent to some program belonging to a specific syntactic subclass of DLPs. The generic problem is as follows:

Instance: A program P (possibly from a restricted class of programs, $\mathcal{C}$).
Question: Does there exist some program Q from a fixed class of programs, $\mathcal{C}'$, such that $P \equiv_e Q$, where $e \in \{s, u\}$ is fixed?

Our results concerning this task are summarized in Table 4, where entries stand for completeness for the respective complexity classes unless stated otherwise. Notice that definite Horn programs are constraint-free Horn programs.

Unsurprisingly, the problem is intractable in general. The upper bounds for the non-polynomial cases can be easily derived from the results in the previous sections and the complexity of $\models_s$, which is coNP-complete for general DLPs (as follows from [70, 71]), and of $\models_u$, which is Π_2^P-complete in general but coNP-complete for HCF (and, in particular, normal) logic programs. Furthermore, in view of our results concerning the elimination of disjunctions, if $\mathcal{C}$ is the class of all DLPs and $\mathcal{C}'$ is the class of all NLPs, the problem is coNP-complete under strong equivalence and trivial under uniform equivalence.

Note that the rows for NLPs and HCF programs coincide under $\equiv_u$, since $P \equiv_u P^{\rightarrow}$ holds for each HCF program P, and the shifting program $P^{\rightarrow}$ is constraint-free iff P is constraint-free. However, under $\equiv_s$, we have a different picture, and we can find some interesting tractable cases from Theorem 4.63:

Theorem 4.83 *Let P be a positive-constraint free DLP such that P^+ is normal. Then, deciding if there is some positive program Q such that $P \equiv_s Q$ is feasible in polynomial time, and likewise deciding whether there is some Horn program Q with $P \equiv_s Q$.*

4.8 First-Order Programs

In the previous sections, we have extensively considered propositional (ground) logic programs. We now turn to first-order programs (with predicates and variables in rules). As usual in the setting of deductive databases, we assume that the semantics of a Datalog$^\neg$ program $\mathcal{P}$ over a database instance $\mathcal{D}$ is given by $\mathcal{P}[\mathcal{D}]$, i.e., amounts to the semantics of the union $\mathcal{P} \cup \mathcal{D}$.

We adopt the common convention that all rules in $\mathcal{P}$ satisfy the usual notion of safety, cf. [49], viz. that each variable X occurring in the head of a rule r also occurs in the positive body $B^+(r)$ of the rule r, and analogously for each variable occurring in the negative part $B^-(r)$ of rule r. This extends to the case where special built-in predicates are used; but we shall only consider inequality ($\neq$, also denoted <>) here.

The notion of safety guarantees that no facts involving new individuals which are not mentioned in the database $\mathcal{D}$ or the program $\mathcal{P}$ can be derived. Then, by the finiteness assumption on the program $\mathcal{P}$ and the database D, any stable model for $\mathcal{P}[\mathcal{D}]$ will be finite, and thus can be viewed as representing a (finite) database on the predicates involved, which is built on the constants occurring in $\mathcal{P}$ and $\mathcal{D}$. This set of constants constitutes the *active domain*, which is a subset of the database domain $\mathcal{U}$ (assuming that the constants in $\mathcal{P}$ are all from $\mathcal{U}$). For further background on the evaluation of datalog programs over databases we refer, e.g., to [13, 1].

We recall that the database domain $\mathcal{U}$ is assumed to be countable, and thus is either finite or infinite (in the latter case, it is equi-pollent to the natural numbers). These two settings significantly differ with respect to complexity in general, and in particular in the latter setting, many optimization problems for datalog programs, already with limited use of negation, are undecidable, cf. [11, 32, 42, 50, 51, 64, 65, 66, 67, 68]. This is intuitively because we have to consider an infinite number of different database instances $\mathcal{D}$ on which a program $\mathcal{P}$ has to be evaluated.

The optimization of datalog programs with negation, and in particular with the use of negation under unstratified negation, has not been much analyzed before. However, for the concerns of the INFOMIX project, such a setting is important, since the datalog programs which are produced for computing programs repair and query answering make use of unstratified negation (which is indispensable, since different stable models must be possible in general respecting the different ways to reconcile the inconsistent source database with the data integration system.) Given that the equivalence problem of (positive, disjunction-free) datalog programs under the assumption of an infinite underlying domain $\mathcal{D}$ is undecidable, cf. [67, 64, 68], i.e., whether two given programs always compute the same query answers over all database instances, it is clear that also in the extended language with negation the problem is undecidable in general. Thus, decidable cases need to be singled out. Furthermore, our interest is, beyond equivalence, also uniform and strong equivalence. As shown in [64], uniform equivalence for positive disjunction-free datalog programs is decidable, and thus there is hope for decidability under the notions of uniform and strong equivalence, respectively.

The results in the previous subsections on propositional logic programs provide, given that the semantics of datalog programs on databases is reduced to a propositional setting, a useful basis for deriving results on uniform and strong equivalence of datalog programs. The notions of uniform and strong equivalence between datalog programs are defined as follows:

Definition 4.84 *Let P and Q be two Datalog$^{\vee,\neg}$ programs over an underlying domain $\mathcal{U}$.*

Then,

(*i*) *P and Q are* strongly equivalent*, or s*-equivalent*, denoted* $P \equiv_s Q$*, iff, for any finite set R of rules, the programs* $P \cup R$ *and* $Q \cup R$ *are equivalent, i.e.,* $P \cup R \equiv Q \cup R$.

(*ii*) *P and Q are* uniformly equivalent*, or u*-equivalent*, denoted* $P \equiv_u Q$*, iff, for any finite set F of non-disjunctive ground facts on* $\mathcal{U}$*,* $P \cup F$ *and* $Q \cup F$ *are equivalent, i.e.,* $P \cup F \equiv Q \cup F$.

Here, we tacitly assume that the rules in R added to P and Q are safe (however, this assumption is not crucial, given the grounding definition of the semantics of first-order logic programs).

The notions of SE and UE-model naturally generalize from the propositional to the first-order case.

Definition 4.85 *Given a Datalog*$^{\vee,\neg}$ *program P (over a relational alphabet* $\mathcal{A}$*) and a set of constants C including all constants in P, any pair of Herbrand interpretations X and Y of* $\mathcal{A}$ *with respect to C such that* $X \subseteq Y$ *and* $Y \models P$ *and* $X \models P^Y$*, denoted* $(X,Y)_C$*, is an* SE-model *of P (that is, if* (X,Y) *is an SE-model of the propositional program* $ground(P,C)$*, where the set of atoms is given by the Herbrand base of P with respect to C).*

Furthermore, $(X,Y)_C$ *is an* UE-model *of P, if for every SE-model* $(X',Y)_C$ *of P, it holds that* $X \subset X'$ *implies* $X' = Y$.

In the remainder, we assume that the alphabet $\mathcal{A}$ and an underlying domain $\mathcal{U}$ are fixed, and we denote by $M_s(P)$ (resp., $M_u(P)$) the set of all SE-models (resp., UE-models) $(X,Y)_C$ of P where $C \subseteq \mathcal{U}$.

We note the following result.

Theorem 4.86 *Let P and Q be Datalog*$^{\vee,\neg}$ *programs on a relational alphabet* $\mathcal{A}$ *and a domain* $\mathcal{U}$*. Then, P and Q are strongly equivalent, i.e.* $P \equiv_s Q$*, iff* $M_s(P) = M_s(Q)$.

Proof. The results are proved by reduction to the propositional case.

($\Leftarrow$) Suppose that $P \not\equiv_s Q$. Then, there exists a finite set of rules R (with constants from $\mathcal{U}$) such that $P \cup R$ and $Q \cup R$ do not have the same stable models. That is, the propositional programs $P_R = ground(P \cup R)$ and $Q_R = ground(Q \cup R)$ have different stable models. By Proposition 4.4, it follows that the propositional programs $ground(P,C)$ and $ground(Q,C)$, which results from the grounding of the rules in P and Q, respectively, do not have the same SE-models, with respect to the set of atoms A given by P_R and Q_R. That is, there exists without loss of generality an SE-model (X,Y) of $ground(P,C)$ which is not an SE-model of $ground(Q,C)$ (w.r.t. $\mathcal{A}$). Consequently, $(X,Y)_C$ is an SE-model of P but not of Q.

($\Rightarrow$) Conversely, suppose that $M_s(P) \neq M_s(Q)$. Without loss of generality suppose that $(X,Y)_C$ is an SE-model of P but not of Q. Then, with respect to the set of propositional atoms A given by the Herbrand base of P and Q with respect to C, (X,Y) is an SE-model of the ground (propositional) program $ground(P,C)$ but not of $ground(Q,C)$. This means that there is a set of ground rules R on A such that $ground(P,C) \cup R$ and

$ground(Q, C) \cup R$ do not have the same stable models. Because of the safety of P, Q, and R, there is a subset of the constants $C' \subseteq C$ and a subset $R' \subseteq R$ with constants only from C', such that $ground(P, C') \cup R'$ and $ground(Q, C') \cup R'$ do not have the same stable models, and each constant in $ground(P, C') \cup R'$ and $ground(Q, C') \cup R'$ occurs either in P, Q or R'. Thus, it follows that the datalog programs $P \cup R'$ and $Q \cup R'$, have different stable models. By definition, this means that P and Q are not strongly equivalent. $\Box$

An analogous result for uniform equivalence of datalog programs can be obtained by similar lifting from the propositional case.

Theorem 4.87 *Let P and Q be Datalog$^{\vee,\neg}$ programs on a relational alphabet $\mathcal{A}$ and domain $\mathcal{U}$. Then, P and Q are uniformly equivalent, i.e. $P \equiv_u Q$, iff $M_u(P) = M_u(Q)$.*

Also the notions of SE-consequence $P \models_s r$, $M \models_s Q$ and UE-consequence $P \models_u r$, $P \models_u Q$ naturally generalize from the propositional to the datalog case, where in $P \models_s r$ resp. $P \models_u r$ the rule r is understood to be universally quantified.

By lifting from the propositional case, the following results can be obtained.

Theorem 4.88 *Let P and Q be Datalog$^{\vee,\neg}$ programs on a relational alphabet $\mathcal{A}$ and domain $\mathcal{U}$. Then, for $e \in \{s, u\}$, we have*

1. *$P \equiv_e Q$ iff $P \models_e Q$ and $Q \models_e P$,*

2. *$P \models_e r$ implies $P \equiv_e P \cup \{r\}$, for any rule r on $\mathcal{A}$ and $\mathcal{U}$.*

Proof. It is sufficient to show part 2, since part 1 follows by repeated application of 2. and the fact that $\equiv_e$ is an equivalence relation.

By definition, $P \models_s r$ implies that $M_s(P) \subseteq M_s(r)$, and hence $M_s(P \cup \{r\}) = M_s(P) \cap M_s(r) = M_s(P)$. Thus, $P \equiv_s P \cup \{r\}$ holds.

Similarly, $P \models_u r$ means that $M_u(P) \subseteq M_s(r)$.

We first show that $M_u(P \cup \{r\}) \subseteq M_u(P)$: Indeed, suppose $(X, Y)_C \in M_u(P \cup \{r\}) - M_u(P)$ would hold. Then there must exist an UE-model $(X', Y)_C$ of P such that $X \subset X' \subset Y$. Since $(X', Y)_C$ is an SE-model of $P \cup \{r\}$, this contradicts that $(X, Y)_C$ is an UE-model of $P \cup \{r\}$.

Conversely, $M_u(P) \subseteq M_u(P \cup \{r\})$ must hold: if $(X, Y)_C$ would be in $M_u(P) - M_u(P \cup \{r\})$, then some SE-model $(X', Y)_C$ of $P \cup \{r\}$ would exist such that $X \subset X' \subset Y$ which clearly contradicts that $(X, Y)_C \in M_u(P)$. $\Box$

The above results form a basis on which optimization methods and can be built which eliminate redundant rules, simplify rules etc., as has been discussed in Subsection 4.2 for propositional programs, properly lifted to the first-order case.

4.8.1 Finite domain

In the case of a finite underlying domain $\mathcal{U}$, we can as usual view a datalog program P as a compact representation of the propositional program $ground(P, \mathcal{U})$ over the set of atoms given by the Herbrand basis of the predicates A with respect to $\mathcal{U}$.

Compared to the propositional case, the size of a Herbrand interpretation I of A has single exponential size (i.e., size $O(2^{p(n)})$ for a polynomial $p(n)$) in the size of $\mathcal{U}$, both if $\mathcal{U}$ is represented explicitly or implicitly by, say, a number n such that $\mathcal{U} = \{0, 1, \ldots, n-1\}$.

For strong and uniform equivalence, as well as for SE-consequence and UE-consequence, we thus obtain upper complexity bounds which are, compared to the propositional complexity, higher by one exponential; P to EXPTIME, NP to NEXPTIME, coNP to co-NEXPTIME, Π_2^P to co-NEXPTIME$^{\mathrm{NP}}$, etc. Given that the inference problem of positive, disjunction-free datalog program is EXPTIME-complete (cf. [13]), it is thus of no surprise that the complexity of deciding strong and uniform equivalence, etc., has lower bounds which match the exponential analogs of the propositional case.

We shall not aim at deriving these results here, but sketch a way how this can be achieved. The hardness proofs given in the previous sections involved reductions from the propositional satisfiability problem, SAT, and the generalization to evaluate quantified Boolean formulas (QBFs) of the form $\exists X \forall Y\, E$ resp. $\forall X \exists Y\, E$ where E is a propositional formula on the atoms in $X \cup Y$; satisfiability of a Boolean expression E is tantamount to the validity of the QBF $\exists\, E$.

Evaluating a given second-order sentence Φ of the form $\exists \mathbf{P} \forall \mathbf{Q} \phi$ resp. $\forall \mathbf{Q} \phi$, where $\mathbf{P} = p_1, \ldots, p_n$ and $\mathbf{Q} = q_1, \ldots, q_m$ are lists of predicate variables and $\phi = \exists \mathbf{x} \psi(\mathbf{x})$ and $\psi(\mathbf{x})$ is a quantifier-free formula in DNF over predicates $\mathbf{P} \cup \mathbf{Q} \cup \mathcal{A}$ (resp., $\mathbf{Q} \cup \mathcal{A}$), where $\mathcal{A}$ consists of predicate constants including special predicates *succ*, *first*, and *last*, over a given first-order structure $\mathcal{M}$ for $\mathcal{A}$ in which *succ*, *first*, and *last* are interpreted as a successor predicate and the first and last element of a linear odering of the universe $|\mathcal{M}|$ of $\mathcal{M}$, respectively, is complete for NEXPTIME$^{\mathrm{NP}}$ resp. co-NEXPTIME (this follows from normal form results for second-order logic over successor structures, cf. [47, 27], and the fact that a conversion into such a normal form is feasible in polynomial time).

Proofs of propositional complexity results, which encode reductions from evaluating $\exists X \forall Y\, E$ resp. $\forall X\, E$, can be lifted to encode the evaluation of $\exists \mathbf{P} \forall \mathbf{Q} \phi$ resp. $\forall \mathbf{Q} \phi$ in the predicate case. We sketch this for the proof of Theorem 4.67.

The propositional atoms x_i and $x_i' \in X$ are replaced by predicate atoms $p_i(X_1, \ldots, X_n)$ and $p_i'(X_1, \ldots, X_n)$ for $p_i \in \mathbf{P}$ corresponding to x_i, respectively, and similarly y_i and y_i' are replaced by predicate atoms $q_i(X_1, \ldots, X_n)$ and $q_i'(X_1, \ldots, X_n)$ for $q_i \in \mathbf{Q}$ corresponding to y_i, respectively, where n is the arity of p resp. q, in all rules except $w \leftarrow D_i^*$ (where $\phi = \exists \mathbf{x} \psi(\mathbf{x})$ and $\psi(\mathbf{x}) = \bigvee_{i=1}^{l} D_i$ is a DNF). In the latter rules, literals $\neg p_j(t_1, \ldots, t_n)$ and $\neg q_j(t_1, \ldots, t_n)$ are replaced by atoms $p_j'(t_1, \ldots, t_n)$ and $q_j'(t_1, \ldots, t_n)$, respectively. Furthermore, facts

$$\begin{array}{l} \mathit{succ}(c, c'), \\ \mathit{first}(c), \text{ and} \\ \mathit{last}(c) \end{array}$$

are added to Q and P for each elements c and c' from the universe $|\mathcal{M}|$ of the given structure $\mathcal{M}$, such that $\mathcal{M} \models \mathit{succ}(c, c')$ (resp., $\mathcal{M} \models \mathit{first}(c)$, $\mathcal{M} \models \mathit{last}(c)$), and constraints

$$\begin{array}{l} \leftarrow \mathit{succ}(c, c'), \\ \leftarrow \mathit{first}(c), \text{ and} \\ \leftarrow \mathit{last}(c) \end{array}$$

for all others. These facts and constraints enforce a unique interpretation of *succ*, *first*, and *last* in each stable model of any program containing P resp. Q, which coincides with the one given by $\mathcal{M}$. Then, the programs P and Q are uniformly equivalent w.r.t. $\mathcal{U} = |\mathcal{M}|$, iff the sentence $\Phi = \exists \mathbf{P} \forall \mathbf{Q} \phi$ is false on $\mathcal{M}$. Since P and Q can be constructed in polynomial time from $\mathcal{M}$ and Φ, this implies co-NEXPTIME$^{\mathrm{NP}}$ hardness of deciding $P \equiv_u Q$ w.r.t. given finite universe $\mathcal{U}$.

SE- and UE-consequence Tests Implementations of an SE- and UE-consequence tests over a $\mathcal{A}$ and given finite domain $\mathcal{U}$ can be obtained by lifting the reductions in the propositional case to determine existence of a stable model for a normal (resp., disjunctive) propositional logic program in Section 4.3 to the datalog case, for which the problem is in co-NEXPTIME resp. co-NEXPTIME$^{\mathrm{NP}}$. The main step which needs to be done here is to replace suitably propositional atoms with first-order atoms.

We remark in closing this subsection that under certain restrictions, the problems might have smaller complexity. For example, if the arities of all predicates are bounded by a constant, then the complexities of the problems fall back in the polynomial hierarchy. This follows from a straightforward application of the results in [18].

4.8.2 Infinite domain

Over infinite domains, decidability of equivalence and inference problems for datalog is no longer guaranteed. We recall the classical result that given positive datalog programs P and Q over relations $\mathcal{A}$ and infinite domain $\mathcal{U}$, it is undecidable whether P and Q are equivalent on all (finite) database instances $\mathcal{D}$ of a designated fixed subset $E \subseteq \mathcal{A}$ of extensional relations formed over $\mathcal{U}$, i.e., whether $P \cup \mathcal{D}$ and $Q \cup \mathcal{D}$ always compute the same value for a designated output predicate, cf. [67, 68].

The following lemma is important for strong equivalence.

Lemma 4.89 (SE submodels) *For any Datalog$^{\vee,\neg}$ program P over $\mathcal{A}$ and domain $\mathcal{U}$, the class of SE-models of P is closed under submodels. That is, if $(X,Y)_C$ is an SE-model of P and $C' \subseteq C$ is any set of constants which includes all constants in P, then the model $(X',Y')_{C'}$, denoted $(X,Y)|C'$, is also an SE-model of P, where $X' = \{p(c_1,\ldots,c_n) \in X \mid p \in \mathcal{A}, c_1,\ldots,c_n \in C'\}$ and $Y' = \{p(c_1,\ldots,c_n) \in Y \mid p \in \mathcal{A}, c_1,\ldots,c_n \in C'\}$ are the restrictions of X and Y to the domain C'.*

The proof of this lemma is straightforward. In view of it, we can derive the following results:

Corollary 4.90 *Given a Datalog$^{\vee,\neg}$ program P and rule r over $\mathcal{A}$ and an infinite domain $\mathcal{U}$, $P \models_s r$ is decidable (more precisely, it is in co-NEXPTIME).*

Proof. If $P \not\models_s r$, then there exists some SE-model $(X,Y)_C$ of P such that $(X,Y)_C \not\models r$. That is, there is a ground instance r' of r such that $(X,Y)_C \not\models r'$. Let $C' \subseteq \mathcal{U}$ be any set of constants which includes all constants in P and r. Then, by the SE-Submodel Lemma, $(X,Y)|C'$ is an SE-model of P, and $(X,Y)|C' \not\models r'$. Consequently, $P \not\models_s r$ iff there exists an SE-model $(X,Y)_C$ of P where C consists of all constants in

$P \cup \{r\}$ and arbitrary further different constants $c_1, \ldots, c_m$ for the variables $X_1, \ldots, X_n$ occurring in r. That is, $P \not\models_s r$ over $\mathcal{A}$ and finite set of constants C, which can be decided in co-NEXPTIME as discussed in the previous section. □

Corollary 4.91 *Given Datalog$^{\vee,\neg}$ programs P and Q over $\mathcal{A}$ and an infinite domain $\mathcal{U}$, $P \models_s Q$ and $P \equiv_s Q$ are decidable (more precisely, solvable in co-NEXPTIME).*

Proof. Immediate from the previous corollary and Theorem 4.88. □

Both of the above results can be sharpened to completeness results, and implementations of the tests can be obtained by reduction to answer set programming. Decidability of $P \equiv_s Q$ was earlier reported by Lin [54], using a different proof and claiming (erroneously, and without proof) that the problem is coNP-complete.

We remark that the SE-Submodel Lemma continues to hold in the presence of certain built-in relations which are preserved under submodels as well (e.g., inequality or a total ordering).

Unfortunately, a property similar to Lemma 4.89 does not hold for UE-models, which however is intuitive. By restricting to a subset $C' \subseteq C$ of the constants, the first component of the UE-model $(X, Y)_C$ might lose ground facts such that the model $(X', Y')_{C'}$ becomes comparable to another model $(X'', Y')_{C'}$ which has a larger first component, i.e., $X' \subset X''$ holds. In other words, the maximality of $(X, Y)_C$ is lost when going to submodels.

In what follows, we formally prove the undecidability of checking uniform equivalence. We commence with the following result:

Lemma 4.92 *Given positive, disjunction-free datalog programs P and Q over relations $\mathcal{A}$, with designated extensional relations $\mathcal{E} \subseteq \mathcal{A}$ and infinite domain $\mathcal{U}$, it holds that for each database instance $\mathcal{D}$ of $\mathcal{E}$ on finite $C \subseteq \mathcal{U}$ including all constants of P and Q and $\mathcal{D}$, $P \cup \mathcal{D}$ and $Q \cup \mathcal{D}$ has the same stable model w.r.t. C iff P and Q are equivalent (w.r.t. extensional relations $\mathcal{E}$).*

Given a program P, we construct from it a program $\bar{P}$ which intuitively complements the models of P.

The program $\bar{P}$ is follows. For each rule

$$r(X_1, \ldots, X_n): \qquad A_0 \leftarrow A_1, \ldots, A_n \tag{12}$$

in P we add the rule

$$\bar{r}(X_1, \ldots, X_n): \qquad A_1 \vee \cdots \vee A_n \leftarrow A_0, d(X_1), \ldots, d(X_n) \tag{13}$$

in $\bar{P}$, where d is a new predicate; intuitively, it desribes the active domain and serves to ensure safety of $\bar{r}(X_1, \ldots, X_n)$.

Furthermore, we add facts $d(c)$ to $\bar{P}$ for each constant c occurring in P, and for each predicate $p \in \mathcal{E}$ of arity $n > 0$ the rules

$$d(X_i) \leftarrow p(X_1, \ldots, X_n) \qquad i \in \{1, \ldots, n\}$$

Intuitively, the rules serve to gather all active domain elements in d.

Lemma 4.93 *Let $\mathcal{D}$ be any finite database instance of $\mathcal{E}$ over $\mathcal{U}$. Then, among the Herbrand interpretations I of $\bar{P} \cup \mathcal{D}$ w.r.t. a set of constants C that includes each constant c occurring in $\bar{P} \cup \mathcal{D}$ and where the predicates E are as in $\mathcal{D}$, there is a unique maximal model, denoted $M(\mathcal{D}, \bar{P})_C$. Furthermore, $d(c) \in M(\mathcal{D}, \bar{P})_C$ holds for all $c \in C$.*

Proof. Clearly, $d(c) \in M$ must hold for every Herbrand model M of $\bar{P} \cup \mathcal{D}$ w.r.t. C. Thus, the rule (13) classically amounts in M to

$$r(X_1, \ldots, X_n): \qquad \neg A_0 \leftarrow \neg A_1, \ldots, \neg A_n,$$

that is, to the rule (13) if all relations are complemented. Thus, if we take the complement interpretation $\bar{M}$ of M (w.r.t. C), then we obtain a model of $Q = P \cup \bar{\mathcal{D}}$, where $\bar{\mathcal{D}}$ is the complement of $\mathcal{D}$ w.r.t. C. Since Q is Horn, it has a least model N which coincides on $\mathcal{E}$ with $\bar{M}$. Its complement w.r.t. C, $\bar{N}$, augmented with all facts $d(c)$, $c \in C$, is a model of $\bar{P} \cup \mathcal{D}$, and, by minimality of N, it is moreover maximal w.r.t. $\mathcal{D}$; thus, it is $M(\mathcal{D}, \bar{P})_C$ as claimed. □

We now extend $\bar{P}$ to a program $\bar{P}_w$ whose UE-models $(X, Y)_C$ where $X \subset Y$ informally encode the maximal models of $\bar{P} \cup \mathcal{D}$ for all $\mathcal{D}$, where the constants in $\bar{P} \cup \mathcal{D}$ are C.

For a fresh propositional atom w, we add

1. the rule $w \leftarrow \neg w$.
2. the atom w in the head of each constraint in $\bar{P}$;
3. the rule $w \leftarrow e(X_1, \ldots, X_n), \bar{e}(X_1, \ldots, X_n)$ for each extensional predicate $e \in \mathcal{E}$, where $\bar{e}$ is a fresh predicate of the same arity.
4. the rule $p(X_1, \ldots, X_n) \leftarrow w, d(X_1), \ldots, d(X_n)$ for each predicate $p \in \mathcal{A}' := \mathcal{A} \cup \{\bar{e} \mid e \in \mathcal{E}\}$;
5. the rules $d(X_i) \leftarrow \bar{e}(X_1, \ldots, X_n)$, $i \in \{1, \ldots, n\}$ for each predicate $e \in \mathcal{E}$;

Informally, the rule in 1. effects that w must be contained in each classical model. Therefore, in each SE-model $(X, Y)_C$ of $\bar{P}_w$, Y must contain w, and by the rules in 4., Y must consist of the maximal interpretation of $\mathcal{A}' \cup \{d\}$ on C. If $X \subset Y$ holds, then X must not contain w. Thus, X must be a model of $\bar{P}$ (w.r.t. C.

If $(X, Y)_C$ is a UE-model of $\bar{P}_w$, by maximality X must contain exactly one of the facts $e(c_1, \ldots, c_n)$ and $\bar{e}(c_1, \ldots, c_n)$, for each $c_1, \ldots, c_n$ from C. Moreover, restricted to $\mathcal{A} \cup \{e\}$ is the maximal model $M(\bar{P}, \mathcal{D})_C$ for the database $\mathcal{D}$ on C given by the extensions of the predicates $\mathcal{E}$ in X.

On the other hand, if X is $M(\bar{P}, \mathcal{D})_C$ plus all facts $\bar{e}(c_1, \ldots, c_n)$ over C such that $e(c_1, \ldots, c_n) \notin M$, then $(X, Y)_C$ is an SE-model of $\bar{P}_w$. Furthermore, by maximality $M(\bar{P}, \mathcal{D})_C$, $(X, Y)_C$ must be a UE-model of $\bar{P}_w$.

We obtain from this and Lemma 4.92, the following lemma.

Lemma 4.94 *For disjunction-free datalog programs P and Q, we have that $M_u(\bar{P}_w) = M_u(\bar{Q}_w)$ iff P and Q are equivalent (w.r.t. extensional relations $\mathcal{E} \subseteq \mathcal{A}$ and infinite domain $\mathcal{U}$).*

By Shmueli's result [67, 68] that equivalence of datalog programs P and Q (w.r.t. extensional relations $\mathcal{E} \subseteq \mathcal{A}$ and infinite domain $\mathcal{U}$) is undecidable, we obtain following rsult.

Theorem 4.95 *Given Datalog*$^{\vee,\neg}$ *programs P and Q, it is undecidable whether $P \equiv_u Q$.*

Corollary 4.96 *Given a Datalog*$^{\vee,\neg}$ *program P and a rule r, it is undecidable whether $P \models_u r$.*

Indeed, if $P \models_u r$ were decidable, then also $P \equiv_u Q$ would be decidable.

Note that [64] showed that uniform equivalence of positive, disjunction-free datalog programs P and Q is equivalent to equivalence of P and Q over Herbrand models. This result easily generalizes to positive disjunctive datalog programs P and Q, and in fact, uniform equivalence, strong equivalence and classical equivalence of positive disjunctive P and Q over Herbrand models all coincide (this can be easily seen from the lifting of the results for the propositional case to the datalog case). As a consequence, uniform equivalence of positive disjunctive programs P and Q is decidable, since classical equivalence of programs P and Q over Herbrand models can be refuted, because of submodel preservation, in a small universe of linear size in NEXPTIME. More precisely, deciding classical equivalence of P and Q, and thus also uniform and strong equivalence, is complete co-NEXPTIME).

We remark that the program $\bar{P}_w$ constructed in the proof of Lemma 4.93 does not contain any constraints, and a single rule which contains default negation ($w \leftarrow \neg w$). For the purpose of our construction, this rule can be equivalently replaced with the constraint $\leftarrow \neg w$. The undecidability results of Theorem 4.95 and Corollary 4.96 thus apply to a minimal extension of positive disjunctive programs, and show that positive disjunctive programs are a maximal decidable class, while a single use of negation might lead to undecidability.

4.8.3 Optimizations

In order not to proliferate results, we confine here to a brief discussion of syntactic transformation rules for local program optimization.

The rules TAUT and CONTRA readily carry over from the propositional to the datalog case, as well as LSH, if for the later head-cycle freeness of a rule r is understood in terms of the head-cycle freeness of all instances of r w.r.t. $\mathcal{U}$. Notice that this is decidable also in the case of an infinite domain $\mathcal{U}$.

For the rules RED$^-$, NONMIN, and S-IMPL on rules r_1 and r_2, we can define similarly generalization to the datalog case by *semantic lifting*, i.e., by imposing that

(∗) for each ground instance $r_1\!\downarrow$ of the rule r_1 to be removed on $\mathcal{U}$ there exists a ground instance $r_2\!\downarrow$ of the other rule r_2 on the constants in $r_1\!\downarrow$ such that RED$^-$ (resp., NONMIN and S-IMPL) applies to the pair $r_1\!\downarrow$ and $r_2\!\downarrow$ as specified in Table 1.

Then, we have the following exchange result:

Proposition 4.97 *Suppose that r_1 and r_2 are different rules to which condition $(*)$ applies. If P is a datalog program such that $P \models_e r_2$, $r_1 \in P$, then $P - \{r_1\} \equiv_e P \cup \{r_2\}$, for $e \in \{s, u\}$.*

Corollary 4.98 *Suppose r_1, r_2 are different rules in a datalog program P, and that condition $(*)$ applies to r_1 and r_2. Then $P - \{r_1\} \equiv_e P$, for $e \in \{s, u\}$.*

The above condition $(*)$ can be effectively decided for each of RED$^-$, NONMIN, and S-IMPL (a counterexample involves at most as many additional arbitrary constants as different variables are in r_1). Thus, these properties can be decided with respect to a small set of constants $C \subseteq \mathcal{U}$ (of linear size) which can be constructed easily. With respect to such C, deciding the property is then obviously possible in Π_2^P, and thus can be polynomially encoded to answer set programming with disjunctions.

In a more restrictive, intuitively simpler generalization (with fewer cases where the property holds, but faster to evaluate), we can require the following conditions on r_1 and r_2:

$(**)$ RED$^-$: there exists a subset $L \subseteq B^-(r_1)$ and a substitution θ such that $H(r_2)\theta = L$.

$(**)$ NONMIN: there exist a rule r_1' and a substitution θ such that $H(r_1') \subseteq H(r_1)$, $B(r_1') \subseteq B(r_1)$ and $r_2\theta = r_1'$.

$(**)$ S-IMPL: there exist a rule r_1', a set of atoms $A \subseteq B^-(r_1)$ and a substitution θ such that $H(r_1') \subseteq H(r_1) \cup A$, $B(r_1') \subseteq B^+(r_1) \cup (B^-(r_1) - A)$ and $r_2\theta = r_1'$.

Then, we have a similar exchange result as above:

Proposition 4.99 *Suppose that r_1 and r_2 are different rules such that variant $(**)$ applies to r_1 and r_2 for any of RED$^-$, NONMIN, or S-IMPL. Then, if P is a datalog program such that $P \models_e r_2$ and $r_1 \in P$, then $P - \{r_1\} \equiv_e P \cup \{r_2\}$, for $e \in \{s, u\}$.*

Corollary 4.100 *Suppose r_1, r_2 are different rules in a datalog program P, and that variant $(**)$ applies to r_1 and r_2 for any of RED$^-$, NONMIN, or S-IMPL. Then $P - \{r_1\} \equiv_e P$, for $e \in \{s, u\}$.*

Each of the conditions $(**)$ can be checked in NP, and thus can be polynomially encoded to propositional answer set programming without disjunctions. On the other hand, they are NP-hard to decide in general.

We show this for NONMIN by a reduction for deciding whether a given directed graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ has a Hamiltonian path $v_{i_1} \to v_{i_2} \to v_{i_3} \ldots \to v_{i_n}$ (where $v_{i_j} \neq v_{i_k}$ for all $j \neq k$).

We construct, using binary predicates e and p, rules $r_1 :\ p(c, c) \leftarrow B_1$ and $r_2 : p(c, c) \leftarrow B_2$ where

- B_1 contains all atoms $e(X_i, X_j)$ such that $v_i \to v_j$ is an arc in G and all atoms $p(X_i, X_j)$ such that $1 \leq i \neq j \leq n$,
- B_2 contains all atoms $e(Y_i, Y_{i+1})$, $1 \leq i < n$ and all atoms $p(Y_i, Y_j)$ such that $1 \leq i \neq j \leq n$.

Then, condition (∗∗) NONMIN holds for r_1 and r_2 if and only if G has some Hamiltonian path. Intuitively, the rule r_1' must select a subset of the arcs which form a Hamiltonian path. The $p(X_i, X_j)$ atoms and $p(Y_i, Y_j)$ atoms in $B(r_1)$ and $B(r_2)$, respectively, effect that the substitution θ is a bijection between $Y = \{Y_1, \ldots, Y_n\}$ and $X = \{X_1, \ldots, X_n\}$, respectively. Via this bijection, $e(Y_1, Y_2)$, ..., $e(Y_{n-1}, Y_n)$ must correspond to a Hamiltonian path in G encoded by $e(X_{i_1}, X_{i_2}), e(X_{i_2}, X_{i_3}), \ldots, e(X_{i_{n-1}}, X_{i_n})$. We omit the formal proof of correctness here. Clearly, r_1 and r_2 are constructible in polynomial time.

The construction also works for (∗∗) S-IMPL, since no partial left-shift is possible. The coNP-hardness for RED$^-$ can be shown applying slight adaptations. Thus, all the conditions (∗∗) RED$^-$, NONMIN, and S-IMPL are NP-complete to decide.

We finally remark that despite the intractability of most of the optimization techniques which have been presented in this subsection, these are nonetheless useful and important for off-line optimization of programs, where the run time spent is less critical. Furthermore, polynomial-time solvability may be established for particular cases.

4.9 Extended Logic Programs

As has been reported in Section 2, the stable model semantics has been extended to various classes of logic programs, including extended logic programs, which offer both strong and weak negation [34], and nested logic programs [52]. As it turns out, the main semantical characterizations of strong and uniform equivalence of (disjunctive) logic programs in terms of M_s-models and M_u-models, respectively, gracefully extend to these and further syntactically enhanced classes of programs, cf. [52, 70, 71, 19, 20, 61, 62]. Furthermore, also the main complexity results carry over. As for extended logic programs, the optimization methods and techniques from above can be adapted with little effort in order to accommodate also strong negation. For a discussion of extended logic programs which permit also the inconsistent answer, we refer to [19, 20].

Extensions of the optimization methods for disjunctive logic programs with aggregates is left for future work.

4.10 Relativized Equivalence

The model-theoretic characterizations of strong and uniform equivalence of disjunctive logic programs under the stable resp. answer set semantics have most recently been generalized in [73] to the setting of relativized strong resp. uniform equivalence, in which the rules resp. facts to be added are formed over a given subset $\mathcal{A}'$ of the atoms $\mathcal{A}$. This notion of relativized equivalence generalizes both ordinary equivalence, which emerges for $\mathcal{A} = \emptyset$, and strong resp. uniform equivalence, which is the special case of $\mathcal{A} = \mathcal{A}'$.

The paper [73] provides a generalization of the notions SE-models and UE-models to the setting of relativized equivalence in a uniform manner, such that similar as in the case of (global) strong resp. uniform equivalence, relativized UE-models are obtained as the correspondents to relativized "maximal" SE-models.

The notion of relativized equivalence is of particular interest for program modules and programs components, in particular in a hierarchically composed program, and provides a more accurate assessment than the (simplifying) notions of global strong and uniform equivalence. It is not of a surprise that this may come at a computational price. Indeed, as shown in [73], testing relativized strong equivalence of propositional disjunctive logic programs is Π_2^P-complete, while testing non-relativized equivalence for such programs is coNP-complete. In the datalog case, both relativized strong and uniform equivalence are undecidable, which is an immediate consequence of the classical undecidability result for the equivalence of datalog programs [67, 68].

Similar as in Section 4.1, based on relativized SE-models and UE-models the notions of relativized SE-consequence and UE-consequence can be defined in the obvious way, which provide a semantical basis for implication and redundancy of rules with respect to program transformations under preservation of relativized equivalence. To this end, the generic methods for program optimization developed in this section above can be adapted for the notion of relativized equivalence, which will be the subject of ongoing work.

5 Focussing

Let us now analyze the repair programs as defined in Section 2.3. The source of complexity in evaluating the program $\Pi_{\mathcal{I}}(q)$ defined in the above section, actually lies in the conflict resolution module Π_Σ, and in the evaluation of Π_q. Indeed, $\Pi_{\mathcal{M}}$ is a Datalog$^{\neg_s}$ program that can be evaluated in polynomial time over the source database $\mathcal{D}$ for constructing $ret(\mathcal{I}, \mathcal{D})$, whereas Π_q is a non-recursive Datalog$^{\neg}$ program that has to be evaluated over each repair of the retrieved global database, and Π_Σ is in general a Datalog$^{\vee,\neg}$ program whose evaluation complexity over varying databases is at the second level of the polynomial hierarchy [38]. Furthermore, also evaluating programs with lower complexity over large data sets by means of stable models solvers, such as DLV [29] or Smodels [69], quickly becomes infeasible. This calls for suitable optimization methods speeding up the evaluation (as recently stated in [7]).

Concentrating on the most relevant and computational expensive aspects of the optimization, we focus here on Π_Σ, assuming that $ret(\mathcal{I}, \mathcal{D})$ is already computed, and devise intelligent techniques for the evaluation of Π_q.

Roughly speaking, in the focussing approach we first localize in the retrieved global database $ret(\mathcal{I}, \mathcal{D})$ those facts that are not "affected" (formally specified below) by any violation. Then, we compute the repairs by taking into account only the affected facts. Since in practice, we expect the size of the set of the affected facts to be much smaller than the size of the retrieved global database, the computation of the stable models of Π_Σ, i.e., repairs of $ret(\mathcal{I}, \mathcal{D})$ (Item 2 in Definition 2.6), over the affected facts is significantly faster than the naive evaluation of $\Pi_{\mathcal{I}}(q)$ on the whole retrieved global database.

5.1 Safe and Affected Databases

5.1.1 Conflict Closure

We start with some concepts for a single database. Let $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ be a relational schema. We call two facts $p_1(a_1, \ldots, a_n)$ and $p_2(b_1, \ldots, b_m)$, where $p_1, p_2 \in \Psi$ and each a_i, b_j is in the domain $\mathcal{U}$, *constraint-bounded in* $\mathcal{RS}$, if they occur in the same ground constraint $\sigma^g \in ground(\Sigma)$. Furthermore, for any $\sigma^g \in ground(\Sigma)$, we use $facts(\sigma^g)$ to denote the set of all facts $p(t)$, $p \in \Psi$, which occur in σ^g.

Let $\mathcal{DB}$ be a database for $\mathcal{RS}$. Then, the *conflict set* for $\mathcal{DB}$ w.r.t. $\mathcal{RS}$ is the set of facts $C^{\mathcal{RS}}_{\mathcal{DB}} = \{p(t) \mid \exists \sigma^g \in ground(\Sigma) \wedge p(t) \in facts(\sigma^g) \wedge \sigma^g \text{ is violated in } \mathcal{DB}\}$, i.e., the set of facts occurring in the constraints of $ground(\Sigma)$ that are violated in $\mathcal{DB}$.

Definition 5.1 Let $\mathcal{RS}$=$\langle\Psi, \Sigma\rangle$ be a relational schema and $\mathcal{DB}$ a database for $\mathcal{RS}$. Then, the *conflict closure* for $\mathcal{DB}$, denoted by $C^{\mathcal{RS}^*}_{\mathcal{DB}}$, is the least set such that $t \in C^{\mathcal{RS}^*}_{\mathcal{DB}}$ if either $t \in C^{\mathcal{RS}}_{\mathcal{DB}}$, or t is constraint-bounded in $\mathcal{RS}$ with a fact $t' \in C^{\mathcal{RS}^*}_{\mathcal{DB}}$.

We drop the superscript $\mathcal{RS}$ if it is clear from the context. Intuitively, $C^*_{\mathcal{DB}}$ contains all facts involved in constraint violations, i.e., facts belonging to $C_{\mathcal{DB}}$, and facts which possibly must be changed in turn to avoid new inconsistency with Σ by repairing.

5.1.2 Deviding the Database

We will divide the database along its conflict closure:

Definition 5.2 Let $\mathcal{RS}$=$\langle\Psi, \Sigma\rangle$ be a relational schema and $\mathcal{DB}$ a database for $\mathcal{RS}$. Then, we call $S^{\mathcal{RS}}_{\mathcal{DB}} = \mathcal{DB} - C^{\mathcal{RS}^*}_{\mathcal{DB}}$ and $A^{\mathcal{RS}}_{\mathcal{DB}} = \mathcal{DB} \cap C^{\mathcal{RS}^*}_{\mathcal{DB}}$ the *safe database* and the *affected database* for $\mathcal{DB}$, respectively. □

We also consider the following two subsets of all ground constraints.

(i) $\Sigma^A_{\mathcal{DB}}$ consists of all the ground constraints in which at least one fact from $C^*_{\mathcal{DB}}$ occurs, i.e., $\Sigma^A_{\mathcal{DB}} = \{\sigma^g \in ground(\Sigma) \mid facts(\sigma^g) \cap C^*_{\mathcal{DB}} \neq \emptyset\}$, and

(ii) $\Sigma^S_{\mathcal{DB}}$ consists of all the ground constraints in which at least one fact occurs which is not in $C^*_{\mathcal{DB}}$, i.e., $\Sigma^S_{\mathcal{DB}} = \{\sigma^g \in ground(\Sigma) \mid facts(\sigma^g) \not\subseteq C^*_{\mathcal{DB}}\}$.

We first show that $\Sigma^A_{\mathcal{DB}}$ and $\Sigma^S_{\mathcal{DB}}$ form a partitioning of $ground(\Sigma)$.

Proposition 5.3 (Separation) *Let $\mathcal{RS} = \langle\Psi, \Sigma\rangle$ be a relational schema, and let $\mathcal{DB}$ a database for $\mathcal{RS}$. Then,*

1. $\bigcup_{\sigma^g \in \Sigma^A_{\mathcal{DB}}} facts(\sigma^g) \subseteq C^*_{\mathcal{DB}}$;
2. $\bigcup_{\sigma^g \in \Sigma^S_{\mathcal{DB}}} facts(\sigma^g) \cap C^*_{\mathcal{DB}} = \emptyset$;
3. $\Sigma^A_{\mathcal{DB}} \cap \Sigma^S_{\mathcal{DB}} = \emptyset$ *and* $\Sigma^A_{\mathcal{DB}} \cup \Sigma^S_{\mathcal{DB}} = ground(\Sigma)$.

Proof. 1. All facts occurring in a ground constraint $\sigma^g \in \Sigma^A_{\mathcal{DB}}$ must belong to $C^*_{\mathcal{DB}}$. Indeed, by definition of $\Sigma^A_{\mathcal{DB}}$, σ^g contains at least one fact t in $C^*_{\mathcal{DB}}$; each other fact in σ^g is constraint-bounded in $\mathcal{RS}$ with t, and hence it also is in $C^*_{\mathcal{DB}}$.
2. Assume by contradiction that some $\sigma^g \in \Sigma^S_{\mathcal{DB}}$ with $facts(\sigma^g) \cap C^*_{\mathcal{DB}} \neq \emptyset$ exists. Then, Definitions 5.1 and 5.2 imply $facts(\sigma^g) \subseteq C^*_{\mathcal{DB}}$, which contradicts $\sigma^g \in \Sigma^S_{\mathcal{DB}}$. Part 3 is straightforward from Part 1 and Part 2. □

5.2 Focussing on the Affected Database

Having divided the database, we now proceed with showing that the hard part is indeed associated with the affected database.

5.2.1 Limiting Repairs to the Affected Database

The separation property allows us to shed light on the structure of repairs for $\mathcal{DB}$:

Proposition 5.4 (Safe Database) *Let $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ be a relational schema, and let $\mathcal{DB}$ be a database for $\mathcal{RS}$. Then, for each repair $\mathcal{R} \in rep(\mathcal{DB})$ w.r.t. Σ, $S_{\mathcal{DB}} = \mathcal{R} - C^*_{\mathcal{DB}}$.*

Proof. Towards a contradiction, suppose there exists a repair $\mathcal{R}$ for $\mathcal{DB}$ w.r.t. Σ such that $\mathcal{R} - C^*_{\mathcal{DB}} \neq S_{\mathcal{DB}}$. Let us construct a new database $\mathcal{R}'$ in the following way: $\mathcal{R}'$ contains all the facts of $\mathcal{R}$ that are also in $C^*_{\mathcal{DB}}$, and all the facts of $\mathcal{DB}$ that are not in $C^*_{\mathcal{DB}}$. Consider any $\sigma^g \in ground(\Sigma)$. By Proposition 5.3, either (i) $\sigma^g \in \Sigma^A_{\mathcal{DB}}$ or (ii) $\sigma^g \in \Sigma^S_{\mathcal{DB}}$. Moreover, since $\mathcal{R}$ satisfies σ^g, it follows that in Case (i) $\mathcal{R}'$ also satisfies σ^g. Similarly, since $S_{\mathcal{DB}}$ satisfies σ^g in Case (ii) $\mathcal{R}'$ also satisfies σ^g.

Thus, $\mathcal{R}'$ is consistent with Σ. However, it differs from $\mathcal{R}$ only in the tuples that are not in $C^*_{\mathcal{DB}}$, and all these tuples do not need to be repaired; it follows that $\Delta(\mathcal{R}', \mathcal{DB}) \subseteq \Delta(\mathcal{R}, \mathcal{DB})$. This contradicts that $\mathcal{R}$ is a repair of $\mathcal{DB}$. □

Prior to the main result of this section, we show the following lemma:

Lemma 5.5 *Let $\mathcal{RS} = \langle \Psi, \Sigma \rangle$ be a relational schema, and let $\mathcal{DB}$ be a database for $\mathcal{RS}$. Then, for each $S \subseteq S_{\mathcal{DB}}$, we have that*

1. *for each $\mathcal{R} \in rep(S \cup A_{\mathcal{DB}})$ w.r.t. Σ, $(\mathcal{R} \cap C^*_{\mathcal{DB}}) \in rep(A_{\mathcal{DB}})$ w.r.t. $\Sigma^A_{\mathcal{DB}}$;*

2. *for each $\mathcal{R}_a \in rep(A_{\mathcal{DB}})$ w.r.t. $\Sigma^A_{\mathcal{DB}}$, there exists some set of facts S', $S' \cap C^*_{\mathcal{DB}} = \emptyset$, such that $(\mathcal{R}_a \cup S') \in rep(S \cup A_{\mathcal{DB}})$ w.r.t. Σ.*

Proof. (Sketch) 1. Let $\mathcal{R}$ be a repair of $S \cup A_{\mathcal{DB}}$ w.r.t. Σ, and let $\mathcal{R}_a = \mathcal{R} \cap C^*_{\mathcal{DB}}$. Since $\mathcal{R}$ is consistent with Σ, Proposition 5.3 implies that $\mathcal{R}_a$ satisfies all constraints in $\Sigma^A_{\mathcal{DB}}$, while $\mathcal{R} - \mathcal{R}_a$ satisfies all constraints in $\Sigma^S_{\mathcal{DB}}$. Assume $\mathcal{R}_a$ is not a repair of $A_{\mathcal{DB}}$ w.r.t. $\Sigma^A_{\mathcal{DB}}$. Then, there exists some $\mathcal{R}'_a \in rep(A_{\mathcal{DB}})$ w.r.t. $\Sigma^A_{\mathcal{DB}}$, such that $\mathcal{R}'_a <_{A_{\mathcal{DB}}} \mathcal{R}_a$. Since $\mathcal{R}'_a$ satisfies $\Sigma^A_{\mathcal{DB}}$, Proposition 5.3 implies that $\mathcal{R}'_a \cup (\mathcal{R} - \mathcal{R}_a)$ satisfies Σ. But $\mathcal{R}'_a \cup (\mathcal{R} - \mathcal{R}_a) <_{S \cup A_{\mathcal{DB}}} \mathcal{R}_a$, which contradicts that $\mathcal{R}$ is a repair of $S \cup A_{\mathcal{DB}}$.
2. Let $\mathcal{R}_a$ be a repair of $A_{\mathcal{DB}}$ w.r.t. $\Sigma^A_{\mathcal{DB}}$. We show that there exists a repair $\mathcal{R}$ of $S \cup A_{\mathcal{DB}}$ w.r.t. Σ of form $\mathcal{R} = \mathcal{R}_a \cup S'$, where S' is a set that satisfies $\Sigma^S_{\mathcal{DB}}$, such that $\mathcal{R} \cap A_{\mathcal{DB}} = \mathcal{R}_a$. In particular, let $S' \in rep(S)$ w.r.t. $\Sigma^S_{\mathcal{DB}}$ be arbitrary. Using Proposition 5.3, we can see that $\mathcal{R}_a \cup S'$ satisfies Σ; note that $S' \cap C^*_{\mathcal{DB}} = \emptyset$. Thus, if $\mathcal{R}_a \cup S'$ is not a repair of $S \cup A_{\mathcal{DB}}$, there must exist some $\mathcal{R}'$ consistent with Σ such that $\mathcal{R}' <_{S \cup A_{\mathcal{DB}}} \mathcal{R}_a \cup S'$. Since we always can write $\mathcal{R}' = \mathcal{R}'_a \cup \mathcal{R}'_s$, where $\mathcal{R}'_a \subseteq C^*_{\mathcal{DB}}$ satisfies $\Sigma^A_{\mathcal{DB}}$ and $\mathcal{R}'_s = \mathcal{R} - \mathcal{R}'_a$ satisfies $\Sigma^S_{\mathcal{DB}}$, this implies either $\mathcal{R}'_a <_{A_{\mathcal{DB}}} \mathcal{R}_a$ or $\mathcal{R}'_s <_S S'$. This contradicts $\mathcal{R}_a \in rep(A_{\mathcal{DB}})$ w.r.t. $\Sigma^A_{\mathcal{DB}}$ and $S' \in rep(S)$ w.r.t. $\Sigma^S_{\mathcal{DB}}$. □

Armed with the above concepts and results, we now turn to the data integration settings in which we have to repair the retrieved global database $ret(\mathcal{I}, \mathcal{D})$. The following theorem shows that its repairs can be computed by looking only at $A_{ret(\mathcal{I},\mathcal{D})}$.

Theorem 5.6 (Main) *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and let $\mathcal{D}$ be a source database for $\mathcal{I}$. Then,*

1. $\forall \mathcal{R} \in rep_{\mathcal{I}}(\mathcal{D})$, $\exists \mathcal{R}' \in rep(A_{ret(\mathcal{I},\mathcal{D})})$ *w.r.t.* Σ *such that* $\mathcal{R} = \mathcal{R}' \cap C^*_{ret(\mathcal{I},\mathcal{D})} \cup S_{ret(\mathcal{I},\mathcal{D})}$;
2. $\forall \mathcal{R}' \in rep(A_{ret(\mathcal{I},\mathcal{D})})$, $\exists \mathcal{R} \in rep_{\mathcal{I}}(\mathcal{D})$ *w.r.t.* Σ *such that* $\mathcal{R} = \mathcal{R}' \cap C^*_{ret(\mathcal{I},\mathcal{D})} \cup S_{ret(\mathcal{I},\mathcal{D})}$.

Proof. Recall that $ret(\mathcal{I},\mathcal{D}) = S_{ret(\mathcal{I},\mathcal{D})} \cup A_{ret(\mathcal{I},\mathcal{D})}$ and that $rep_{\mathcal{I}}(\mathcal{D})$ coincides with $rep(ret(\mathcal{I},\mathcal{D}))$ w.r.t. Σ. Thus, applying Lemma 5.5, first Part 1 for $S = S_{ret(\mathcal{I},\mathcal{D})}$ and then Part 2 for $S = \emptyset$, we obtain that for every $\mathcal{R} \in rep_{\mathcal{I}}(\mathcal{D})$, there exists some $\mathcal{R}' \in rep(A_{ret(\mathcal{I},\mathcal{D})})$ w.r.t. Σ of form $\mathcal{R}' = (\mathcal{R} \cap C^*_{ret(\mathcal{I},\mathcal{D})}) \cup S'$, where $S' \cap C^*_{ret(\mathcal{I},\mathcal{D})} = \emptyset$. Hence, $\mathcal{R}' \cap C^*_{ret(\mathcal{I},\mathcal{D})} = R \cap C^*_{ret(\mathcal{I},\mathcal{D})}$. By Proposition 5.4, every $\mathcal{R} \in rep_{\mathcal{I}}(D)$ is of form $\mathcal{R} = (\mathcal{R} \cap C^*_{ret(\mathcal{I},\mathcal{D})}) \cup S_{ret(\mathcal{I},\mathcal{D})}$. Therefore, $\mathcal{R} = (\mathcal{R}' \cap C^*_{ret(\mathcal{I},\mathcal{D})}) \cup S_{ret(\mathcal{I},\mathcal{D})}$; this proves 1.

Similarly, applying Lemma 5.5, first Part 1 for $S = \emptyset$ and then Part 2 for $S = S_{ret(\mathcal{I},\mathcal{D})}$, we obtain that for every $\mathcal{R}' \in rep(A_{ret(\mathcal{I},\mathcal{D})})$ w.r.t. Σ, there exists some $\mathcal{R} \in rep_{\mathcal{I}}(\mathcal{D})$ w.r.t. Σ such that $\mathcal{R} = (\mathcal{R}' \cap C^*_{ret(\mathcal{I},\mathcal{D})}) \cup S'$, where $S' \cap C^*_{ret(\mathcal{I},\mathcal{D})} = \emptyset$. Moreover, Proposition 5.4 implies that $S' = S_{ret(\mathcal{I},\mathcal{D})}$, which proves 2. □

As a consequence, for computing the stable models of the retrieved global database $rep_{\mathcal{I}}(\mathcal{D})$, it is sufficient to evaluate the program Π_Σ on $A_{ret(\mathcal{I},\mathcal{D})}$, i.e., to exploit the correspondence $rep_{\mathcal{I}}(\mathcal{D}) \rightleftharpoons \mathrm{SM}(\Pi_\Sigma[A_{ret(\mathcal{I},\mathcal{D})}])$, intersect with $C^*_{ret(\mathcal{I},\mathcal{D})}$, and unite with $S_{ret(\mathcal{I},\mathcal{D})}$.

5.2.2 Further Focussing

Nonetheless, computing $A_{ret(\mathcal{I},\mathcal{D})}$ is expensive in general since it requires computing the closure of $C_{ret(\mathcal{I},\mathcal{D})}$. Furthermore, in repairs of $A_{ret(\mathcal{I},\mathcal{D})}$ many facts not in $C^*_{ret(\mathcal{I},\mathcal{D})}$ might be computed which are stripped off subsequently. Fortunately, in practice, for many important cases this can be avoided: repairs can be made fully local and even focused just on the immediate conflicts in the database.

Proposition 5.7 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and let $\mathcal{D}$ be a source database for $\mathcal{I}$. Then,*

1. *if each $\sigma \in \Sigma$ is of the form (1) with $l > 0$, then each $\mathcal{R}'$ repair of $A_{ret(\mathcal{I},\mathcal{D})}$ w.r.t. Σ satisfies $\mathcal{R}' \subseteq C^*_{ret(\mathcal{I},\mathcal{D})}$;*
2. *if each $\sigma \in \Sigma$ is of the form (1) with $m = 0$, then*
   i) each $\mathcal{R}'$ repair of $A_{ret(\mathcal{I},\mathcal{D})}$ w.r.t. Σ satisfies $\mathcal{R}' \subseteq C_{ret(\mathcal{I},\mathcal{D})}$,
   ii) $C_{ret(\mathcal{I},\mathcal{D})} \subseteq ret(\mathcal{I},\mathcal{D})$, and
   ii) $rep(A_{ret(\mathcal{I},\mathcal{D})})$ w.r.t. Σ coincides with $rep(C_{ret(\mathcal{I},\mathcal{D})})$ w.r.t. Σ. □

This proposition allows us to exploit Theorem 5.6 in a constructive way for many significant classes of constraints, for which it implies a bijection between the repairs of the retrieved global database, $ret(\mathcal{I},\mathcal{D})$, and the repairs of its affected part $A_{ret(\mathcal{I},\mathcal{D})}$ w.r.t. the constraints Σ. In particular, Condition 1 is satisfied by all constraints that do not unconditionally enforce inclusion of some fact in every repair, while Condition 2 is satisfied by constraints that can be repaired by just deleting facts from the database, such as key constraints, functional dependencies, and exclusion dependencies.

According to Theorem 5.6, in case of Condition 2 the set $rep_{\mathcal{I}}(\mathcal{D})$ can be obtained by simply computing the repairs of the conflicting facts, $C_{ret(\mathcal{I},\mathcal{D})}$, in place of $A_{ret(\mathcal{I},\mathcal{D})}$ and by adding $S_{ret(\mathcal{I},\mathcal{D})}$ to each repair. We also point out that the computation of the set $C_{ret(\mathcal{I},\mathcal{D})}$ can be carried out very efficiently, by means of suitable SQL statements.

The following corollary formalizes the above discussion for Condition 2.

Corollary 5.8 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and let $\mathcal{D}$ be a source database for $\mathcal{I}$. Assume all constraints in Σ are of the form (1) with $m = 0$. Then, there exists a bijection $\mu : rep_{\mathcal{I}}(\mathcal{D}) \rightarrow rep(C_{ret(\mathcal{I},\mathcal{D})})$, such that for each repair $\mathcal{R} \in rep_{\mathcal{I}}(\mathcal{D})$, $\mathcal{R} = \mu(\mathcal{R}) \cup S_{ret(\mathcal{I},\mathcal{D})}$ (where $C_{ret(\mathcal{I},\mathcal{D})} \subseteq ret(\mathcal{I},\mathcal{D})$).* □

5.3 Focussing with Query Answering

Let us turn our attention to the evaluation of a user query q. According to the definition of consistent answers (Section 2.3), we need to evaluate q over each repair $\mathcal{R} \in rep_{\mathcal{I}}(\mathcal{D})$, and by Theorem 5.6 we can exploit the correspondence $rep_{\mathcal{I}}(\mathcal{D}) \rightleftharpoons \mathrm{SM}(\Pi_\Sigma[A_{ret(\mathcal{I},\mathcal{D})}])$ for computing each such $\mathcal{R}$. More precisely, we need to recombine the repairs of $A_{ret(\mathcal{I},\mathcal{D})}$ with $S_{ret(\mathcal{I},\mathcal{D})}$ computed in the decomposition step as stated by the following theorem.

Theorem 5.9 *Let $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, let $\mathcal{D}$ be a source database for $\mathcal{I}$, and let q be a query over $\mathcal{G}$. Then,*

$$ans(q, \mathcal{I}, \mathcal{D}) = \bigcap_{\mathcal{R} \in rep(A_{ret(\mathcal{I},\mathcal{D})})} \Pi_q[\mathcal{R} \cap C^*_{ret(\mathcal{I},\mathcal{D})} \cup S_{ret(\mathcal{I},\mathcal{D})}] \tag{14}$$

Note that the number of repairs of $A_{ret(\mathcal{I},\mathcal{D})}$ is exponential in the number of violated constraints, and hence efficient computation of the intersection in (14) requires some intelligent strategy, as will be discussed in the next section.

We close this section with briefly addressing databases over infinite domains. Here, usually a safety condition is imposed on constraints and queries in order to assure finite database instances and query results. Namely, a constraint of the form (1) is safe, if each variable occurring in some B_j or ϕ_k also occurs in some A_i, and a query $q(\vec{\mathbf{x}})$ is safe, if each variable in $\vec{\mathbf{x}}$ is recursively bound to some value occurring in the database. Note that important types of constraints such as key, functional and exclusion dependencies are safe. It appears that under safe constraints and queries, database repairs fully localize to the *active domain*, $AD(\mathcal{DB})$, of a database $\mathcal{DB}$, i.e., the values occurring in $\mathcal{DB}$. As for repair, the (finite or infinite) domain can thus be equivalently replaced with $AD(\mathcal{DB})$, and for query answering, by $AD(\mathcal{DB})$ plus the constants in $q(\vec{\mathbf{x}})$. Together with further domain pruning, this may lead to considerable savings.

6 Efficient Recombination

In this section, we describe a technique for implementing the recombination step in a way which circumvents the evaluation of Π_q on each repair of $ret(\mathcal{I},\mathcal{D})$ separately. For the

${player_m}^{ret(\mathcal{I}_0,\mathcal{D}_0)}$:

10	*Totti*	*RM*	'11'
9	*Beckham*	*MU*	'11'

${team_m}^{ret(\mathcal{I}_0,\mathcal{D}_0)}$:

RM	*Roma*	10	'10'
MU	*Man. Utd.*	8	'11'
RM	*Real Madrid*	10	'01'

${coach_m}^{ret(\mathcal{I}_0,\mathcal{D}_0)}$:

7	*Ferguson*	*MU*	'11'

Figure 7: The retrieved global database of our running example after marking.

sake of simplicity, we deal here with constraints of the form (1), when $l > 0$ and $m = 0$. In this case, according to Proposition 5.7, $rep(A_{ret(\mathcal{I},\mathcal{D})})$ coincide with $rep(C_{ret(\mathcal{I},\mathcal{D})})$, and $\mathcal{R} \subseteq C_{ret(\mathcal{I},\mathcal{D})} \subseteq ret(\mathcal{I},\mathcal{D})$ for each $\mathcal{R} \in rep(C_{ret(\mathcal{I},\mathcal{D})})$. Furthermore, the thesis in Theorem 5.9 can be rewritten as $ans(q,\mathcal{I},\mathcal{D}) = \bigcap_{\mathcal{R}\in rep(C_{ret(\mathcal{I},\mathcal{D})})} \Pi_q[\mathcal{R} \cup S_{ret(\mathcal{I},\mathcal{D})}]$.

The basic idea of our approach is to encode all repairs into a single database over which the query Π_q can be evaluated by means of standard database techniques.

6.1 Marked Database

For each global relation r, we construct a new relation r_m by adding an auxiliary attribute *mark*. The mark value is a string $'b_1 \dots b_n'$ of bits $b_i \in \{0,1\}$ such that, given any tuple t, $b_i = 1$ if and only if t belongs to the i-th repair $\mathcal{R}_i \in rep(C_{ret(\mathcal{I},\mathcal{D})}) = \{\mathcal{R}_1,\dots,\mathcal{R}_n\}$, for every $i \in \{1,\dots,n\}$ (indexing the repairs is easy, e.g. using the order in which the deductive database system computes them). The set of all marked relations constitutes a *marked* database, denoted by $M_{ret(\mathcal{I},\mathcal{D})}$. Note that the facts in $S_{ret(\mathcal{I},\mathcal{D})}$ (the bulk of data) can be marked without any preprocessing, as they belong to every repair $\mathcal{R}_i$; hence, their mark is $'11\cdots1'$. For our running example, the marked database derived from the repairs in Figure 3 is shown in Figure 7.

We next show how the original query q can be reformulated in a way such that its evaluation over the marked database computes the set of consistent answers to q. Before proceeding, we point out that our recombination technique also applies in the presence of constraints of general form. In such a case, the source of complexity lies in the computation of $C^*_{ret(\mathcal{I},\mathcal{D})}$.

6.2 Query Reformulation

We next focus on non-recursive Datalog queries and provide a technique for rewriting them into SQL. The extension to non-recursive Datalog$^\neg$ queries is straightforward, and will not be described here.

Let a query $q(\vec{\mathbf{x}})$, where $\vec{\mathbf{x}} = X_1,\dots X_n$, be given by rules $q(\vec{\mathbf{x}}) \leftarrow q_j(\vec{\mathbf{y}}_j)$, $1 \leq j \leq n$, where each $q_j(\vec{\mathbf{y}}_j)$ is a conjunction of atoms $p_{j,1}(\vec{\mathbf{y}}_{j,1}), ..., p_{j,m}(\vec{\mathbf{y}}_{j,m})$, where $\vec{\mathbf{y}}_j = \bigcup_{i=1}^{m} \vec{\mathbf{y}}_{j,i}$. Moreover, let us call each variable Y such that $Y \in \vec{\mathbf{y}}_{j,i}$ and $Y \in \vec{\mathbf{y}}_{j,h}$ a *join variable of* $p_{j,i}$ *and* $p_{j,h}$.

In reformulating the query, we use the following two functions `ANDBIT` and `SUMBIT`:

- `ANDBIT` is a binary function that takes as its input two bit strings $'a_1 \dots a_n'$ and $'b_1 \dots b_n'$ and returns $'(a_1 \wedge b_1) \dots (a_n \wedge b_n)'$, where $\wedge$ denotes Boolean and;

- SUMBIT is an aggregate function such that given m strings of the form $'b_{i,1} \ldots b'_{i,n}$, it returns $'(b_{1,1} \vee \ldots \vee b_{m,1}) \ldots (b_{1,n} \vee \ldots \vee b_{m,n})'$, where $\vee$ denotes Boolean or.

Then, each $q(\vec{\mathbf{x}}) \leftarrow q_j(\vec{\mathbf{y}}_j)$ can be translated in the SQL statement SQL_{q_j} of the form

$\texttt{SELECT } \mathbf{X_1}, \ldots \mathbf{X_n}, (\mathbf{p_{j,1}.mark} \texttt{ ANDBIT} \ldots \texttt{ANDBIT } \mathbf{p_{j,m}.mark}) \texttt{ AS } \mathbf{mark}$
$\texttt{FROM } \mathbf{p_1} \ldots \mathbf{p_m}$
$\texttt{WHERE } \mathbf{p_{j,i}.Y{=}p_{j,h}.Y}$ *(for each join variable* $\mathbf{Y}$ *of* $\mathbf{p_{j,i}}$ *and* $\mathbf{p_{j,h}}, 1 \leq i, h \leq m$*).*

In addition to the answers to q_j, the statement computes the marks of the repairs in which an answer holds by applying the ANDBIT operator to the *mark* attributes of the joined relations. The results of all SQL_{q_j} can be collected into a view uq_m by the SQL statement SQL_{q_1} UNION $SQL_{q_2} \ldots$ UNION SQL_{q_n}. Finally, SQL_q is the SQL statement

$\texttt{SELECT } \mathbf{X_1}, \ldots \mathbf{X_n}, \texttt{SUMBIT}(\mathbf{mark})$
$\texttt{FROM } \mathbf{uq_m}$
$\texttt{GROUP BY } \mathbf{X_1}, \ldots \mathbf{X_n}$
$\texttt{HAVING SUMBIT}(\mathbf{mark}) =' 1 \ldots 1'.$

It computes query answers by considering only the facts that belong to all repairs.

Proposition 6.1 *Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, a source database $\mathcal{D}$ and a query q over $\mathcal{G}$, the set $ans(q, \mathcal{I}, \mathcal{D})$ coincides with the set computed by executing SQL_q over the marked database $M_{ret(\mathcal{I},\mathcal{D})}$.*

Example 6.2 *For $q(X) \leftarrow player(X, Y, Z)$; $q(X) \leftarrow team(V, W, X)$, SQL_q is as follows:*

$\texttt{CREATE VIEW } \mathbf{uq_m}\ (\mathbf{X},\ \mathbf{mark}) \texttt{ AS}$
$\texttt{SELECT } Pcode, \mathbf{mark} \texttt{ FROM } player$
$\texttt{UNION}$
$\texttt{SELECT } Tleader, \mathbf{mark} \texttt{ FROM } team;$

$\texttt{SELECT } \mathbf{X}, \texttt{SUMBIT}(\mathbf{mark})$
$\texttt{FROM } \mathbf{uq_m}$
$\texttt{GROUP BY } \mathbf{X}$
$\texttt{HAVING SUMBIT}(\mathbf{mark}) =' 11';$

It is easy to see that the answers consist of the codes 8, 9, 10. □

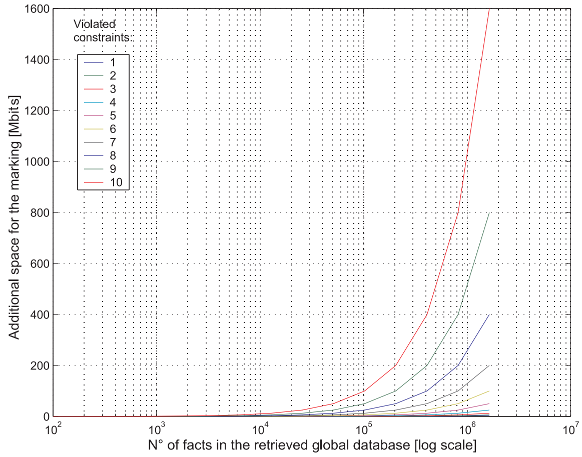


Figure 8: Additional space for marking w.r.t. the size of the global database $ret(\mathcal{I}, \mathcal{D})$

6.3 Scaling the Technique

Since the number of repairs is exponential in the number of violated constraints, the marking string can be of considerable length. For instance, 10 constraint violations, involving two facts each, give rise to 2^{10} repairs, and hence 1 Kbit is needed for marking each tuple.

Figure 8 shows the additional space needed in our running example, depending on the size of the retrieved global database, $ret(\mathcal{I}, \mathcal{D})$, for different numbers of constraint violations (assuming each violation has two conflicting facts). This clearly limits the effectiveness of the technique. For instance, a retrieved global database of 'only' 100000 facts requires almost 100 Mbits additional space, most of it for marking the safe part.

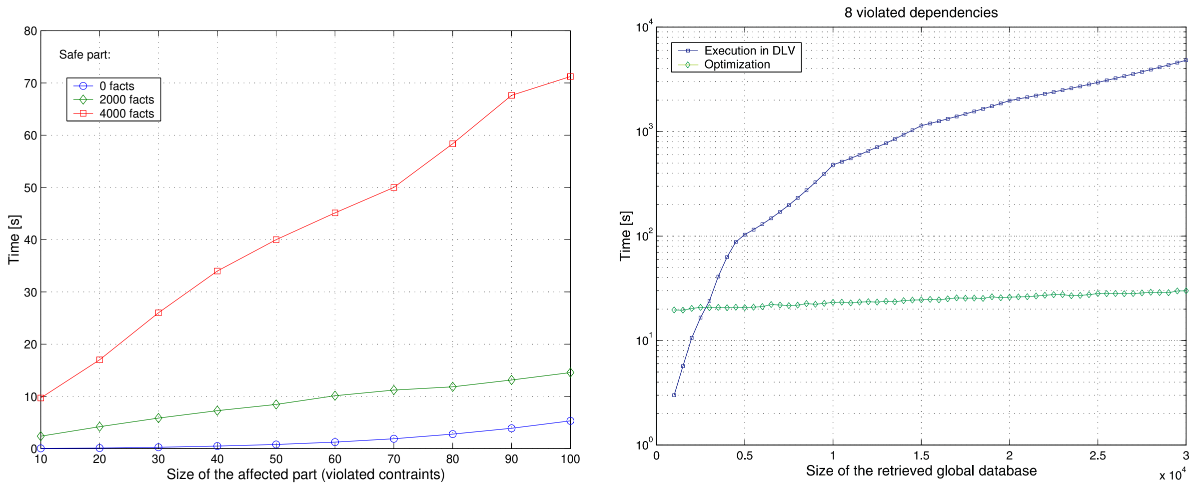
We thus refine our technique to mark only the affected database part. More specifically, we split each global relation r into a "safe" part $r_{safe}(\mathcal{I}, \mathcal{D}) = \{r(t) \mid r(t) \in S_{ret(\mathcal{I},\mathcal{D})}\}$ and an "affected" part $r_{aff}(\mathcal{I}, \mathcal{D}) = \{r(t) \mid r(t) \in A_{ret(\mathcal{I},\mathcal{D})}\}$.

With this approach, the additional space depends only on the size of $A_{ret(\mathcal{I},\mathcal{D})}$ but not on the size of $S_{ret(\mathcal{I},\mathcal{D})}$. For example, for 10 constraint violations involving two tuples each, the additional space is bound by 20 Kbits, independently on the size od $ret(\mathcal{I}, \mathcal{D})$. The improvements w.r.t. to the naive implementation should be evident.

We now need to use a different rewriting of q, obtained as follows:

- in each rule $q(\vec{\mathbf{x}}) \leftarrow q_j(\vec{\mathbf{y}}_j)$, we replace each atom $p(\vec{\mathbf{y}})$, with $p_{aff}(\vec{\mathbf{y}}) \vee p_{safe}(\vec{\mathbf{y}})$;
- we then rewrite each resulting formula into disjunctive normal form $q_{j,1}(\vec{\mathbf{z}}) \vee \cdots \vee q_{j,n_j}(\vec{\mathbf{z}})$ and replace each $q(\vec{\mathbf{x}}) \leftarrow q_j(\vec{\mathbf{y}}_j)$ with the rules $q(\vec{\mathbf{x}}) \leftarrow q_{j,i}(\vec{\mathbf{z}})$ for each $1 \leq i \leq n_j$;
- each $q_{j,i}(\vec{\mathbf{z}})$ is of form $p_{safe_{j,1}}(\vec{\mathbf{z}}_1), \ldots, p_{safe_{j,n}}(\vec{\mathbf{z}}_n), p_{aff_{j,n+1}}(\vec{\mathbf{z}}_{n+1}), \ldots, p_{aff_{j,m}}(\vec{\mathbf{z}}_m)$, and is evaluated by a statement $SQL_{q_{j,i}}$, in which $'1 \ldots 1'$ replaces every $p_{safe_{j,k}}.mark$.

Notice that this rewriting is exponential in the size of q (more precisely, in the number of atoms). However, as commonly agreed in the database community, the overhead in query complexity usually pays off the advantage gained in data complexity.



(a) (b)

Figure 9: (a) Execution time in DLV system w.r.t. $|A_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}|$, for different sizes of $S_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}$. (b) Comparison with the optimization method.

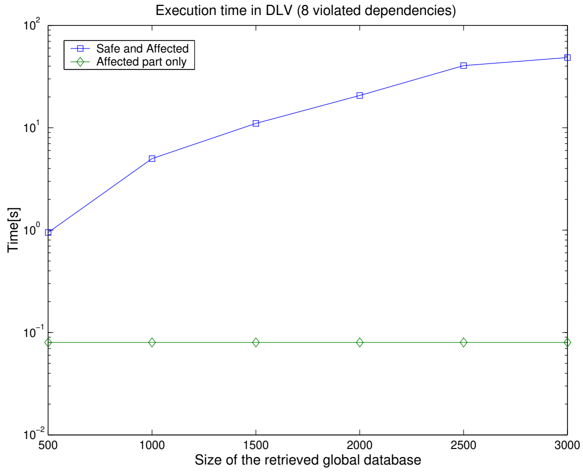
It should be easy to see that, by allotting 5 Mbytes marking space, our technique may scale up to 20 violated constraints involving two tuples each. With this marking space, any number of violations n can be handled by evaluating q incrementally over a sequence of partially marked databases $M^i_{ret(\mathcal{I},\mathcal{D})}$, $1 \leq i \leq \lceil n/20 \rceil$, grouping the marks of 20 repairs at a time, i.e., each relation r in $M^i_{ret(\mathcal{I},\mathcal{D})}$ is marked with the bits $'b_{20(i-1)+1} \cdots b'_{20i}$ of $'b_1 \cdots b'_n$.

6.4 Experimental Results

Here, we present experimental results for the focussing and repair compilation techniques obtained by means of a prototype implementation that couples the DLV deductive database system [29] with Postgresql, a relational DBMS which allows for a convenient encoding of the `ANDBIT` and `SUMBIT` operators. Notice that DLV supports disjunction, which is needed for encoding universal constraints into programs Π_Σ, since computing consistent answers in this setting is Π^p_2-complete [38]. The experiments have been carried out on a 1200MHz - 256MB AMD processor under the Linux operating system.

Football Teams For our running example, we built a synthetic data set $\mathcal{D}_{syn}$, in which the facts in *coach* and *team* satisfy the key constraints, while facts in *player* violate it. Each violation consists of two facts that coincide on the attribute *Pcode* but differ on the attributes *Pname* or *Pteam*; note that these facts constitute $A_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}$. For the query $q(X) \leftarrow player(X,Y,Z)$; $q(X) \leftarrow team(V,W,X)$, we measured the execution time of the program $\Pi_{\mathcal{I}_0}(q)$ in DLV depending on $|A_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}|$ for different fixed values of $|S_{ret(\mathcal{I}_0,\mathcal{D}_{syn})}|$, viz. (i) 0, (ii) 2000, and (iii) 4000. The results are shown in Fig. 9.(a).

We point out that in Case (i), in which $\Pi_{\mathcal{I}_0}(q)$ is evaluated only on the affected part, the execution time scales well to a significant number of violations. On the other hand,



(a) (b)

Figure 10: (a) Execution time w.r.t. the size of the global database: advantages of the decomposition step. (b) Execution time in DLV and in the optimization method.

the evaluation of $\Pi_{\mathcal{I}_0}(q)$ on the whole database is significantly slower; in fact, a small database of 4000 facts will require up to 40 seconds for 50 violated constraints.

Figure 9.(b) shows a comparison (in log scale) between the consistent query answering by a single DLV program and the optimization approach proposed in this paper. Interestingly, for a fixed number of violations (8 in the figure) the growth of the running time of our optimization method under varying database size is negligible. In fact, a major share ($\sim$ 20 seconds) is used for computing repairs of the affected database, plus marking and storing them in Postgresql; the time for query evaluation itself is negligible. In conclusion, for very small databases (1000 facts), consistent query answering by straight evaluation in DLV may be considered viable; nonetheless, for real databases, the asymptotic behavior shows that our approach outperforms a naive implementation (e.g., a database of 30,000 facts requires more then one hour for computing consistent answers).

3Coloring As a further example, we encoded the classical NP-complete graph 3-coloring problem into querying a data integration system $\mathcal{I}_1 = \langle \mathcal{G}_1, \mathcal{S}_1, \mathcal{M}_1 \rangle$. The global schema $\mathcal{G}_1$ has relations $edge(Node, Node)$ and $colored(Node, Color)$, and $Node$ is the key for $colored$.

For a global database $\mathcal{DB}$, we fixed a number of nodes and generated facts in $edge$ producing a graph; moreover, for each node i, we generated three facts: $colored(i, red)$, $colored(i, blue)$, $colored(i, yellow)$. Clearly $\mathcal{DB}$ is inconsistent, and due to the key constraint only one of the three facts can remain in the database. The remaining nodes are precolored, i.e., for them only one of the three facts above is in $\mathcal{DB}$.

Now consider the query $q \leftarrow edge(X, Y), colored(X, C), colored(Y, C)$. As easily seen, it evaluates to true on $\mathcal{DB}$ iff there is no legal 3-coloring for the graph in $\mathcal{DB}$.

Figure 10.(a) reports the execution time for deriving consistent answers by looking at the whole database $\mathcal{DB}$ and by looking at its affected part only, varying the size of $\mathcal{DB}$. Obviously, our approach requires some further effort in the recombination: Figure 10.(b) reports the time needed for the whole process (almost 10 seconds).

We stress that the implementation is a rather crude prototype, leaving quite some room for improvements. Nevertheless, its advantage over the standard technique is evident.

7 Query Plan Generation

The Query Reformulator & Optimizer module creates both repair program and query plan. While we have discussed how to optimize repair programs in detail in the earlier sections, we now focus on the query plan.

7.1 Role of the Query Plan

The query plan is the main interface from upper levels to the DAT layer in the runtime architecture of the INFOMIX system. Its main purpose is to communicate to the DAT layer which of the external source data is to be loaded.

The Query Reformulator & Optimizer module singles out

- which sources are needed to answer query Q,
- whether wrapper execution might be bypassed in some cases (using DLV^{DB} and information in the metadata repository),
- whether queries to a source might be modified (e.g., to push selections).

These informations must then be passed on to the Wrapper Executor via the query plan. In order to incorporate these requirements, we define a query plan as follows:

Definition 7.1 *A* Query Plan *is a partially ordered set of elements $\langle W, Q_W \rangle$ where W is a wrapper ID and Q_W is the actual query associated to it. Q_W can be empty (denoted as $\Box$).*

The query plan is therefore able to express a temporal order for a set of wrappers to be invoked, where each wrapper might be specialized according to Q_W. If $Q_W = \Box$ and wrapper W requires a query, or if Q_W is not compatible with W, then the standard query as specified in the wrapper description in the Metadata Repository should be used. If $Q_W \neq \Box$ and W does not support queries, then Q_W is simply ignored.

7.2 Query Composition

For actually constructing a query plan, the Query Reformulator & Optimizer module has to determine the external sources to be loaded, and, on a more fine-grained level, what fraction of these external sources are to be loaded.

Both tasks involve determining which sources (and hence wrappers) are relevant to the query. This involves a task called *query composition* (cf. [1]), because both the user query and the mapping can be seen as a query (recall that we consider only GAV mappings in this report), and the task is then to "plug" the user query onto the mapping query and to simplify it (possibly getting rid of global relations, as they are only views).

An important issue in our setting is that in addition to predicates in the query also predicates linked to query predicates via constraints over the global schema are relevant to the query. In some sense, this can be seen as an extended query, which then needs composition via the mapping.

Let us first formalize how to determine the extended query predicates, which are termed relevant global predicates:

Definition 7.2 Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and a query q over $\mathcal{I}$, we say that a relation r in $\mathcal{G}$ is *relevant w.r.t.* q if

(a) r is a predicate in the body of q, or

(b) r is involved in a dependency in Σ^* with a predicate s which is relevant for q.

We denote by $ret_r(\mathcal{I}, \mathcal{D}, q)$ the subset of $ret(\mathcal{I}, \mathcal{D})$ that contains only the extension of tuples that are relevant w.r.t. q, i.e.,

$$ret_r(\mathcal{I}, \mathcal{D}, q) = \{r(t) \in ret(\mathcal{I}, \mathcal{D}) \mid r \text{ is relevant w.r.t. } q\}.$$

$\square$

Example 7.3 *Reconsider Example 2.5 and the query* $q(X) \leftarrow player(X, Y, Z)$; $q(X) \leftarrow team(V, W, X)$ *the relations player and team are relevant due to condition (a), whereas relation coach is relevant due to condition (b), as it is involved in a constraint with the relation player. Hence,* $ret_r(\mathcal{I}_0, \mathcal{D}_0, q) = ret(\mathcal{I}_0, \mathcal{D}_0)$.

After having determined which of the global relations are relevant to the query, we can—via the mapping—determine which of the source relations (and hence wrappers) are relevant to the query.

Definition 7.4 Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G} = \langle \Psi, \Sigma \rangle$, and a query q over $\mathcal{I}$, we say that a relation r_s in $\mathcal{S}$ is *relevant w.r.t.* q if r_s occurs in the body of a rule in $\mathcal{M}$ such that the head of the rule is a global relation r which is relevant to q.

We denote by $\mathcal{D}_q$ the subset of $\mathcal{D}$ that contains only tuples that are relevant w.r.t. q, i.e., $\mathcal{D}_q = \{r_s(t) \in \mathcal{D} \mid r_s \text{ is relevant w.r.t. } q\}$. $\square$

Proposition 7.5 *Given a data integration system* $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, *a source database* $\mathcal{D}$ *and a query* q *over* $\mathcal{I}$, *it holds that* $ans(q, \mathcal{I}, \mathcal{D}) = ans(q, \mathcal{I}, \mathcal{D}_q)$.

Query composition is still an active area of research, also at the theoretical level, cf. [30]. However, we do not investigate this direction in INFOMIX in depth, since this is a complex research task of its own right which seems to be independent of the particular INFOMIX approach.

7.3 Pushing Selections to Wrappers

In the previous section, the query plan was composed of a subset of the source relations to be retrieved. In this section, we will try to move yet one step further and pose restrictions on these relations.

Indeed, when looking at the INFOMIX query answering design in Figure 5, the process of wrapping data (the lowest level of query answering) can be an expensive one. For example this process might entail accessing several Internet sites even for one source, which is naturally quite slow.

So it would be desirable to minimize the data which is needed to be transmitted, and this could be done by propagating restrictions which are given implicitly inside the query or the mapping down to the source relations and hence to the wrappers.

However, there are some caveats. As in the previous section, the global constraints must be taken into account, as we will explain below. Additionally, it depends on the type of the wrapper whether it is possible at all to restrict the data it should retrieve. Recall from deliverable D6.2 that INFOMIX supports three classes of wrappers:

1. Code Wrappers;
2. Query Wrappers;
3. Visual Wrappers.

Concerning Code Wrappers, there is no hope to restrict this kind of wrapper during runtime. Its sole purpose is to execute the same code again and again. Note that code wrappers feature a parameter, but its sole purpose is to facilitate wrapper generation during design time, as the semantics of the parameter might be known to the wrapper designer, but cannot be known to the runtime system, as it is potentially different for each code wrapper.

Query Wrappers are quite different in this respect. Even though a query wrapper can support an arbitrary query language, it is known which one through the metadata repository. So, one can implement generic restrictions for some of the query languages. In practice, most wrappers will support SQL as a query language. It is therefore advisable to devise methods for imposing restrictions onto SQL queries. In the simplest case, this means adding conditions (selections) to the WHERE clause of an SQL query. As for quiery composition, combining different query languages becomes and issue, like e.g. SQL and XML in SilkRoute [31]). This is a challenging issue, but like discussed for query composition above, we do not focus on it since this seems to be of its own right and orthogonal to the particular INFOMIX approach.

Visual Wrappers also have a means for query specialization, but no generic language such as SQL can be exploited by them. Rather, some parameters can be bound to values, or some tool-specific query might be posed. It is therefore possible to employ specialization techniques for visual wrappers, but they will in general be less generic and hence require more coding efforts.

Let us first study why it is not correct to consider only the user query without taking into account the global constraints.

Example 7.6 *Recall Example 2.5, but now with the query* $q_1(X) \leftarrow team(X,' Roma', Z)$*. Without considering the constraints (as specified in Example 2.1), one could believe that it is sufficient to access sources in a way such that the global relations are filled only with tuples where* $Tname =' Roma'$*. But then, referring to the database of Example 2.3, any such retrieved database does not violate the key constraint on team, and hence the answer to the query is* $q_1('RM')$*, while* $ans(q_1, \mathcal{I}_0, \mathcal{D}_0) = \emptyset$*.*

On the other hand, for the query $q_2(Y) \leftarrow team('RM', Y, Z)$ *such an anomaly cannot be witnessed, here it is sufficient to consider retrieved databases where* $Tcode =' RM'$ *without changing the semantics.*

We will now formalize this intuition.

Definition 7.7 *Given a query* q*, and an integration system* $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$*, where* $\mathcal{G} = \langle \Psi, \Sigma \rangle$*, all atoms occurring in the body of a query rule are necessary to answer* q *(w.r.t.* $\mathcal{I}$*). Let* a *be an atom which is necessary to answer* q *w.r.t.* $\mathcal{I}$ *and which is unifiable with an* A_{i_a} *of a constraint* $\forall(\vec{\mathbf{x}}) \bigwedge_{i=1}^{l} A_i \supset \bigvee_{j=1}^{m} B_j \vee \bigvee_{k=1}^{n} \phi_k \in \Sigma$ *(where all variables in the constraint are distinct from those in* a*) by means of a most general unifier* θ*. Then, all* $A_i\theta$ $(1 \leq i \leq l,\ i \neq i_a)$ *and* $B_j\theta$ $(1 \leq j \leq m)$ *are necessary to answer* q *w.r.t.* $\mathcal{I}$*. Variable names are immaterial for the notion of being necessary to answer a query, hence variants of atoms (atoms which differ only by variable renaming) should not be considered.*

Finally, an atom a *is relevant to* q *w.r.t.* $\mathcal{I}$*, if it is necessary to answer* q *w.r.t.* $\mathcal{I}$ *and if* a *is not subsumed by another atom* b *which is necessary to answer* q *w.r.t.* $\mathcal{I}$*.*

Example 7.8 *Considering* q_1*,* $team(X,' Roma', Z)$ *is clearly necessary to answer* q_1*. But this atom is clearly unifiable with the constraint*

$$team(X_1, Y_1, Z_1) \wedge team(X_1, Y_2, Z_2) \supset Y_1{=}Y_2$$

using $\theta = \{Y_1/'Roma'\}$ *making also* $team(X_1, Y_2, Z_2)$ *necessary to answer the query. Similarly, one can also determine that* $team(X_1, Y_1, Z_1)$ *is necessary to answer the query, but this atom is clearly only a variant of* $team(X_1, Y_2, Z_2)$*. Also,* $team(X_3, Y_3, Z_3) \wedge team(X_3, Y_4, Z_4) \supset Z_3{=}Z_4$ *only gives rise to variants of* $team(X_1, Y_2, Z_2)$*. Finally, also* $coach(X_5, Y_5, Z_5)$ *and* $player(X_6, Y_6, Z_6)$ *can be determined to be necessary to answer* q_1*.*

So, $team(X,' Roma', Z)$*,* $team(X_1, Y_2, Z_2)$*,* $coach(X_5, Y_5, Z_5)$*, and* $player(X_6, Y_6, Z_6)$ *are necessary to answer* q_1*. Since* $team(X,' Roma', Z)$ *is subsumed by* $team(X_1, Y_2, Z_2)$*, only* $team(X_1, Y_2, Z_2)$*,* $coach(X_5, Y_5, Z_5)$*, and* $player(X_6, Y_6, Z_6)$ *are relevant to* q_1 *w.r.t.* $\mathcal{I}_0$*.*

For q_2 *the situation is different:* $team('RM', Y, Z)$ *is necessary to answer* q_2*, but* $team(X_1, Y_1, Z_1) \wedge team(X_1, Y_2, Z_2) \supset Y_1{=}Y_2$ *renders only variants of* $team('RM', Y, Z)$ *necessary. For coach and player, the situation is the same as above, so* $team('RM', Y, Z)$*,* $coach(X_5, Y_5, Z_5)$*, and* $player(X_6, Y_6, Z_6)$ *are relevant to* q_2 *w.r.t.* $\mathcal{I}_0$*.*

We will next define a modified query which is composed of the atoms which are relvant to the original query. This query is not to be understood as being equivalent to the original query, it is used only for the purpose of optimizing source data access, while the original query is still to be used for the answering process.

Definition 7.9 *Given a query q and data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, we define its data access query $\hat{q}$ as $\hat{q} \leftarrow A$, for each atom A which is relevant to q w.r.t. $\mathcal{I}$.*

Since the mapping $\mathcal{M}$ is assumed to be a Datalog$^{\neg_s}$ program and $\hat{q}$ is a non-recursive Datalog program, one can subsequently use arbitrary query optimization methods (see, e.g., [36, 43, 1, 44] for overviews) on $\hat{q} \cup \mathcal{M}$, e.g., in order to push constants from $\hat{q}$ to the source relations. Let PO be the result of such an optimization. One can then analyze the atoms over source relations in PO, taking only the most general ones for each source relation. These atoms implicitly define a specialization on the source relation, which can then be propagated to the associated wrapper, if the type of the wrapper allows it. Let $\mathcal{D}_{\hat{q}}$ be a source database which has been restricted in such a way. We can then show the following result, which is the analogon to Proposition 7.5:

Proposition 7.10 *Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, a source database $\mathcal{D}$ and a query q over $\mathcal{I}$, it holds that $ans(q, \mathcal{I}, \mathcal{D}) = ans(q, \mathcal{I}, \mathcal{D}_{\hat{q}})$.*

Given PO, also more general restriction would be possible, e.g., also source interrelations could be taken into account, allowing for semi-joins and similar distributed operations. Again we refer to [43] and [1, 44] for details, and leave these issues for future work. Furthermore, as already mentioned above, research on efficient composition of mappings from sources to global relations and query wrappers in general, is judged to be orthogonal to the INFOMIX approach and further investigation left for future work.

8 Conclusion

In this document, we have addressed several aspects of optimization which are applicable in the INFOMIX system. After having discussed in general at which parts of the system optimizations are possible, and outlining many of the possibilities, we have discussed some specific optimization methods:

Equivalences of Logic Programs Since the use of computational logic, and of logic programming in particular, is one of the distinguishing features of INFOMIX, we have dealt with optimization of logic programming in detail. Starting in a propositional setting for simplicity, we have picked up on current research and made significant contributions in this area. We have subsequently lifted some of these results to the first-order case, arriving at a flexible optimization framework for logic programs as they occur in the INFOMIX system.

Focussing and Efficient Recombination While in Equivalences of Logic Programs we were dealing with a general optimization method for logic programs, Focussing and Recombination takes the particular data-integration-optimizations setting, and in particular the use of logic programs for repairs of the retrieved database, into account. First, a technique has been devised, which allows for considering only a part of the retrieved data for repair ("Focussing"). Subsequently, we have tackled the problem of recombining the repaired data with the data which has not been submitted for repair ("Efficient Recombination"). The latter technique is also useful in general, and not only in connection with Focussing. Experiments with a prototype implementation clearly show the practical benefit of these techniques.

Query Plan Generation Finally, we have addressed the problem of retrieving the source data, which is subsequently submitted to repair, from the external data sources. To this end, we have described the structure of the Query Plan in more detail, and discussed methods for restricting source data retrieval. First, we have described a simple technique, which determines whether some sources are not necessary at all to answer the user query. Subsequently, we have generalized this method to also allow for propagation of restrictions from the user query and the mapping to the sources. It turns out that a large part of these issues can be solved using standard optimization techniques from database research. Only the question of how to deal with integrity constraints on the global schema is somewhat different and has been discussed in this report.

To summarize, we have described several novel techniques for optimization in the INFOMIX system and sketched many issues which have potential for optimization which could be dealt with in future work.

Acronyms

- DAT = Data Acquisition and Transformation.
- DBMS = Database Management System.
- GAV = Global As View.
- IIDF = Internal Integration Data Format.
- IQL = INFOMIX Query Language
- ISDF = INFOMIX Source Data Format.
- ODBC = Open Database Connectivity.
- RDBMS = Relational DBMS.
- SQL = Structured Query Language.
- WP = Workpackage.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic Sets and Other Strange Ways to Implement Logic Programs. In *Proc. Int. Symposium on Principles of Database Systems*, pages 1–16, 1986.

[3] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. *Journal of Logic Programming*, 10(1–4):255–259, 1991.

[4] Rachel Ben-Eliyahu and Rina Dechter. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

[5] Stefan Brass and Jürgen Dix. Disjunctive Semantics Based upon Partial and Bottom-Up Evaluation. In Leon Sterling, editor, *Proceedings of the 12th Int. Conf. on Logic Programming*, pages 199–213, Tokyo, June 1995. MIT Press.

[6] Stefan Brass and Jürgen Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 38(3):167–213, 1999. Extended Abstract appeared as [5].

[7] Loreto Bravo and Leopoldo Bertossi. Logic programming for consistently querying data integration systems. In *Proceedings of IJCAI'03*, 2003.

[8] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. Space Efficiency of Propositional Knowledge Representation Formalisms. *Journal of Artificial Intelligence Research*, 13:1–31, 2000.

[9] Marco Cadoli, Thomas Eiter, and Georg Gottlob. Default Logic as a Query Language. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):448–463, May/June 1997.

[10] Andrea Calì, Domenico Lembo, Riccardo Rosati, and Marco Ruzzi. Experimenting Data Integration with DIS@DIS. In *Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE 2004)*, 2004. To appear.

[11] Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 55–66. ACM Press, 1992.

[12] Chiara Cumbo, Wolfgang Faber, and Gianluigi Greco. Improving Query Optimization for Disjunctive Datalog. In Francesco Buccafurri, editor, *Proceedings of the Joint Conference on Declarative Programming APPIA-GULP-PRODE 2003*, pages 252–262, September 2003.

[13] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[14] Jürgen Dix, Georg Gottlob, and V. Wiktor Marek. Causal Models for Disjunctive Logic Programs. In Pascal Van Hentenryck, editor, *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*, pages 290–302, Santa Margherita Ligure, Italy, June 1994. MIT Press.

[15] Jürgen Dix, Georg Gottlob, and V. Wiktor Marek. Reducing Disjunctive to Non-Disjunctive Semantics by Shift-Operations. *Fundamenta Informaticae*, 28:87–100, 1996. (This is a full version of [14].).

[16] Thomas Eiter, Wolfgang Faber, Michael Fink, Gerald Pfeifer, and Stefan Woltran. Complexity of answer set checking and bounded predicate arities for non-ground answer set programming. In Marina de Vos and Alessandro Provetti, editors, *Proceedings ASP03 - Answer Set Programming: Advances in Theory and Implementation, Messina, Italy, September 2003*, number 78 in CEUR Workshop Proceedings, pages 69–83, 2003. Online `http://CEUR-WS.org/Vol78/`, to appear.

[17] Thomas Eiter, Wolfgang Faber, Michael Fink, Gerald Pfeifer, and Stefan Woltran. Complexity of answer set checking and bounded predicate arities for non-ground answer set programming. Technical Report INFSYS RR-1843-03-11, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, November 2003. Preliminary report, extending an ASP03 abstract.

[18] Thomas Eiter, Wolfgang Faber, Michael Fink, Gerald Pfeifer, and Stefan Woltran. Complexity of Model Checking and Bounded Predicate Arities for Non-ground Answer Set Programming. In *Proceedings Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), June 2-5, Whistler, British Columbia, Canada*. Morgan Kaufmann, 2004. To appear.

[19] Thomas Eiter and Michael Fink. Uniform Equivalence of Logic Programs under the Stable Model Semantics. In Catuscia Palamidessi, editor, *Proceedings 19th International Conference on Logic Programming (ICLP 2003)*, number 2916 in Lecture Notes in Computer Science, pages 224–238. Springer, 2003.

[20] Thomas Eiter and Michael Fink. Uniform Equivalence of Logic Programs under the Stable Model Semantics. Technical Report INFSYS RR-1843-03-08, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, May 2003. Preliminary Report. Short version in [19].

[21] Thomas Eiter, Michael Fink, Gianluigi Greco, and Domenico Lembo. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In Catuscia Palamidessi, editor, *Proceedings 19th International Conference on Logic Programming (ICLP 2003)*, number 2916 in LNCS, pages 163–177. Springer, 2003.

[22] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. Eliminating disjunction from propositional logic programs under stable model preservation. In Marina de Vos and Alessandro Provetti, editors, *Proceedings ASP03 - Answer Set Programming: Advances in Theory and Implementation, Messina, Italy, September*

2003, number 78 in CEUR Workshop Proceedings, pages 151–165, 2003. Online `http://CEUR-WS.org/Vol78/`.

[23] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. On eliminating disjunctions in stable logic programming. Technical Report INFSYS RR-1843-03-12, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, November 2003. Preliminary report, extending an ASP03 abstract.

[24] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. On Eliminating Disjunctions in Stable Logic Programming. In *Proceedings Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR 2004), June 2-5, Whistler, British Columbia, Canada*. Morgan Kaufmann, 2004. To appear.

[25] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. Simplifying logic programs under uniform and strong equivalence. In Ilkka Niemelä and Vladimir Lifschitz, editors, *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2004)*, number 2923 in LNCS, pages 87–99. Springer, 2004.

[26] Thomas Eiter and Georg Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, 15(3/4):289–323, 1995.

[27] Thomas Eiter, Georg Gottlob, and Yuri Gurevich. Normal Forms for Second-Order Logic over Finite Structures, and Classification of NP Optimization Problems. *Annals of Pure and Applied Logic*, 78:111–125, 1996.

[28] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, September 1997.

[29] Wolfgang Faber and Gerald Pfeifer. `DLV` homepage, since 1996. `http://www.dlvsystem.com/`.

[30] Ronald Fagin, Phokion Kolaitis, Lucian Popa, and Wang-Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. In *Proceedings of PODS 2004*, 2004. To appear.

[31] Mary F. Fernandez, Yana Kadiyska, Dan Suciu, Atsuyuki Morishima, and Wang Chiew Tan. Silkroute: A framework for publishing relational data in xml. *ACM Transactions on Database Systems*, 27(4):438–493, 2002.

[32] Haim Gaifman, Harry Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. In *LICS*, pages 106–115. IEEE, 1987.

[33] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[34] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[35] Michael Gelfond, Halina Przymusinska, Vladimir Lifschitz, and Mirosław Truszczyński. Disjunctive Defaults. In *Proceedings Second International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pages 230–237, 1991.

[36] Parke Godfrey and Jarek Gryz. Partial Evaluation of Views. *Journal of Intelligent Information Systems*, 16:21–39, 2001.

[37] Goran Gogic, Henry A. Kautz, Christos H. Papadimitriou, and Bart Selman. The comparative linguistics of knowledge representation. In C. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 862–869. Morgan Kaufmann, 1995.

[38] Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *Proceedings of ICLP'01*, pages 348–364, 2001.

[39] Sergio Greco. Optimization of Disjunction Queries. In Danny De Schreye, editor, *Proceedings of the 16th International Conference on Logic Programming (ICLP'99)*, pages 441–455, Las Cruces, New Mexico, USA, November 1999. The MIT Press.

[40] Sergio Greco. Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):368–385, March/April 2003. Extended Abstract appeared as [39].

[41] Ashish Gupta and Inderpal Singh Mumick. Magic-sets Transformation in Nonrecursive Systems. In *Proceedings of the Thirteenth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-92)*, pages 354–367, 1992.

[42] Alon Y. Halevy, Inderpal Singh Mumick, Yehoshua Sagiv, and Oded Shmueli. Static analysis in datalog extensions. *J. ACM*, 48(5):971–1012, 2001.

[43] Chun-Nan Hsu and Craig A. Knoblock. Semantic Query Optimization for Query Plans of Heterogeneous Multidatabase Systems. *IEEE Transactions on Knowledge and Data Engineering*, 12:959–978, 2000.

[44] Matthias Jarke and Jürgen Koch. Query Optimization in Database Systems. *ACM Computing Surveys*, 16:111–152, 1984.

[45] Richard Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity Of Computer Computations*, pages 85–103. Plenum Press, 1972.

[46] David B. Kemp, Divesh Srivastava, and Peter J. Stuckey. Bottom-up evaluation and query optimization of well-founded models. *Theoretical Computer Science*, 146:145–184, July 1995.

[47] Daniel Leivant. Descriptive Characterizations of Computational Complexity. *Journal of Computer and System Sciences*, 39:51–83, 1989.

[48] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of PODS 2002*, pages 233–246, 2002.

[49] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. Technical Report cs.AI/0211004, arXiv.org, November 2002. Accepted for ACM TOCL.

[50] Alon Levy, Inderpal Singh Mumick, Yehoshua Sagiv, and Oded Shmueli. Equivalence, query-reachability and satisfiability in datalog extensions. In *Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 109–122. ACM Press, 1993.

[51] Alon Levy and Yehoshua Sagiv. Constraints and redundancy in datalog. In *Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 67–80. ACM Press, 1992.

[52] Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

[53] Vladimir Lifschitz and Hudson Turner. Splitting a Logic Program. In Pascal Van Hentenryck, editor, *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*, pages 23–37, Santa Margherita Ligure, Italy, June 1994. MIT Press.

[54] Fangzhen Lin. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In D. Fensel, F. Giunchiglia, D. McGuiness, and M.-A. Williams, editors, *Proceedings Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR-02), April 22-25, Toulouse, France*, pages 170–176. Morgan Kaufmann, 2002.

[55] Michael J. Maher. Equivalences of logic programs. In Minker [56], pages 627–658.

[56] Jack Minker, editor. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, Inc., Washington DC, 1988.

[57] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic is relevant. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 247–258, 1990.

[58] Mauricio Osorio, Juan A. Navarro, and Jos Arrazola. Equivalence in Answer Set Programming. In Alberto Pettorossi, editor, *Proceedings of the 11th International Workshop on Logic Based Program Synthesis and Transformation (LOPSTR 2001), Paphos, Cyprus, November 28-30, 2001*, number 2372 in Lecture Notes in Computer Science, pages 57–75. Springer, 2001.

[59] David Pearce. A new logical characterisation of stable models and answer sets. In *Non-Monotonic Extensions of Logic Programming (NEMLP 1996)*, volume 1216 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 1997.

[60] David Pearce. Simplifying logic programs under answer set semantics. In *Proceedings 20th International Conference on Logic Programming (ICLP 2004)*, 2004. To appear.

[61] David Pearce and Augustin Valverde. Some types of equivalence for logic programs and equilibrium logic. In Nicola Leone and Pasquale Rullo, editors, *Proceedings Joint Conference on Declarative Programming (APPIA-GULP-PRODE 2003), Reggio Calabria, Italy, September 3-5, 2003*, pages 350–361, 2003.

[62] David Pearce and Augustin Valverde. Uniform equivalence for equilibrium logic and logic programs. In Ilkka Niemelä and Vladimir Lifschitz, editors, *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2004)*, number 2923 in LNCS, pages 194–206. Springer, 2004.

[63] Teodor C. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9:401–424, 1991.

[64] Yehoshua Sagiv. Optimizing datalog programs. In Minker [56], pages 659–698.

[65] Yehoshua Sagiv and Moshe Y. Vardi. Safety of datalog queries over infinite databases. In *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 160–171. ACM Press, 1989.

[66] Yatin P. Saraiya. Hard problems for simple logic programs. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 64–73. ACM Press, 1990.

[67] Oded Shmueli. Decidability and Expressivness Aspects of Logic Queries. In *Proceedings of the Sixth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-87)*, pages 237–249, May 1987.

[68] Oded Shmueli. Equivalence of datalog queries is undecidable. *J. Log. Program.*, 15(3):231–241, 1993.

[69] Patrik Simons. Smodels Homepage, since 1996. `http://www.tcs.hut.fi/Software/smodels/`.

[70] Hudson Turner. Strong Equivalence for Logic Programs and Default Theories (Made Easy). In Thomas Eiter, Wolfgang Faber, and Mirosław Truszczyński, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, number 2173 in LNCS, pages 81–92. Springer, 2001.

[71] Hudson Turner. Strong equivalence made easy: nested expressions and weight constraints. *Journal of the Theory and Practice of Logic Programming*, 3(4–5):609–622, 2003.

[72] Kewen Wang and Lizhu Zhou. Comparisons and computation of well-founded semantics for disjunctive logic programs. Manuscript, CoRR Achive, cs.AI/0301010. Available at `http://arxiv.org/abs/cs.AI/0301010`, 2003.

[73] Stefan Woltran. Characterizations for relativized notions of equivalence in answer set programming. Manuscript, submitted for publication, May 2004.