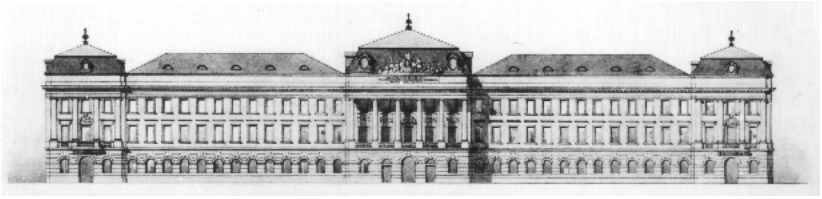
dbai

TECHNICAL REPORT



INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

Model-Based Recasting in Answer-Set Programming

DBAI-TR-2013-83

Thomas Eiter **Michael Fink** **Jörg Pührer**
Hans Tompits **Stefan Woltran**

Institut für Informationssysteme
Abteilung Datenbanken und
Artificial Intelligence
Technische Universität Wien
Favoritenstr. 9
A-1040 Vienna, Austria
Tel: +43-1-58801-18403
Fax: +43-1-58801-18493
sekret@dbai.tuwien.ac.at
www.dbai.tuwien.ac.at

DBAI TECHNICAL REPORT
2013

TU WIEN
TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

DBAI TECHNICAL REPORT
DBAI TECHNICAL REPORT DBAI-TR-2013-83, 2013

Model-Based Recasting in Answer-Set Programming

Thomas Eiter [1] **Michael Fink** [1] **Jörg Pührer** [1]
Hans Tompits [1] **Stefan Woltran** [1]

Abstract. As well-known, answer-set programs do not satisfy the replacement property in general, i.e., programs P and Q that are equivalent may cease to be so when they are put in the context of some other program R, i.e., $R \cup P$ and $R \cup Q$ may have different (sets of) answer sets. Pearce et al. thus introduced *strong equivalence* for context-independent equivalence, and proved that such equivalence holds between given programs P and Q iff P and Q are equivalent theories in the monotonic *logic of here-and-there*. In this article, we consider a related question: given a program P, *does there exist some program Q from a certain class $\mathcal{C}$ of programs such that P and Q are equivalent under a given notion of equivalence*? Furthermore, if the answer to this question is positive, how can we recast P to an equivalent program from $\mathcal{C}$ (i.e., construct such a Q)? In particular, we consider classes of programs that emerge by (dis)allowing disjunction and/or negation, and as equivalence notions we consider strong, uniform, and ordinary equivalence. Based on general model-theoretic properties and a novel form of canonical programs, we develop semantic characterisations for the existence of such a program Q, determine the computational complexity of checking existence, and provide (local) rewriting rules for recasting.

[1]Technische Universität Wien. E-mails: eiter,fink,puehrer,tompits,stefan@kr.tuwien.ac.at

Acknowledgements: This work was partially supported by the Austrian Science Fund (FWF) under projects P21698, P25518, and P25521; by the Vienna Science and Technology Fund (WWTF) project ICT08-020; and by the Vienna University of Technology special fund "Innovative Projekte" (9006.09/008).

1 Introduction

Answer-set programming, which is rooted in the answer-set semantics of logic programs (Gelfond & Lifschitz, 1988, 1991), has become an increasingly popular formalism for declarative problem solving for a range of different application areas (Brewka, Eiter, & Truszczyński, 2011). Besides the declarative and elegant approach of the latter to deal with negation as failure in programs, which has occupied the research community quite some time, the availability of efficient solvers like `clasp`, `DLV`, or `Smodels` to evaluate programs that encode solutions of a problem at hand is an important factor of the success of answer-set programming.

All this builds on a rich body of foundational results, regarding both semantic and computational properties, with a detailed understanding of the computational complexity and optimal algorithms in the light of this insight.

While Gelfond and Lifschitz (1988) based their seminal definition of a stable model on the notion of a program reduct, inspired by ideas underlying default logic (Reiter, 1980) to handle negation as failure, an interpretation of the latter in terms of a weakening of classical logic took some time to be realised. Enlightening for this question was the work of David Pearce, who aimed at a logical reconstruction of answer sets in terms of intuitionistic logic, the most well-known weakening of classical logic. He gave a characterisation of answer-set semantics in terms of intuitionistic provability (Pearce, 1999) and he developed a non-monotonic version of the *logic of here-and-there* (HT), which is a monotonic intermediate logic between (full) intuitionistic logic and classical logic. His *equilibrium logic* (EQL) (Pearce, 2006) is a milestone for answer-set programming, as it enables us to bypass answer-set construction through the auxiliary device of reduct and perform it in pure model-theoretic terms, using a Kripke structure with two worlds, "here" and "there", where the latter intuitively serves to evaluate negation. Not only is EQL strikingly elegant from a logical perspective, it also comes with the benefit that it immediately extends from a simple core language as that of Gelfond and Lifschitz (1988) to richer languages including constructs such as *strong negation* (often also called *classical negation*) and disjunction in rule heads; this and other properties nurture the view that EQL is a proper characterisation of answer-set semantics from a logical perspective. Among these properties is another one concerning modularity, which is in fact crucial for the present article. As well-known, programs under answer-set semantics suffer from a violation of the replacement property. That is, given programs P and Q which are equivalent, i.e., have the same (sets of) answer sets, and given another program R, then it is not guaranteed that the programs $R \cup P$ and $R \cup Q$ are equivalent, i.e., having the same sets of answer sets—note that for classical logic this property holds.

In a landmark paper (Lifschitz, Pearce, & Valverde, 2001), Pearce et al. developed the notion of *strong equivalence* which ensures the replacement property: P and Q are regarded as strongly equivalent, denoted $P \equiv_s Q$, if for every program R the programs $R \cup P$ and $R \cup Q$ are equivalent in the ordinary sense, i.e., they have the same collections of answer sets. Pearce et al. established the beautiful and surprising result that $P \equiv_s Q$ holds if and only if P and Q are equivalent theories in the logic of here-and-there. This in fact drew attention to EQL as a tool to work with and analyse answer-set programs, and proved extremely fruitful to tackle, e.g., a number of issues regarding modular replacement that refine and extend the basic notion of strong equivalence. For instance,

starting with uniform equivalence, where R is restricted to be a set of facts, many other notions have been studied; see the survey article by Woltran (2010) and the works of Eiter, Tompits, and Woltran (2005), Woltran (2008), and Fink (2011) for fairly general frameworks capturing various such notions.

The possibility of program replacement, expressed by a suitable notion $\equiv_e$ of equivalence between programs, is at the foundation of program optimisation and modular program reuse. The question whether $P \equiv_e Q$ for given programs P and Q has been studied in a number of papers, for different program classes, and semantic and computational complexity results have been obtained. The work of Eiter, Fink, and Woltran (2007) is an example of a thorough study that has considered this issue exhaustively for strong ($\equiv_s$) and uniform ($\equiv_u$) equivalence between disjunctive logic programs (DLPs) and fragments of this class, including Horn logic programs and normal logic programs (NLPs).

In the present article, we consider a similar yet different question: given a program P, *does there exist some program Q from a certain class $\mathcal{C}$ of programs such that $P \equiv_e Q$*, i.e., P and Q are equivalent under a notion e of equivalence? Furthermore, if the answer to this question is positive, how can we construct from P a respective program Q? More specifically, we shall consider these questions for classes of programs that emerge by (dis)allowing disjunction and/or negation, and for e being either strong, uniform, or ordinary equivalence.

The existence and construction of a program Q equivalent to P is crucial for modular replacement in combination with a *program recast*, i.e., we also change the syntactic class of the underlying program. In other terms, the issue is to recognise programs in disguise. This may be exploited to arrive at a program that can be be evaluated more easily, or with an algorithm that is tailored for a specific program class. The availability of a suite of algorithms for different program classes that allows for graceful evaluation and offers nice down-scaling properties (if a program falls into an "easy class", a solver should evaluate it efficiently) is a versatile feature of advanced answer-set solvers. Prominent examples of such program classes that are exploited in practice are Horn programs, stratified programs (Apt, Blair, & Walker, 1988) and head-cycle free programs (Ben-Eliyahu & Dechter, 1994).

For an illustrative example, the disjunctive program $P = \{a \vee b \leftarrow\}$ is ordinarily equivalent to $Q = \{a \leftarrow not\, b;\ b \leftarrow not\, a\}$; this can be exploited to recast P to Q for evaluation on an ASP solver for normal (i.e., non-disjunctive) programs. On the other hand, it is well known that P is not strongly equivalent to Q. In fact, as it turns out, P is not strongly equivalent to any disjunction-free program Q. This means that P cannot, for the sake of modular replacement in an arbitrary context R, be rewritten to a non-disjunctive program. However, such a rewriting is possible if the context R is a set of facts.

Briefly, the main results of this article are as follows.

- We present some general properties of HT-models of disjunctive programs, and a new form of canonical program (Section 3). Canonical programs have already been addressed in the literature. Cabalar and Ferraris (2007) have shown that for any set $\mathcal{S}$ of HT-interpretations, satisfying $(X, Y) \in \mathcal{S}$ implies $(Y, Y) \in \mathcal{S}$, there exists a *generalised* disjunctive logic program (i.e., a program which also permits negations-as-failure in the head of rules) having as its set of HT-models exactly the set $\mathcal{S}$. Eiter, Tompits, and Woltran (2005) use a similar method to construct

counterexamples within their correspondence framework. We provide here a new approach to obtain disjunctive logic programs from a given set of HT-interpretations, which turns out to be convenient for eliminating disjunctions. Roughly speaking, the canonical program $P_{\mathcal{A},\mathcal{S},\mathcal{S}'}$ for alphabet $\mathcal{A}$ has two parameters, $\mathcal{S}$ and $\mathcal{S}'$, that are sets of HT-interpretations, such that the HT-models of the program include all elements of $\mathcal{S}$ but no element of $\mathcal{S}'$.

- We provide results for the elimination of disjunction from a logic program P, i.e., recasting it to an equivalent normal program Q. We first give a semantic characterisation of the existence of such a program Q for strong equivalence. More precisely, we show that this amounts to a natural intersection condition on the models of P in the logic HT (which is coNP-complete); for uniform and ordinary equivalence, Q always does exist.

We then consider *local recasting*, i.e., whether an equivalent Q can be constructed using local rewriting rules. More specifically, we consider the common operation of body-shifting, in which all literals but one in the head of a rule are moved to the body. This operation preserves uniform and ordinary equivalence for head-cycle free programs, but not for arbitrary disjunctive programs. We show that by adding further rules to the shift program, uniform resp. strong equivalence can be preserved. In particular, it suffices to add the canonical program $P_{\mathcal{A},\mathcal{S},\mathcal{S}'}$ for suitable $\mathcal{S}$ and $\mathcal{S}'$. This rewriting subsumes previous rewritings discussed in earlier work (Eiter, Faber, Fink, Pfeifer, & Woltran, 2004).

- Furthermore, we consider the elimination of negation from logic programs, i.e., recasting to positive programs. We provide a semantic characterisation for settings when under strong resp. uniform equivalence a program P is equivalent to some positive (i.e., negation-free) program Q (for ordinary equivalence, such a Q always exists). We show that this is the case if and only if a particular subset of the HT-models of P satisfies some natural condition (which is coNP-complete, respectively Π_2^P-complete, to check on P); furthermore, we show that there is a simple candidate for Q that is given by the left (head) shift of literals that occur in the body of a rule under default negation to the head of the same rule.

- Finally, we also consider the joint elimination of disjunction and negation from a logic program. We provide a semantic criterion which combines the respective criteria for strong and uniform equivalence; the precise computational complexity of this criterion remains open.

We thus significantly extend preliminary results towards semantic recasting that appeared in preliminary works (Eiter, Fink, Tompits, & Woltran, 2004b; Eiter, Faber, et al., 2004), establishing a theoretical basis for improving and enhancing the computation of answer sets of a given program.

The remainder of this article is organised as follows. In the next section, we recall basic notions from ASP and equilibrium logic that are relevant for our study. We then consider the elimination of disjunction in Section 4, followed by the elimination of negation in Section 5, including the joint elimination of disjunction and negation. After a discussion of possible generalisations in Section 6, we conclude in Section 7.

2 Background

We start by first introducing the syntax of logic programs and corresponding program classes considered in this article. We recall their semantics, specifically also in terms of equilibrium logic, which provides suitable means for the investigation of semantic properties and characterisations. Eventually, we introduce the notions of equivalence, and review seminal corresponding results, that shall later be faithfully retained while transforming between different syntactical program classes.

2.1 Syntax and Program Classes

We deal with disjunctive logic programs, which allow the use of default negation not in rules. A *rule* r is a triple $\langle H(r), B^+(r), B^-(r)\rangle$, alternatively written as

$$H(r) \leftarrow B^+(r), not\, B^-(r),$$

where $H(r) = \{A_1, \ldots, A_l\}$, $B^+(r) = \{A_{l+1}, \ldots, A_m\}$, and $B^-(r) = \{A_{m+1}, \ldots, A_n\}$, where $0 \leq l \leq m \leq n$ and A_i, $1 \leq i \leq n$, are atoms from a first-order language. We also use the traditional representation of a rule as an expression of the form

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

We call $H(r)$ the *head* of r and $B(r) = \{A_{l+1}, \ldots, A_m, not\, A_{m+1}, \ldots, not\, A_n\}$ the *body* of r. If $H(r) = \emptyset$, then r is a *constraint*. A constraint with $B^-(r) = \emptyset$ is called *positive constraint*, otherwise it is called *negative constraint*. As usual, r is a *disjunctive fact* if $B(r) = \emptyset$, and r is a (non-disjunctive) *fact* if $B(r) = \emptyset$ and $|H(r)| \leq 1$, both also represented by $H(r)$; for $H(r) = \emptyset$, we occasionally write $\bot$.

A rule r is *normal* (or *non-disjunctive*), if $|H(r)| \leq 1$; and *definite*, if $|H(r)| = 1$. For $H(r) = \{h\}$, we denote normal rules also as

$$h \leftarrow B^+(r), not\, B^-(r).$$

A rule r is *positive*, if $B^-(r) = \emptyset$, and *Horn*, if it is normal and positive.

A *disjunctive logic program* (DLP), is a finite set of rules. Unless stated otherwise the term *program* refers to a disjunctive logic program.

In the rest of this article, we focus on propositional programs over a finite set $\mathcal{A}$ of atoms—programs with variables reduce to their ground (propositional) versions as usual. We say that a program P is *given over* $A \subseteq \mathcal{A}$, if each propositional atom in P is from A.

We shall deal with classes of programs which are defined via syntactical restrictions of each of their rules. A program P is a *normal logic program* (NLP) if all rules in P are normal. Accordingly, P is *positive*, if each rule in P is positive and *constraint-free*, if no rule in P is a constraint.

We need also the notion of head-cycle free programs and rules. A program P is *head-cycle free* (HCF) (Ben-Eliyahu & Dechter, 1994), if each $r \in P$ is head-cycle free (in P), i.e., if the dependency graph of P (which is defined as usual) where literals of form $not\, A$ are disregarded, has no directed cycle that contains two atoms belonging to $H(r)$.

2.2 Semantics

Concerning semantics, we first briefly review (propositional) equilibrium semantics (Pearce, 2006, 1997), which is based on the logic of here-and-there (HT). The latter provides a logical underpinning of the answer-set semantics, and the corresponding semantic structures, i.e., HT-*models*, play a fundamental role in the study of semantic properties, specifically equivalences, of logic programs. The relation to the original definition of answer sets is pointed out by a quick review of the latter.

Propositional Logic of Here-and-There (HT). The logic of here-and-there is an intermediate logic between intuitionistic logic and classical logic. Like intuitionistic logic it can be semantically characterised by Kripke models, in particular using just two worlds, namely "here" and "there" (assuming that the "here" world is prior to the "there" world). Accordingly, HT-interpretations are pairs (X, Y) of sets of atoms from $\mathcal{A}$, such that $X \subseteq Y$. An HT-interpretation is *total* if $X = Y$. The intuition is that atoms in X (the *here* part) are considered to be true, atoms not in Y (the *there* part) are considered to be false, while the remaining atoms (from $Y \setminus X$) are undefined. By $INT_{\mathcal{A}}$ we denote the set of all HT-interpretations over $\mathcal{A}$.

These pairs are used to define the semantics of HT by a satisfaction relation for formulas over the propositional signature $\mathcal{A}$, the binary connectives $\wedge$, $\vee$, and $\rightarrow$ (for conjunction, disjunction, and implication, respectively), and the nullary connective $\bot$ (for falsity). Moreover, for convenience, we consider the following abbreviations: $\phi \equiv \psi$ for $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$; $\neg\phi$ for $\phi \rightarrow \bot$; and $\top$ for $\bot \rightarrow \bot$.

Let X be an interpretation, i.e., a set of atoms from $\mathcal{A}$. We use $X \models \phi$ to denote classical satisfaction of a formula ϕ by X, and define satisfaction in the logic of here-and-there by an HT-interpretation (X, Y), denoted as $(X, Y) \models \phi$, inductively as follows:

1. $(X, Y) \not\models \bot$,
2. $(X, Y) \models a$ if $a \in X$, for any atom a,
3. $(X, Y) \models \phi \wedge \psi$ if $(X, Y) \models \phi$ and $(X, Y) \models \psi$,
4. $(X, Y) \models \phi \vee \psi$ if $(X, Y) \models \phi$ or $(X, Y) \models \psi$,
5. $(X, Y) \models \phi \rightarrow \psi$ if (i) $(X, Y) \not\models \phi$ or $(X, Y) \models \psi$, and (ii) $Y \models \phi \rightarrow \psi$.[1]

An HT-interpretation (X, Y) such that $(X, Y) \models \phi$ is called an HT-*model* of ϕ. This definition is extended to theories, i.e., sets of formulas, as usual: (X, Y) satisfies a theory $\mathcal{T}$, iff it satisfies all formulas $\phi \in \mathcal{T}$. The set of all HT-models of $\mathcal{T}$ (over $\mathcal{A}$) is denoted by $HT_{\mathcal{A}}(\mathcal{T})$ (or simply by $HT(\mathcal{T})$ if $\mathcal{A}$ is fixed). For a single formula ϕ, we write $HT(\phi)$ instead of $HT(\{\phi\})$. Moreover, we will make use of the following simple properties: if $(X, Y) \models \mathcal{T}$ then $(Y, Y) \models \mathcal{T}$; and $(X, Y) \models \neg\phi$ iff $Y \models \neg\phi$. For an axiomatic proof system of the logic of here-and-there, we refer, e.g., to the work of Pearce (2006).

Equilibrium semantics is obtained by imposing an additional equilibrium condition. A total HT-interpretation (Y, Y) is called an *equilibrium model* of a theory $\mathcal{T}$, iff $(Y, Y) \models \mathcal{T}$ and for all

[1] That is, Y satisfies $\phi \rightarrow \psi$ classically.

HT-interpretations (X, Y) such that $X \subset Y$, it holds that $(X, Y) \not\models \mathcal{T}$. An interpretation Y is an *answer set* of $\mathcal{T}$ iff (Y, Y) is an equilibrium model of $\mathcal{T}$. By $\mathcal{AS}(\mathcal{T})$ we denote the set of all answer sets of $\mathcal{T}$.

The applicability of the above semantic definitions to our logic programming setting is given by a strict correspondence between rules and formulas of the above form: A rule of the form (1) is identified with the formula

$$A_{l+1} \wedge \ldots \wedge A_m \wedge \neg A_{m+1} \wedge \ldots \wedge \neg A_n \rightarrow A_1 \vee \cdots \vee A_l.$$

Consequently, DLPs correspond to propositional theories with respective syntactic restrictions. Since disjunctive logic programs and subclasses are the subject of our investigations, in the rest of this article we stick to the syntax of rules (also when implicitly referring to them as formulas in HT).

Relation to Answer-Set Semantics. We recall the answer-set semantics for DLPs (Gelfond & Lifschitz, 1991), which generalises the answer-set semantics for NLPs (Gelfond & Lifschitz, 1988). An interpretation Y *satisfies* (or *models*) the head of a rule r, denoted $Y \models H(r)$, iff $A \in Y$ for some $A \in H(r)$. It satisfies $B(r)$, i.e., $Y \models B(r)$, iff (i) each $A \in B^+(r)$ is true in Y, i.e., $A \in Y$, and (ii) each $A \in B^-(r)$ is false in Y, i.e., $A \notin Y$. Furthermore, Y satisfies a rule r, or Y is a model of r, symbolically $Y \models r$, iff $Y \models H(r)$ whenever $Y \models B(r)$. Finally, Y is a model of a program P, denoted $Y \models P$, iff $Y \models r$, for all $r \in P$.

The *Gelfond-Lifschitz reduct* of a program P *relative to* a set of atoms Y, denoted P^Y, is defined as $P^Y = \{H(r) \leftarrow B^+(r) \mid r \in P, Y \cap B^-(r) = \emptyset\}$. For convenience, we write, for a single rule r, r^Y instead of $\{r\}^Y$. An interpretation Y is an answer set (or a *stable model* (Przymusinski, 1991)) of a program P iff Y is a minimal model (under inclusion $\subseteq$) of P^Y.

Beyond yielding equivalent definitions of answer sets (for a generalisation of the original reduct-based approach from programs to general propositional theories, cf. the work of Ferraris (2005), Ferraris and Lifschitz (2005), and Truszczyński (2010)), another well-known relationship is that $(X, Y) \in \mathit{INT}_\mathcal{A}$ is an HT*-model* (over $\mathcal{A}$) of a DLP P, iff $Y \models P$ and $X \models P^Y$. This is a straightforward consequence of the basic properties mentioned above, i.e., that $(X, Y) \models P$ implies $(Y, Y) \models P$, and that $(X, Y) \models \mathit{not}\, a$ iff $Y \models \mathit{not}\, a$.

2.3 Notions of Equivalence

Next, we review the central notions of equivalence in answer-set programming, together with their characterisations and complexity. Two programs P and Q are *ordinarily equivalent*, in symbols $P \equiv Q$, iff P and Q possess the same answer sets. Not surprisingly, checking ordinary equivalence between programs of a class $\mathcal{C}$ is as hard as checking whether a program from $\mathcal{C}$ has at least one answer set (which is referred to as the *consistency problem*). More precisely, deciding $P \equiv Q$ is Π_2^P-complete for DLPs in general, coNP-complete if both P and Q are HCF or normal, and P-complete if both programs are Horn.

Strong Equivalence. Two programs P and Q are *strongly equivalent*, denoted $P \equiv_s Q$, iff for any program R, the programs $P \cup R$ and $Q \cup R$ are equivalent, i.e., $P \cup R \equiv Q \cup R$. That strong

equivalence coincides with logical equivalence in HT (Lifschitz et al., 2001; Pearce, 2006), is one of the most important findings of the foundation of answer-set programming, and specifically for the study of program equivalences.

Proposition 1 ((Lifschitz et al., 2001)). *For every DLP P and Q, $P \equiv_s Q$ iff $HT(P) = HT(Q)$.*

To check strong equivalence of two programs P and Q, it is obviously sufficient to consider HT-interpretations (X, Y) over $\mathcal{A}$ such that $\mathcal{A}$ contains the atoms occurring in P or Q. We implicitly make use of this simplification whenever convenient.

We also introduce the following notation for HT-consequences, in particular for a DLP P and a rule r: we say that r is an *HT-consequence* (or *strong consequence*) of P, denoted $P \models_s r$, iff for each $(X, Y) \in HT(P)$, it holds that $(X, Y) \in HT(r)$. Furthermore, we write $P \models_s Q$ iff $P \models_s r$, for every $r \in Q$.

Proposition 2. *For every DLP P and Q, $P \equiv_s Q$ iff $P \models_s Q$ and $Q \models_s P$.*

The complexity of checking strong equivalence between two programs is as follows.

Proposition 3 ((Eiter, Fink, & Woltran, 2007)). *Checking strong equivalence between given DLPs P and Q is* coNP*-complete, even if one of the programs is normal or positive and the other is Horn. If both P and Q are Horn, the problem is* P*-complete.*

Uniform Equivalence. We now turn to *uniform equivalence*, which holds between two programs P and Q, in symbols $P \equiv_u Q$, iff for each set F of facts, the program $P \cup F$ and $Q \cup F$ are ordinarily equivalent. The characterisation for uniform equivalence again relies on certain HT-models of the involved programs.

Definition 1. *Let P be a program and (X, Y) an HT-model (over $\mathcal{A}$) of P. Then, (X, Y) is an* UE-model *(over $\mathcal{A}$) of P iff, for every HT-model (X', Y) of P, it holds that $X \subset X'$ implies $X' = Y$. The set of all UE-models (over $\mathcal{A}$) of P is denoted by $UE_{\mathcal{A}}(P)$.*

As before, we mostly leave $\mathcal{A}$ implicit, and consider that it contains all atoms from the compared programs.

Proposition 4 ((Eiter & Fink, 2003)). *For every DLP P and Q, $P \equiv_u Q$ iff $UE(P) = UE(Q)$.*

In accordance to strong consequence, uniform consequence is defined as follows. Let P be a DLP and r a rule. Then, r is a *uniform consequence* of P, denoted $P \models_u r$, iff for each $(X, Y) \in UE(P)$, it holds that $(X, Y) \in HT(r)$. Furthermore, we write $P \models_u Q$ iff $P \models_u r$, for every $r \in Q$.

Proposition 5. *For any DLP P and Q, $P \equiv_u Q$ iff $P \models_u Q$ and $Q \models_u P$.*

We again highlight corresponding complexity results.

Proposition 6 ((Eiter, Fink, & Woltran, 2007)). *Checking uniform equivalence between DLPs P and Q is Π_2^P-complete, and Π_2^P-hardness holds even if one of the programs is positive. If one of the programs is non-disjunctive, then the problem is* coNP*-complete, and hardness holds even if the program is Horn. If both P and Q are Horn, the problem is* P*-complete.*

Recall that deciding whether $P \equiv_e Q$, for $e \in \{s, u, o\}$, where P is from a class $\mathcal{C}$ and Q from a class $\mathcal{C}'$, is different from deciding whether for a program P from $\mathcal{C}$, there *exists some* program Q from class $\mathcal{C}'$ such that $P \equiv_e Q$. We shall investigate this problem in the rest of this article.

3 General Properties and Canonical Forms for DLPs

In this section, we first discuss properties on the set of HT-models that all disjunctive programs share. We also introduce some new concepts which relate HT-models to UE-models and respectively to answer sets. After that, we introduce a new notion of canonical programs, which we finally use to obtain further syntactic descriptions of certain sets of HT-interpretations.

Canonical programs have already been considered in the literature. Cabalar and Ferraris (2007) have shown that for any set $\mathcal{S}$ of HT-interpretations such that $(X, Y) \in \mathcal{S}$ implies $(Y, Y) \in \mathcal{S}$, i.e., where $\mathcal{S}$ is *there-total*, there exists a *generalised* disjunctive logic program (i.e., a program which also permits default negation in the head of rules) having as its set of HT-models exactly the set $\mathcal{S}$. Eiter, Tompits, and Woltran (2005) use a similar method to construct counterexamples within their correspondence framework.

We provide here a new approach to obtain disjunctive logic programs from a given set of HT-interpretations, which will later turn out to be convenient for results on eliminating disjunctions. We introduce a construction guided by two disjoint sets $\mathcal{S}$ and $\mathcal{S}'$ of HT-interpretations. The central idea is that the canonical program P incorporates both $\mathcal{S}$ and $\mathcal{S}'$ in the sense that $\mathcal{S} \subseteq HT(P)$ and $HT(P) \cap \mathcal{S}' = \emptyset$ are jointly satisfied. As we will show, if we choose $\mathcal{S}'$ to be "maximal disjoint" to $\mathcal{S}$, we end up with a program such that $HT(P) = \mathcal{S}$ for every $\mathcal{S}$. However, the selection of $\mathcal{S}'$ provides us with a tool to locally repair programs, as we show in the forthcoming section.

3.1 General Properties

Definition 2. *A there-total set $\mathcal{S}$ of HT-interpretations is* complete *iff for all (X, Y), $(Z, Z) \in \mathcal{S}$ such that $Y \subseteq Z$ it holds that $(X, Z) \in \mathcal{S}$.*

Lemma 1. *For each DLP P, $HT(P)$ is complete.*

Proof. By definition of HT-models, $HT(P)$ is always there-total, i.e., $(X, Y) \in HT(P)$ implies $(Y, Y) \in HT(P)$. Assume that $(X, Y), (Z, Z) \in HT(P)$ for some $Y \subseteq Z$. By anti-monotonicity of the reduct of DLPs, $Y \subseteq Z$ implies $P^Z \subseteq P^Y$, thus $X \models P^Y$ implies $X \models P^Z$. Consequently, $(X, Z) \in HT(P)$. ∎

Later we show that for *any* complete set $\mathcal{S}$ of HT-interpretations, the existence of a program possessing exactly $\mathcal{S}$ as its set of HT-models is guaranteed. Thus, the property of completeness is equivalent to expressibility by a DLP.

Next, we take a look at the general patterns of the UE-models of a DLP. To this end, we have to weaken the notion of completeness.

Definition 3. *A there-total set $\mathcal{S}$ of HT-interpretations is* quasi-complete *iff, for every $(X, Y) \in \mathcal{S}$, the following conditions hold:*

(*i*) *for each X' with $X \subset X' \subset Y$, $(X', Y) \notin \mathcal{S}$; and*

(*ii*) *for each $(Z, Z) \in \mathcal{S}$, where $Y \subset Z$, there exists a Y' with $Y \subseteq Y' \subset Z$ such that $(Y', Z) \in \mathcal{S}$.*

Lemma 2. *For each DLP P, $UE(P)$ is quasi-complete.*

Proof. By definition, $UE(P)$ is there-total. Let $(X, Y) \in UE(P)$, then (i) is satisfied by definition of UE-models. Recall that $Y \subseteq Z$ implies $P^Z \subseteq P^Y$, thus $Y \models P^Y$ (which indeed holds for $(X, Y) \in UE(P)$) implies $Y \models P^Z$. Hence, $(Y, Z) \in HT(P)$. Again by definition of UE-models, there exists a Y' with $Y \subseteq Y' \subset Z$ such that $(Y', Z) \in UE(P)$. Thus, (ii) is satisfied as well. ∎

In words, the properties of being complete, resp. quasi-complete capture implict relations between the models of different reducts. The weaker notion of quasi-completeness mirrors the fact that not all models X of a reduct P^Y "materialize" as an UE-model (X, Y) of P.

For the sake of completeness, we also review some known results on answer sets, but rephrase them in the setting of HT-interpretations.

Definition 4. *Let $\mathcal{S}$ be a set of HT-interpretations. A total interpretation $(Y, Y) \in \mathcal{S}$ is* stable in $\mathcal{S}$ *iff there exists no $(X, Y) \in \mathcal{S}$ with $X \subset Y$.*

Corollary 1. *For any program P, $Y \in \mathcal{AS}(P)$ iff (Y, Y) is stable in $HT(P)$.*

We will also need the notion of stable equivalence of two sets of HT-interpretations.

Definition 5. *Two sets $\mathcal{S}$ and $\mathcal{S}'$ of HT-interpretations are* stably equivalent *if exactly the same HT-interpretations are stable in $\mathcal{S}$ and $\mathcal{S}'$.*

Stable equivalence reflects ordinary equivalence on the level of HT-models.

Corollary 2. *Two programs P and Q are ordinarily equivalent iff $HT(P)$ and $HT(Q)$ are stably equivalent.*

Definition 6. *A there-total set of HT-interpretations $\mathcal{S}$ is said to* induce a stable anti-chain *iff, for each pair (Y, Y) and (Z, Z) of HT-interpretations stable in $\mathcal{S}$, $Y \subseteq Z$ implies $Y = Z$.*

Proposition 7. *For each DLP P, $HT(P)$ induces a stable anti-chain.*

Proof. By Lemma 1, $HT(P)$ is complete. Thus for each pair, (Y, Y) and (Z, Z) from $HT(P)$, $Y \subseteq Z$ implies $(Y, Z) \in HT(P)$, by completeness of $HT(P)$. Thus, for $Y \subset Z$, (Z, Z) cannot be stable in $HT(P)$. This shows that $HT(P)$ induces a stable anti-chain. ∎

3.2 The Canonical Program

To start with, we introduce some basic concepts used in the program generation. First of all, given a set $\mathcal{S}$ of HT-interpretations, the following operator provides, starting from an HT-interpretation (X, Z) (which not necessarily belongs to $\mathcal{S}$), the distances (in terms of set-difference $Y \setminus X$) to the "next" HT-interpretations (Y, Z) in $\mathcal{S}$.

Definition 7. *Let $\mathcal{S}$ be a set of HT-interpretations and let (X, Z) be a further HT-interpretation. Then,*

$$next_{\mathcal{S}}(X,Z) := \{ \; (Y \setminus X) \mid (Y,Z) \in \mathcal{S}, X \subseteq Y \subseteq Z, \; \forall Y' : X \subseteq Y' \subset Y \Rightarrow (Y',Z) \notin \mathcal{S}\}.$$

For an example[2], let $\mathcal{S}_0 = \{(ab, ab), (a, ab), (b, ab)\}$ and consider the HT-interpretation $(\emptyset, ab)$. Then, $next_{\mathcal{S}_0}(\emptyset, ab) = \{\{a\}, \{b\}\}$.

Next, we define a special form of a generic rule-set which is guided by interpretations X, Z, and a set $\mathcal{Y}$ of interpretations. For the latter, we will later plug in the $next(\cdot)$ operator from above. This also explains subtle difference between the cases for $\mathcal{Y} = \{\emptyset\}$ and $\mathcal{Y} = \emptyset$. The former applies if $next_{\mathcal{S}}(X, Z)$ is built from a single interpretation Y with $Y = X$, i.e. if and only if $(X, Z) \in \mathcal{S}$; the latter applies exactly if $(Z, Z) \notin \mathcal{S}$.

Definition 8. *Let X and Z be interpretations and let $\mathcal{Y} = \{Y_1, \ldots, Y_n\}$ be a set of interpretations. Then,*

$$\mathcal{Y} \Leftarrow (X,Z) := \begin{cases} \emptyset & \text{if } \mathcal{Y} = \{\emptyset\}; \\ \{\bot \leftarrow X, not\ Z\} & \text{if } \mathcal{Y} = \emptyset; \\ \{\{y_1, \ldots, y_n\} \leftarrow X, not\ Z \mid y_1 \in Y_1, \ldots, y_n \in Y_n\} & \text{otherwise.} \end{cases}$$

For illustration, let $X = \{c\}$, $\mathcal{Y} = \{\{a\}, \{b\}\}$ and $Z = \{d, e\}$. Then,

$$\mathcal{Y} \Leftarrow (X,Z) \; = \; \{a \vee b \leftarrow c, not\ d, not\ e\}.$$

For $X = Z = \emptyset$, and $\mathcal{Y} = \{\{a, a'\}, \{b, b'\}\}$ we get four rules, viz.

$$\mathcal{Y} \Leftarrow (X,Z) \; = \; \{a \vee b \leftarrow;\ a \vee b' \leftarrow;\ a' \vee b \leftarrow;\ a' \vee b' \leftarrow\}.$$

We finally need a further set $\mathcal{S}'$ which provides those elements (X, Z) used to initialise rules of the form $\mathcal{Y} \Leftarrow (X, \mathcal{A} \setminus Z)$ for $\mathcal{Y} = next_{\mathcal{S}}(X, Z)$.

Definition 9. *A set $\mathcal{S}'$ of HT-interpretations is called* complementary *to a set $\mathcal{S}$ of HT-interpretations iff, $\mathcal{S} \cap \mathcal{S}' = \emptyset$, and, for all $(X, Y) \in \mathcal{S}'$ it holds that either $X = Y$ or $(Y, Y) \in \mathcal{S}$.*

We are now prepared to define our notion of a canonical program. Recall that $\mathcal{A}$ denotes the universe of atoms.

[2]We sometimes denote interpretations $\{a_1, \ldots, a_n\}$ by juxtaposition $a_1 \cdots a_n$ of their elements.

Definition 10. *For every complete set $\mathcal{S}$ of HT-interpretations and every set $\mathcal{S}'$ complementary to $\mathcal{S}$, define the* canonical program over $\mathcal{A}$ for $\mathcal{S}$ with respect to $\mathcal{S}'$ *as*

$$P_{\mathcal{A},\mathcal{S},\mathcal{S}'} := \bigcup_{(X,Z)\in\mathcal{S}'} next_{\mathcal{S}}(X,Z) \Leftarrow (X, \mathcal{A} \setminus Z).$$

Lemma 3. *Consider $\mathcal{A}$, $\mathcal{S}$, and $\mathcal{S}'$ as in Definition* 10. *Then, $\mathcal{S} \subseteq HT(P_{\mathcal{A},\mathcal{S},\mathcal{S}'})$.*

Proof. Let P abbreviate $P_{\mathcal{A},\mathcal{S},\mathcal{S}'}$, and, towards a contradiction, assume an HT-interpretation $(X,Z) \in \mathcal{S}$ exists such that $(X,Z) \notin HT(P)$. From the latter, there exists a rule $r \in P$ such that $(X,Z) \notin HT(r)$. By construction of P, it follows that there exists an HT-interpretation $(X',Z') \in \mathcal{S}'$ such that $r \in next_{\mathcal{S}}(X',Z') \Leftarrow (X', \mathcal{A} \setminus Z')$. As $\mathcal{S}'$ is complementary to $\mathcal{S}$, we have that $(X',Z') \notin \mathcal{S}$, and furthermore one of the following two cases applies: (i) $X' = Z'$; or (ii) $(Z',Z') \in \mathcal{S}$.

(i) As $\mathcal{S}'$ is complementary to $\mathcal{S}$, we have $(Z',Z') \notin \mathcal{S}$. Thus, by construction, $next_{\mathcal{S}}(X',Z') \Leftarrow (X', \mathcal{A} \setminus Z')$ contains r as the single constraint. As r is a constraint, $(X,Z) \notin HT(r)$ iff Z is a model of the body of r. Hence, $B^+(r) = Z' \subseteq Z$ and $Z \subseteq Z' = \mathcal{A} \setminus B^-(r)$ jointly have to hold, yielding $Z = Z'$. Since $\mathcal{S}$ is there-total, we get from the assumption $(X,Z) \in \mathcal{S}$ that also $(Z,Z) = (Z',Z') \in \mathcal{S}$, a contradiction.

(ii) Suppose $X' \subset Z'$ and $(Z',Z') \in \mathcal{S}$. In what follows, we set $X = Z$ in case $(Z,Z) \notin HT(r)$. As $r \in next_{\mathcal{S}}(X',Z') \Leftarrow (X', \mathcal{A} \setminus Z')$ and $(X,Z) \notin HT(r)$, we have $B^+(r) = X' \subseteq X$ and, as before, $Z \subseteq Z'$. Now, $H(r) \cap X = \emptyset$. Hence, each $Y \in next_{\mathcal{S}}(X',Z')$ contains at least one atom y which is not contained in X. Hence, each $Y \in next_{\mathcal{S}}(X',Z')$ fulfills $Y \not\subseteq X$. However, since $X' \subseteq X$, $(X,Z') \notin \mathcal{S}$ then has to hold. On the other hand, $\mathcal{S}$ is complete, and since $(X,Z) \in \mathcal{S}$ by assumption, $Z \subseteq Z'$, and $(Z',Z') \in \mathcal{S}$, we get $(X,Z') \in \mathcal{S}$, a contradiction. ∎

The next result provides an inverse characterisation in terms that no HT-interpretation from $\mathcal{S}'$ is HT-model of the canonical program $P_{\mathcal{A},\mathcal{S},\mathcal{S}'}$.

Lemma 4. *Consider $\mathcal{A}$, $\mathcal{S}$, and $\mathcal{S}'$ as in Definition* 10. *Then, $HT(P_{\mathcal{A},\mathcal{S},\mathcal{S}'}) \cap \mathcal{S}' = \emptyset$.*

Proof. Let P abbreviate $P_{\mathcal{A},\mathcal{S},\mathcal{S}'}$, and, towards a contradiction, assume there is some $(X,Z) \in HT(P) \cap \mathcal{S}'$. Now, $(X,Z) \in \mathcal{S}'$ implies that $R = next_{\mathcal{S}}(X,Z) \Leftarrow (X, \mathcal{A} \setminus Z)$ is a subset of $P_{\mathcal{S}}$ and R is non-empty, since $(X,Z) \notin \mathcal{S}$ by the assumption that $\mathcal{S}'$ is complementary to $\mathcal{S}$. First, consider $X = Z$. By construction, $R = \{\bot \leftarrow Z, not\,(\mathcal{A} \setminus Z)\}$. But then, $(Z,Z) \notin HT(P)$. Hence, suppose $X \subset Z$. Take any rule $r = \{y_1, \ldots, y_n\} \leftarrow X, not\, \mathcal{A} \setminus Z$ from R. Clearly, we have $r^Z = \{\{y_1, \ldots, y_n\} \leftarrow X\}$, and by definition of the set $next_{\mathcal{S}}(X,Z)$, no y_i of this rule is contained in X. Hence, $X \not\models r^Z$, and consequently $(X,Z) \notin HT(P_{\mathcal{S}})$. So, in both cases, we arrive at a contradiction to $(X,Z) \in HT(P_{\mathcal{S}})$. ∎

So far, we have shown that the program $P_{\mathcal{A},\mathcal{S},\mathcal{S}'}$ satisfies $\mathcal{S} \subseteq HT(P) \subseteq (INT_{\mathcal{A}} \setminus \mathcal{S}')$. Finally, to define a canonical program with respect to a set $\mathcal{S}$, i.e., with $\mathcal{S} = HT(P)$, it is sufficient to consider a set being "maximal" complementary to $\mathcal{S}$.

Definition 11. *Let $\mathcal{A}$ be the universe, let $\mathcal{S}$ be a complete set of HT-interpretations, and let*

$$\bar{\mathcal{S}}_{\mathcal{A}} := \{(X,Z) \in INT_{\mathcal{A}} \setminus \mathcal{S} \mid (X = Z) \vee (Z,Z) \in \mathcal{S}\}.$$

Whenever $\mathcal{A}$ is clear from the context, we leave it implicit in $\bar{\mathcal{S}}_{\mathcal{A}}$ and use $P_{\mathcal{S}}$ as a shorthand for $P_{\mathcal{A},\mathcal{S},\bar{\mathcal{S}}_{\mathcal{A}}}$, i.e., for the canonical DLP over $\mathcal{A}$ for $\mathcal{S}$ with respect to $\bar{\mathcal{S}}_{\mathcal{A}}$.

Theorem 1. *For any complete set $\mathcal{S}$ of HT-interpretations, $HT(P_{\mathcal{S}}) = \mathcal{S}$.*

Proof. First note that $\bar{\mathcal{S}}_{\mathcal{A}}$ is indeed complementary to $\mathcal{S}$. Hence, by Lemma 3, we get $\mathcal{S} \subseteq HT(P_{\mathcal{S}})$. We show $HT(P_{\mathcal{S}}) \subseteq \mathcal{S}$. Towards a contradiction, suppose some $(X, Y) \in HT(P_{\mathcal{S}})$ exists such that $(X, Y) \notin \mathcal{S}$. First, let $X = Y$, i.e., $(Y, Y) \notin \mathcal{S}$. By definition, then $(Y, Y) \in \bar{\mathcal{S}}_{\mathcal{A}}$, and by Lemma 4, $HT(P_{\mathcal{S}}) \cap \bar{\mathcal{S}}_{\mathcal{A}} = \emptyset$. Hence, $(Y, Y) \notin HT(P_{\mathcal{S}})$. Second, let $X \subset Y$, $(X, Y) \notin \mathcal{S}$, and $(Y, Y) \in \mathcal{S}$. Again by definition of $\bar{\mathcal{S}}_{\mathcal{A}}$, we have $(X, Y) \in \bar{\mathcal{S}}_{\mathcal{A}}$, and Lemma 4 gives us that $(X, Y) \notin HT(P_{\mathcal{S}})$. Hence, in both cases, we derive a contradiction to the assumption $(X, Y) \in HT(P_{\mathcal{S}})$ which shows the claim. ∎

Hence, we have shown the inverse of Lemma 1, i.e., that for any complete set $\mathcal{S}$ of HT-interpretations, there exists a program P such that $HT(P) = \mathcal{S}$.

Example 1. *Consider the complete set*

$$\mathcal{S}_1 = \{(ab, ab), (a, ab), (b, ab), (a, a), (b, b)\}.$$

Setting the universe to $\mathcal{A} = \{a, b\}$, we have

$$\bar{\mathcal{S}}_1 = \{(\emptyset, \emptyset), (\emptyset, a), (\emptyset, b), (\emptyset, ab)\}.$$

We obtain for the elements in $\bar{\mathcal{S}}_1$ the following successors:

$$\begin{aligned} next_{\mathcal{S}_1}(\emptyset, \emptyset) &= \emptyset; \\ next_{\mathcal{S}_1}(\emptyset, a) &= \{\{a\}\}; \\ next_{\mathcal{S}_1}(\emptyset, b) &= \{\{b\}\}; \\ next_{\mathcal{S}_1}(\emptyset, ab) &= \{\{a\}, \{b\}\}. \end{aligned}$$

The four (in this case, singleton) rule sets according to Definition 10 are as follows:

$$\begin{aligned} \emptyset \Leftarrow (\emptyset, \mathcal{A} \setminus \emptyset) &= \{\bot \leftarrow not\, a, not\, b\}; \\ \{\{a\}\} \Leftarrow (\emptyset, \mathcal{A} \setminus \{a\}) &= \{a \leftarrow not\, b\}; \\ \{\{b\}\} \Leftarrow (\emptyset, \mathcal{A} \setminus \{b\}) &= \{b \leftarrow not\, a\}; \\ \{\{a\}, \{b\}\} \Leftarrow (\emptyset, \mathcal{A} \setminus \{a, b\}) &= \{a \vee b \leftarrow\}. \end{aligned}$$

One can check that the composed program $P_{\mathcal{S}_1}$ satisfies $HT(P_{\mathcal{S}_1}) = \mathcal{S}_1$.

In the next example, we start from a slightly different set of HT-interpretations. Observe that the outcome now is a non-disjunctive program.

Example 2. *Consider*

$$\mathcal{S}_2 = \{(ab, ab), (a, ab), (b, ab), (a, a), (b, b), (\emptyset, ab)\}.$$

In this case, we have

$$\bar{\mathcal{S}}_2 = \{(\emptyset, \emptyset), (\emptyset, a), (\emptyset, b)\},$$

and we obtain, for the three elements in $\bar{\mathcal{S}}_2$*,*

$$\begin{array}{rcl} next_{\mathcal{S}_2}(\emptyset, \emptyset) & = & \emptyset, \\ next_{\mathcal{S}_2}(\emptyset, a) & = & \{\{a\}\}, \\ next_{\mathcal{S}_2}(\emptyset, b) & = & \{\{b\}\}, \end{array}$$

and the corresponding rules

$$\begin{array}{rcl} \emptyset \Leftarrow (\emptyset, \mathcal{A} \setminus \emptyset) & = & \{\bot \leftarrow not\, a, not\, b\}, \\ \{\{a\}\} \Leftarrow (\emptyset, \mathcal{A} \setminus \{a\}) & = & \{a \leftarrow not\, b\}, \\ \{\{b\}\} \Leftarrow (\emptyset, \mathcal{A} \setminus \{b\}) & = & \{b \leftarrow not\, a\}. \end{array}$$

Once again one can verify that $HT(P_{\mathcal{S}_2}) = \mathcal{S}_2$*.*

One might have observed that a subset of the rules in the above programs would already be sufficient to exactly capture the respective input sets $\mathcal{S}_1$ and $\mathcal{S}_2$. In particular, the last rule in $P_{\mathcal{S}_1}$, $a \vee b \leftarrow$, subsumes the other three rules in terms of their HT-models. In other words, the other three rules in $P_{\mathcal{S}_1}$ are redundant. In $P_{\mathcal{S}_2}$, only the first rule is redundant, i.e., $P_{\mathcal{S}_2}$ is strongly equivalent to the program $\{a \leftarrow not\, b;\ b \leftarrow not\, a\}$.

We can obtain these more compact programs by selecting the complementary sets more suitably.

Example 3. *Let us consider certain subsets of* $\bar{\mathcal{S}}_1$ *and* $\bar{\mathcal{S}}_2$ *that directly yield programs more compact than those given Examples 1 and 2. For instance, we can select*

$$\begin{array}{rcl} P_{\mathcal{A},\mathcal{S}_1,\{(\emptyset,ab)\}} & = & \{a \vee b \leftarrow\}, \\ P_{\mathcal{A},\mathcal{S}_2,\{(\emptyset,a),(\emptyset,b)\}} & = & \{a \leftarrow not\, b;\ b \leftarrow not\, a\}, \end{array}$$

and derive programs without redundant rules.

It remains for further study how the complementary sets have to be chosen in general to provide more compact or even irredundant canonical forms.

3.3 Applying the Canonical Program

We now make use of our canonical programs such that their UE-models match a given set of HT-interpretations. To this end, we first introduce a function that generalises the selection of UE-models from HT-models of a program to sets of HT-interpretations.

Definition 12. *The* uniform selection operator $\sigma_u : INT_{\mathcal{A}} \to INT_{\mathcal{A}}$ *is given as*

$$\sigma_u(\mathcal{S}) = \mathcal{S} \setminus \{(X, Y) \mid \exists\, (X', Y) \in \mathcal{S} \text{ with } X \subset X' \subset Y\}.$$

The following relation is easy to check.

Proposition 8. *For any program P, $UE(P) = \sigma_u(HT(P))$.*

Quasi-complete sets turn out to be closed under uniform selection.

Lemma 5. *For any quasi-complete set $\mathcal{S}$ of HT-interpretations, $\sigma_u(\mathcal{S}) = \mathcal{S}$.*

Proof. Clearly, $\sigma_u(\mathcal{S}) \subseteq \mathcal{S}$ holds by definition. To show $\mathcal{S} \subseteq \sigma_u(\mathcal{S})$, for any quasi-complete set $\mathcal{S}$, just observe that if $(X, Y) \in \mathcal{S}$ then there is no X' such that $X \subset X' \subset Y$ and $(X', Y) \in \mathcal{S}$ (due to condition (i) in Definition 3). But then $(X, Y) \in \sigma_u(\mathcal{S})$ by definition. ∎

Next, we define certain operators on sets of HT-interpretations, in order to turn quasi-complete into complete sets such that the result of the uniform selection operator σ_u remains unchanged.

Definition 13. *A* uniform mapping *is any function $U : INT_{\mathcal{A}} \to INT_{\mathcal{A}}$ such that, for any quasi-complete set $\mathcal{S}$ of HT-interpretations, (i) $U(\mathcal{S})$ is complete, and (ii) $\sigma_u(U(\mathcal{S})) = \sigma_u(\mathcal{S})$.*

Using such mappings we are able to turn our canonical program to be defined with respect to UE-models.

Theorem 2. *For every quasi-complete set $\mathcal{S}$ of HT-interpretations and uniform mapping U, $UE(P_{U(\mathcal{S})}) = \mathcal{S}$.*

Proof. By definition, $U(\mathcal{S})$ is complete if $\mathcal{S}$ is quasi-complete. Hence, by Theorem 1, $HT(P_{U(\mathcal{S})}) = U(\mathcal{S})$. We apply σ_u to both sets and derive by Proposition 8 $UE(P_{U(\mathcal{S})}) = \sigma_u(\mathcal{S})$. Since $\mathcal{S}$ is quasi-complete, we get $\sigma_u(\mathcal{S}) = \mathcal{S}$ from Lemma 5, which proves the result. ∎

Note that different uniform mappings are possible and yield different programs. We define two concrete uniform mappings:

$$\begin{aligned} U_{cover}(\mathcal{S}) &= \mathcal{S} \cup \{(X, Z) \mid X \subseteq Y \subset Z, (Y, Z) \in \mathcal{S}\}; \\ U_{compl}(\mathcal{S}) &= \mathcal{S} \cup \{(X, Y) \mid (X, Z) \in \mathcal{S}, (Y, Y) \in \mathcal{S}, Z \subseteq Y\}. \end{aligned}$$

Intuitively, $U_{cover}(\mathcal{S})$ adds any HT-interpretation (X, Z) "below" a non-total UE-model (Y, Z), while $U_{compl}(\mathcal{S})$ in turn just "repairs" the quasi-complete set to a complete one by adding exactly those HT-interpretations which violate completeness for $\mathcal{S}$. Both mappings satisfy the conditions from Definition 13.

Lemma 6. *The mappings U_{cover} and U_{compl} are both uniform.*

Proof. We show the result for U_{cover}. The proof for U_{compl} is similar.

Let $\mathcal{S}$ be quasi-complete. We first show that condition (i) holds, i.e. $\mathcal{U} = U_{cover}(\mathcal{S})$ is complete. Consider any $(X, Z), (Y, Y) \in \mathcal{U}$ such that $X \subseteq Z \subseteq Y$. We have to show that $(X, Y) \in \mathcal{U}$. For $X = Y$ this is trivial, thus in the following let $X \subset Y$. First observe that $(Y, Y) \in \mathcal{S}$. Also $(Z, Z) \in \mathcal{S}$ holds: this is clear in case $X = Z$ by definition; otherwise some X' exists such that $X \subseteq X' \subset Z$ and $(X', Z) \in \mathcal{S}$. Since $\mathcal{S}$ is there-total, it follows that $(Z, Z) \in \mathcal{S}$, proving the claim. Now, since $\mathcal{S}$ is quasi-complete, we know that there exists some X'' with $X' \subseteq X'' \subset Y$ such that $(X'', Y) \in \mathcal{S}$. Since $X \subseteq X''$, $(X, Y) \in \mathcal{U}$ follows by definition of $U_{cover}(\mathcal{S})$.

To show (ii), i.e., $\sigma_u(U_{cover}(\mathcal{S})) = \sigma_u(\mathcal{S})$, observe that U_{cover} adds only non-total HT-interpretations (X, Z) where $X \subset Y$ with $(Y, Z) \in \mathcal{S}$. Hence, all these additional elements are removed by σ_u. ∎

Example 4. *Consider the set $\mathcal{S}_1$ from Example 1. We want to construct programs such that their UE-models exactly match $\mathcal{S}_1$. To this end, we consider the two uniform mappings introduced above. First observe that $U_{cover}(\mathcal{S}_1) = \mathcal{S}_1 \cup \{(\emptyset, ab)\} = \mathcal{S}_2$. Hence, we get $P_{\mathcal{S}_2}$ as the canonical program which, as discussed in Example 2, can be reduced to $P = \{a \leftarrow not\, b;\ b \leftarrow not\, a\}$. Observe that $UE(P) = \mathcal{S}_1$ since $(\emptyset, ab)$ is an HT-model but not a UE-model of P.*

For our second mapping U_{compl}, one can check that we get $U_{compl}(\mathcal{S}_1) = \mathcal{S}_1$. Thus, we simply take $P_{\mathcal{S}_1}$ as already derived in Example 1, which can be simplified to $Q = \{a \vee b \leftarrow\}$. Clearly, $UE(Q) = HT(Q) = \mathcal{S}_1$ mirroring the fact that $P \equiv_u Q$.

We finally employ our method to derive canonical programs with respect to desired answer sets.

Definition 14. *A* stable mapping *is any function $I : INT_{\mathcal{A}} \rightarrow INT_{\mathcal{A}}$ such that, for every set $\mathcal{S}$ of HT-interpretations inducing a stable anti-chain, (i) $I(\mathcal{S})$ is complete, and (ii) $I(\mathcal{S})$ and $\mathcal{S}$ are stably equivalent.*

Theorem 3. *For every set $\mathcal{S}$ of HT-interpretations inducing a stable anti-chain and every stable mapping I, the set $HT(P_{I(\mathcal{S})})$ is stably equivalent to $\mathcal{S}$.*

Proof. Since I is a stable mapping, (i) $I(\mathcal{S})$ is complete yielding $HT(P_{I(\mathcal{S})}) = I(\mathcal{S})$; and (ii) $I(\mathcal{S})$ and $\mathcal{S}$ are stably equivalent. ∎

Corollary 3. *Let $\mathcal{S}$ be a set of HT-interpretations inducing a stable anti-chain and let I be any stable mapping. Then, $\mathcal{AS}(P_{I(\mathcal{S})}) = \{Y \mid (Y, Y)$ is stable in $\mathcal{S}\}$.*

Hence, to obtain a program that has a set $\mathcal{Y}$ of interpretations as its answer sets, we just set $\mathcal{S} = \{(Y, Y) \mid Y \in \mathcal{Y}\}$ and select a stable mapping I to provide $P_{I(\mathcal{S})}$ as such a program.

We introduce three mappings which can be shown to be stable.

$$\begin{aligned}
I_{lcover}(\mathcal{S}) &= \{(Y,Y) \mid (Y,Y) \text{ is stable in } \mathcal{S}\} \cup \\
&\quad \{(X,Z) \mid (Y,Y) \text{ is stable in } \mathcal{S}, X \subseteq Z, Y \subset Z\}. \\
I_{ucover}(\mathcal{S}) &= \{(Y,Z) \mid (X,X) \text{ is stable in } \mathcal{S}, X \subseteq Y, Y \subseteq Z\}. \\
I_{nocover}(\mathcal{S}) &= \{(X,X) \mid (X,X) \text{ is stable in } \mathcal{S}\}.
\end{aligned}$$

Note that in case $\mathcal{S}$ comes only with total HT-interpretations, the test for stability is not necessary.

We provide a final example to illustrate the different behaviours of the three mappings.

Example 5. *Consider $\mathcal{S} = \{(a, a), (b, b)\}$ to generate programs which have a and b as their answer sets. Obviously, $\mathcal{S}$ induces a stable anti-chain. Over universe $\mathcal{A} = \{a, b\}$, we get*

$$\begin{aligned}
I_{lcover}(\mathcal{S}) &= \{(a,a), (b,b), (\emptyset, ab), (a, ab), (b, ab), (ab, ab)\}, \\
I_{ucover}(\mathcal{S}) &= \{(a,a), (b,b), (a, ab), (b, ab), (ab, ab)\}, \\
I_{nocover}(\mathcal{S}) &= \{(a,a), (b,b)\}.
\end{aligned}$$

We already have partly shown in earlier examples how programs are obtained from these sets. In particular, since $I_{lcover}(\mathcal{S}) = \mathcal{S}_2$, we get $\{a \leftarrow not\, b; b \leftarrow not\, a\}$ as resulting (and optimised) program. Moreover, $I_{ucover}(\mathcal{S}) = \mathcal{S}_1$ yielding $\{a \vee b \leftarrow\}$.

For $\mathcal{S}_3 = I_{nocover}(\mathcal{S}) = \{(a,a),(b,b)\}$*, we have to compute* $P_{\mathcal{S}_3} = P_{\mathcal{A},\mathcal{S}_3,\bar{\mathcal{S}}_3}$*. Observe that*

$$\bar{\mathcal{S}}_3 = \{(\emptyset,\emptyset),(\emptyset,a),(\emptyset,b),(ab,ab)\}.$$

Hence, the resulting program is (*only the first rule is redundant with respect to* HT*-models*)*:*

$$\{\bot \leftarrow not\, a, not\, b;\ a \leftarrow not\, b;\ b \leftarrow not\, a;\ \bot \leftarrow a, b\}.$$

To summarise, we obtained three programs (*in their reduced variants*)*:*

$$\begin{aligned} P_1 = P_{I_{lcover(\mathcal{S})}} &= \{a \leftarrow not\, b;\ b \leftarrow not\, a\}, \\ P_2 = P_{I_{ucover(\mathcal{S})}} &= \{a \vee b \leftarrow\}, \\ P_3 = P_{I_{nocover(\mathcal{S})}} &= \{a \leftarrow not\, b;\ b \leftarrow not\, a;\ \bot \leftarrow a, b\}. \end{aligned}$$

Each of these programs has two answer sets, $\{a\}$ *and* $\{b\}$*, as intended, although none of them is strongly equivalent to another, since they are built from different sets of* HT*-interpretations.*

4 Elimination of Disjunction

This section provides two approaches to eliminate disjunctions from a given program P such that the resulting program Q satisfies $P \equiv_e Q$, for $e \in \{s, u, e\}$.

We start with *global* rewritings where the entire program is recomputed. Then, we provide a *local* rewriting, which changes the program rule-by-rule, taking however the semantics of the entire program into account. As we will show, the local approach subsumes previous results (Eiter, Fink, Tompits, & Woltran, 2004a). We shall also address complexity issues in this section.

Indeed, the most prominent rewriting for disjunctive programs is the *head-to-body shift* due to Ben-Eliyahu and Dechter (1994). This simple syntactical rewriting is faithful for ordinary and, as shown by Eiter, Fink, et al. (2004b), uniform equivalence. We show that for these notions of equivalence, a generalised (semantics-based) version of this shift applies to all disjunctive programs. For strong equivalence, a further semantic criterion has to be satisfied.

We first analyse this particular difference between disjunction-free and disjunctive programs with respect to the properties of their HT-models. The forthcoming term "closed under here-intersection" was coined by Eiter, Fink, et al. (2004a) and refers to a property characteristic for the HT-models of normal logic programs. In particular, the here-worlds of the HT-models are closed under intersection upon fixing the there-world.

Definition 15. *A set* $\mathcal{S}$ *of* HT*-interpretations is* closed under here-intersection *iff, for any two* HT*-interpretations* $(X,Y),(X',Y) \in \mathcal{S}$*, also* $(X \cap X', Y) \in \mathcal{S}$*.*

Lemma 7. *Let* P *be an NLP. Then,* $HT(P)$ *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, as well known (cf. McKinsey (1943)), each Horn program P' satisfies the following intersection property: if $X \models P'$ and $X' \models P'$, then $X \cap X' \models P'$. ∎

Obviously, not every complete set of HT-interpretations is closed under here-intersection. Hence,

Corollary 4. *There exist disjunctive programs which cannot be rewritten into strongly equivalent normal programs.*

Indeed, if the set $HT(P)$ for a disjunctive program P is *not* closed under here-intersection, no strongly equivalent normal program exists.

4.1 Global Elimination of Disjunction

In what follows we first show that in the case where $HT(P)$ of a disjunctive program P is closed under here-intersection, a normal program Q such that $P \equiv_s Q$ is guaranteed to exist. To this end, we employ our program generation $P_{\mathcal{S}}$. The following observation is central.

Lemma 8. *For any set $\mathcal{S}$ of HT-interpretations closed under here-intersection, it holds that $|next_{\mathcal{S}}(X, Z)| \leq 1$, for any (X, Z).*

Proof. Consider some HT-interpretation (X, Z) such that $|next_{\mathcal{S}}(X, Z)| > 1$. By definition of $next_{\mathcal{S}}(X, Z)$, there exist $(Y_1, Z) \in \mathcal{S}$ and $(Y_2, Z) \in \mathcal{S}$, with $Y_1 \neq Y_2$, such that $X \subseteq Y_i \subseteq Z$, for $i \in \{1, 2\}$. Note that $X \subseteq Y_1 \cap Y_2$ and from $Y_1 \neq Y_2$ follows that either $Y_1 \cap Y_2 \subset Y_1$ or $Y_1 \cap Y_2 \subset Y_2$. Without loss of generality assume the former. By the second condition for $next_{\mathcal{S}}(X, Z)$, i.e., that for Y', if $X \subseteq Y' \subset Y$, then $(Y', Z) \notin \mathcal{S}$, as $X \subseteq Y_1 \cap Y_2 \subset Y_1$ we get $(Y_1 \cap Y_2, Z) \notin \mathcal{S}$, which is a contradiction to $\mathcal{S}$ being closed under here-intersection, because $(Y_1, Z) \in \mathcal{S}$ and $(Y_2, Z) \in \mathcal{S}$. ∎

Thus, by inspecting Definition 10, the following observation comes nearly for free.

Lemma 9. *Let $\mathcal{A}$ be the universe and let $\mathcal{S}$ be a complete set of HT-interpretations closed under here-intersection. Then, for every $\mathcal{S}'$ complementary to $\mathcal{S}$, the program $P_{\mathcal{A},\mathcal{S},\mathcal{S}'}$ is normal.*

As a corollary we thus get:

Corollary 5. *For every complete set $\mathcal{S}$ of HT-interpretations that is closed under here-intersection, the canonical program $P_{\mathcal{S}}$ is normal.*

This result together with Theorem 1 immediately provides us with the following observation:

Theorem 4. *Let P be a DLP over $\mathcal{A}$. Then, there exists a normal logic program Q over $\mathcal{A}$ such that $P \equiv_s Q$ are strongly equivalent iff $HT(P)$ is closed under here-intersection.*

Indeed, the construction of the normal program relies on the canonical program. Hence, given a DLP P, first one has to compute the HT-models $HT(P)$ of P, and then generate the canonical program $P_{UE(P)}$. By construction, the latter is the desired normal program, whenever $HT(P)$ is closed under here-intersection.

The case of uniform equivalence is elegantly settled by considering the cover-mapping U_{cover} from above. Recall that U_{cover} is a uniform mapping (see Lemma 6). Now we observe:

Lemma 10. *For every set $\mathcal{S}$ of HT-interpretations, $U_{cover}(\mathcal{S})$ is closed under here-intersection.*

From Theorem 2 and Lemma 10, we thus obtain:

Theorem 5. *For every DLP P there exists some normal logic program Q over $\mathcal{A}$ such that $P \equiv_u Q$.*

Again, the construction of the normal program relies on the canonical program. Given a DLP P, compute the UE-models $UE(P)$ of P, select a uniform mapping U, and then generate the canonical program $P_{U(HT(P))}$. By construction, the latter is the desired normal program whenever $U(\mathcal{S})$ is a set closed under here-intersection. As we have seen, $U_{cover}(\mathcal{S})$ satisfies this property for arbitrary $\mathcal{S}$.

As uniform equivalence implies ordinary equivalence, we immediately obtain:

Corollary 6. *For every DLP P, there exists some normal logic program Q such that $P \equiv Q$.*

In the case of ordinary equivalence, however, we can make use of stable mappings $I(\cdot)$ as in Theorem 3, as long as their outcome satisfies here-intersection. This holds for two of the presented mappings:

Lemma 11. *For any set $\mathcal{S}$ of HT-interpretations, $I_{lcover}(\mathcal{S})$ and $I_{nocover}(\mathcal{S})$ are closed under here-intersection.*

Proof. This is easy to see: as for each non-total $(X, Y) \in \mathcal{S}'$, we have $(Z, Y) \in \mathcal{S}'$ for any $Z \subseteq X$, closure of $\mathcal{S}' = I_{lcover}(\mathcal{S})$ under here-intersection cannot be violated. Furthermore, $I_{nocover}(\mathcal{S})$ only contains total HT-models; again, this immediately shows that $\mathcal{S}$ is closed under here-intersection. ∎

Hence, we can obtain a normal program $P_{\mathcal{S}}$ as a faithful rewriting of an arbitrary DLP P by using $\mathcal{S} = \{(Y, Y) \mid Y \in \mathcal{AS}(P)\}$ or $\mathcal{S} = I_{lcover}((Y, Y) \mid Y \in \mathcal{AS}(P)\}$.

Note that the "standard" rewriting

$$\bigcup_{Y \in \mathcal{AS}(P)} \{\{y\} \leftarrow not\, (\mathcal{A} \setminus Y) \mid y \in Y\}$$

can also be obtained via our canonical program $P_{\mathcal{A},\mathcal{S},\mathcal{S}'}$, by setting $\mathcal{S} = \{(Y, Y) \mid Y \in \mathcal{AS}(P)\}$ and $\mathcal{S}' = \{(\emptyset, Y) \mid Y \in \mathcal{AS}(P), Y \neq \emptyset\}$. It is ordinarily equivalent to P in case $\mathcal{AS}(P) \neq \emptyset$. Note that $\mathcal{S}'$ is complementary to $\mathcal{S}$, $\mathcal{I}_{nocover}(\mathcal{S}) = \mathcal{S}$, and $\mathcal{S}$ is closed under here-intersection.

Computational Complexity. We finally provide complexity results.

Theorem 6. *Given a DLP P, checking closure under here-intersection for $HT(P)$ is complete for* coNP. *Hardness holds even for a constraint-free HCF program with a single disjunction.*

Proof. Membership for the complementary problem is by guessing two HT-interpretations (X, Y) and (X', Y), and checking whether $(X, Y) \in HT(P)$, $(X', Y) \in HT(P)$, $(X \cap X', Y) \notin HT(P)$. As HT-model-checking is feasible in polynomial time (Eiter, Fink, & Woltran, 2007), membership in coNP follows.

For hardness, we reduce the problem of validity of a DNF formula ϕ to a program P such that $HT(P)$ is closed under here-intersection iff ϕ is valid. To this end, consider ϕ in 3-DNF, i.e., $\phi = \bigvee_{i=1}^{m} l_{i,1} \wedge l_{i,2} \wedge l_{i,3}$, and let

$$\begin{aligned} P \;=\; & \{w \leftarrow l_{i,1}, l_{i,2}, l_{i,3} \mid 1 \leq i \leq m\} \cup \\ & \{a \vee b \leftarrow not\, w\}, \end{aligned}$$

where in P negative literals $l_{i,j} = \neg v$ are written as $not\ v$ and a, b, and w are new atoms. Obviously, the program is head-cycle free, contains no constraints, and is obtained from ϕ in polynomial time.

We show that $HT(P)$ satisfies here-intersection iff ϕ is valid. Remember that $(X, Y) \in HT(P)$ iff $Y \models P$ and $X \models P^Y$. Consider the case that ϕ is valid. Then, $w \in Y$ holds for every model Y of P. Hence, for each $(Y, Y) \in HT(P)$, P^Y is Horn. As models of Horn programs are closed under intersection, $HT(P)$ is closed under here-intersection. On the other hand, if ϕ is not valid we have some $(Y, Y) \in HT(P)$ with $w \notin Y$, and $\{a, b\} \subseteq Y$. One can check that then $(Y \setminus \{a\}, Y) \in HT(P)$, $(Y \setminus \{b\}, Y) \in HT(P)$, but $(Y \setminus \{a, b\}, Y) \notin HT(P)$, i.e., $HT(P)$ is not closed under here-intersection. ∎

Corollary 7. *Checking whether, for a given DLP P, there exists a normal program Q such that $P \equiv_s Q$ is* coNP*-complete. Hardness holds for the settings as in Theorem* 6*.*

For uniform, and respectively, ordinary equivalence, we have seen that such a normal program always exists and thus the decision problem is trivial.

4.2 Local Elimination of Disjunction

To employ disjunction elimination locally, i.e., rule per rule, we start from the concept of shifting which is known to be faithful for a rule r with respect to uniform (and thus, ordinary) equivalence in case r is head-cycle free in the given program P (Eiter, Fink, & Woltran, 2007). As we shall see, this shift in general generates additional HT-models. In order to avoid this, we add further rules whenever necessary by using our canonical program. This is done in such a way, that whenever the given program's HT-models are closed under here-intersection, these "repair" rules are normal.

We start with a formal definition of the well-known head-to-body shift.

Definition 16. *Given a rule r, we define*

$$r^{\rightarrow} = \begin{cases} \{h \leftarrow B^+(r), not\ B^-(r), not\ (H(r) \setminus h) \mid h \in H(r)\} & \text{if } |H(r)| > 1; \\ \{r\} & \text{otherwise}. \end{cases}$$

Moreover, $P_r^{\rightarrow}$ denotes the program $(P \setminus \{r\}) \cup r^{\rightarrow}$ and $P^{\rightarrow} = \bigcup_{r \in P} r^{\rightarrow}$.

Proposition 9 ((Eiter, Fink, & Woltran, 2007))**.** *For every DLP P and rule $r \in P$ that is HCF in P, $P \equiv_u P_r^{\rightarrow}$. Moreover, for every HCF program P, $P \equiv_u P^{\rightarrow}$.*

Shifting does not work in case of strong equivalence, even for head-cycle free programs, as the prominent example $P = \{a \vee b \leftarrow\}$ shows. Observe that $P^{\rightarrow} = \{a \leftarrow not\ b;\ b \leftarrow not\ a\}$, and we already have seen several times that the latter program has an additional HT-model $(\emptyset, ab)$. Indeed, $HT(P)$ is not closed under here-intersection and thus, in view of Theorem 4, we know that no normal program that is strongly equivalent to P exists.

Another program over atoms $\{a, b\}$ which does not possess a strongly equivalent normal program is $\{a \vee b \leftarrow;\ a \leftarrow not\ a\}$; it has the same HT-models as $\{a \vee b \leftarrow\}$ except for (b, b).

The next result characterises the exact difference between r and $r^{\rightarrow}$ in terms of HT-models.

Lemma 12. *For any rule r,*

(*i*) $HT(r) \subseteq HT(r^{\rightarrow})$ *and*

(*ii*) $HT(r^{\rightarrow}) \setminus HT(r) = \mathcal{S}_r^{\rightarrow}$*, with*

$$\mathcal{S}_r^{\rightarrow} = \{(X,Y) \mid X \subseteq Y,\ |H(r) \cap Y| \geq 2,\ X \not\models r^Y\}.$$

(*iii*) *for every interpretation Y it holds that $(Y,Y) \in HT(r)$ iff $(Y,Y) \in HT(r^{\rightarrow})$.*

Proof. As for (iii), this is easily seen by the fact that total HT-models do not make a difference between $H(r)$ and $B^-(r)$.

In what follows, let r_p denote the rule in $r^{\rightarrow}$ with $H(r_p) = p$.

To show (i), suppose, towards a contradiction, that a non-total HT-interpretation $(X,Y) \in HT(r)$ exists such that $(X,Y) \notin HT(r^{\rightarrow})$. From the latter, we know that there exists a rule $r_p \in r^{\rightarrow}$ such that $(X,Y) \notin HT(r_p)$. Since $(Y,Y) \in HT(r)$, we get from item (iii) that also $(Y,Y) \in HT(r^{\rightarrow})$. Since $(X,Y) \notin HT(r^{\rightarrow})$, the following then must hold jointly: $Y \cap B^-(r_p) = \emptyset$, $B^+(r_p) \subseteq X$, and $X \cap H(r_p) = \emptyset$. Since $H(r) \subseteq B^-(r_p) \cup H(r_p)$ and $B^+(r) = B^+(r_p)$, we thus would get $(X,Y) \notin HT(r)$, a contradiction.

For (ii), note that $\mathcal{S}_r^{\rightarrow} \cap HT(r) = \emptyset$. It remains to show that $HT(r^{\rightarrow}) \subseteq HT(r) \cup S_r^{\rightarrow}$. Therefore, consider some $(X,Y) \in HT(r^{\rightarrow})$, and suppose that $(X,Y) \notin HT(r)$. We show that $(X,Y) \in S_r^{\rightarrow}$. Towards a contradiction, suppose that $(X,Y) \notin S_r^{\rightarrow}$. Since $(X,Y) \notin HT(r)$, $X \not\models H(r)$, $Y \models B(r)$, 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 $(X,Y) \in HT(r^{\rightarrow})$. Otherwise, if $Y \models H(r)$, we have $X \models B^+(r)$, and we get $|Y \cap H(r)| = 1$, otherwise $(X,Y) \in S_r^{\rightarrow}$. 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 $(X,Y) \in HT(r^{\rightarrow})$. ∎

In other words $\mathcal{S}_r^{\rightarrow}$ contains those HT-interpretations (X,Y), such that $(X,Y) \notin HT(r)$ and $|H(r) \cap Y| \geq 2$. Note that the latter implies $(Y,Y) \in HT(r)$, and together with $(X,Y) \notin HT(r)$ this implies $Y \cap B^-(r) = \emptyset$, otherwise r^Y would be void. Also observe that $\mathcal{S}_r^{\rightarrow}$ contains only non-total HT-interpretations witnessing the fact that r and $r^{\rightarrow}$ are indeed classically equivalent.

We define now notions of extended shifting exploiting concepts from the previous section. In fact, we just need to add the canonical program for $HT(P)$ with respect to $\mathcal{S}_r^{\rightarrow}$, since this program rules out all additional HT-models.

Definition 17. *Let P be a program over $\mathcal{A}$ and $r \in P$. We define*

$$P_r^{\Rightarrow} = P_r^{\rightarrow} \cup P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow}}.$$

Moreover, let $P^{\Rightarrow}$ denote the program $\bigcup_{r \in P} \left(r^{\rightarrow} \cup P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow}}\right)$.

Note that in case r is already normal, $P_r^{\rightarrow} = P$, and moreover, $\mathcal{S}_r^{\rightarrow}$ is empty, which yields that $P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow}}$ is the empty program. Hence, in this case $P_r^{\Rightarrow} = P$.

Theorem 7. *For every program P and every rule $r \in P$, the following holds:*

(*i*) $P \equiv_s P_r^{\Rightarrow}$;

(*ii*) *if $HT(P)$ satisfies here-intersection, then $P_r^{\Rightarrow} \setminus P$ is normal, i.e., all new rules are normal.*

Proof. The second part is an immediate consequence of Lemma 9.

The proof of the first part is as follows: First, from Lemma 12, we know $HT(P) \subseteq HT(P_r^{\rightarrow})$ and $HT(P_r^{\rightarrow}) \setminus HT(P) \subseteq \mathcal{S}_r^{\rightarrow}$. Hence, $\mathcal{S}_r^{\rightarrow}$ is complementary to $HT(P)$ in the sense of Definition 9. By Lemma 3, $HT(P) \subseteq HT(P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow}})$. Hence, since $P_r^{\Rightarrow} = P_r^{\rightarrow} \cup P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow}}$, we get $HT(P) \subseteq HT(P_r^{\Rightarrow})$ and $HT(P_r^{\Rightarrow}) \setminus HT(P) \subseteq \mathcal{S}_r^{\rightarrow}$. Now, by Lemma 4, we get that $HT(P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow}}) \cap \mathcal{S}_r^{\rightarrow} = \emptyset$, thus we end up with $HT(P) = HT(P_r^{\Rightarrow})$. ∎

Therefore, given a program P, the rewriting $P^{\Rightarrow}$ yields a strongly equivalent program which is normal whenever $HT(P)$ is closed under here-intersection. Hence, we defined a "local variant" of rewritings from DLPs to normal programs, however, taking the semantics of the entire program with respect to its HT-models into account. This "local variant" might be preferable, if there is only a small number of disjuntive rules to be rewritten and the structure of the remaining normal rules should remain unchanged. This might also be beneficial if the non-disjunctive part of the program to be recasted enjoys certain structural features that are typically lost by the global rewriting as used in Theorem 4.

With a slightly different rewriting, viz. using $P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow} \cap HT(P \setminus \{r\})}$ instead of $P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow}}$ in $P_r^{\Rightarrow}$ from Definition 17, we obtain a simple head-to-body shift whenever it is faithful under strong equivalence, i.e., we only add the "repair rules" if necessary. It can be checked that this modification also satisfies Theorem 7.

Example 6. *Consider the program $\{a \vee b \leftarrow; \bot \leftarrow a, b;\}$. Our result shows that shifting $r = a \vee b \leftarrow$ in P is faithful with respect to strong equivalence. This is seen by the fact that $HT(P \setminus \{r\}) = \{(a,a), (b,b), (\emptyset,a), (\emptyset,b), (\emptyset,\emptyset)\}$. No such pair (X,Y) satisfies $|Y \cap H(r)| \geq 2$, and consequently $\mathcal{S}_r^{\rightarrow} \cap HT(P \setminus \{r\}) = \emptyset$. But then, $P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow} \cap HT(P \setminus \{r\})}$ is the empty program and we have $P_r^{\Rightarrow} = P_r^{\rightarrow}$ for this modified variant of $P_r^{\Rightarrow}$.*

If we want to shift rules in such a way that uniform equivalence is retained, we just simply adapt our concept as follows: First, we take for the "repair rule" the uniform cover mapping for $HT(P)$, i.e., $U_{cover}(HT(P))$, into account instead of $HT(P)$. Second, we can now refrain from elements in $\mathcal{S}_r^{\rightarrow}$ which are contained in $U_{cover}(HT(P))$, since this repair is not necessary anymore.

Definition 18. *Let P be a program over $\mathcal{A}$, $r \in P$, $\mathcal{S} = U_{cover}(HT(P))$, and $\mathcal{S}' = \mathcal{S}_r^{\rightarrow} \setminus \mathcal{S}$. Then,*

$$P_r^{\Rightarrow_U} = P_r^{\rightarrow} \cup P_{\mathcal{A},\mathcal{S},\mathcal{S}'},$$

and $P^{\Rightarrow_U} = \bigcup_{r \in P} \left(r^{\rightarrow} \cup P_{\mathcal{A},\mathcal{S},\mathcal{S}'}\right)$.

Again, we have the property that $P_r^{\Rightarrow_U} = P$ whenever r is normal.

Example 7. *Once more, we use our canonical example program P consisting of a single disjunctive rule r given by $a \vee b \leftarrow$. We already have used $HT(P) = \mathcal{S}_1$ in previous examples, in particular in Example 1, and, moreover, we have seen that*

$$\mathcal{S}_2 = U_{cover}(\mathcal{S}_1) = \{(ab, ab), (a, ab), (b, ab), (a, a), (b, b), (\emptyset, ab)\}$$

and thus $\mathcal{S}_r^{\rightarrow} \setminus \mathcal{S}_2$ is empty. We thus obtain the program $P_r^{\Rightarrow} = \{a \leftarrow not\, b; b \leftarrow not\, a\}$ as expected.

The general relation is as follows.

Theorem 8. *For any program P and any rule $r \in P$ the following holds:*

(*i*) $P \equiv_u P_r^{\Rightarrow U}$ *and*

(*ii*) $P_r^{\Rightarrow U} \setminus P$ *is normal.*

A certain optimisation of this rewriting is by taking programs $P_{\mathcal{A},\mathcal{S},\mathcal{S}'} \neq \emptyset$ such that $\mathcal{S}' \subseteq HT(P \setminus \{r\})$, otherwise we may add redundant rules in $P_r^{\Rightarrow}$. Thus, for instance, instead of $P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow}}$ we could take $P_{\mathcal{A},HT(P),\mathcal{S}_r^{\rightarrow} \cap HT(P_r^{\rightarrow})}$. With this optimisation one can show that our method generalises the head-to-body shift for HCF program.

Having a local rewriting which retains uniform equivalence for any DLP, we thus have also obtained such a rewriting for ordinary equivalence, as well.

We mention here that our presentation, although quite different, subsumes all the rewritings that we have presented in preliminary work (Eiter, Faber, et al., 2004). In fact, one can check that that the object $P_r^{\Rightarrow}$ amounts to the rewriting $P_{r,s}$ in Definition 5 from Eiter, Fink, et al. (2004a); and $P_r^{\Rightarrow U}$ amounts to $P_{r,u}$ in Definition 6 from that paper. The rewriting with respect to ordinary equivalence introduced in Lemma 4 of Eiter, Fink, et al. (2004a) is obtained in our new framework as as well by taking $P_{\mathcal{A},\mathcal{S},\mathcal{S}_r^{\rightarrow}}$ with $\mathcal{S} = \{(Y, Y) \mid Y \in \mathcal{AS}(P)\}$ into account.

Hence, we have shown that employing the new canonical notion of programs subsumes all rewritings presented in earlier work, which have been defined in a rather ad-hoc manner. In other words, our results here provide a more general access to program rewriting since we made precise how canonical programs can be used for a local rewriting. Together with the choice of suitable operators as uniform mappings, a broad range of several different rewriting approaches can be obtained.

5 Eliminating Negation

Besides disjunction, the second connective in DLPs that may considerably affect the efficiency of program evaluation, and thus has given rise to various syntactic program classes, is (default) negation. In this section, we thus study the elimination of negation.

We start by considering the global elimination of negation from DLPs retaining strong, respectively uniform, equivalence and then briefly draw some conclusions for NLPs. Throughout this section we use $\mathcal{S}_e(P)$, where $e \in \{s, u\}$, to denote $HT(P)$ in case $e = s$, and $UE(P)$ otherwise.

5.1 Elimination of Negation in DLPs

We start with introducing two further notions on sets of HT-interpretations in analogy to here-intersection, viz. *here-totality* and *there-intersection*.

Definition 19. *A set $\mathcal{S}$ of HT-interpretations is* here-total *iff $(X, Y) \in \mathcal{S}$ implies $(X, X) \in \mathcal{S}$, and $\mathcal{S}$ is* closed under there-intersection *iff for every $(X, X), (Y, Y) \in \mathcal{S}$ it holds that $(X \cap Y, X \cap Y) \in \mathcal{S}$.*

Lemma 13. *For every positive logic program P, $HT(P)$ is here-total.*

Proof. If P is positive, then $X \models P$ for every $(X, Y) \in HT(P)$, which implies $(X, X) \in HT(P)$. ∎

Since, for any program P, $HT(P) \setminus UE(P)$ contains only non-total HT-interpretations, the following result is an immediate consequence of Lemma 13.

Corollary 8. *For every positive logic program P, $UE(P)$ is here-total.*

Next, we restate (in terms of the above notion of here-totality) and slightly generalise (also considering strong equivalence) a result (more precisely Theorem 5) from Eiter and Fink (2003).

Lemma 14. *Let P be a DLP. Then the following conditions are equivalent, for $e \in \{s, u\}$:*

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

(*ii*) *$\mathcal{S}_e(P)$ is here-total.*

Proof. First, observe that $\mathcal{S}_e(P)$ is here-total iff $X \models P$ holds for every $(X, Y) \in \mathcal{S}_e(P)$: Indeed, by definition here-totality implies $(X, X) \in \mathcal{S}_e(P)$, for every $(X, Y) \in \mathcal{S}_e(P)$, and thus $X \models P$. On the other hand, if $X \models P$ holds for every $(X, Y) \in \mathcal{S}_e(P)$, then $(X, X) \in HT(P)$ holds, and since $HT(P)$ and $UE(P)$ coincide on total models, $(X, X) \in \mathcal{S}_e(P)$ follows, proving here-totality of $\mathcal{S}_e(P)$.

Given the above, for uniform equivalence, the lemma coincides with Theorem 5 from Eiter and Fink (2003). Moreover, the proof of this result (ibidem) also works for the case of strong equivalence, i.e., when $HT(P)$ and $\models_s$ are considered instead of $UE(P)$ and $\models_u$. To wit, we proceed by essentially repeating the argument (generalising it) for $\mathcal{S}_e(P)$ and $\models_e$.
(i) $\Rightarrow$ (ii) Suppose $P \models_e r$ iff $P \models r$, for every rule r, but there exists an HT-model (X, Y) of P such that $X \not\models P$. Hence $X \not\models r$ for some rule $r \in P$. Let r' be the rule which results from r by shifting the negative literals to the head, i.e., $H(r') = H(r) \cup B^-(r)$, $B^+(r') = B^+(r)$, and $B^-(r') = \emptyset$. Since r and r' are classically equivalent, i.e., $I \models r$ iff $I \models r'$ holds for every interpretation I, and because $P \models r$, we conclude that $P \models r'$ and by (i) that $P \models_e r'$. The latter implies that $(X, Y) \models r'$. Moreover, $B^-(r') = \emptyset$ implies that $X \models r'$, and thus $X \models r$. This is a contradiction. Therefore, $X \models P$ holds for every (X, Y) in $\mathcal{S}_e(P)$, proving (ii).
(ii) $\Rightarrow$ (i) Suppose that $\mathcal{S}_e(P)$ is here-total, i.e., $X \models P$ holds for every (X, Y) in $\mathcal{S}_e(P)$. Then, $P \models_e r$ implies $P \models r$, for every rule r, because $X \models P$ iff $(X, X) \in \mathcal{S}_e(P)$, i.e., classical models of P coincide with the total HT-models in $\mathcal{S}_e(P)$. It is left to show that $P \models r$ implies $P \models_e r$, for

every rule r. Towards a contradiction assume $P \models r'$ but $P \not\models_e r'$, for some r'. Then, $(X, Y) \not\models r'$ for some $(X, Y) \in \mathcal{S}_e(P)$. However, since $\mathcal{S}_e(P)$ is here-total, both X and Y are models of P, and thus $P \models r'$ implies $X \models r'$ and $Y \models r'$. This implies $(X, Y) \models r'$, a contradiction. Consequently, $P \models_e r$ iff $P \models r$, i.e. (i), holds for every rule r. ∎

Turning to the question whether we can globally eliminate negation from a DLP under strong (respectively uniform) equivalence—that is we ask for the existence of a positive program that is strongly (respectively uniformly) equivalent to the given program—the above lemma gives a necessary criterion in terms of a property of the HT-models for the existence of an equivalent positive program, namely here-totality. It can be shown that this necessary condition is also sufficient. More precisely, we prove:

Theorem 9. *Let P be a DLP. Then, there exists a positive program Q such that $P \equiv_e Q$ iff $\mathcal{S}_e(P)$ is here-total, where $e \in \{s, u\}$.*

Proof. ($\Leftarrow$) Suppose $\mathcal{S}_e(P)$ is here-total. Then, by Lemma 14, it holds that $P \models_e Q$ iff $P \models Q$, for any program Q. Now let Q be any set of positive rules such that the models of P and Q coincide (such a Q clearly exists). Then, $P \models Q$, and hence $P \models_e Q$. Since Q is positive and $Q \models P$, by a simple corollary to Lemma 14, it follows that $Q \models_e P$. This proves $P \equiv_e Q$.
($\Rightarrow$) $P \equiv_e Q$ implies $Q \models P$ and $\mathcal{S}_e(P) = \mathcal{S}_e(Q)$. It follows for all $(X, Y) \in \mathcal{S}_e(P)$ that $X \models P$ and thus $(X, X) \in \mathcal{S}_e(P)$, i.e., that $\mathcal{S}_e(P)$ is here-total. ∎

Note that Theorem 9 implies that if P is e-equivalent to some positive program Q, then we can obtain Q by shifting not -literals to the head. Let, for any rule r, r^{ls} be the rule such that

- $H(r^{ls}) = H(r) \cup B^-(r)$,
- $B^+(r^{ls}) = B^+(r)$, and
- $B^-(r^{ls}) = \emptyset$

(i.e., the "left shift" of r). Furthermore, let $P^{ls} = \{r^{ls} \mid r \in P\}$.

Theorem 10. *Let P be any DLP and $e \in \{s, u\}$. Then, the following statements are equivalent:*

(i) There exists a positive program Q such that $P \equiv_e Q$.

(ii) $P \equiv_e P^{ls}$.

(iii) $P \models_e r^{ls}$, for every $r \in P$ such that $B^-(r) \neq \emptyset$.

Proof. Note that for any positive program Q and any rule r, $Q \models_e r$ iff $Q \models r^{ls}$. Furthermore, $P \equiv_e Q$ implies $Q \models_e r$, for all $r \in P$. Thus, $P \equiv_e Q$, where Q is positive, implies $Q \models r^{ls}$, for all $r \in P$. Hence, $Q \equiv P^{ls}$, in this case.

This means that there exists a positive program Q such that $P \equiv_e Q$ iff $P \equiv_e P^{ls}$, i.e., iff $P \models_e P^{ls}$. Since $r^{ls} = r$ if $B^-(r) = \emptyset$ (and then trivially $P \models_e r$), we get $P \models_e P^{ls}$ iff $P \models_e r^{ls}$, for all $r \in P$ such that $B^-(r) \neq \emptyset$. ∎

Before turning to an example, let us briefly recall, regarding ordinary equivalence, that for any P there always exists a positive program Q such that $P \equiv Q$.

Example 8. *Consider* $P = \{a \vee b \leftarrow \mathit{not}\, a\}$. *Then,* $P^{ls} = \{a \vee b \leftarrow\}$ *and observe that neither* $P \equiv_e P^{ls}$ *holds nor that* $\mathcal{S}_e(P)$ *is here-total (e.g.,* $(\emptyset, ab) \in \mathcal{S}_e(P)$ *but* $\emptyset \not\models P$*), for* $e \in \{s, u\}$. *Hence, it is impossible to remove negation while retaining strong or uniform equivalence in this case.*

As for computational complexity, the following result holds:

Theorem 11. *Checking here-totality, for a given DLP* P*, is* coNP*-complete for* $HT(P)$ *and* Π_2^P*-complete for* $UE(P)$.

Proof. For proving the membership part of the theorem, observe that the complementary problems are in NP and Σ_2^P, respectively. To wit, subsets X and Y over the language $A \subseteq \mathcal{A}$ of P can be chosen nondeterministically, and the subsequent tests for $(X, Y) \in \mathcal{S}_e(P)$ and $(X, X) \notin \mathcal{S}_e(P)$ can be carried out in polynomial time, the former with an additional NP-oracle call in case $e = u$. This proves coNP-membership of here-totality checking for $HT(P)$, respectively Π_2^P-membership for $UE(P)$.

To show the coNP-hardness part of the theorem, we reduce UNSAT to the problem as follows. Let $F = \bigwedge_{i=1}^{n} C_i$ be a Boolean formula in CNF over atoms X. Let a and $\bar{c}$, where $c \in X$, be fresh atoms, and consider

$$P = \{x \vee \bar{x} \leftarrow;\ \leftarrow x, \bar{x} \mid x \in X\} \cup \{\leftarrow \overline{C}_i \mid 1 \leq i \leq n\} \cup \{a \leftarrow \mathit{not}\, a\},$$

where $\overline{C}_i$ is the conjunction obtained from $\neg C_i$ after applying DeMorgan's law and replacing negative literals $\neg x$ by $\bar{x}$. This program has a classical model, and thus $HT(P)$ is non-empty, iff F is satisfiable. However, whenever $HT(P) \neq \emptyset$, then it is also not here-total due to the rule $a \leftarrow \mathit{not}\, a$. Therefore, $HT(P)$ is here-total iff F is satisfiable (note that $HT(P) = \emptyset$ trivially is here-total). This proves coNP-hardness of deciding whether $HT(P)$ is here-total.

To prove Π_2^P-hardness of deciding here-totality for $UE(Q)$ of a given DLP Q, consider a QBF of the form $F = \forall X \exists Y \phi$ with $\phi = \bigwedge_{i=1}^{m} C_i$, where each C_i is a disjunction of literals over the Boolean variables in $X \cup Y$. Deciding whether a given such F is true is well-known to be Π_2^P-complete.

Let a, b, and $\bar{c}$, where $c \in X \cup Y$, be fresh atoms and define

$$\begin{aligned} Q \;=\; & \{x \vee \bar{x} \leftarrow;\ \leftarrow x, \bar{x} \mid x \in X\} \cup \\ & \{y \vee \bar{y} \leftarrow z \mid y \in Y;\ z \in Y \cup \bar{Y} \cup \{a\}\} \cup \\ & \{b \leftarrow y, \bar{y};\ y \leftarrow b;\ \bar{y} \leftarrow b \mid y \in Y\} \cup \\ & \{b \leftarrow \overline{C}_i \mid 1 \leq i \leq m\} \cup \\ & \{a \leftarrow b;\ a \leftarrow \mathit{not}\, b;\ a \leftarrow \mathit{not}\, a\}. \end{aligned}$$

The set of all HT-models of Q is

$$\begin{aligned} & \{(\sigma_{X \cup Y}(I) \cup \{a\}, \sigma_{X \cup Y}(I) \cup \{a\}) \mid I \in M_\phi\} \cup \\ & \{(\sigma_{X \cup Y}(I) \cup \{a\}, \sigma_X(I) \cup \mathcal{A}) \mid I \in M_\phi\} \cup \\ & \{(\sigma_X(J) \cup \mathcal{A}, \sigma_X(J) \cup \mathcal{A}) \mid J \subseteq X\} \cup \\ & \{(\sigma_X(J), \sigma_X(J) \cup \mathcal{A}) \mid J \subseteq X\}, \end{aligned}$$

where $\sigma_A(I)$ denotes the set of atoms from A that are true under a given interpretation I of ϕ, and M_ϕ denotes the set of all models of ϕ.

It thus holds that $UE(Q)$ is here-total iff none of the HT-models of the form $\{(\sigma_X(J), \sigma_X(J) \cup \mathcal{A})\}$ is an UE-model of Q. This is the case iff, for each $J \subseteq X$, there exists a value assignment to Y that makes ϕ true, i.e., iff the QBF $\forall X \exists Y \phi$ evaluates to true. ∎

5.2 Elimination of Negation in NLPs

When restricting our attention to NLPs and eliminate (under strong or uniform equivalence) negation from a program, there are two different cases to consider.

The first is that the resulting program should be normal; that is, the outcome should be an equivalent Horn program. The resulting problem turns out to be a specific case (P is normal) of the more general problem of jointly eliminating disjunction and negation from a DLP, which we consider in the next subsection. (For an illustrative instance where P is a NLP, simply consider P^{ls} instead of P in the subsequent Example 9.)

The second case is that one is allowed to introduce disjunction in rule heads; that is, the outcome should be an equivalent positive DLP. This setting is a special case (P is normal) of the more general case (P is any DLP) considered above, and the same characterisations apply. Also, the problem does not trivialise for NLPs (consider, for instance, the shift program $P^{\rightarrow}$ of the DLP P in Example 8), not even simplify (in the hardness reductions of Theorem 11 one can use default negation instead of disjunction).

5.3 Joint Elimination of Disjunction and Negation

It is tempting to combine the sufficient and necessary criteria for eliminating negation (here-totality) and for eliminating disjunction (here-intersection) to obtain a criterion for eliminating both negation and disjunction. However, this does not work as shown by the following example.

Example 9. *Let $P = \{a \vee b \leftarrow;\ \leftarrow a, b\}$ and observe that $\mathcal{S}_e(P) = \{(a,a), (b,b)\}$ is both closed under here-intersection and here-total for $e \in \{s, u\}$. Nevertheless, we cannot avoid disjunction and negation simultaneously, i.e., we cannot remove the disjunction without introducing negation and vice versa. In other words, there exists no Horn program Q such that $P \equiv_e Q$.*

However, we can establish such a criterion by means of there-intersection:

Theorem 12. *For every DLP P and $e \in \{s, u\}$, there exists some Horn program Q such that $P \equiv_e Q$ iff $\mathcal{S}_e(P)$ is here-total and closed under there-intersection.*

Proof. Note that $\mathcal{S}_e(P)$ contains all classical models of P. For the if direction, suppose that $\mathcal{S}_e(P)$ is here-total and closed under there-intersection. The latter implies the existence of a Horn program Q such that the classical models of P and Q coincide (and equal $\mathcal{S}_e(P)$). By Lemma 14, it follows that $P \models_e Q$ iff $P \models Q$, and $Q \models_e P$ iff $Q \models P$, i.e., $P \equiv_e Q$ iff their classical models coincide. This is the case.

For the only-if direction, let Q be a Horn program such that $\mathcal{S}_e(P) = \mathcal{S}_e(Q)$. Since Q is positive, for every $(X, Y) \in \mathcal{S}_e(Q)$, X and Y are models of Q and thus (as a consequence of Lemma 14) models of P as well. Thus, $(X, X) \in \mathcal{S}_e(P)$, i.e., $\mathcal{S}_e(P)$ is here-total. Furthermore, as Q is Horn, $(X, X) \in \mathcal{S}_e(Q)$ and $(Y, Y) \in \mathcal{S}_e(Q)$ implies that $(X \cap Y, X \cap Y) \in \mathcal{S}_e(P)$. Hence, $\mathcal{S}_e(Q)$, and therefore also $\mathcal{S}_e(P)$, is closed under there-intersection. ∎

Note, however, that the canonical program in general is not Horn in this case, and we leave corresponding (local) rewriting rules for recasting as a subject for further work. Under ordinary equivalence though, the task is trivial (by encoding the unique answer set as a set of facts).

Concerning computational complexity, we eventually remark that the upper bounds established in Theorem 11 are readily extended to the corresponding joint decision problems. However, matching lower bounds are not evident and intuitively may not always exist. A more precise characterisation of the computational complexity is also left for future work.

6 Generalisations

In this section, we review some notions of equivalence between nonmonotonic logic programs, in the light of semantic properties that guarantee the existence of equivalent programs from some specific class of programs. We then discuss how some of the results (including ones in the previous sections) can be obtained using a general approach at an abstract level. Finally, we discuss generalisations towards non-ground programs.

There is a huge gap between the notions of strong equivalence (equivalence for substitution) and ordinary equivalence (the compared programs only have to possess the same answer sets; no addition of programs is considered). Uniform equivalence indeed is a notion between strong and ordinary equivalence by restricting the syntactic structure of the possibly added programs (i.e., restricting them to facts); however, it was shown by Pearce and Valverde (2004) that rule-wise restrictions of the syntax necessarily results in either strong, uniform, or ordinary equivalence.

Another option is to restrict the language of the the added programs. The latter approach has been introduced as *relativised strong equivalence* and has been investigated for ground programs (Eiter, Fink, & Woltran, 2007; Inoue & Sakama, 2004; Woltran, 2004). Similarly, a notion of relativised uniform equivalence was investigated in these papers.

Later, a more fine-grained parametrised notion of equivalence was investigated where the rules which might be added can be specified in terms of two alphabets $\mathcal{H}, \mathcal{B} \subseteq \mathcal{A}$, one for their heads and one for their bodies (Woltran, 2008). This concept, also termed *hyperequivalence* (Truszczyński & Woltran, 2009), allows for a common characterisation of strong and uniform equivalence and thus to understand the difference between these notions on model-theoretic grounds. More specifically, two programs P and Q over $\mathcal{A}$ are hyperequivalent relative to $\mathcal{H}$ and $\mathcal{B}$, if, for each DLP R such that $\bigcup_{r \in P} H(r) \subseteq \mathcal{H}$ and $\bigcup_{r \in P}(B^+(r) \cup B^-(r)) \subseteq \mathcal{B}$, it holds that $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$.

Another line of research considered equivalence notions where the comparison is restricted to *projected answer sets* (Eiter, Tompits, & Woltran, 2005). One such notion is *relativised strong equivalence with projection*: Given sets $A, O \subseteq \mathcal{A}$, two programs P and Q over $\mathcal{A}$ are strongly equivalent relative to A under projection to O, if, for any program R over A, it holds that $\{I \cap O \mid$

$I \in \mathcal{AS}(P \cup R)\} = \{I \cap O \mid I \in \mathcal{AS}(Q \cup R)\}$.

Based on the results presented in this article, we generalised the semantic properties for the existence of equivalent programs of a lower syntactic class to hyperequivalence (Pührer, Tompits, & Woltran, 2008) and relativised strong equivalence with projection (Pührer & Tompits, 2009). For all instances e of these parametrised equivalence relations, there is a semantic characterisation in terms of sets $M_e(\cdot)$ of HT-interpretations, analogous to HT-models for strong equivalence and UE-models for uniform equivalence, such that $M_e(P) = M_e(Q)$ for two programs P and Q iff $P \equiv_e Q$. In what follows, we will describe the general approach to obtain the extended results on an abstract level. The concrete properties and translations are detailed in the respective papers.

Let $\mathcal{NLP}$, $\mathcal{PLP}$, and $\mathcal{HP}$ be the classes of normal, positive, and Horn programs. For every $\mathcal{C} \in \{\mathcal{NLP}, \mathcal{PLP}, \mathcal{HP}\}$, a property $\phi_e^{\mathcal{C}}(\mathcal{S})$ can be defined that intuitively holds on the particular set $\mathcal{S} = M_e(P)$ if a casting and a completion function $\tau_e^{\mathcal{C}}(\mathcal{S})$ exist such that:

(i) whenever $\phi_e^{\mathcal{C}}(\mathcal{S})$ holds for a set $\mathcal{S}$ of HT-interpretations, then $\tau_e^{\mathcal{C}}(\mathcal{S})$ is complete and closed under here-intersection for $\mathcal{C} = \mathcal{NLP}$, here-total for $\mathcal{C} = \mathcal{NLP}$, and here-total and closed under there-intersection for $\mathcal{C} = \mathcal{HP}$;

(ii) if, for a set $\mathcal{S}$ of HT-interpretations, $\phi_e^{\mathcal{C}}(\mathcal{S})$ holds and some program $Q \in \mathcal{C}$ exists such that $\tau_e^{\mathcal{C}}(\mathcal{S}) = HT(Q)$, then $M_e(Q) = \mathcal{S}$;

(iii) for every program $Q \in \mathcal{C}$, $\phi_e^{\mathcal{C}}(M_e(Q))$ holds.

From these properties, we obtain the following result.

Proposition 10. *Let $\equiv_e$ denote an instance of hyper-equivalence or relativised strong equivalence with projection and let $\mathcal{C} \in \{\mathcal{NLP}, \mathcal{PLP}, \mathcal{HP}\}$. Then, for every set $\mathcal{S}$ of HT-interpretations, $\phi_e^{\mathcal{C}}(\mathcal{S})$ holds iff there exists a program $Q \in \mathcal{C}$ with $\mathcal{S} = M_e(Q)$.*

Proof. Assume that $\phi_e^{\mathcal{C}}(\mathcal{S})$ holds. Then, by (i) and Theorems 1, 4, 9, and 12, it follows that there is a program $Q \in \mathcal{C}$ with $HT(Q) = \tau_e^{\mathcal{C}}(\mathcal{S})$. Hence, by (ii), $M_e(Q) = \mathcal{S}$. Conversely, let $Q \in \mathcal{C}$ be a program such that $M_e(Q) = \mathcal{S}$. By (iii), $\phi_e^{\mathcal{C}}(M_e(Q))$ holds. Since $M_e(Q) = \mathcal{S}$, we get that $\phi_e^{\mathcal{C}}(\mathcal{S})$ holds. ∎

It follows that, for a given DLP P, $\phi_e^{\mathcal{C}}(M_e(P))$ holds iff there exists a program $Q \in \mathcal{C}$ such that $M_e(P) = M_e(Q)$. We thus obtain:

Corollary 9. *Let $\equiv_e$ denote an instance of hyperequivalence or relativised strong equivalence with projection and let $\mathcal{C} \in \{\mathcal{NLP}, \mathcal{PLP}, \mathcal{HP}\}$. Then, for a given DLP P, there exists a program $Q \in \mathcal{C}$ with $P \equiv_e Q$ iff $\phi_e^{\mathcal{C}}(M_e(P))$ holds.*

Note that Pührer et al. (2008) and Pührer and Tompits (2009) also reported concrete means to construct such a Q whenever $\phi_e^{\mathcal{C}}(M_e(P))$ holds, based on the canonical program as defined in this article.

So far, the study of recasting has focused on propositional programs; however, notions of equivalence have also been introduced for non-ground programs. More precisely, the research on notions of equivalence (akin to and) as considered in this article started out in the standard datalog

setting with the seminal undecidability result by Shmueli (1987) on *query equivalence* from 1987. After that, research focused on the one hand on identifying the exact frontier between decidable and undecidable language fragments (see, e.g., the work of Halevy, Mumick, Sagiv, and Shmueli (2001)), and, on the other hand, on sound approximations to query equivalence. Uniform equivalence was introduced by Sagiv (1988) as one such approach. *Equivalence of program segments* (which is the pendant to strong equivalence introduced above) was introduced by Maher (1988) but coincides with uniform equivalence for datalog queries. Lin (2002) was the first to discuss the non-ground variant of strong equivalence and gave a translation into the Bernays-Schönfinkel fragment of first-order logic as a decision procedure. The model-theoretic characterisation in terms of HT-models was finally lifted to the non-ground case by Eiter, Fink, Tompits, and Woltran (2005), who also presented first complexity and undecidability results which were subsequently complemented by Eiter, Fink, Tompits, and Woltran (2007). Using first-order variants of the logic of here-and-there to decide strong equivalence was later discussed by Lifschitz, Pearce, and Valverde (2007). Relativised notions of strong and uniform equivalence have been introduced to non-ground programs as well (Oetsch & Tompits, 2008). The most recent and general account of non-ground versions of equivalence, providing model-theoretic characterisations in terms of counter-models in HT, including relativised notions and hyper-equivalence, is the work by Fink (2011).

7 Conclusion

In this article, we dealt with the question whether for a given logic program P another logic program Q from a specific class of logic programs exists such that P and Q are equivalent with respect to different notions of equivalence on top of answer sets for nonmonotonic logic programs (Gelfond & Lifschitz, 1991). To this end, we exploited the seminal result by Lifschitz et al. (2001) on the relation between the logic of here-and-there and strong equivalence in answer-set programming, for developing semantic characterisations of different program classes (normal, positive, and Horn programs) in terms of the HT-models of the programs. Indeed, whenever the HT-models of a program P satisfy one of these properties, we are able to construct a corresponding program Q of the respective class that is equivalent to P under the considered notion of equivalence, using our notion of a canonical program.

As regards possible directions for future work, one issue is to consider recasting to further classes of programs, such as stratified (Apt et al., 1988) or tight logic programs (Erdem & Lifschitz, 2003), i.e., programs without loops in the positive dependency graphs. Work on the latter could profit from previous results on the relation of loops and HT-models (Gebser, Schaub, Tompits, & Woltran, 2008). Strong equivalence has also been studied under other semantics, including well-founded semantics (Cabalar, Odintsov, & Pearce, 2006; Nomikos, Rondogiannis, & Wadge, 2005) and supported model semantics (Truszczyński & Woltran, 2008). It could also be worthwhile to study program recasting in these settings. Further topics of interest include recasting for further notions of equivalence and for non-ground programs.

References

Apt, K., Blair, H., & Walker, A. (1988). Towards a theory of declarative knowledge. In J. Minker (Ed.), *Foundations of deductive databases and logic programming* (pp. 89–148). Washington DC: Morgan Kaufman.

Ben-Eliyahu, R., & Dechter, R. (1994). Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, *12*, 53–87.

Brewka, G., Eiter, T., & Truszczyński, M. (2011). Answer set programming at a glance. *Communications of the ACM*, *54*(12), 92-103.

Cabalar, P., & Ferraris, P. (2007). Propositional theories are strongly equivalent to logic programs. *Theory and Practice of Logic Programming*, *7*(6), 745–759.

Cabalar, P., Odintsov, S. P., & Pearce, D. (2006). Logical foundations of well-founded semantics. In P. Doherty, J. Mylopoulos, & C. A. Welty (Eds.), *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning* (*KR'06*)*, Lake District of the United Kingdom, June 2-5, 2006* (p. 25-35). AAAI Press.

Eiter, T., Faber, W., Fink, M., Pfeifer, G., & Woltran, S. (2004). Complexity of answer set checking and bounded predicate arities for non-ground answer set programming. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning* (*KR'04*) (pp. 377–387). AAAI Press.

Eiter, T., & Fink, M. (2003). Uniform equivalence of logic programs under the stable model semantics. In *Proceedings 19th International Conference on Logic Programming* (*ICLP'03*) (Vol. 2916, pp. 224–238). Springer Verlag.

Eiter, T., Fink, M., Tompits, H., & Woltran, S. (2004a). On eliminating disjunctions in stable logic programming. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning* (*KR'04*) (pp. 447–458). AAAI Press.

Eiter, T., Fink, M., Tompits, H., & Woltran, S. (2004b). Simplifying logic programs under uniform and strong equivalence. In V. Lifschitz & I. Niemelä (Eds.), *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04*) (Vol. 2923, pp. 87–99). Springer Verlag.

Eiter, T., Fink, M., Tompits, H., & Woltran, S. (2005). Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In *Proceedings of the 20th National Conference on Artificial Intelligence* (*AAAI'05*) (pp. 695–700). AAAI Press.

Eiter, T., Fink, M., Tompits, H., & Woltran, S. (2007). Complexity results for checking equivalence of stratified logic programs. In M. M. Veloso (Ed.), *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (*IJCAI'07*) (pp. 330–335). AAAI Press.

Eiter, T., Fink, M., & Woltran, S. (2007). Semantical characterizations and complexity of equivalences in answer set programming. *ACM Transactions on Computational Logic*, *8*(3). (53 pages)

Eiter, T., Tompits, H., & Woltran, S. (2005). On solution correspondences in answer set programming. In L. P. Kaelbling & A. Saffiotti (Eds.), *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (*IJCAI'05*) (pp. 97–102). Professional Book Center.

Erdem, E., & Lifschitz, V. (2003). Tight logic programs. *Theory and Practice of Logic Programming*, *3*(4-5), 499-518.

Ferraris, P. (2005). Answer sets for propositional theories. In C. Baral, G. Greco, N. Leone, & G. Terracina (Eds.), *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'05*) (Vol. 3662, p. 119-131). Springer Verlag.

Ferraris, P., & Lifschitz, V. (2005). Mathematical foundations of answer set programming. In S. N. Artëmov, H. Barringer, A. S. d'Avila Garcez, L. C. Lamb, & J. Woods (Eds.), *We Will Show Them!* (*1*) (p. 615-664). College Publications.

Fink, M. (2011). A general framework for equivalences in answer-set programming by countermodels in the logic of here-and-there. *Theory and Practice of Logic Programming*, *11*(2-3), 171-202.

Gebser, M., Schaub, T., Tompits, H., & Woltran, S. (2008). Alternative characterizations for program equivalence under answer-set semantics based on unfounded sets. In S. Hartmann & G. Kern-Isberner (Eds.), *Proceedings of the 5th International Symposium on Foundations of Information and Knowledge Systems* (*FoIKS'08*)*, Pisa, Italy, February 11-15, 2008* (Vol. 4932, p. 24-41). Springer Verlag.

Gelfond, M., & Lifschitz, V. (1988). The stable model semantics for logic programming. In R. Kowalski & K. Bowen (Eds.), *Proceedings of the 5th International Conference on Logic Programming* (*ICLP'88*) (pp. 1070–1080). The MIT Press.

Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, *9*, 365–385.

Halevy, A., Mumick, I., Sagiv, Y., & Shmueli, O. (2001). Static analysis in datalog extensions. *Journal of the ACM*, *48*(5), 971–1012.

Inoue, K., & Sakama, C. (2004). Equivalence of logic programs under updates. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence* (*JELIA'04*) (Vol. 3229, pp. 174–186). Springer Verlag.

Lifschitz, V., Pearce, D., & Valverde, A. (2001). Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, *2*(4), 526–541.

Lifschitz, V., Pearce, D., & Valverde, A. (2007). A characterization of strong equivalence for logic programs with variables. In C. Baral, G. Brewka, & J. Schlipf (Eds.), *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'07*) (Vol. 4483, pp. 188–200). Springer Verlag.

Lin, F. (2002). Reducing strong equivalence of logic programs to entailment in classical propositional logic. In D. Fensel, F. Giunchiglia, D. McGuiness, & M.-A. Williams (Eds.), *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning* (*KR'02*) (pp. 170–176). Morgan Kaufmann.

Maher, M. J. (1988). Equivalences of logic programs. In J. Minker (Ed.), *Foundations of deductive databases and logic programming* (pp. 627–658). Washington DC: Morgan Kaufman.

McKinsey, J. (1943). The decision problem for some classes of sentences without quantifiers. *Journal of Symbolic Logic*, *8*, 61–76.

Minker, J. (Ed.). (1988). *Foundations of deductive databases and logic programming*. Washington DC: Morgan Kaufman.

Nomikos, C., Rondogiannis, P., & Wadge, W. W. (2005). A sufficient condition for strong equivalence under the well-founded semantics. In M. Gabbrielli & G. Gupta (Eds.), *Proceedings of the 21st International Conference on Logic Programming* (*ICLP'05*)*, Sitges, Spain, October 2-5, 2005* (Vol. 3668, p. 414-415). Springer Verlag.

Oetsch, J., & Tompits, H. (2008). Program correspondence under the answer-set semantics: The non-ground case. In M. G. de la Banda & E. Pontelli (Eds.), *Proceedings of the 24th International Conference on Logic Programming* (*ICLP'08*) (Vol. 5366, pp. 591–605). Springer Verlag.

Pearce, D. (1997). A new logical characterisation of stable models and answer sets. In J. Dix, L. M. Pereira, & T. C. Przymusinski (Eds.), *Selected Papers of the 2nd International Workshop on Non-Monotonic Extensions of Logic Programming* (*NMELP '96*)*, Bad Honnef, Germany, September 5-6, 1996* (Vol. 1216, p. 57-70). Springer Verlag.

Pearce, D. (1999). Stable inference as intuitionistic validity. *Journal of Logic Programming*, *38*(1), 79-91.

Pearce, D. (2006). Equilibrium logic. *Annals of Mathematics and Artificial Intelligence*, *47*(1-2), 3-41.

Pearce, D., & Valverde, A. (2004). Uniform equivalence for equilibrium logic and logic programs. In V. Lifschitz & I. Niemelä (Eds.), *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'04*) (Vol. 2923, pp. 194–206). Springer Verlag.

Przymusinski, T. (1991). Stable semantics for disjunctive programs. *New Generation Computing*, *9*, 401–424.

Pührer, J., & Tompits, H. (2009). Casting away disjunction and negation under a generalisation of strong equivalence with projection. In E. Erdem, F. Lin, & T. Schaub (Eds.), *Proceedings of the 10th international conference on logic programming and nonmonotonic reasoning* (*LPNMR'09*) (pp. 264–276). 5753: Springer Verlag.

Pührer, J., Tompits, H., & Woltran, S. (2008). Elimination of disjunction and negation in answer-set programs under hyperequivalence. In M. G. de la Banda & E. Pontelli (Eds.), *Proceedings of the 24th Conference on Logic Programming* (*ICLP'08*) (pp. 561–575). 5366: Springer Verlag.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, *13*, 81–132.

Sagiv, Y. (1988). Optimizing datalog programs. In J. Minker (Ed.), *Foundations of deductive databases and logic programming* (pp. 659–698). Washington DC: Morgan Kaufman.

Shmueli, O. (1987). Decidability and expressiveness aspects of logic queries. In *Proceedings of the 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (*PODS'87*) (pp. 237–249). ACM Press.

Truszczyński, M. (2010). Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. *Artificial Intelligence*, *174*(16-17), 1285-1306.

Truszczyński, M., & Woltran, S. (2008). Hyperequivalence of logic programs with respect to supported models. *Annals of Mathematics and Artificial Intelligence*, *53*(1-4), 331-365.

Truszczyński, M., & Woltran, S. (2009). Relativized hyperequivalence of logic programs for modular programming. *Theory and Practice of Logic Programming*, *9*(6), 781–819.

Woltran, S. (2004). Characterizations for relativized notions of equivalence in answer set program-

ming. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence* (*JELIA'04*) (Vol. 3229, pp. 161–173). Springer Verlag.

Woltran, S. (2008). A common view on strong, uniform, and other notions of equivalence in answer-set programming. *Theory and Practice of Logic Programming*, *8*(2), 217–234.

Woltran, S. (2010). Equivalence between extended datalog programs - a brief survey. In O. de Moor, G. Gottlob, T. Furche, & A. J. Sellers (Eds.), *Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers* (Vol. 6702, pp. 106–119). Springer.