Strong Equivalence for Epistemic Logic Programs Made Easy

Wolfgang Faber, Michael Morak
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria

Stefan Woltran
TU Wien
Vienna, Austria

Abstract

Epistemic Logic Programs (ELPs), that is, Answer Set Programming (ASP) extended with epistemic operators, have received renewed interest in recent years, which led to a flurry of new research, as well as efficient solvers. An important question is under which conditions a sub-program can be replaced by another one without changing the meaning, in any context. This problem is known as strong equivalence, and is well-studied for ASP. For ELPs, this question has been approached by embedding them into epistemic extensions of equilibrium logics. In this paper, we consider a simpler, more direct characterization that is directly applicable to the language used in state-of-the-art ELP solvers. This also allows us to give tight complexity bounds, showing that strong equivalence for ELPs remains coNP-complete, as for ASP. We further use our results to provide syntactic characterizations for tautological rules and rule subsumption for ELPs.

1 Introduction

Epistemic Logic Programs (ELPs) are an extension of the well-established formalism of Answer Set Programming (ASP), a generic, fully declarative logic programming language that allows for encoding problems such that the resulting answers (called *answer sets*) directly correspond to solutions of the encoded problem (Brewka, Eiter, and Truszczynski 2011; Schaub and Woltran 2018). Negation in ASP is generally interpreted according to the stable model semantics (Gelfond and Lifschitz 1988), that is, as negation-as-failure, also called default negation. Such a default negation $\neg a$ of an atom a is true if there is no justification for a in the same answer set, making it a "local" operator in the sense that it is defined relative to one considered answer set. ELPs (in the version of (Shen and Eiter 2016)), on the other hand, extend ASP with the epistemic negation operator $\mathbf{not}$ that allows for a form of meta-reasoning, that is, reasoning over multiple answer sets. Intuitively, an epistemically negated atom $\mathbf{not}\, a$ expresses that a cannot be *proven* true, meaning that it is not true in every answer set. Thus, epistemic negation is defined relative to a collection of answer sets, referred to as a *world view*. Deciding whether a world view exists, is Σ_P^3-complete (Shen and Eiter 2016), one level higher on the polynomial hierarchy than deciding answer set existence (Eiter and Gottlob 1995).

Gelfond (1991; 1994) recognized epistemic negation as a desired construct for ASP early on and introduced the modal operators $\mathbf{K}$ ("known" or "provably true") and $\mathbf{M}$ ("possible" or "not provably false") to address this. $\mathbf{K}a$ and $\mathbf{M}a$ correspond to $\neg\mathbf{not}\, a$ and $\mathbf{not}\, \neg a$, respectively. Renewed interest in recent years has revealed several flaws in the original semantics, and various new approaches (cf. e.g. (Gelfond 2011; Truszczynski 2011; Kahl 2014; Fariñas del Cerro, Herzig, and Su 2015; Shen and Eiter 2016)) were proposed. Also the development of ELP solving systems (Kahl et al. 2015; Son et al. 2017; Bichler, Morak, and Woltran 2018) has gained momentum.

A main application and major motivation of ELPs is the formalization of the Closed World Assumption (CWA), as pointed out already by Gelfond (1991). Interestingly, there are two formalizations of the CWA using ELPs in the literature. The first, given in (Gelfond 1991)[1], shall be referred to as Gelfond-CWA and introduces one rule for each atom p:

$$p' \leftarrow \neg\mathbf{not}\, \neg p. \tag{1}$$

Intuitively, this says that p' (i.e. the negation of p) shall be true if there is no possible world where p is true. Shen and Eiter (2016) propose a different rule for CWA, which we will refer to as Shen-Eiter-CWA for p:

$$p' \leftarrow \mathbf{not}\, p. \tag{2}$$

Again, intuitively, this formulation makes p' true iff there is a possible world where p is false.

A natural question is whether (1) and (2) yield the same results in any context, which will be answered in this paper. To this end, we study notions of equivalence between ELPs. For instance, two ELPs Π_1 and Π_2 are strongly ELP-WV-equivalent iff, for any third ELP Π, the combined programs $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are equivalent (i.e. have the same world views). This notion is useful to transform ELPs into equivalent versions where one wants to verify that a local change

[1] Gelfond (1991) proposed the rule $\sim p \leftarrow \neg\mathbf{M}p$, where $\sim$ is a third kind of negation, usually referred to as strong negation, not considered in this paper. It can be simulated by replacing occurrences of $\sim p$ by a fresh atom p' and adding a constraint rule $\leftarrow p, p'$ that excludes p and p' to hold simultaneously.

preserves equivalence without considering the whole program. Other notions of strong equivalence can be defined for comparing candidate world views (rather than world views) or considering only the addition of programs that do not contain epistemic operators. In (plain) ASP, strong equivalence is a well-studied problem. Lifschitz, Pearce, and Valverde (2001) have provided an elegant characterization of the problem in terms of Heyting's logic of here-and-there (HT). Strong equivalence also proved useful as a means to simplify programs (Cabalar, Pearce, and Valverde 2007; Lin and Chen 2007; Eiter et al. 2013).

It has been shown in (Wang and Zhang 2005) and (Fariñas del Cerro, Herzig, and Su 2015), among others, that epistemic extensions of logic HT can be used to characterize strong equivalence of ELPs. These approaches, however, are very general and lead to a very abstract characterization that cannot be immediately used for ELPs written in the language of current solving systems. It is also not easy to obtain tight complexity results in such a general setting. The semantics considered in (Wang and Zhang 2005) is the original one of (Gelfond 1991), which is now considered obsolete, while (Fariñas del Cerro, Herzig, and Su 2015) consider a different semantics from the one in (Shen and Eiter 2016), which is what we use. An in-depth comparison of the differences in the semantics can also be found in (Shen and Eiter 2016).

In this paper, we therefore propose a simpler, more direct characterization for the well-understood ELP semantics given in (Shen and Eiter 2016). Our characterization is in the spirit of (Turner 2003), which is useful to study ELPs written in the input language of the ELP solvers mentioned above, as it can be directly applied in this setting. This also allows us to obtain tight complexity bounds for checking strong equivalence of ELPs. We further investigate several use cases of strong equivalence by using our technique to syntactically characterize tautological rules and rule subsumptions.

Contributions. The main contributions of this paper are the following[2]:

- We propose different notions of strong equivalence of ELPs (based on the input language of ELP solvers) that strictly generalize strong equivalence for plain ASP.
- We provide a model-theoretic characterization of strong equivalence for ELPs showing that the different notions proposed coincide.
- We use our characterization to show that, surprisingly, testing strong equivalence of two ELPs remains in coNP, that is, the complexity of this test does not increase when considering ELPs instead of plain ASP.
- Finally, we use our proposed notion to syntactically characterize tautological ELP rules and when one ELP rule subsumes another.

2 Preliminaries

Answer Set Programming (ASP). A *ground logic program* with nested negation (also called answer set program, ASP program, or, simply, logic program) is a pair $\Pi = (\mathcal{A}, \mathcal{R})$, where $\mathcal{A}$ is a set of propositional (i.e. ground) atoms and $\mathcal{R}$ is a set of rules of the form

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \neg\ell_1, \ldots, \neg\ell_n; \qquad (3)$$

where the comma symbol stands for conjunction, $0 \leq l \leq m$, $0 \leq n$, $a_i \in \mathcal{A}$ for all $1 \leq i \leq m$, and each ℓ_i is a *literal*, that is, either an atom a or its (default) negation $\neg a$ for any atom $a \in \mathcal{A}$. Note that, therefore, doubly negated atoms may occur. We will sometimes refer to such rules as *standard rules*. Each rule $r \in \mathcal{R}$ of form (3) consists of a *head* $H(r) = \{a_1, \ldots, a_l\}$ and a *body* $B(r) = \{a_{l+1}, \ldots, a_m, \neg\ell_1, \ldots, \neg\ell_n\}$. We denote the *positive* body by $B^+(r) = \{a_{l+1}, \ldots, a_m\}$.

An *interpretation* I (over $\mathcal{A}$) is a set of atoms, that is, $I \subseteq \mathcal{A}$. A literal ℓ is true in an interpretation $I \subseteq \mathcal{A}$, denoted $I \vDash \ell$, if $a \in I$ and $\ell = a$, or if $a \notin I$ and $\ell = \neg a$; otherwise ℓ is false in I, denoted $I \nvDash \ell$. Finally, for some literal ℓ, we define that $I \vDash \neg\ell$ if $I \nvDash \ell$. This notation naturally extends to sets of literals. An interpretation M is called a *model* of r, denoted $M \vDash r$, if, whenever $M \vDash B(r)$, it holds that $M \vDash H(r)$. We denote the set of models of r by $mods(r)$; the models of a logic program $\Pi = (\mathcal{A}, \mathcal{R})$ are given by $mods(\Pi) = \bigcap_{r \in \mathcal{R}} mods(r)$. We also write $I \vDash r$ (resp. $I \vDash \Pi$) if $I \in mods(r)$ (resp. $I \in mods(\Pi)$).

The GL-reduct Π^I of a logic program $\Pi = (\mathcal{A}, \mathcal{R})$ with respect to an interpretation I is the program $(\mathcal{A}, \mathcal{R}^I)$, where $\mathcal{R}^I = \{H(r) \leftarrow B^+(r) \mid r \in \mathcal{R}, \forall \neg\ell \in B(r) : I \vDash \neg\ell\}$.

Definition 1. *(Gelfond and Lifschitz 1988; 1991; Lifschitz, Tang, and Turner 1999)* $M \subseteq \mathcal{A}$ *is an* answer set *of a logic program* Π *if (1)* $M \in mods(\Pi)$ *and (2) there is no subset* $M' \subset M$ *such that* $M' \in mods(\Pi^M)$.

The set of answer sets of a logic program Π is denoted by $AS(\Pi)$. The *consistency problem* of ASP, that is, to decide whether for a given logic program Π it holds that $AS(\Pi) \neq \emptyset$, is Σ_P^2-complete (Eiter and Gottlob 1995), and remains so also in the case where doubly negated atoms are allowed in rule bodies (Pearce, Tompits, and Woltran 2009).

Strong Equivalence for Logic Programs. Two logic programs $\Pi_1 = (\mathcal{A}, \mathcal{R}_1)$ and $\Pi_2 = (\mathcal{A}, \mathcal{R}_2)$ are *equivalent* iff they have the same set of answer sets, that is, $AS(\Pi_1) = AS(\Pi_2)$. The two logic programs are *strongly equivalent* iff for any third logic program $\Pi = (\mathcal{A}, \mathcal{R})$ it holds that the combined logic program $\Pi_1 \cup \Pi = (\mathcal{A}, \mathcal{R}_1 \cup \mathcal{R})$ is equivalent to the combined logic program $\Pi_2 \cup \Pi = (\mathcal{A}, \mathcal{R}_2 \cup \mathcal{R})$. Note that strong equivalence implies equivalence, since the empty program $\Pi = (\mathcal{A}, \emptyset)$ would already contradict strong equivalence for two non-equivalent programs Π_1 and Π_2.

An *SE-model* (Turner 2003) of a logic program $\Pi = (\mathcal{A}, \mathcal{R})$ is a tuple (X, Y), where $X \subseteq Y \subseteq \mathcal{A}$, $Y \vDash \Pi$, and $X \vDash \Pi^Y$. The set of SE-models of a logic program Π is denoted $\mathcal{SE}(\Pi)$. Note that for every model Y of Π, (Y, Y) is an SE-model of Π, since $Y \vDash \Pi$ implies that $Y \vDash \Pi^Y$.

Two logic programs (over the same atoms) are strongly equivalent iff they have the same SE-models (Lifschitz, Pearce, and Valverde 2001; Turner 2003). Checking whether two logic programs are strongly equivalent is coNP-complete (Turner 2003; Pearce, Tompits, and Woltran 2009).

[2] An extended version, containing full proofs where absent, can be found here: https://arxiv.org/abs/1811.04800.

Epistemic Logic Programs. An *epistemic literal* is a formula $\mathbf{not}\,\ell$, where ℓ is a literal and $\mathbf{not}$ is the epistemic negation operator. A *ground epistemic logic program (ELP)* is a triple $\Pi = (\mathcal{A}, \mathcal{E}, \mathcal{R})$, where $\mathcal{A}$ is a set of propositional atoms, $\mathcal{E}$ is a set of epistemic literals over the atoms $\mathcal{A}$, and $\mathcal{R}$ is a set of *ELP rules*, which are

$$a_1 \vee \cdots \vee a_k \leftarrow \ell_1, \ldots, \ell_m, \xi_1, \ldots, \xi_j, \neg\xi_{j+1}, \ldots, \neg\xi_n,$$

where each $a_i \in \mathcal{A}$ is an atom, each ℓ_i is a literal, and each $\xi_i \in \mathcal{E}$ is an epistemic literal. Note that usually $\mathcal{E}$ is defined implicitly to be the set of all epistemic literals appearing in the rules $\mathcal{R}$; however, making the domain of epistemic literals explicit will prove useful for our purposes. The *union* of two ELPs $\Pi_1 = (\mathcal{A}_1, \mathcal{E}_1, \mathcal{R}_1)$ and $\Pi_2 = (\mathcal{A}_2, \mathcal{E}_2, \mathcal{R}_2)$ is the ELP $\Pi_1 \cup \Pi_2 = (\mathcal{A}_1 \cup \mathcal{A}_2, \mathcal{E}_1 \cup \mathcal{E}_2, \mathcal{R}_1 \cup \mathcal{R}_2)$.

For a set $\mathcal{E}$ of epistemic literals, a subset $\Phi \subseteq \mathcal{E}$ of epistemic literals is called an *epistemic guess* (or, simply, a *guess*). The following definition provides a way to check whether a set of interpretations is compatible with a guess Φ.

Definition 2. *Let $\mathcal{A}$ be a set of atoms, $\mathcal{E}$ be a set of epistemic literals over $\mathcal{A}$, and $\Phi \subseteq \mathcal{E}$ be an epistemic guess. A set $\mathcal{I}$ of interpretations over $\mathcal{A}$ is called* Φ-compatible w.r.t. *$\mathcal{E}$, iff*

1. *$\mathcal{I} \neq \emptyset$;*
2. *for each epistemic literal $\mathbf{not}\,\ell \in \Phi$, there exists an interpretation $I \in \mathcal{I}$ such that $I \not\models \ell$; and*
3. *for each epistemic literal $\mathbf{not}\,\ell \in \mathcal{E} \setminus \Phi$, for all interpretations $I \in \mathcal{I}$ it holds that $I \models \ell$.*

For an ELP $\Pi = (\mathcal{A}, \mathcal{E}, \mathcal{R})$, the *epistemic reduct* (Shen and Eiter 2016) of the program Π w.r.t. a guess Φ, denoted Π^Φ, consists of the rules $\mathcal{R}^\Phi = \{r^\neg \mid r \in \mathcal{R}\}$, where $r^\neg$ is defined as the rule $r \in \mathcal{R}$ where all occurrences of epistemic literals $\mathbf{not}\,\ell \in \Phi$ are replaced by $\top$, and all remaining epistemic negation symbols $\mathbf{not}$ are replaced by default negation $\neg$. Note that, after this transformation, $\Pi^\Phi = (\mathcal{A}, \mathcal{R}^\Phi)$ is a logic program without epistemic negation[3]. This leads to the following, central definition.

Definition 3. *Let $\Pi = (\mathcal{A}, \mathcal{E}, \mathcal{R})$ be an ELP. A set $\mathcal{M}$ of interpretations over $\mathcal{A}$ is a* candidate world view (CWV) *of Π if there is an epistemic guess $\Phi \subseteq \mathcal{E}$ such that $\mathcal{M} = AS(\Pi^\Phi)$ and $\mathcal{M}$ is compatible with Φ w.r.t. $\mathcal{E}$. The set of all CWVs of an ELP Π is denoted by $cwv(\Pi)$.*

Let us reconsider the CWA formulations as examples.

Example 4. *Let $\mathcal{A}_C = \{p, p'\}$, $\mathcal{E}_C = \{\mathbf{not}\,p, \mathbf{not}\,\neg p\}$, $\Pi_G = (\mathcal{A}_C, \mathcal{E}_C, \mathcal{R}_G)$ with $\mathcal{R}_G$ containing only rule (1), and $\Pi_S = (\mathcal{A}_C, \mathcal{E}_C, \mathcal{R}_S)$ with $\mathcal{R}_S$ containing only rule (2).*

We obtain $cwv(\Pi_G) = \{\{\{p'\}\}\}$ as guess $\Phi = \{\mathbf{not}\,p\}$ is compatible with $AS(\Pi_G^\Phi = \{p' \leftarrow \neg p\}) = \{\{p'\}\}$, while no other guesses are compatible with the answer sets of the respective epistemic reducts. We also obtain $cwv(\Pi_S) = \{\{\{p'\}\}\}$ as Φ is compatible with $AS(\Pi_S^\Phi = \{p' \leftarrow \top\}) = \{\{p'\}\}$, while no other guesses are compatible with the answer sets of the respective epistemic reducts.

[3] In fact, Π^Φ may contain triple-negated atoms $\neg\neg\neg a$. But, according to the definition of the GL-reduct in (Lifschitz, Tang, and Turner 1999), such formulas are equivalent to simple negated atoms $\neg a$, and we treat them as such.

Following the principle of knowledge minimization, Shen and Eiter (2016) define a *world view* as follows.

Definition 5. *Let $\Pi = (\mathcal{A}, \mathcal{E}, \mathcal{R})$ be an ELP. $\mathcal{C} \in cwv(\Pi)$ is called* world view (WV) *of Π if its associated guess Φ is subset-maximal, i.e. there is no $\mathcal{C}' \in cwv(\Pi)$ with associated guess $\Phi' \supset \Phi$.*

Note that in Example 4 there is only one CWV per program; hence the associated guesses are subset-maximal, and the sets of CWVs and WVs coincide.

One of the main reasoning tasks regarding ELPs is the *world view existence problem*, that is, given an ELP Π, decide whether a WV (or, equivalently, CWV) exists. This problem is Σ_3^P-complete (Shen and Eiter 2016).

We close this section with a statement that shows that extending $\mathcal{A}$ or $\mathcal{E}$ of an ELP does not change their CWVs (and hence also not their WVs).

Theorem 6. *Let $\Pi = (\mathcal{A}, \mathcal{E}, \mathcal{R})$ be an ELP and let $\Pi' = (\mathcal{A}', \mathcal{E}', \mathcal{R})$ be an ELP with the same set of rules, but with $\mathcal{A}' \supset \mathcal{A}$ and $\mathcal{E}' \supset \mathcal{E}$. Then, $cwv(\Pi) = cwv(\Pi')$.*

While not difficult to see, one issue that needs closer examination is when $\mathcal{E}$ is extended with an epistemic literal whose atom is already in $\mathcal{R}$, but it can be shown that no additional CWV can be created and that no CWV can become invalid either. This also implies that, given two ELPs $\Pi_1 = (\mathcal{A}_1, \mathcal{E}_1, \mathcal{R}_1)$ and $\Pi_2 = (\mathcal{A}_2, \mathcal{E}_2, \mathcal{R}_2)$, we can always assume that $\mathcal{A}_1 = \mathcal{A}_2$ and $\mathcal{E}_1 = \mathcal{E}_2$ (since the domains can be extended without changing the CWVs, as per the above theorem).

3 Strong Equivalence for ELPs

In this section, we will investigate notions of equivalence of ELPs, in particular, focusing on how to extend the concept of strong equivalence (Lifschitz, Pearce, and Valverde 2001; Turner 2003) from logic programs to ELPs. We will start by defining (ordinary) equivalence of two ELPs.

Definition 7. *Two ELPs are* WV-equivalent *(resp.* CWV-equivalent*) iff their world views (resp. candidate world views) coincide.*

We observe that CWV-equivalence is the stronger notion, as it immediately implies WV-equivalence. Moreover, for two ELPs to be equivalent as defined above, not only must the set of guesses leading to WVs/CWVs be the same, but also the answer sets in each of these WVs/CWVs.

We now continue by defining strong equivalence for ELPs. One motivation for such a kind of equivalence is modularization: we want to be able to replace a sub-program by another one such that the semantics (i.e. WVs or CWVs) of the overall program does not change. Based on the two equivalence notions defined above and using ideas from work done in the area of logic programs (Eiter et al. 2007), we propose four notions of strong equivalence for ELPs.

Definition 8. *Let Π_1 and Π_2 be two ELPs. Π_1 and Π_2 are*

- strongly ELP-WV-equivalent *iff, for every ELP Π, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are WV-equivalent;*
- strongly ASP-WV-equivalent *iff, for every (plain) logic program Π, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are WV-equivalent;*

- strongly ELP-CWV-equivalent *iff, for every ELP* Π, $\Pi_1 \cup \Pi$ *and* $\Pi_2 \cup \Pi$ *are CWV-equivalent;*
- strongly ASP-CWV-equivalent *iff, for every (plain) logic program* Π, $\Pi_1 \cup \Pi$ *and* $\Pi_2 \cup \Pi$ *are CWV-equivalent.*

Having defined these equivalence notions for ELPs, the main aim of this section is to characterize strong equivalence in a similar fashion as was done for logic programs by Turner (2003). Note that one could be tempted to define strong equivalence for ELPs simply in terms of Turner's SE-models of the epistemic reducts, for each possible epistemic guess. However, this approach does not capture our intentions, as the following example shows:

Example 9. *Take the two ELPs* $\Pi_1 = (\mathcal{A}, \mathcal{E}, \mathcal{R}_1)$ *and* $\Pi_2 = (\mathcal{A}, \mathcal{E}, \mathcal{R}_2)$ *with* $\mathcal{R}_1 = \{p \leftarrow \mathbf{not}\, p\}$, $\mathcal{R}_2 = \{p \leftarrow \neg p\}$ *and* $\mathcal{E} = \{\mathbf{not}\, p\}$. *Now, for the guess* $\Phi_1 = \emptyset$, *note that* $\Pi_1^{\Phi_1} = \Pi_2^{\Phi_1}$ *and hence, trivially, the SE-models are also the same. However, for the guess* $\Phi_2 = \mathcal{E}$, $\Pi_1^{\Phi_2}$ *consists of the rule* $p \leftarrow \top$, *while* $\Pi_2^{\Phi_2}$ *reduces to* $p \leftarrow \neg p$. *It can be checked that the SE-models of these two epistemic reducts w.r.t.* Φ_2 *are not the same, hence these two programs are not strongly equivalent in the sense of (Turner 2003). However, it turns out that the guess* Φ_2 *can never give rise to a CWV, since it requires that there is an answer set not containing* p, *but both* $\Pi_1^{\Phi_2}$ *and* $\Pi_2^{\Phi_2}$ *require that* p *is true in all answer sets of the CWV. Hence, it seems that the epistemic reducts for* Φ_2 *should not have any bearing on evaluating strong equivalence.*

The example above implies that, when establishing strong equivalence for ELPs, we should discard guesses that can never give rise to a CWV. We formalize this as follows:

Definition 10. *Let* $\mathcal{I}$ *be a set of interpretations over a domain of atoms* $\mathcal{A}$, *let* $\mathcal{E}$ *be a set of epistemic literals over* $\mathcal{A}$, *and* $\Phi \subseteq \mathcal{E}$ *be a guess. Then,* Φ *is* realizable *in* $\mathcal{I}$ *iff there is a subset* $\mathcal{I}' \subseteq \mathcal{I}$ *such that* $\mathcal{I}'$ *is compatible with* Φ *w.r.t.* $\mathcal{E}$.

Given an ELP $\Pi = (\mathcal{A}, \mathcal{E}, \mathcal{R})$ and a guess $\Phi \subseteq \mathcal{E}$, we say that Φ is *realizable in* Π iff Φ is realizable in the set of models of Π^Φ. We say that Φ is *realizable in a set of SE-models* $\mathcal{S}$ iff Φ is realizable in the set $\{Y \mid (X, Y) \in \mathcal{S}\}$. We are now ready to define our central construct, the SE-function $\mathcal{SE}_\Pi$ of an ELP, which will be the key concept to characterize strong equivalence for ELPs. Note that realizability plays an important role in this.

Definition 11. *The* SE-function $\mathcal{SE}_\Pi(\cdot)$ *of an ELP* $\Pi = (\mathcal{A}, \mathcal{E}, \mathcal{R})$ *maps guesses* $\Phi \subseteq \mathcal{E}$ *for* Π *to sets of SE-models as follows.*

$$\mathcal{SE}_\Pi(\Phi) = \begin{cases} \mathcal{SE}(\Pi^\Phi) & \text{if } \Phi \text{ realizable in } \Pi \\ \emptyset & \text{otherwise.} \end{cases}$$

Example 12. *Recall that programs* Π_G *and* Π_S *of Example 4 are both CWV- and WV-equivalent. However, we find that their respective SE-functions differ:* $\mathcal{SE}_{\Pi_G}(\{\mathbf{not}\, p\})$ *contains the tuple* $(\emptyset, \{p\})$, *but* $\mathcal{SE}_{\Pi_S}(\{\mathbf{not}\, p\})$ *does not; yet* $\{\mathbf{not}\, p\}$ *is realizable in both* Π_G *and* Π_S.

Note that if Φ is realizable in Π, then $\{Y \mid (X, Y) \in \mathcal{SE}_\Pi(\Phi)\} = mods(\Pi^\Phi)$. Before proceeding to our main results, we first state some observations that can be made about the SE-function of an ELP.

Lemma 13. *Let* $\Pi = (\mathcal{A}, \mathcal{E}, \mathcal{R})$ *be an ELP with* $\mathcal{SE}_\Pi$ *its SE-function. Further, let* $\mathcal{M}$ *be a set of interpretations, and* $\Phi \subseteq \mathcal{E}$ *be a guess. Then,* $\mathcal{M}$ *is a CWV of* Π *w.r.t.* Φ *iff* $\mathcal{M} = \{Y \mid (Y, Y) \in \mathcal{SE}_\Pi(\Phi), \neg\exists X \subset Y\ (X, Y) \in \mathcal{SE}_\Pi(\Phi)\}$ *and* $\mathcal{M}$ *is compatible with* Φ.

Proof. The left-to-right direction can be shown as follows. By definition of the SE-function it holds that for each CWV $\mathcal{M}$ w.r.t. a guess Φ for Π, $\mathcal{SE}_\Pi(\Phi)$ contains the set $\{(Y, Y) \mid Y \in \mathcal{M}\}$, and, since $\mathcal{M}$ contains only answer sets of the epistemic reduct Π^Φ, there cannot be a pair (X, Y) in $\mathcal{SE}_\Pi(\Phi)$ with $X \subset Y$. There also cannot be some other pair $(Y', Y') \in \mathcal{SE}_\Pi(\Phi)$ for which no pair (X', Y') with $X' \subset Y'$ exists, since then this would mean that Y' is an answer set of Π^Φ, and therefore must be in $\mathcal{M}$. Finally, since by assumption $\mathcal{M}$ is a CWV, it is compatible with Φ by definition.

For the right-to-left direction, note that $\mathcal{M}$ clearly contains all those sets of atoms Y that are models of Π^Φ, such that there is no subset of Y model of the GL-reduct $[\Pi^\Phi]^Y$. Hence, $\mathcal{M}$ contains precisely the answer sets of Π^Φ. Since, by assumption, $\mathcal{M}$ is compatible with Φ, this immediately implies that $\mathcal{M}$ is a CWV of Π as per Definition 3. □

The next statement is a direct consequence of the previous lemma, since, as we have seen, the SE-function of an ELP defines its CWVs.

Lemma 14. *Programs with the same SE-function are CWV-equivalent (and hence WV-equivalent).*

We are now ready to state the main result of this section, namely that the SE-function precisely characterizes strong equivalence for ELPs.

Theorem 15. *Let* Π_1 *and* Π_2 *be two ELPs. The following statements are equivalent:*

1. Π_1 *and* Π_2 *are ELP-CWV-equivalent;*
2. Π_1 *and* Π_2 *are ASP-CWV-equivalent;*
3. Π_1 *and* Π_2 *are ELP-WV-equivalent;*
4. Π_1 *and* Π_2 *are ASP-WV-equivalent; and*
5. $\mathcal{SE}_{\Pi_1} = \mathcal{SE}_{\Pi_2}$.

Proof. $(1) \Rightarrow (2) \Rightarrow (4), (1) \Rightarrow (3) \Rightarrow (4)$**.** These follow directly from Definition 8 and from the fact that every WV is a CWV and every ASP program is an ELP.

$(5) \Rightarrow (1)$**.** Assume that statement (5) holds. We need to show that for any third program Π it holds that $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are equivalent. To this end, pick any guess Φ. Then,

$$\mathcal{SE}_{\Pi_1 \cup \Pi}(\Phi) =$$

$$\left\{ \begin{array}{ll} \mathcal{SE}_{\Pi_1}(\Phi) \cap \mathcal{SE}_\Pi(\Phi) & \text{if } \Phi \text{ is realizable in } \\ & \mathcal{SE}_{\Pi_1}(\Phi) \cap \mathcal{SE}_\Pi(\Phi) \\ \emptyset & \text{otherwise} \end{array} \right\}$$

$$= \mathcal{SE}_{\Pi_2 \cup \Pi}(\Phi),$$

since $\mathcal{SE}_{\Pi_1}(\Phi) = \mathcal{SE}_{\Pi_2}(\Phi)$ by assumption, and thus $\mathcal{SE}_{\Pi_1}(\Phi) \cap \mathcal{SE}_\Pi(\Phi) = \mathcal{SE}_{\Pi_2}(\Phi) \cap \mathcal{SE}_\Pi(\Phi)$. Lemma 14 then proves that $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are equivalent.

(4) $\Rightarrow$ (5)**.** We will prove the contrapositive. Let $\Pi_1 = (\mathcal{A}_1, \mathcal{E}_1, \mathcal{R}_1)$ and $\Pi_2 = (\mathcal{A}_2, \mathcal{E}_2, \mathcal{R}_2)$ be two ELPs and assume, w.l.o.g., that $\mathcal{A}_1 = \mathcal{A}_2 = \mathcal{A}$ and $\mathcal{E}_1 = \mathcal{E}_2 = \mathcal{E}$ (cf. Theorem 6). Further, let $\Phi \subseteq \mathcal{E}$ be a guess such that $\mathcal{Y}_1 = \mathcal{SE}_{\Pi_1}(\Phi) \neq \mathcal{SE}_{\Pi_2}(\Phi) = \mathcal{Y}_2$. Finally, assume that there is a pair (X, Y) in $\mathcal{Y}_1$ but not in $\mathcal{Y}_2$ (again, w.l.o.g., by symmetry). We need to show that there exists a logic program Π (i.e. without epistemic literals) such that the WVs of the ELP $\Pi_1 \cup \Pi$ differ from those of $\Pi_2 \cup \Pi$. We only need to consider the case where Π_1 and Π_2 are WV-equivalent, since the claim is trivially true otherwise.

By Definition 11, with $\mathcal{Y}_1$ non-empty by assumption, there is a subset $\mathcal{C} \subseteq \{Y \mid (X, Y) \in \mathcal{Y}_1\}$ compatible with Φ, since Φ is realizable in Π. Let $\mathcal{C} = \{Y_1, \ldots, Y_m\}$ and let $\{Y_{m+1}, \ldots, Y_n\} = 2^{\mathcal{A}} \setminus \mathcal{C}$.

The idea is to construct Π in such a way that the potential WV represented by $\mathcal{C}$ is actually realized in $\Pi_1 \cup \Pi$. To construct Π, let $y_1, \ldots, y_n$ be fresh atoms not occurring in $\mathcal{A}$. Let Π contain the rule $y_1 \vee \ldots \vee y_n \leftarrow \top$, and, furthermore, for all $1 \leq i \leq n$ and $a \in Y_i$, the rule $\bot \leftarrow y_i, \neg a$, and for all $a \in \mathcal{A} \setminus Y_i$, the rule $\bot \leftarrow y_i, a$. This makes sure that for every model Y_i of Π_1^Φ, the corresponding model Y_i' of $(\Pi_1 \cup \Pi)^\Phi$ contains the atom y_i (i.e. $Y_i' = Y_i \cup \{y_i\}$).

Now, take the pair (X, Y) that is, by assumption, contained in $\mathcal{Y}_1$ (but not in $\mathcal{Y}_2$). Clearly, there is some integer k, $1 \leq k \leq n$, such that $Y = Y_k$. Now, for each model Y_i, $1 \leq i \leq m$, $i \neq k$, and each atom $a \in Y_i$, add the rule $a \leftarrow y_i$ to Π. For each model Y_i, $m < i \leq n$, $i \neq k$, add the rule $\bot \leftarrow y_i$ to Π. This makes sure that, in $(\Pi_1 \cup \Pi)^\Phi$, exactly the models from $\mathcal{C}$ (except Y_k, if $k \leq m$) become answer sets, and all other models are destroyed, except for the model $Y = Y_k$, if $k > m$.

At this point, if we have that $\{Y_1, \ldots, Y_m\} \not\subseteq mods(\Pi_2^\Phi)$, we simply do the same as above also for the model $Y = Y_k$ (i.e. realize Y_k as an answer set if $k \leq m$, or destroy Y_k in case $k > m$). Then, clearly, $(\Pi_1 \cup \Pi)^\Phi$ will have the answer sets $\{Y_1 \cup \{y_1\}, \ldots, Y_m \cup \{y_m\}\}$ which form a CWV of $\Pi_1 \cup \Pi$, but not of $\Pi_2 \cup \Pi$. Since all other models are destroyed, independent of the guess Φ, this CWV is actually the only CWV of $\Pi_1 \cup \Pi$, and hence, it is a WV, proving our claim that $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are not WV-equivalent ($\star$). It therefore remains to show the claim for the case where the set $\{Y_1, \ldots, Y_m\} \subseteq mods(\Pi_2^\Phi)$. To this end, we need to distinguish the following two cases:

Case (1). Assume that $Y \not\models \Pi_2^\Phi$. In this case, for each atom $a \in Y$, add the rule $a \leftarrow y_k$ to Π. Now, $Y \cup \{y_k\}$ is an answer set of $(\Pi_1 \cup \Pi)^\Phi$, but not of $(\Pi_2 \cup \Pi)^\Phi = \Pi_2^\Phi \cup \Pi$.

Case (2). Assume that $Y \models \Pi_2^\Phi$. In this case, for each atom $a \in X$, add the rule $a \leftarrow y_k$ to Π, and, in addition, for all atoms $a, b \in Y \setminus X$, add the rule $a \leftarrow b, y_k$ to Π. We will show that, in this case, Y is an answer set of $(\Pi_2 \cup \Pi)^\Phi$, but not of $(\Pi_1 \cup \Pi)^\Phi$. Since $Y \models \Pi_2^\Phi$ and every model of a program is also a model of its GL-reduct, by definition of SE-models we know that $Y \neq X$, since, by assumption, $(X, Y) \in \mathcal{Y}_1$ but $(X, Y) \notin \mathcal{Y}_2$. Since $(X, Y) \in \mathcal{Y}_1$, we have that $X \models [\Pi_1^\Phi]^Y$. But then, by construction of Π it holds that $X \cup \{y_k\} \models [(\Pi_1 \cup \Pi)^\Phi]^Y = [\Pi_1^\Phi]^Y \cup [\Pi^\Phi]^Y$ and therefore $Y \cup \{y_k\}$ is not an answer set of $(\Pi_1 \cup \Pi)^\Phi$. On the other hand, for $(\Pi_2 \cup \Pi)^\Phi$, assume that there is some $X' \subset Y$ such that $X' \cup \{y_k\} \models [(\Pi_2 \cup \Pi)^\Phi]^Y$. Clearly, by construction of Π, $X \subset X'$. Thus, there is some atom a in $X' \subseteq Y$ but not in X. But, by construction of Π, we then have that $X' = Y$. Hence, $Y \cup \{y_k\}$ is an answer set of $(\Pi_2 \cup \Pi)^\Phi$, as desired.

The above shows that, for both cases (1) and (2), the set $\mathcal{C}' = \{Y_1 \cup \{y_1\}, \ldots, Y_m \cup \{y_m\}\}$ is the set of answer sets of one of the two programs $(\Pi_1 \cup \Pi)^\Phi$ and $(\Pi_2 \cup \Pi)^\Phi$, and therefore, by assumption, a CWV of that program. But, by construction of Π, $\mathcal{C}'$ cannot be a CWV for the other program (because $Y \cup \{y_k\}$ is an answer set of one of the two programs, but not both, and hence distinguishes the CWVs)[4]. It remains to show that $\mathcal{C}'$ is not only a CWV but also a WV of exactly one of the two programs, which can be done via the argument for ($\star$). This concludes the proof. □

The above theorem states that the SE-function precisely characterizes all the notions of strong equivalence for ELPs and that these notions are all equivalent. We therefore will, from now on, jointly refer to these four equivalent notions as *strong equivalence*. The next statement follows immediately.

Corollary 16. *ELPs are strongly equivalent iff they have the same SE-function.*

Example 17. *Continuing from Example 12, we observe that the difference in the SE-function can be made manifest by combining the two programs with the program* $\Pi = (\mathcal{A}_C, \mathcal{E}_C, \mathcal{R})$*, where* $\mathcal{R} = \{p \leftarrow \neg p'\}$*. We observe that* $cwv(\Pi_G \cup \Pi) = \{\{\{p\}\}\}$ *(due to guess* $\{\mathbf{not}\, p\}$*) and* $cwv(\Pi_S \cup \Pi) = \{\{\{p'\}\}\}$ *(due to guess* $\{\mathbf{not}\, \neg p\}$*).*

We close this section by showing that our definition of strong equivalence for ELPs generalizes the established notion of strong equivalence for logic programs.

Corollary 18. *Let* $\Pi_1 = (\mathcal{A}, \mathcal{R}_1)$ *and* $\Pi_2 = (\mathcal{A}, \mathcal{R}_2)$ *be two logic programs, and let* $\Pi_1' = (\mathcal{A}, \emptyset, \mathcal{R}_1)$ *and* $\Pi_2' = (\mathcal{A}, \emptyset, \mathcal{R}_2)$ *be two ELPs with the same sets of rules as* Π_1 *and* Π_2*, respectively. Then,* Π_1 *and* Π_2 *are strongly equivalent iff* Π_1' *and* Π_2' *are.*

Proof. (Sketch) For Π_1' and Π_2' there is only one possible guess, namely, $\Phi = \emptyset$. Moreover, in this setting it holds that $\mathcal{SE}_\Pi(\emptyset) = \mathcal{SE}(\Pi)$ for any ELP Π: the crucical observation is that $\mathcal{SE}(\Pi)$ is compatible with Φ w.r.t. the empty domain of epistemic literals, exactly if $\mathcal{SE}(\Pi) \neq \emptyset$. □

4 Complexity of ELP Strong Equivalence

We now make use of our characterization to settle the computational complexity of deciding strong equivalence between ELPs. The following lemma shows that to check the realizability of a guess Φ for a given ELP Π it suffices to consider a polynomially-sized subset of the models of Π.

Lemma 19. *Let* $\Pi = (\mathcal{A}, \mathcal{E}, \mathcal{R})$ *be an ELP,* $\Phi \subseteq \mathcal{E}$ *a guess for* Π*, and* $\mathcal{Y} = \mathcal{SE}_\Pi(\Phi)$ *be non-empty. Then, there is a set* $\mathcal{C} \subseteq \{Y \mid (X, Y) \in \mathcal{Y}\}$ *of polynomial size in* Π *that is compatible with* Φ *w.r.t.* $\mathcal{E}$*.*

[4]Note that, in particular, in case (1), $\mathcal{C}'$ is a CWV of $\Pi_1 \cup \Pi$ but not of $\Pi_2 \cup \Pi$, and in case (2), the same holds if $k \leq m$, and the reverse holds if $k > m$.

Proof. According to Definition 11, since Φ is realizable in Π, there is a subset $\mathcal{Y}' \subseteq \mathcal{Y}$, such that the set $\mathcal{C}' = \{Y \mid (X, Y) \in \mathcal{Y}'\}$ is compatible with Φ. By Definition 2, for each $\mathbf{not}\,\ell \in \mathcal{E}$, when Φ contains $\mathbf{not}\,\ell$, there is a $Y_{\mathbf{not}\,\ell} \in \mathcal{C}'$ such that ℓ is false in $Y_{\mathbf{not}\,\ell}$. Take the subset $\mathcal{C} \subseteq \mathcal{C}'$ containing $Y_{\mathbf{not}\,\ell}$ for each $\mathbf{not}\,\ell \in \mathcal{E}$, or, if $\Phi = \emptyset$, let $\mathcal{C}$ be any singleton subset of $\mathcal{C}'$. Note that $\mathcal{C}$ is of polynomial size in Π. Clearly, $\mathcal{C}$ is also compatible with Φ. □

With this crucial observation in place, we are now ready to state the complexity of checking strong equivalence for ELPs, which remains in coNP as for plain logic programs. This is surprising, since reasoning tasks for ELPs generally are one level higher on the polynomial hierarchy than the corresponding tasks for logic programs (cf. Section 2).

Theorem 20. *Checking whether two ELPs are strongly equivalent is* coNP*-complete.*

Proof. We give a non-deterministic polynomial time procedure for checking that two ELPs are *not* strongly equivalent, that is, that there is a difference in their respective SE-functions. W.l.o.g. let $\Pi_1 = (\mathcal{A}, \mathcal{E}, \mathcal{R}_1)$ and $\Pi_2 = (\mathcal{A}, \mathcal{E}, \mathcal{R}_2)$. The procedure works as follows:

1. Guess an epistemic guess $\Phi \subseteq \mathcal{E}$.
2. Guess a polynomially-sized, non-empty set of interpretations $\mathcal{C}$ over $\mathcal{A}$, compatible with Φ.
3. Verify in polynomial time that each $Y \in \mathcal{C}$ is a model of both Π_1^Φ and Π_2^Φ. If not, REJECT.
4. Guess a pair (X, Y) with $X \subseteq Y \subseteq \mathcal{A}$.
5. Verify in polynomial time that $(X, Y) \in \mathcal{SE}_{\Pi_1}(\Phi)$ but $(X, Y) \notin \mathcal{SE}_{\Pi_2}(\Phi)$ or vice versa. If so, ACCEPT. If not, REJECT.

Clearly, the above procedure terminates in polynomial time, since it is known that model checking for SE-models can be done in polynomial time (Eiter et al. 2007). To obtain correctness, it is not difficult to verify that the above procedure is sound and complete given the following two observations. Firstly, note that Lemma 19 implies that we only need to focus on polynomially sized subsets when evaluating the realizability of a guess as stated in Definition 11; hence guessing a polynomially sized set of interpretations is enough in step 2. Secondly, assume that two programs Π_1 and Π_2 have differing SE-functions. Then this means that there must be some guess Φ, such that both sets $\mathcal{SE}_{\Pi_1}(\Phi)$ and $\mathcal{SE}_{\Pi_2}(\Phi)$ are non-empty (clearly, as otherwise $\mathcal{SE}_{\Pi_1}(\Phi) = \mathcal{SE}_{\Pi_2}(\Phi) = \emptyset$ for all guesses Φ). But, by Definition 11, this means that both sets contain a potential CWV. Now there are two cases: either they do not share any potential CWVs, or if they do, then there is at least one SE-model (X, Y) contained in one but not both of the two sets.

Corollary 18 allows us to inherit the coNP lower bound from the case of logic programs (Pearce, Tompits, and Woltran 2009), which completes the proof. □

5 Case Studies

In this section, we apply our characterisation to investigate basic principles for simplifying ELPs.

Tautological Rules

Tautological rules are rules that can simply be removed from any program without affecting its outcome.

Definition 21. *An ELP rule r is* tautological *iff the single-rule ELP $\Pi_r = (\mathcal{A}, \mathcal{E}, \{r\})$ is strongly equivalent to the empty program $\Pi_\emptyset = (\mathcal{A}, \mathcal{E}, \emptyset)$.*

Let $\mathcal{E}$ be a set of epistemic literals over $\mathcal{A}$. We say that an epistemic guess $\Phi \subseteq \mathcal{E}$ is *consistent* iff, whenever $\mathcal{E}$ contains both $\mathbf{not}\,a$ and $\mathbf{not}\,\neg a$ for some atom $a \in \mathcal{A}$, it holds that Φ contains at least one of $\{\mathbf{not}\,a, \mathbf{not}\,\neg a\}$.[5] Moreover, let $\mathcal{S}_\mathcal{A}$ denote the set of all pairs (X, Y) such that $X \subseteq Y \subseteq \mathcal{A}$.

Lemma 22. *An ELP rule r is tautological iff for the single-rule ELP $\Pi_r = (\mathcal{A}, \mathcal{E}, \{r\})$ it holds that $\mathcal{SE}_{\Pi_r}(\Phi) = \mathcal{S}_\mathcal{A}$ for each consistent guess $\Phi \subseteq \mathcal{E}$.*

Proof. This follows from the fact that for the empty program $\Pi_\emptyset = (\mathcal{A}, \mathcal{E}, \emptyset)$, $\mathcal{SE}_{\Pi_\emptyset}(\Phi) = \mathcal{S}_\mathcal{A}$ for each consistent guess $\Phi \subseteq \mathcal{E}$, and the observation that if $\mathcal{SE}(\Pi^\Phi) = \mathcal{S}_\mathcal{A}$, then Φ is realizable in Π. □

Before syntactically characterizing tautological ELP rules, we recall a corresponding result for standard ASP rules from the literature. For convenience, we shall denote ASP rules of the form (3) as

$$A \leftarrow B, \neg C, \neg\neg D, \tag{4}$$

using sets of atoms A, B, C, and D as is common practice. In what follows, we denote single-rule logic programs consisting of a rule r by $\Pi_r = (\mathcal{A}, \mathcal{R})$ with $\mathcal{R} = \{r\}$ and call r tautological if Π_r is strongly equivalent to the empty program $\Pi = (\mathcal{A}, \emptyset)$, i.e. $\mathcal{SE}(\Pi_r) = S_\mathcal{A}$. The following result is due to (Cabalar, Pearce, and Valverde 2007, Lemma 2).

Lemma 23. *A rule r of the form (4) is tautological iff (α) $A \cap B \neq \emptyset$, (β) $B \cap C \neq \emptyset$, or (γ) $C \cap D \neq \emptyset$.*

We are now ready to characterize tautological ELP rules. For the sake of presentation let us write them as

$$A \leftarrow B, \neg C, \mathbf{not}\,D, \mathbf{not}\,\neg E, \neg\mathbf{not}\,F, \neg\mathbf{not}\,\neg G, \tag{5}$$

where, again, each capital letter represents a set of atoms, analogously to ASP rules of the form (4). Note that D here plays a different role than in (4); this is due to the fact that in ELP rules we have no explicit double negation.

Let us also consider the notion of epistemic reduct w.r.t. the above notation. Analogously to the ASP case, let $\Pi_r = (\mathcal{A}, \mathcal{E}, \mathcal{R})$ be the single-rule ELP with $\mathcal{R} = \{r\}$. Now, for $\Phi \subseteq \mathcal{E}$, we have $\Pi_r^\Phi = (\mathcal{A}, \mathcal{E}, \mathcal{R}^\Phi)$ with $\mathcal{R}^\Phi = \emptyset$ if $\mathbf{not}\,f \in \Phi$ for some $f \in F$ or if $\mathbf{not}\,\neg g \in \Phi$ for $g \in G$; otherwise,

$$\mathcal{R}^\Phi = \{r^\Phi\} = \{A \leftarrow B, \neg C, \neg D^\Phi, \neg\neg E^\Phi, \neg\neg F, \neg G\}$$

with $D^\Phi = \{d \in D, \mathbf{not}\,d \notin \Phi\}$, $E^\Phi = \{e \in E, \mathbf{not}\,\neg e \notin \Phi\}$. We are now ready to give our full syntactical characterization of tautological ELP rules. It shows that deciding whether a rule is tautological amounts to a simple syntactic check and can be done individually, for each rule.

[5] Clearly, if Φ contains neither of the two epistemic literals, then Φ can never be realizable in any ELP, since there is no (non-empty) set of models where a is both always true and always false.

Theorem 24. *An ELP rule r of form (5) is tautological iff (a) $A \cap B \neq \emptyset$, (b) $B \cap (C \cup G) \neq \emptyset$, (c) $C \cap F \neq \emptyset$, (d) $D \cap F \neq \emptyset$, (e) $E \cap G \neq \emptyset$, or (f) $F \cap G \neq \emptyset$.*

Proof. Let $\Pi_r = (\mathcal{A}, \mathcal{E}, \mathcal{R})$ be the single-rule ELP with $\mathcal{R} = \{r\}$, and, for an epistemic guess Φ, let r^Φ denote the unqiue rule in $\mathcal{R}^\Phi$ in case $\mathcal{R}^\Phi \neq \emptyset$.

($\Rightarrow$) Let r be an ELP rule satisfying at least one of the conditions (a)–(f), and let $\Phi \subseteq \mathcal{E}$ be a consistent guess. By Lemma 22, we have to show that either $\mathcal{R}^\Phi = \emptyset$ or that r^Φ fulfills the conditions of Lemma 23. This can be easily verified for conditions (a)–(c). For (d), note that if $\mathbf{not}\, f \in \Phi$ for some $f \in F$ we get $\mathcal{R}^\Phi = \emptyset$; otherwise for all $f \in F$ it holds that $\mathbf{not}\, f \notin \Phi$, and thus that $D \cap F \neq \emptyset$, and it follows that $f \in D^\Phi$ for some $f \in F$. Hence we have $\neg f$ and $\neg\neg f$ in r^Φ, which is tautological as per Lemma 23, condition (γ). The argument is similar for (e), Finally, for (f) note that since Φ is consistent, it must contain one of $\mathbf{not}\, a$ and $\mathbf{not}\, \neg a$ for each $a \in F \cap G$. It follows that $\mathcal{R}^\Phi = \emptyset$.

($\Leftarrow$) Let r be an ELP rule such that none of (a)–(f) hold. Let $\Phi = \mathcal{E} \setminus (\mathbf{not}\, F \cup \mathbf{not}\, \neg G)$. Φ is consistent since (f) does not hold. Moreover, $\mathbf{not}\, D \cup \mathbf{not}\, \neg E \subseteq \Phi$ (since neither (d) nor (e) holds). Suppose, Φ is realizable in Π_r. By construction of Φ, $\mathcal{R}^\Phi = \{A \leftarrow B, \neg C, \neg\neg F, \neg G\}$; by Lemma 23 and the fact that (α)–(γ) of that lemma do not hold for r^Φ, we have that r^Φ is not tautological, i.e. $\mathcal{SE}(\Pi_{r^\Phi}) \neq S_\mathcal{A}$, and thus $\mathcal{SE}_{\Pi_r}(\Phi) \neq \mathcal{S}_\mathcal{A}$. Otherwise, we get $\mathcal{SE}_{\Pi_r}(\Phi) = \emptyset$. Hence, for both cases, r is not tautological by Lemma 22. □

Rule Subsumption

Rule subsumption identifies when a rule s can safely be removed from a program Π, given another rule r is in Π.

Definition 25. *An ELP rule s is* subsumed *by an ELP rule r iff Π_r is strongly equivalent to $\Pi_r \cup \Pi_s$.*

The next result follows from the definition of the SE-function for a union of two ELPs (cf. proof of Theorem 15).

Lemma 26. *An ELP rule s is subsumed by an ELP rule r iff, for ELPs $\Pi_r = (\mathcal{A}, \mathcal{E}, \{r\})$ and $\Pi_s = (\mathcal{A}, \mathcal{E}, \{s\})$, it holds that $\mathcal{SE}_{\Pi_r}(\Phi) \subseteq \mathcal{SE}_{\Pi_s}(\Phi)$, for all guesses $\Phi \subseteq \mathcal{E}$.*

Clearly, a tautological rule s is subsumed by any other rule, hence in what follows we focus on subsumption of non-tautological rules only. Again, we exploit known results from ASP. With some abuse of terminology, we say that an ASP rule r subsumes another such rule s iff $\Pi_r = (\mathcal{A}, \{r\})$ is strongly equivalent to $\Pi_{r,s} = (\mathcal{A}, \{r, s\})$, i.e. iff $\mathcal{SE}(\Pi_r) \subseteq \mathcal{SE}(\Pi_s)$. We first adapt a result from (Cabalar, Pearce, and Valverde 2007, Theorem 7) to our notation.

Lemma 27. *An ASP rule $r = A \leftarrow B, \neg C, \neg\neg D$ subsumes a non-tautological ASP rule $s = A' \leftarrow B', \neg C', \neg\neg D'$ iff the following conditions jointly hold: (α) $A \subseteq A' \cup C'$, (β) $B \subseteq B' \cup D'$, (β') if $A \cap (A' \setminus C') \neq \emptyset$ then $B \subseteq B'$, (γ) $C \subseteq C'$, and (δ) $D \subseteq B' \cup D'$.*

We can now give a syntactic criterion for ELP rule subsumption, which turns out to be somewhat involved, but still feasible to check. It requires two technical notions that link a rule r to a rule s whenever r has sufficiently many elements that are not "absorbed" by default-negated epistemic literals in s.

Theorem 28. *Let r be an ELP-rule of form (5) and $s = A' \leftarrow B', \neg C', \mathbf{not}\, D', \mathbf{not}\, \neg E', \neg\mathbf{not}\, F', \neg\mathbf{not}\, \neg G'$ be non-tautological. Furthermore, let $r \rhd s$ denote that $|(A \cup C \cup D) \setminus G'| > 1$ or $(B \cup E) \setminus F' \neq \emptyset$, and $r \blacktriangleright s$ denote that $(A \cup C \cup D) \setminus G' \neq \emptyset$ or $|(B \cup E) \setminus F'| > 1$.*
Then, r subsumes s iff the following conditions jointly hold:

- *(a)* $A \subseteq A' \cup C' \cup D' \cup G'$;
- *(a*)* *if $r \rhd s$ then* $A \subseteq A' \cup C' \cup G'$;
- *(b)* $B \subseteq B' \cup E' \cup F'$;
- *(b*)* *if $r \blacktriangleright s$ then* $B \subseteq B' \cup F'$;
- *(b')* *if $A \cap (A' \setminus (C' \cup D' \cup G')) \neq \emptyset$ then* $B \subseteq B'$;
- *(b*')* *if $r \rhd s$ and $A \cap (A' \setminus (C' \cup G')) \neq \emptyset$ then* $B \subseteq B'$;
- *(c)* $C \cup D \subseteq C' \cup D' \cup G'$;
- *(c*)* *if $r \rhd s$ then* $C \subseteq C' \cup G'$;
- *(d)* $E \subseteq B' \cup E' \cup F'$;
- *(e)* $F \subseteq F'$ *and* $G \subseteq G'$.

Note that, in the above, items (a)–(d) generalize the same items in Lemma 27, and the proof for those parts is a generalization of the proof of that lemma. The full proof is rather tedious and relies on a number of case distinctions for each of the conditions. We omit it here due to space reasons.

Interestingly, applying the above theorem shows, for example, that $r = p \leftarrow \mathbf{not}\, p$ subsumes $s = p \leftarrow \neg p$ and vice versa (in particular, since neither $r \rhd s$ nor $s \rhd r$), showing that the two programs in Example 9 are strongly equivalent.

6 Conclusions

In this paper, a simple characterization of strong equivalence for epistemic logic programs was proposed, which also demonstrates that various notions of strong equivalence coincide. The characterization generalizes strong equivalence for plain logic programs, and, somewhat unexpectedly, shows that checking strong equivalence for ELPs is not harder than for ASP in terms of computational complexity. The results also give rise to a syntactic characterization of tautological ELP rules and ELP rule subsumption.

As another byproduct, we studied the relationship between two formalizations of CWA, Gelfond-CWA and Shen-Eiter-CWA, as our running example. Indeed, while they are (ordinarily) equivalent, they are not strongly equivalent, as shown in Examples 12 and 17. In particular, Example 17 shows that, combining Π in that example with the Shen-Eiter-CWA rule, yields the world view $\{\{p'\}\}$, which does not seem intuitive in this setting. However, in (Shen and Eiter 2016), it seems that the CWA rule is proposed for Reiter's CWA (Reiter 1977), and thus was not intended to work with rules containing default negation.

For future work, we want to apply our findings to obtain a normal form for ELPs, as was done for ASP in (Cabalar, Pearce, and Valverde 2007). Furthermore, we plan to study weaker forms of equivalence (Eiter, Fink, and Woltran 2007) for ELPs. In particular, it will be interesting to see whether our notion of SE-functions can be similarly re-used for characterizing uniform equivalence like SE-models did serve as a basis for UE-models.

Acknowledgements

Michael Morak and Stefan Woltran were supported by the Austrian Science Fund (FWF) under grant number Y698.

References

Bichler, M.; Morak, M.; and Woltran, S. 2018. Single-shot epistemic logic program solving. In Lang, J., ed., *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, 1714–1720. ijcai.org.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12):92–103.

Cabalar, P.; Pearce, D.; and Valverde, A. 2007. Minimal logic programs. In Dahl, V., and Niemelä, I., eds., *Logic Programming, 23rd International Conference, ICLP 2007, Porto, Portugal, September 8-13, 2007, Proceedings*, volume 4670 of *Lecture Notes in Computer Science*, 104–118. Springer.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.* 15(3-4):289–323.

Eiter, T.; Faber, W.; Fink, M.; and Woltran, S. 2007. Complexity results for answer set programming with bounded predicate arities and implications. *Ann. Math. Artif. Intell.* 51(2-4):123–165.

Eiter, T.; Fink, M.; Pührer, J.; Tompits, H.; and Woltran, S. 2013. Model-based recasting in answer-set programming. *Journal of Applied Non-Classical Logics* 23(1-2):75–104.

Eiter, T.; Fink, M.; and Woltran, S. 2007. Semantical characterizations and complexity of equivalences in answer set programming. *ACM Trans. Comput. Log.* 8(3).

Fariñas del Cerro, L.; Herzig, A.; and Su, E. I. 2015. Epistemic equilibrium logic. In Yang, Q., and Wooldridge, M., eds., *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 2964–2970. AAAI Press.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K. A., eds., *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, 1070–1080. MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Comput.* 9(3/4):365–386.

Gelfond, M. 1991. Strong introspection. In Dean, T. L., and McKeown, K. R., eds., *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 1*, 386–391. AAAI Press / The MIT Press.

Gelfond, M. 1994. Logic programming and reasoning with incomplete information. *Ann. Math. Artif. Intell.* 12(1-2):89–116.

Gelfond, M. 2011. New semantics for epistemic specifications. In Delgrande, J. P., and Faber, W., eds., *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, 260–265. Springer.

Kahl, P. T.; Watson, R.; Balai, E.; Gelfond, M.; and Zhang, Y. 2015. The language of epistemic specifications (refined) including a prototype solver. *J. Log. Comput.* 25.

Kahl, P. T. 2014. *Refining the Semantics for Epistemic Logic Programs*. Ph.D. Dissertation, Texas Tech University, Texas, USA.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Trans. Comput. Log.* 2(4):526–541.

Lifschitz, V.; Tang, L. R.; and Turner, H. 1999. Nested expressions in logic programs. *Ann. Math. Artif. Intell.* 25(3-4):369–389.

Lin, F., and Chen, Y. 2007. Discovering classes of strongly equivalent logic programs. *J. Artif. Intell. Res.* 28:431–451.

Pearce, D.; Tompits, H.; and Woltran, S. 2009. Characterising equilibrium logic and nested logic programs: Reductions and complexity. *TPLP* 9(5):565–616.

Reiter, R. 1977. On closed world data bases. In *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977.*, 55–76.

Schaub, T., and Woltran, S. 2018. Special issue on answer set programming. *KI* 32(2-3).

Shen, Y., and Eiter, T. 2016. Evaluating epistemic negation in answer set programming. *Artif. Intell.* 237:115–135.

Son, T. C.; Le, T.; Kahl, P. T.; and Leclerc, A. P. 2017. On computing world views of epistemic logic programs. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 1269–1275. ijcai.org.

Truszczynski, M. 2011. Revisiting epistemic specifications. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*.

Turner, H. 2003. Strong equivalence made easy: nested expressions and weight constraints. *TPLP* 3(4-5):609–622.

Wang, K., and Zhang, Y. 2005. Nested epistemic logic programs. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings*, volume 3662 of *LNCS*, 279–290. Springer.