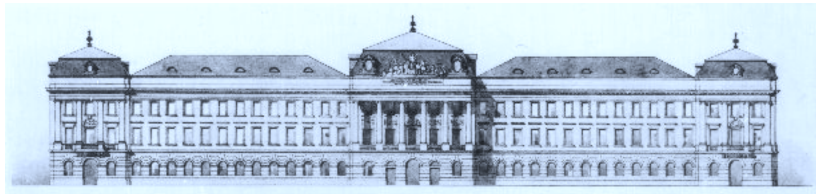
LOGCOMP

RESEARCH

REPORT



INSTITUT FÜR LOGIC AND COMPUTATION

FACHBEREICH WISSENSBASIERTE SYSTEME

INLINING EXTERNAL SOURCES IN ANSWER SET PROGRAMS

CHRISTOPH REDL

LOGCOMP RESEARCH REPORT 18-01

JANUARY 2018

Institut für Logic and Computation
FB Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

LOGCOMP RESEARCH REPORT

LOGCOMP RESEARCH REPORT 18-01, JANUARY 2018

INLINING EXTERNAL SOURCES IN ANSWER SET PROGRAMS

Christoph Redl[1]

Abstract. HEX-programs are an extension of answer set programming (ASP) towards external sources. To this end, *external atoms* provide a bidirectional interface between the program and an external source. The traditional evaluation algorithm for HEX-programs is based on guessing truth values of external atoms and verifying them by explicit calls of the external source. The approach was optimized by techniques that reduce the number of necessary verification calls or speed them up, but the remaining external calls are still expensive. In this paper we present an alternative evaluation approach based on *inlining* of external atoms, motivated by existing but less general approaches for specialized formalisms such as DL-programs. External atoms are then compiled away such that no verification calls are necessary. The approach is implemented in the dlvhex reasoner. Experiments show a significant performance gain. Besides performance improvements, we further exploit inlining for extending previous notions of programs equivalence from ASP to HEX-programs, including *strong equivalence*, *uniform equivalence* and $\langle\mathcal{H},\mathcal{B}\rangle$*-equivalence*. Based on this extended equivalence notion we finally characterize also inconsistency of programs wrt. extensions. Since well-known ASP extensions (such as constraint ASP) amount to special cases of HEX, the results are interesting beyond the particular formalism.

[1]Institut für Logic and Computation, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria; email: redl@kr.tuwien.ac.at.

Acknowledgements: This article is an extension of preliminary work presented at AAAI 2017 (Redl 2017b; Redl 2017c). This work has been supported by the Austrian Science Fund (FWF) Grant P27730.

Contents

1 Introduction

HEX-programs are an extension of answer set programming (ASP) (Gelfond and Lifschitz 1991) towards external sources. As ASP, HEX-programs are based on nonmonotonic programs and have a multi-model semantics. External sources are used to represent knowledge and computation sources such as, for instance, description logic ontologies and Web resources. To this end, so-called *external atoms* are used to send information from the logic program to an external source, which returns values to the program. Cyclic rules that involve external atoms are allowed, such that recursive data exchange between the program and external sources is possible. Moreover, *value invention* allows for returning values which are not contained in the input program, i.e., which expand the domain. A concrete example is the external atom $\&edge[G](X, Y)$ which returns for a filename G, pointing to a file which stores a graph, the contained edges (X, Y).

The traditional evaluation procedure for HEX-programs is based on rewriting external atoms to ordinary atoms and guessing their truth values. This yields answer set candidates, which are subsequently checked to ensure that the guessed values coincide with the actual semantics of the external atoms. Furthermore, an additional minimality check is necessary to exclude self-justified atoms, which involves even more external calls. Although this approach has been refined by integrating advanced techniques for learning (Eiter et al. 2012) and efficient minimality checking (Eiter et al. 2014), which tightly integrate the solver with the external sources and reduce the number of external calls, the remaining calls are still expensive. In addition to the complexity of the external sources themselves, also overhead on the implementation side, such as calls of external libraries and cache misses after jumps out of core algorithms, may decrease efficiency compared to ordinary ASP-programs.

In this paper we present a **novel method for HEX-program evaluation based on inlining of external atoms**. In contrast to existing approaches for DL-programs (Heymans et al. 2010; Xiao and Eiter 2011; Bajraktari et al. 2017), ours is generic and can be applied to arbitrary external sources. Therefore, it is interesting beyond HEX-programs and also applicable to specialized formalisms such as constraint ASP (Gebser et al. 2009; Ostrowski and Schaub 2012). The approach uses *support sets* (cf. e.g. Darwiche and Marquis (2002)), i.e., sets of literals which define assignments of input atoms that guarantee that an external atom is true. Support sets were previously exploited for HEX-program evaluation (Eiter et al. 2014); however, this was only for speeding up but not for eliminating the necessary verification step. In contrast, our new approach compiles external atoms away altogether such that there are no guesses at all which need to be verified. i.e., the semantics of external atoms is embedded in the ASP-program. We use a **benchmark suite to show significant performance improvements for certain classes of external atoms**.

Next, we have a look at equivalence notions for ASP such as *strong equivalence* (Lifschitz et al. 2001), *uniform equivalence* (Eiter and Fink 2003) and the more general notion of $\langle \mathcal{H}, \mathcal{B} \rangle$-equivalence (Woltran 2008); all these notions identify programs as equivalent also wrt. program extensions. Equivalence notions have received quite some attention and in fact have also been developed for other formalisms such as abstract argumentation (Baumann et al. 2017). Thus it is a natural goal to also **extend equivalence notions from ordinary ASP- to HEX-programs** (and again, also special cases thereof), which turns out to be possible based on our inlining approach. We are able to show that equivalence can be characterized similarly as for ordinary ASP-programs; while this is convenient, due to the support for external atoms and the use of the FLP-reduct (Faber et al. 2011) instead of the GL-reduct (Gelfond and Lifschitz 1988) in the semantics of HEX-programs, this result is not immediate. Based on the extended equivalence notion, we further derive a **characterization of inconsistency of a program wrt. program extensions**, which we call *persistent inconsistency*. While the main results are decision criteria based on programs and their reducts, we further derive a criterion for checking persistent inconsistency based on unfounded sets, which are – due to the fact that implementations usually do not explicitly construct the reduct – convenient with view on applications, which we also discuss.

To summarize the **main contributions**, we present

1. **a technique for external source inlining** and three applications thereof, namely
2. **a new evaluation technique for HEX-programs**,
3. **a generalization of equivalence notions from ASP- to HEX-programs**, and
4. **a novel notion of inconsistency of HEX-programs wrt. program extensions**.

After the preliminaries in Section 2 we proceed as follows:

- In Section 3 we show how external atoms can be *inlined* (embedded) into a program. To handle nonmonotonicity we use a *saturation encoding* based on *support sets*. For the sake of a simpler presentation we first restrict the discussion to positive external atoms and then extend our approach to handle also negated ones.
- In Section 4 we exploit this approach for performance gains. To this end, we implement the approach in the dlvhex system and perform an experimental evaluation, which shows a significant speedup for certain classes of external atoms. The speedup is both over traditional evaluation and over a previous approach based on support sets for guess verification.
- Based on this inlining approach, Section 5 characterizes equivalence of HEX-programs, which generalizes results by Woltran (2008). The generalizations of strong (Lifschitz et al. 2001) and uniform equivalence (Eiter and Fink 2003) correspond to special cases thereof.
- In Section 6 we present a characterization of *inconsistency of* HEX*-programs wrt. program extensions*, which we call *persistent inconsistency*. This characterization is derived from the previously presented notion of equivalence. We then discuss an application of the criteria in context of potential further improvements of the evaluation algorithm.
- Section 7 discusses related work and concludes the paper.
- Proofs are outsourced to Appendix A.

A preliminary version of the results in this paper has been presented at AAAI 2017 (Redl 2017b; Redl 2017c); the extensions in this work comprise of more extensive discussions of the theoretical contributions, additional experiments and formal proofs of the results.

2 Preliminaries

We start with basic concepts. Our alphabet consists of possibly infinite, mutually disjoint sets of constant symbols $\mathcal{C}$, predicate symbols $\mathcal{P}$, and external predicates $\mathcal{X}$; in this paper we resign from using variables in the formal part, as will be justified below.

In the following, a (ground) atom a is of form $p(c_1, \dots, c_\ell)$ with predicate $p \in \mathcal{P}$ and constant symbols $c_1, \dots, c_\ell \in \mathcal{C}$, abbreviated as $p(\mathbf{c})$; we write $c \in \mathbf{c}$ if $c = c_i$ for some $1 \leq i \leq \ell$. For $\ell = 0$ we might drop the parentheses and write $p()$ simply as p. An *assignment* Y over the (finite) set A of atoms is a set $Y \subseteq A$, where $a \in Y$ expresses that a is true under Y, also denoted $Y \models a$, and $a \notin Y$ that a is false, also denoted $Y \not\models a$. For a *default-literal* $\mathit{not}\, a$ over an atom a we let $Y \models \mathit{not}\, a$ if $Y \not\models a$ and $Y \not\models \mathit{not}\, a$ otherwise.

HEX-Programs. We recall HEX-programs (Eiter et al. 2016), which generalize (disjunctive) logic programs under the answer set semantics (Gelfond and Lifschitz 1991), as follows.

Syntax. HEX-programs extend ordinary ASP-programs by *external atoms* which provide a bidirectional interface between the program and external sources. A *ground external atom* is of the form $\&g[\mathbf{p}](\mathbf{c})$, where $\&g \in \mathcal{X}$ is an external predicate, $\mathbf{p} = p_1, \ldots, p_k$ is a list of input parameters (predicates from $\mathcal{P}$ or object constants from $\mathcal{C}$), called *input list*, and $\mathbf{c} = c_1, \ldots, c_l$ are output constants from $\mathcal{C}$.

Definition 1 *A HEX-program P consists of rules*

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \mathit{not}\ b_{m+1}, \ldots, \mathit{not}\ b_n \ ,$$

where each a_i is an atom and each b_j is either an ordinary atom or an external atom.

For such a rule r, its *head* is $H(r) = \{a_1, \ldots, a_k\}$, its *body* is $B(r) = \{b_1, \ldots, b_m, \mathit{not}\ b_{m+1}, \ldots, \mathit{not}\ b_n\}$, its *positive body* is $B^+(r) = \{b_1, \ldots, b_m\}$ and its *negative body* is $B^-(r) = \{b_{m+1}, \ldots, b_n\}$. For a program P we let $X(P) = \bigcup_{r \in P} X(r)$ for $X \in \{H, B, B^+, B^-\}$.

For a program P and a set of constants $\mathcal{C}$, let $HB_{\mathcal{C}}(P)$ denote the *Herbrand base* containing all atoms constructible from the predicates occurring in P and constants $\mathcal{C}$.

We restrict the formal discussion to programs without variables as suitable safety conditions guarantee the existence of a finite grounding which suffices for answer set computation, see e.g. Eiter et al. (2016).

Semantics. In the following, assignments are over the set $A(P)$ of ordinary atoms that occur in the program P at hand. The semantics of an external atom $\&g[\mathbf{p}](\mathbf{c})$ wrt. an assignment Y is given by the value of a decidable $1{+}k{+}l$-ary *two-valued (Boolean) oracle function* $f_{\&g}$ that is defined for all possible values of Y, $\mathbf{p}$ and $\mathbf{c}$. We say that $\&g[\mathbf{p}](\mathbf{c})$ is true relative to Y if $f_{\&g}(Y, \mathbf{p}, \mathbf{c}) = \mathbf{T}$, and it is false otherwise. We make the restriction that for a fixed assignment Y and input list $\mathbf{p}$, $f_{\&g}(Y, \mathbf{p}, \mathbf{c}) = \mathbf{T}$ holds only for finitely many different vectors $\mathbf{c}$. Satisfaction of ordinary rules and ASP-programs (Gelfond and Lifschitz 1991) is then extended to HEX-rules and programs as follows. A rule r as by Definition 1 is true under Y, denoted $Y \models r$, if $Y \models h$ for some $h \in H(r)$ or $Y \not\models b$ for some $b \in B(r)$.

The answer sets of a HEX-program P are defined as follows. Let the *FLP-reduct* of P wrt. an assignment Y be the set $fP^Y = \{r \in P \mid Y \models b \text{ for all } b \in B(r)\}$. Then:

Definition 2 *An assignment Y is an answer set of a HEX-program P, if Y is a subset-minimal model of fP^Y.*

Example 1 *Consider the program $P = \{p \leftarrow \&id[p]()\}$, where $\&id[p]()$ is true iff p is true. Then P has the answer set $Y_1 = \emptyset$; indeed it is a subset-minimal model of $fP^{Y_1} = \emptyset$.*

For an ordinary program P, the above definition of answer sets is equivalent to Gelfond & Lifschitz' answer sets.

Traditional Evaluation Approach. A HEX-programs P is transformed to an ordinary ASP-program $\hat{P}$ as follows. Each external atom $\&g[\mathbf{p}](\mathbf{c})$ in P is replaced by an ordinary *replacement atom* $e_{\&g[\mathbf{p}]}(\mathbf{c})$ and a rule $e_{\&g[\mathbf{p}]}(\mathbf{c}) \vee ne_{\&g[\mathbf{p}]}(\mathbf{c}) \leftarrow$ is added. The answer sets of the resulting *guessing program* $\hat{P}$ are computed by an ASP solver. However, the assignment Y extracted from an answer set $\hat{Y}$ of $\hat{P}$ by projecting it to the ordinary atoms $A(P)$ in P may not satisfy P as $\&g[\mathbf{p}](\mathbf{c})$ under $f_{\&g}$ may differ from the guessed value of $e_{\&g[\mathbf{p}]}(\mathbf{c})$. The answer set is merely a *candidate*. If a compatibility check against the external source succeeds, it is a *compatible set* as formalized as follows:

Definition 3 *A* compatible set *of a program P is an answer set $\hat{Y}$ of the guessing program $\hat{P}$ such that $f_{\&g}(\hat{Y}, \mathbf{p}, \mathbf{c}) = \mathbf{T}$ iff $e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \hat{Y}$ for all external atoms $\&g[\mathbf{p}](\mathbf{c})$ in P.*

Example 2 *Consider $P = \{\, p(a) \vee p(b) \leftarrow \&atMostOne[p]() \,\}$, where $\&atMostOne[p]()$ is true under an assignment Y if $\{p(a), p(b)\} \not\subseteq Y$, i.e., at most one of $p(a)$ or $p(b)$ is true under Y, and it is false otherwise. Then we have $\hat{P} = \{p(a) \vee p(b) \leftarrow e_{\&atMostOne[p]};\ e_{\&atMostOne[p]} \vee ne_{\&atMostOne[p]} \leftarrow\}$, which has the answer sets $\hat{Y_1} = \{p(a), e_{\&atMostOne[p]}\}$, $\hat{Y_2} = \{p(b), e_{\&atMostOne[p]}\}$, $\hat{Y_3} = \{ne_{\&atMostOne[p]}\}$ (while $\{p(a), p(b), e_{\&atMostOne[p]}\}$ is not an answer set of $\hat{P}$). However, although $\hat{Y_3}$ is an answer set of $\hat{P}$, its projection $Y_3 = \emptyset$ to atoms $A(P)$ in P is not an answer set of P because $Y_3 \models \&atMostOne[p]()$ but $e_{\&atMostOne[p]} \notin \hat{Y_3}$, and thus the compatibility check for $\hat{Y_3}$ fails. In contrast, the compatibility checks for $\hat{Y_1}$ and $\hat{Y_2}$ pass, i.e., they are compatible sets of P, and their projections $Y_1 = \{p(a)\}$ and $Y_2 = \{p(b)\}$ to atoms $A(P)$ in P are answer sets of P.*

However, if the compatibility check succeeds, the projected interpretation is not always automatically an answer set of the original program. Instead, after the compatibility check of an answer set $\hat{Y}$ of P was passed, another final check is needed to guarantee also subset-minimality of its projection Y wrt. fP^Y. Each answer set Y of P is the projection of some compatible set $\hat{Y}$ to $A(P)$, but not vice versa.

Example 3 *Reconsider $P = \{\, p \leftarrow \&id[p]() \,\}$ from above. Then $\hat{P} = \{p \leftarrow e_{\&id[p]}();\ e_{\&id[p]} \vee ne_{\&id[p]} \leftarrow\}$ has the answer sets $\hat{Y_1} = \{ne_{\&id[p]}\}$ and $\hat{Y_2} = \{p, e_{\&id[p]}\}$. Here, $Y_1 = \emptyset$ is a $\subseteq$-minimal model of $fP^{Y_1} = \emptyset$, but $Y_2 = \{p\}$ not of $fP^{Y_2} = P$.*

There are several approaches for checking this minimality, e.g. based on *unfounded sets*, which are sets of atoms that support each other only cyclically (Faber 2005). However, the details of this check are not relevant for this paper, which is why we refer the interested reader to Eiter et al. (2014) for a discussion and evaluation of various approaches.

Learning Techniques. In practice, the guessing program $\hat{P}$ has usually many answer sets, but many of them fail the compatibility check against external sources (often because of the same wrong guess), which turns out to be an evaluation bottleneck. To overcome the problem, techniques which extend *conflict-driven learning* have been introduced as *external behavior learning (EBL)* (Eiter et al. 2012).

As in ordinary ASP solving, the traditional HEX-algorithm translates the guessing program to a set of *nogoods*, i.e., a set of literals which must not be true at the same time. Given this representation, techniques from SAT solving are applied to find an assignment which satisfies all nogoods (Gebser et al. 2012). Notably, as the encoding as a set of nogoods is of exponential size due to *loop nogoods* which avoid cyclic justifications of atoms, those parts are generated only on-the-fly. Moreover, additional nogoods are learned from conflict situations, i.e., violated nogoods which cause the solver to backtrack; this is called *conflict-driven nogood learning*, see e.g. (Franco and Martin 2009).

EBL extends this algorithm by learning additional nogoods not only from conflict situations in the ordinary part, but also from verification calls to external sources. Whenever an external atom $e_{\&e[\mathbf{p}]}(\mathbf{c})$ is evaluated under an assignment Y for the sake of compatibility checking, the actual truth value under the assignment becomes evident. Then, regardless of whether the guessed value was correct or not, one can add a nogood which represents that $e_{\&e[\mathbf{p}]}(\mathbf{c})$ must be true under Y if $Y \models \&e[\mathbf{p}](\mathbf{c})$ or that $e_{\&e[\mathbf{p}]}(\mathbf{c})$ must be false under Y if $Y \not\models \&e[\mathbf{p}](\mathbf{c})$. If the guess was incorrect, the newly learned nogood will trigger backtracking, if the guess was correct, the learned nogood will prevent future wrong guesses.

Example 4 *Suppose $\&atMostOne[p]()$ is evaluated under $Y = \{p(a), p(b)\}$. Then the real truth value of $\&atMostOne[p]()$ under Y becomes evident: in this case $Y \not\models \&atMostOne[p]()$. One can then learn the nogood $\{p(a), p(b), e_{\&atMostOne[p]}()\}$ to represent that $p(a)$, $p(b)$ and $\&atMostOne[p]()$ cannot be true at the same time.*

Learning realizes a tight coupling of the reasoner and the external source by adding parts of the semantics on-demand to the program instance, which is similar to theory propagation in SMT (see e.g. Nieuwenhuis and Oliveras (2005)). Experimental results show that this technique leads to a significant, up to exponential speedup, which is explained by the exclusion of up to exponentially many guesses by the learned nogoods, but the remaining verification calls are still expensive and – depending on the type of the external source – can account for large parts of the overall runtime (Eiter et al. 2014).

Evaluation Based on Support Sets. Later, an alternative evaluation approach was developed. While the basic idea of guessing the values of external atoms as in the traditional approach remains, the verification is now accomplished by using so-called *support sets* instead of explicit evaluation (Eiter et al. 2014). Here, a *positive resp. negative support set* for an external atom e is a set of literals over the input atoms of e whose satisfaction implies satisfaction resp. falsification of e. Informally, the verification is done by checking if the answer set candidate matches with a support set of the external atom. If this is the case, the guess is verified resp. falsified.

More precisely, for a set S of literals a or $\neg a$, where a is an atom, let $\neg S = \{\neg a \mid a \in S\} \cup \{a \mid \neg a \in S\}$ be the set of literals S with swapped sign. We formalize support sets as follows:

Definition 4 (Support Set) *Let $e = \&g[\mathbf{y}](\mathbf{x})$ be an external atom in a program P. A* support set *for e is a consistent set $S_\sigma = S_\sigma^+ \cup S_\sigma^-$ with $\sigma \in \{\mathbf{T}, \mathbf{F}\}$, $S_\sigma^+ \subseteq HB_{\mathcal{C}}(P)$, and $S_\sigma^- \subseteq \neg HB_{\mathcal{C}}(P)$ s.t. $Y \supseteq S_\sigma^+$ and $Y \cap \neg S_\sigma^- = \emptyset$ implies $Y \models e$ if $\sigma = \mathbf{T}$ and $Y \not\models e$ if $\sigma = \mathbf{F}$ for all assignments Y.*

We call the support set S_σ *positive* if $\sigma = \mathbf{T}$ and *negative* if $\sigma = \mathbf{F}$.

Example 5 *Suppose $\&diff[p, q](c)$ computes the set of all elements c which are in the extension of p but not in that of q. Then $\{p(a), \neg q(a)\}$ is a positive support set for $\&diff[p, q](a)$ because any assignment Y with $\{p(a)\} \subseteq Y$ but $Y \cap \{q(a)\} = \emptyset$ satisfies $\&diff[p, q](a)$.*

We are in particular interested in *families (=sets) of support sets* which describe the behavior of external atoms completely:

Definition 5 ((Complete) Support Set Family) *A* positive resp. negative family of support sets *$\mathcal{S}_\sigma$ with $\sigma \in \{\mathbf{T}, \mathbf{F}\}$ for external atom e is a set of positive resp. negative support sets of e; $\mathcal{S}_\sigma$ is* complete *if for each assignment Y with $Y \models e$ resp. $Y \not\models e$ there is an $S_\sigma \in \mathcal{S}_\sigma$ s.t. $Y \supseteq S_\sigma^+$ and $Y \cap \neg S_\sigma^- = \emptyset$.*

Complete support set families $\mathcal{S}_\sigma$ can be used for the verification of external atoms as follows. One still uses the rewriting $\hat{P}$, but instead of explicit evaluation and comparison of the guess of a replacement atom to the actual value under the current assignment, one checks if for some $S_\sigma \in \mathcal{S}_\sigma$ we have $Y \supseteq S_\sigma^+$ and $Y \cap \neg S_\sigma^- = \emptyset$ for the current assignment Y. If this is the case, the external atom must be true if $\sigma = \mathbf{T}$

and false if $\sigma = \mathbf{F}$; otherwise, it must be false if $\sigma = \mathbf{T}$ and true if $\sigma = \mathbf{F}$. This method is in particular advantageous if the support sets in $\mathcal{S}_\sigma$ are *small and few*.

As a further improvement, positive support sets $S_\mathbf{T}$ for $\&g[\mathbf{p}](\mathbf{c})$ can be added as constraints $\leftarrow S^+_\mathbf{T}, \{not\, a \mid \neg a \in S^-_\mathbf{T}\}, not\ \&g[\mathbf{p}](\mathbf{c})$ to the program in order to exclude false negative guesses. Analogously, for negative support sets we can add $\leftarrow S^+_\mathbf{F}, \{not\, a \mid \neg a \in S^-_\mathbf{F}\}, \&g[\mathbf{p}](\mathbf{c})$ to exclude false positive guesses. This improvement was exploited in existing approaches for performance improvements (Eiter et al. 2014); we will also use this technique in Section 4 when comparing our new approach to the previous support-set-based approach. This amounts to a learning technique similar to EBL. However, note that this learns only a fixed number of nogoods at the beginning, while learning by EBL is not possible here as external sources are not evaluated during solving. Also note that the verification check is still necessary even if all $S_\sigma \in \mathcal{S}_\sigma$ are added as constraints because these constraints prevent only false negative guesses for $\sigma = \mathbf{T}$ and false positive guesses for $\sigma = \mathbf{F}$, but not vice versa.

The approach was also lifted to the non-ground level (Eiter et al. 2014). Intuitively, non-ground support sets may contain variables as shortcuts for all ground instances. Prior to the use of non-ground support sets, the variables are substituted by all relevant constants which appear in the program. However, in the following we restrict the formal discussion to the ground level for simplicity.

To summarize, improvements in the traditional evaluation approach (learning) have reduced the number of verification calls, and the alternative support set approach has replaced explicit verification calls by matching an assignment with support sets, but neither of them did eliminate the need for guessing and subsequent verification altogether. In the next section we go a step further and eliminate this need.

3 External Source Inlining

In this section we present a rewriting which compiles HEX-programs into equivalent ordinary ASP-programs (modulo auxiliary atoms) based on support sets, and thus embeds external sources into the program; we call the technique *inlining*. Due to nonmonotonic behavior of external atoms, inlining is not straightforward. In particular, it is *not* sufficient to substitute external atoms by ordinary replacement atoms and derive their truth values based on their support sets, which is surprising at first glance. Intuitively, this is because rules, which define replacement atoms, can be missing in the reduct and it is not guaranteed any longer that the replacement atoms resemble the original semantics; we will demonstrate this in more detail in Section 3.2. Afterwards we present a sound and complete encoding based on the saturation technique (cf. e.g. Eiter et al. (2009)) in Section 3.3. Obviously, in order to make use of support sets there must be procedures that can effectively and efficiently construct them, which is why we have a look at this aspect first.

3.1 Constructing Support Sets

Constructing support sets depends on the external source (Eiter et al. 2014). In general, the developer of an external atom is aware of its semantic structure, which usually allows her/him to provide this knowledge in form of support sets. Then, providing support sets can be seen as an alternative way to define and implement oracle functions. For certain classes of external atoms, procedures for constructing support sets are in fact already in place.

Compactness of families of support sets is an important aspect for our contribution. It is crucial for our contribution that, although our encoding may be exponential in the worst case due to exponentially many support sets, many realistic external sources have small support set families; for certain types of external sources, their small size is even *provable* and *known before evaluating the program*. External sources with provably small support set families include, for instance, the description logic $\mathit{DL}\text{-}\mathit{Lite}_{\mathcal{A}}$ (Calvanese et al. 2007). As the number of support sets is directly linked to the size of our rewriting, also the latter is guaranteed to be small in such cases. Generally, support set families tend to be small for sources whose behavior is structured, i.e., whose output often depends only on parts of the input and does not change completely with small changes in the input (Eiter et al. 2014). Note that such a structure in many realistic applications is also the key to parameterized complexity. In this paper, we focus on such sources; also the sources used in our benchmarks are guaranteed to have small families of support sets (whose sizes we will discuss together with the respective benchmark results).

As an example we have a closer at constructing support sets for a $\mathit{DL}\text{-}\mathit{Lite}_{\mathcal{A}}$-ontology, which is accessed from the logic program using dedicated external atoms (also called *DL-atoms* (Eiter et al. 2008)). DL-atoms allow for answering queries over the ontology under a (possibly) extended Abox based on input from the program. We use an external atom of form $\&\mathit{DL}[\mathit{ont}, \mathit{inpc}, \mathit{inpr}, \mathit{con}](X)$ to access an ontology ont and retrieve all individuals X in the concept con, where the binary resp. ternary predicates inpc and inpr allow for answering the query under the assumption that certain concept resp. role assertions are added to the Abox of the ontology before answering the query. More precisely, the query is answered wrt. an assignment Y under the assumption that concept assertion $c(i)$ is added for each $\mathit{inpc}(c, i) \in Y$. and role assertion $r(i_1, i_2)$ is added for each $\mathit{inpr}(r, i_1, i_2) \in Y$.

For instance, suppose the program contains atoms of form $\mathit{inpc}(\text{“}\mathit{Person}\text{”}, \cdot)$ to specify persons and atoms of form $\mathit{inpr}(\text{“}\mathit{childOf}\text{”}, \cdot, \cdot)$ to specify parent-child relations. Then the external atom $\&\mathit{DL}[\mathit{ont}, \mathit{inpc}, \mathit{inpr}, \text{“}\mathit{OnlyChild}\text{”}](X)$ queries all members of concept $\mathit{OnlyChild}$ under the assumption that concepts Person and rols $\mathit{childOf}$ has been extended according to the truth values of the inpc and inpr atoms in the program. [1]

For this type of description logic, Calvanese et al. (2007) have proven that at most one assertion is needed to derive an instance query from a consistent ontology. Hence, for each concept c and individual i there is a (positive) support set either of form $\emptyset$ or of form $\{p(\mathbf{x})\}$, where the latter encodes that if $p(\mathbf{x}) \in Y$, then $Y \models \&\mathit{DL}[\mathit{ont}, \mathit{inpc}, \mathit{inpr}, c](i)$ for all assignments Y. Moreover, at most two added ABox assertions are needed to make such an ontology inconsistent (in which case all queries are true). For each possibility where the ontology becomes inconsistent there is a (positive) support set of form $\{p(\mathbf{x}), p'(\mathbf{x}')\}$. Then, each support set is of one of only three different forms, which are all at most binary. Moreover, Lembo et al. (2011) have proven that the number of different constants appearing in $\mathbf{x}$ resp. $\mathbf{x}'$ in these support sets is limited by three. The limited cardinality and number of constants also limits the number of possible support sets required to describe the overall ontology to a quadratic number in the size of the program and the Abox.

Moreover, as one can see, the support sets are easy to construct by a syntactic analysis of the ontology and the DL-atoms. For details regarding the construction of support sets for $\mathit{DL}\text{-}\mathit{Lite}_{\mathcal{A}}$ we refer to Eiter et al. (2014).

3.2 Observations

We now start with observations that can be made when attempting to inline external sources in a straightforward way. The first intuitive attempt to inline an external atom e might be to replace it by an ordinary atom x_e and add rules of kind $x_e \leftarrow L$, where L is constructed from a positive support set $S_{\mathbf{T}}$ of e by adding $S^+_{\mathbf{T}}$ as positive atoms and $S^-_{\mathbf{T}}$ as default-negated ones. However, this alone is in general incorrect even if

[1]This is often written as $\mathit{DL}[\mathit{ont}; \mathit{Person} \uplus p, \mathit{childOf} \uplus c; \mathit{OnlyChild}](X)$ using a more convenient syntax tailored to DL-atoms, where additions to concepts and rules is expressed by operator $\uplus$ and p and c are unary and binary (instead of binary and ternary) predicates, respectively.

repeated for all $S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}$ for a complete family of support sets $\mathcal{S}_{\mathbf{T}}$, as the following example demonstrates.

Example 6 *Consider $P = \{a \leftarrow \&true[a]()\}$ where $e = \&true[a]()$ is always true. The program is expected to have the answer set $Y = \{a\}$. However, the translated program $P' = \{x_e \leftarrow a;\ x_e \leftarrow not\, a;\ a \leftarrow x_e\}$ has no answer set because the only candidate is $Y' = \{a, x_e\}$ and $fP'^{Y} = \{x_e \leftarrow a;\ a \leftarrow x_e\}$ has the smaller model $\emptyset$.*

In the example, P' fails to have an answer set because the former external atom $\&true[a]()$ is true also if $not\, a$ holds, but the rule $x_e \leftarrow not\, a$, which represents this case, is dropped from the reduct wrt. Y' because its body $not\, a$ is unsatisfied by Y'. Hence, although the external atom e holds both under Y' and under the smaller model $\emptyset$ of the reduct which dismisses Y', this is not detected since the representation of the external atom in the reduct is incomplete. In such a case, the value of x_e and e under a model of the reduct can differ.

An attempt to fix this problem might be to explicitly guess the value of the external atom and represent both when it is true and when it is false. Indeed, $P'' = \{x_e \vee \overline{x_e};\ \leftarrow a, not\, x_e;\ \leftarrow not\, a, not\, x_e;\ a \leftarrow x_e\}$ is a valid rewriting of the previous program (Y' is an answer set). However, this rewriting is also incorrect in general, as the next example shows.

Example 7 *Consider $P = \{a \leftarrow \&id[a]()\}$ where $e = \&id[a]()$ is true iff a is true. The program is expected to have the answer set $Y = \emptyset$. However, the translated program $P' = \{x_e \vee \overline{x_e};\ \leftarrow a, not\, x_e;\ \leftarrow not\, a, x_e;\ a \leftarrow x_e\}$ has not only the intended answer set $\{\overline{x_e}\}$ but also $Y' = \{a, x_e\}$ because $fP'^{Y'} = \{x_e \vee \overline{x_e};\ a \leftarrow x_e\}$ has no smaller model.*

While the second rewriting attempt from Example 7 works for Example 6, and, conversely, the one applied in Example 6 works for Example 7, a general rewriting schema must be more elaborated. In fact, since HEX-programs with recursive, nonmonotonic external atoms are on the second level of the polynomial hierarchy (Faber et al. 2011), such a rewriting must involve disjunctions with head-cycles.

3.3 Encoding in Disjunctive ASP

We now present such a general rewriting. In the following, for an external atom e in a program P, let $I(e, P)$ be the set of all ordinary atoms in P whose predicate occurs as a predicate parameter in e, i.e., the set of all input atoms to e. Let further $\mathcal{S}_{\mathbf{T}}(e, P)$ be an arbitrary but fixed complete positive support set family over atoms in P.

For a simpler presentation, we proceed in two steps. We first restrict the discussion to positive external atoms, and then extend the approach to negative ones in Section 3.3.2.

3.3.1 Inlining Positive External Atoms

We present the encoding for inlining single positive external atoms into a program and explain it rule by rule afterwards.

Definition 6 (External Atom Inlining) *For a HEX-program P and external atom e which occurs only positively in P, let*

$$P_{[e]} = \{x_e \leftarrow S^+_{\mathbf{T}} \cup \{\overline{a} \mid \neg a \in S^-_{\mathbf{T}}\} \mid S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}(e, P)\} \quad (1)$$

$$\cup\ \{\overline{a} \leftarrow not\, a; \overline{a} \leftarrow x_e; a \vee \overline{a} \leftarrow not\, \overline{x_e} \mid a \in I(e, P)\} \quad (2)$$

$$\cup\ \{\overline{x_e} \leftarrow not\, x_e\} \quad (3)$$

$$\cup\ P|_{e \rightarrow x_e} \quad (4)$$

where $\overline{a}$ is a new atom for each a, x_e and $\overline{x_e}$ are new atoms for e, and $P|_{e \to x_e} = \bigcup_{r \in P} r|_{e \to x_e}$ where $r|_{e \to x_e}$ denotes rule r with every occurrence of e replaced by x_e.

The rewriting works as follows. The atom x_e represents the former external atom, i.e., that e is true, while $\overline{x_e}$ represents that it is false. The rules in (1) represent all input assignments which satisfy x_e (resp. e). More specifically, each rule in $\{x_e \leftarrow S^+_{\mathbf{T}} \cup \{\overline{a} \mid \neg a \in S^-_{\mathbf{T}}\} \mid S \in \mathcal{S}_{\mathbf{T}}(e,P)\}$ represents one possibility to satisfy the former external atom e, using the complete positive family of support sets $\mathcal{S}_{\mathbf{T}}$. Next, for an input atom a, the atom $\overline{a}$ represents that a is false *or* that x_e (resp. e) is true, as formalized by the rules (2). This is in order to ensure that for an assignment Y, all relevant rules in (1), i.e. those which might apply to subsets of Y, are contained in the reduct wrt. Y (because a could become false in a smaller model of the reduct); recall that in Example 6 the reason for incorrectness of the rewriting was exactly that these rules were dropped. The derivation of $\overline{a}$ despite a being true is only necessary if x_e is true wrt. Y; if x_e is false then all rules containing x_e are dropped from the reduct anyway. The idea amounts to a saturation encoding (Eiter et al. 2009); the use of disjunctions with head-cycles is in general unavoidable for complexity reasons (unless the polynomial hierarchy collapses). Next, rule (3) enforces $\overline{x_e}$ to be true whenever x_e is false. Finally, rules (4) resemble the original program with x_e in place of e.

We show now that the rewriting is sound and complete. For the following Proposition 1 we first assume that the complete family of support sets $\mathcal{S}_{\mathbf{T}}(e,P)$ is chosen such that for all $S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}(e,P)$ we have that $S^+_{\mathbf{T}} \cup \neg S^-_{\mathbf{T}} = I(e,P)$, i.e., all input atoms to e in P are explicitly constrained to be true or false. Note that each complete family of support sets can be modified to fulfill this criterion: replace each $S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}(e,P)$ with $S^+_{\mathbf{T}} \cup \neg S^-_{\mathbf{T}} \subsetneq I(e,P)$ by the set of all support sets constructible by distributing undefined atoms to $S^+_{\mathbf{T}}$ or $S^-_{\mathbf{T}}$ in all possible ways. This might lead to an exponential blowup of the size of the family of support sets, but is made in order to simplify the first result and its proof; however, we show below that the result still goes through without this blowup.

Proposition 1 *For all* HEX*-programs P, external atoms e in P and a positive complete family of support sets $\mathcal{S}_{\mathbf{T}}(e,P)$ such that $S^+_{\mathbf{T}} \cup \neg S^-_{\mathbf{T}} = I(e,P)$ for all $S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}(e,P)$, the answer sets of P are equivalent to those of $P_{[e]}$, modulo the atoms newly introduced in $P_{[e]}$.*

However, the idea still works for arbitrary complete positive families of support sets $\mathcal{S}_{\mathbf{T}}(e,P)$. To this end, we first show that two rules $x_e \leftarrow B, b$ and $x_e \leftarrow B, \overline{b}$ in the above encoding, stemming from two support sets that differ only in b resp. $\overline{b}$, can be replaced by a single rule $x_e \leftarrow B$ without affecting the semantics of the program. Intuitively, this corresponds to the case that two support sets imply that e is true whenever all of B and one of b or $\overline{b}$ hold, which might be also be expressed by a single support sets consisting of B only.

Proposition 2 *Let X be a set of atoms and P be a* HEX*-program such that*

$$
\begin{aligned}
P \supseteq\ & \{r_1 \colon x_e \leftarrow B, b; r_2 \colon x_e \leftarrow B, \overline{b}\} \\
& \cup \{\overline{a} \leftarrow \mathit{not}\, a;\ \overline{a} \leftarrow x_e;\ a \vee \overline{a} \leftarrow \mathit{not}\, \overline{x_e} \mid a \in X\} \\
& \cup \{\overline{x_e} \leftarrow \mathit{not}\, x_e\}
\end{aligned}
$$

where $B \subseteq \{a, \overline{a} \mid a \in X\}$, $b \in X$, and $\overline{x_e}$ occurs only in the rules explicitly shown above. Then P is equivalent to $P' = (P \setminus \{r_1, r_2\}) \cup \{r \colon x_e \leftarrow B\}$.

This idea can be iterated such that pairs of support sets can be recursively combined. Based on this result, one can then show that inlining is correct for arbitrary complete families of support sets.

Corollary 1 *For all* HEX*-programs P, external atoms e in P and a positive complete family of support sets $\mathcal{S}_{\mathbf{T}}(e, P)$, the answer sets of P are equivalent to those of $P_{[e]}$, modulo the atoms newly introduced in program $P_{[e]}$.*

We demonstrate the rewriting with an example.

Example 8 *Consider $P = \{a \leftarrow \&aOrNotB[a, b]()\}$, where $e = \&aOrNotB[a, b]()$ evaluates to true if a is true or b is false. Let $\mathcal{S}_{\mathbf{T}}(e, P) = \{\{a\}, \{\neg b\}\}$. Then we have:*

$$
\begin{aligned}
P_{[e]} = \{ & x_e \leftarrow a;\ x_e \leftarrow \overline{b} \\
& \overline{a} \leftarrow not\, a;\ \overline{a} \leftarrow x_e;\ \overline{b} \leftarrow not\, b;\ \overline{b} \leftarrow x_e; a \vee \overline{a} \leftarrow not\, \overline{x_e};\ b \vee \overline{b} \leftarrow not\, \overline{x_e} \\
& \overline{x_e} \leftarrow not\, x_e \\
& a \leftarrow x_e \}
\end{aligned}
$$

The program has the unique answer set $Y' = \{a, x_e, \overline{a}, \overline{b}\}$, which represents the answer set $Y = \{a\}$ of P.

Multiple external atoms can be inlined by iterative application. For a program P and a set E of external atoms in P we denote by $P_{[E]}$ the program after all external atoms from E have been inlined. Importantly, separate auxiliaries must be introduced for atoms that are input to multiple external atoms.

3.3.2 Inlining Negated External Atoms

Until now we restricted the discussion to positive external atoms based on positive support sets. One can observe that the rewriting from Definition 6 does indeed not work for external atoms e, which occur (also) in form $not\, e$ because programs P and $P[e]$ are in this case *not* equivalent in general.

Example 9 *Consider $P = \{p \leftarrow not\ \&neg[p]()\}$, where $\&neg[p]()$ is true if p is false and vice versa. The only answer set of P is $Y = \emptyset$ but the rewriting from Definition 6 yields*

$$
\begin{aligned}
P_{[\&neg[p]()]} = \{ & x_e \leftarrow \overline{p} \\
& \overline{p} \leftarrow not\, p;\ \overline{p} \leftarrow x_e;\ p \vee \overline{p} \leftarrow not\, \overline{x_e} \\
& \overline{x_e} \leftarrow not\, x_e \\
& p \leftarrow not\, x_e \}
\end{aligned}
$$

which has the answer sets $Y_1' = \{x_e, \overline{p}\}$ and $Y_2' = \{\overline{x_e}, p\}$ that represent the assignments $Y_1 = \emptyset$ and $Y_2 = \{p\}$ over P. However, only Y_1 $(= Y)$ is an answer set of P.

Intuitively, the rewriting does not work for negated external atoms because their input atoms may support themselves. More precisely, due to rule (3), an external atom is false by default if none of the rules (1) apply. If one of the external atom's input atoms depends on falsehood of the external atom, as in Example 9, then the input atom might be supported by falsehood of the external atom, although this falsehood itself depends on the input atom.

In order to extend our approach to the inlining of negated external atoms $not\, e$ in a program P, we make use of an arbitrary but fixed negative complete family $\mathcal{S}_{\mathbf{F}}(e, P)$ of support sets as by Definition 5. The idea is to replace a negated external atom $not\, e$ by a positive one e' such that $Y \models e'$ iff $Y \not\models e$ for all assignments Y; obviously, the resulting program has the same answer sets as before. Then the semantics of

e' is fully described by the negative complete family of support sets of e and we may apply the rewriting of Definition 6.

The idea is formalized by the following definition:

Definition 7 (Negated External Atom Inlining) *For a* HEX*-program P and negated external atom $not\,e$ in P, let*

$$\begin{aligned} P_{[not\,e]} = & \{x_e \leftarrow S^+_{\mathbf{F}} \cup \{\overline{a} \mid \neg a \in S^-_{\mathbf{F}}\} \mid S_{\mathbf{F}} \in \mathcal{S}_{\mathbf{F}}(e,P)\} && (5)\\ & \cup \{\overline{a} \leftarrow not\,a; \overline{a} \leftarrow x_e; a \vee \overline{a} \leftarrow not\,\overline{x_e} \mid a \in I(e,P)\} && (6)\\ & \cup \{\overline{x_e} \leftarrow not\,x_e\} && (7)\\ & \cup P|_{not\,e \rightarrow x_e} && (8) \end{aligned}$$

where $\overline{a}$ is a new atom for each a, x_e and $\overline{x_e}$ are new atoms for e, and $P|_{not\,e\rightarrow x_e} = \bigcup_{r \in P} r|_{not\,e \rightarrow x_e}$ where $r|_{not\,e\rightarrow x_e}$ denotes rule r with every occurrence of $not\,e$ replaced by x_e.

One can show that this rewriting is sound and complete.

Proposition 3 *For all* HEX*-programs P, negated external atoms $not\,e$ in P and a negative complete family of support sets $\mathcal{S}_{\mathbf{F}}(e,P)$, the answer sets of P are equivalent to those of $P_{[not\,e]}$, modulo the atoms newly introduced in program $P_{[not\,e]}$.*

As before, iterative application allows for inlining multiple negated external atoms. In the following, for a program P and a set E of either positive or negated external atoms in P, we denote by $P_{[E]}$ the program after all external atoms from E have been inlined.

Transforming Complete Families of Support Sets. For the sake of completeness we show that one can change the polarity of complete families of support sets:

Proposition 4 *Let $\mathcal{S}_\sigma$ be a positive resp. negative complete family of support sets for some external atom e in a program P, where $\sigma \in \{\mathbf{T}, \mathbf{F}\}$. Then $\mathcal{S}_{\overline{\sigma}} = \{S_{\overline{\sigma}} \in \prod_{S_\sigma \in \mathcal{S}_\sigma} \neg S_\sigma \mid S_{\overline{\sigma}}$ is consistent$\}$ is a negative resp. positive complete family of support sets, where $\overline{\mathbf{T}} = \mathbf{F}$ and $\overline{\mathbf{F}} = \mathbf{T}$.*

Intuitively, since a complete family family of positive support sets $\mathcal{S}_{\mathbf{T}}$ fully describes under which conditions the external atom is true, one can construct a negative support set by picking an arbitrary literal from each $S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}$ and changing its sign. Then, whenever the newly generated set is contained in the assignment, none of the original support sets in $\mathcal{S}_{\mathbf{T}}$ can match. The case for families of negative support sets is symmetric.

However, similarly to the transformation of the formula from conjunctive normal form to disjunctive normal form or vice versa, this may result in an exponential blow-up. In the spirit of our initial assumption that compact complete families of support sets exist, it is suggested to construct families of support sets of the required polarity right from the beginning, which we will also do in our experiments.

4 Exploiting External Source Inlining for Performance Boosts

An application of the techniques from the previous section are algorithmic improvements by skipping explicit verification calls for the sake of performance gains. As stated in Section 2, learning techniques may reduce the number of required verification calls, and – alternatively – using support sets for verification instead of explicit calls may lead to an efficiency improvement when checking external source guesses, but neither of these techniques eliminates the checks altogether (Eiter et al. 2014). In contrast, inlining embeds the semantics of external sources directly in the logic program. Thus, no more checks are needed; the resulting program can actually be evaluated by an ordinary ASP solver.

4.1 Implementation

We implemented this approach in the dlvhex[2] system, which is based on gringo and clasp from the Potassco suite[3]. External sources are supposed to provide a complete set of support sets. The system allows also for using universally quantified variables in the specification of support sets, which are automatically substituted by all constants occurring in the program. After support set learning in the initialization for the sake of external source inlining, the HEX-program is then evaluated entirely by the backend without any external calls.

The rewriting makes both the compatibility check (cf. Definition 3) and the minimality check wrt. the reduct and external sources (cf. Section 2 and Eiter et al. (2014)) obsolete. With the traditional approach, compatible sets are not necessarily answer sets because of possibly self-justified atoms whose cyclic support involves external atoms and is not detected by the ordinary ASP solver that evaluates the guessing program $\hat{P}$. On the other hand, after inlining, the minimality check performed by the ordinary ASP solver suffices.

We evaluated the approach using the experiments described in the following.

4.2 Experimental Setup

We present several benchmarks with 100 randomly generated instances each, which were run on a Linux server with two 12-core AMD 6176 SE CPUs and 128GB RAM using a 300 seconds timeout. The instances are available from http://www.kr.tuwien.ac.at/research/projects/inthex/inlining, while the program encodings and scripts used for running the benchmarks are included in the sourcecode repository of the dlvhex system, which is available from https://github.com/hexhex. Although some of the benchmark problems are similar to those used by Eiter et al. (2014) and in the conference versions of this paper, the runtime results are not directly comparable because of technical improvements in the implementation of support set generation and other (unrelated) solver improvements. Moreover, for the taxi benchmark we use a different scenario since the previous one was too easy in this context. However, for the pre-existing approaches the fundamental trend that the approach based on support sets outperforms the traditional approach is the same.

In our tables we compare three evaluation approaches (*configurations*), which we evaluate both for computing all and the first answer set only; the numbers in parentheses indicate the number of timeout instances, which were counted as 300 seconds when computing the average runtime of the instances. The **traditional** evaluation algorithm guesses the truth values of external atoms and verifies them by evaluation. In our experiments we use the learning technique EBL (Eiter et al. 2012) to learn parts of the external atom's behavior, i.e., there is a tight coupling of the reasoner with external sources. The second approach as by Eiter et al. (2014) is based on **support sets** (**sup.sets**), which are provided by the external source and learned at the beginning of the evaluation process. It then guesses external atoms as in the traditional approach, but verifies them by matching candidate compatible sets against support sets rather than by evaluation. While we add learned support sets as nogoods at the beginning, which exclude some but not all wrong guesses, recall that on-the-fly learning as by EBL is not possible in this approach since external sources are only

[2] www.kr.tuwien.ac.at/research/systems/dlvhex
[3] https://potassco.org

called at the beginning; this may be a drawback compared to **traditional**. The new **inlining** approach, based on the results from this paper, also learns support sets at the beginning similar to **sup.sets**, but uses them for rewriting external atoms as demonstrated in Section 3. Then, all answer sets of the rewritten ASP-program are accepted without the necessity for additional checks. Wrong guesses, which are not detected by the ordinary ASP solver backend, cannot occur here.

Note that our goal is to show improvements compared to previous HEX-algorithms, but not to compare HEX to other formalisms or encodings in ordinary (disjunctive) ASP, which might be feasible for some of the benchmark programs. Our fundamental assumption that compact complete families of support sets exist (see Section 3) is satisfied for all our scenarios, as discussed in more detail when we present the results.

Our hypothesis is that **inlining** outperforms both **traditional** and **sup.sets** for external sources with compact complete support set families. More precisely, we expect that **inlining** leads to a further speedup over **sup.sets** in many cases, especially when there are many candidate answer sets. Moreover, we expect that in cases where **inlining** cannot yield further improvements over **sup.sets**, then it does at least not harm much. This is because with **inlining**, (i) no external calls and (ii) no additional minimality checks are needed, which potentially leads to speedups. On the other hand, the only significant costs when generating the rewriting are caused by support set learning; however, this is also necessary with **sup.sets**, which was already shown to outperform **traditional** if small complete families of support sets exist. Hence, we expect further benefits but negligible additional costs.

House Problem. We first consider an abstraction of configuration problems, consisting of sets of *cabinets*, *rooms*, *objects* and *persons* (Mayer et al. 2009). The goal is to assign cabinets to persons, cabinets to rooms, and objects to cabinets, such that there are no more than four cabinets in a room or more than five objects in a cabinet. Objects belonging to a person must be stored in a cabinet belonging to the same person, and a room must not contain cabinets of more than one person. We assume that we have already a partial assignment to be completed. We use an existing guess-and-check encoding[4] which implements the check as external source. Instances of size n have n persons, $n{+}2$ cabinets, $n{+}1$ rooms, and $2n$ objects randomly assigned to persons; $2n{-}2$ objects are already stored.

The number and size of support sets is polynomially bounded by $(2n)^5$; this is due to the constraints that no more than four cabinets can be in a room and no more than five objects can be in a cabinet.

Table 1 shows the results, where numbers in parentheses indicate timeout instances. As expected, we have that **sup.sets** clearly outperforms **traditional** both when computing all answer sets and the first answer set only, which is because of faster candidate checking as already observed by Eiter et al. (2014). When computing all answer sets, the new **inlining** approach leads to a further speedup as it eliminates wrong guesses and the checking step altogether, while the additional initialization overhead is negligible. This is consistent with our hypothesis. When computing only a single answer set, **inlining** does not yield a further visible speedup, which can be explained by the fact that only few candidates (in the extreme case for consistent instances: a single one) must be checked before an answer set is found. In this case the additional initialization overhead compared to **sup.sets** is slightly visible, but as can be seen it is little such that the new technique does in fact not harm, as expected.

Taxi Assignment. We consider a program which uses external atoms to access a $DL\text{-}Lite_{\mathcal{A}}$-ontology (called a *DL-atom* (Eiter et al. 2008)). As demonstracted in Section 3.1, Calvanese et al. (2007) have proven that for this type of description logic at most one assertion is needed to derive an instance query from a consistent ontology. Moreover, at most two added ABox assertions are needed to make such an ontology inconsistent. Hence, the support sets required to describe the ontology are of only few different and small forms, which

[4]The encoding was taken from http://143.205.174.183/reconcile/tools.

n	all answer sets			first answer set		
	traditional	sup.sets	inlining	traditional	sup.sets	inlining
5	99.88 (17)	5.81 (0)	3.57 (0)	5.17 (0)	0.39 (0)	0.40 (0)
6	193.56 (35)	19.40 (1)	11.51 (0)	13.03 (0)	0.75 (0)	0.77 (0)
7	252.61 (81)	35.72 (3)	22.04 (2)	23.68 (2)	1.50 (0)	1.54 (0)
8	267.01 (85)	93.39 (13)	59.25 (11)	64.89 (10)	3.06 (0)	3.14 (0)
9	274.23 (85)	129.37 (29)	85.85 (13)	79.52 (13)	6.15 (0)	6.34 (0)
10	281.55 (83)	154.29 (42)	120.66 (16)	107.86 (12)	11.80 (0)	12.17 (0)
11	297.28 (86)	206.15 (53)	166.84 (45)	160.25 (49)	21.84 (0)	22.55 (0)
12	300.00 (100)	246.40 (57)	179.59 (41)	162.33 (47)	39.31 (0)	40.62 (0)
13	297.43 (99)	281.02 (91)	239.08 (69)	214.30 (65)	68.07 (0)	70.43 (0)
14	300.00 (100)	287.11 (91)	253.58 (65)	213.63 (63)	114.56 (0)	118.81 (0)
15	300.00 (100)	296.36 (92)	287.66 (75)	240.21 (75)	187.94 (0)	195.09 (0)

Table 1: House configuration

limits also the number of possible support sets to a quadratic number in the size of the program and the Abox. Moreover, the support sets are easy to construct by a syntactic analysis of the ontology and the DL-atoms, for details we refer to Eiter et al. (2014).

The task in this benchmark is to assign taxi *drivers* to *customers*. Each customer and driver is in a *region*. A customer may only be assigned to a driver in the same region. Up to four customers may be assigned to a driver. We let some customers be *e-customers* who use only electronic cars, and some drivers be *e-drivers* who drive electronic cars. The ontology stores information about individuals such as their locations (randomly chosen but balanced among regions). The encoding is taken from http://www.kr.tuwien.ac.at/research/projects/inthex/partialevaluation. An instance of size $4 \leq n \leq 9$ consists of n drivers, n customers including $n/2$ e-customers and $n/2$ regions.

Table 2 shows the results. The **sup.sets** approach is faster than the **traditional** one. When computing all answer sets, the difference is still clearly visible but less dramatic than when computing only the first answer set or in other benchmarks. This is because there is a large number of candidates and answer sets in this benchmark, which allow the learning techniques used in **traditional** to learn the behavior of the external sources well over time. The reasoner can then prevent wrong guesses and verification calls effectively, such that the advantage of improved verification calls as in **sup.sets** decreases the longer the solver runs. However, the **inlining** approach leads to a significant speedup since wrong guesses are impossible from the beginning and all verification calls are spared.

LUBM Diamond. While description logics correspond to fragments of first-order logic and are monotonic, their cyclic interaction with rules allow for default reasoning, i.e., making assumptions which might have to be withdrawn if more information becomes available (such as classifying an object based on absence of information). We consider default reasoning over the LUBM $DL\text{-}Lite_{\mathcal{A}}$ ontology (http://swat.cse.lehigh.edu/projects/lubm/). Defaults express that assistants are normally employees and students are normally not employees. The ontology entails that assistants are students, resembling Nixon's diamond. The instance size is the number of persons, which are randomly marked as students, assistants or employees. The task is to classify all persons in the ontology. Due to incomplete information the result is not unique.

Table 3 shows the results. As already observed by Eiter et al. (2014), **sup.sets** outperforms **traditional**. Compared to the taxi benchmark there is a significantly smaller number of model candidates, which makes

n	all answer sets			first answer set		
	traditional	sup.sets	inlining	traditional	sup.sets	inlining
4	0.54 (0)	0.47 (0)	0.22 (0)	0.19 (0)	0.16 (0)	0.16 (0)
5	5.40 (0)	5.92 (0)	1.10 (0)	0.82 (0)	0.21 (0)	0.18 (0)
6	88.93 (9)	63.24 (2)	8.92 (0)	8.86 (0)	0.28 (0)	0.21 (0)
7	295.94 (98)	277.64 (84)	149.56 (19)	154.71 (42)	0.90 (0)	0.26 (0)
8	300.00 (100)	299.99 (99)	290.00 (94)	249.79 (81)	3.55 (1)	0.32 (0)
9	300.00 (100)	300.00 (100)	300.00 (100)	281.35 (92)	2.77 (0)	0.39 (0)
10	300.00 (100)	300.00 (100)	300.00 (100)	289.54 (96)	3.33 (1)	0.49 (0)

Table 2: Driver-customer assignment

n	all answer sets			first answer set		
	traditional	sup.sets	inlining	traditional	sup.sets	inlining
20	1.17 (0)	0.33 (0)	0.30 (0)	0.34 (0)	0.31 (0)	0.30 (0)
30	30.05 (3)	0.98 (0)	0.33 (0)	6.29 (0)	0.61 (0)	0.33 (0)
40	148.57 (40)	16.66 (2)	0.37 (0)	86.69 (22)	8.88 (0)	0.37 (0)
50	250.26 (75)	80.51 (15)	0.44 (0)	214.68 (65)	51.94 (4)	0.43 (0)
60	286.58 (89)	183.79 (47)	0.52 (0)	265.91 (87)	153.05 (36)	0.52 (0)
70	297.94 (99)	253.66 (73)	0.65 (0)	297.16 (99)	225.54 (65)	0.65 (0)
80	300.00 (100)	282.01 (91)	0.81 (0)	300.00 (100)	271.19 (84)	0.81 (0)
90	300.00 (100)	298.71 (99)	1.04 (0)	300.00 (100)	296.06 (97)	1.04 (0)
100	300.00 (100)	300.00 (100)	1.27 (0)	300.00 (100)	298.45 (99)	1.27 (0)
110	300.00 (100)	300.00 (100)	1.59 (0)	300.00 (100)	300.00 (100)	1.58 (0)
120	300.00 (100)	300.00 (100)	2.00 (0)	300.00 (100)	300.00 (100)	2.00 (0)

Table 3: Default rules over LUBM in $DL\text{-}Lite_{\mathcal{A}}$

learning in the **traditional** approach less effective. This can in particular be seen when computing all answer sets, since when computing the first answer set only, learning is less effective anyway (as described in the previous benchmark). The decreased effectiveness of learning from external calls is then more easily compensated by the more efficient compatibility check as by **sup.sets**, which is why the relative speedup is larger now. However, **inlining** is again the most efficient approach due to elimination of the compatibility check. Thanks to the existence of a compact family support sets for $DL\text{-}Lite_{\mathcal{A}}$-ontologies, the speedup is dramatic.

Non-3-Colorability. We consider the problem of deciding if a given graph is *not* 3-colorable, i.e., if it is not possible to color the nodes such that adjacent nodes have different colors. We use an encoding which splits the guessing part P_{col} from the checking part P_{check}. The latter, which is itself implemented as logic program, is used as an external source from the guessing part. For a color assignment, given by facts of kind $inp(col, v, c)$ where v is a vertex and c is a color, P_{check} derives the atom inv in its only answer set, otherwise it has an empty answer set. We then use the following program P_{col} to guess a coloring and check it using the external atom $\&query[P_{check}, inp, inv]()$, which is true iff program P_{check}, extended with facts over predicate inp, delivers an answer set which contains inv. We add a constraint which eliminates answer sets other than the saturated one, thus each instance has either no or exactly one answer set. The size of the

n	all answer sets			first answer set		
	traditional	sup.sets	inlining	traditional	sup.sets	inlining
20	299.01 (99)	0.20 (0)	0.16 (0)	0.12 (0)	0.12 (0)	0.12 (0)
60	300.00 (100)	1.65 (0)	1.38 (0)	0.47 (0)	0.46 (0)	0.46 (0)
100	300.00 (100)	8.74 (0)	8.05 (0)	2.03 (0)	2.02 (0)	2.02 (0)
140	300.00 (100)	29.65 (0)	28.62 (0)	6.42 (0)	6.46 (0)	6.46 (0)
180	300.00 (100)	76.91 (0)	75.65 (0)	16.60 (0)	16.48 (0)	16.39 (0)
220	300.00 (100)	154.24 (20)	153.17 (20)	35.64 (0)	35.60 (0)	35.30 (0)
260	300.00 (100)	203.91 (49)	201.88 (50)	67.63 (0)	67.20 (0)	67.48 (0)
300	300.00 (100)	233.00 (60)	231.12 (60)	117.26 (0)	117.75 (1)	117.09 (0)
340	300.00 (100)	252.45 (70)	250.02 (70)	164.72 (29)	164.59 (30)	164.62 (30)
380	300.00 (100)	265.30 (80)	262.85 (80)	196.86 (41)	196.76 (40)	196.46 (40)
420	300.00 (100)	276.76 (80)	274.24 (80)	219.18 (58)	219.65 (58)	219.48 (60)
460	300.00 (100)	282.09 (90)	280.31 (90)	235.81 (60)	236.19 (61)	236.10 (61)

Table 4: Non-3-colorability

instances is the number of nodes n.

A compact complete family of support sets for $\&query[P_{check}, inp, inv]()$ exists: the number of edges to be checked is no greater than quadratic in the number of nodes and the number of colors is constant, which allows the check to be encoded by a quadratic number of binary support sets.

The encoding is as follows:

$$\begin{aligned} P_{col} = \big\{ col(V,r) \vee col(V,g) \vee col(V,b) &\leftarrow node(V), \\ inp(p,X,Y) &\leftarrow p(X,Y) \mid p \in \{col, edge\}, \\ inval &\leftarrow \&query[P_{check}, inp, inv](), \\ col(V,c) &\leftarrow inval, node(V) \mid c \in \{r,g,b\} \\ &\leftarrow inval \big\} \end{aligned}$$

The results are shown in Table 4. While **sup.sets** already outperforms **traditional**, **inlining** leads to a further small speedup when computing all answer sets. Compared to previous benchmarks, there are significantly fewer support sets, which makes candidate checking in **sup.sets** inexpensive. This explains the large speedup of **sup.sets** over **traditional**, and that avoiding the check in **inlining** does not lead to a large further speedup. However, due to a negligible additional overhead, **inlining** does at least not harm, which is in line with our hypothesis.

Nonexistence of a Vertex Covering. Next, we consider the coNP-complete problem of checking if for a given undirected graph there is no vertex covering of a certain maximal size. More precisely, given a graph $\langle V, E\rangle$, a vertex covering is a node selection $C \subseteq V$ such that for each edge $\{v,u\} \in E$ we have $\{v,u\} \cap C \neq \emptyset$. Our instances consist of such a graph $\langle V, E\rangle$, given by predicates $node(\cdot)$ and $edge(\cdot,\cdot)$, and an integer L (*limit*), given by $limit(L)$. The task is to decide if there is *no* vertex covering containing at most L nodes. The size of the instances is the number of nodes $n = |V|$.

Similarly as for the previous benchmark, we use an encoding which splits the guessing part P_{nonVC} from the checking part, where the latter is realized as an external source. An important difference to the previous benchmark is that the checking component must now aggregate over the node selection to check the size constraint. An encoding as an ordinary ASP-program is challenging since the coNP-completeness requires a saturation encoding and the size check requires aggregate atoms, which means that aggregate atoms must be used in a cycle; many reasoners do not support this. However, HEX-program, which inherently support cyclic external atoms, allow for pushing the check into an external source.

The number and size of support sets is polynomial in the size of the graph, but exponential in the limit L. In this benchmark we consider L to be a constant number which is for each instance randomly chosen from the range $1 \leq L \leq 20$; we exclude instances with graphs $\langle V, E\rangle$ and limits L such that $L \geq |V|$ as in such cases the final answer to the considered problem is trivially false (since V is trivially a vertex covering of size no greater than L).

The encoding is as follows. The guessing part is similar as before. For a candidate vertex covering given by atoms of kind $in(n)$ or $out(n)$ for nodes n, the external atom $\&checkVC[in, out, edge, S]()$ is true iff in and out encode a valid vertex covering of the graph specified by $edge$ of size no greater than limit L. A complete family of support sets for $\&checkVC[in, out, edge, L]()$ is of size at most n^L, where L is bounded in our scenario.

$$
\begin{aligned}
P_{nonVC} = \big\{ in(V) \vee out(V) &\leftarrow node(V),\\
inval &\leftarrow \&checkVC[in, out, edge, L](), limit(L),\\
in(V) &\leftarrow inval, node(V)\\
out(V) &\leftarrow inval, node(V)\\
&\leftarrow inval\big\}
\end{aligned}
$$

The results are shown in Table 5. Although the maximum size L of the vertex covering is limited and the number of support sets is thus polynomial in the instance size, it is significantly larger compared to the non-3-colorability problem; the benchmark shows that the approach is still feasible in such cases. Checking guesses based on support sets in the **sup.sets** configuration is then more expensive than for non-3-colorability, which makes the relative speedup of **sup.sets** over **traditional** smaller (but still clearly visible). Thus, there is now more room for further improvement by eliminating this check as in the **inlining** configuration. Once more, eliminating the compatibility check altogether yields a a further speedup.

Interestingly, the runtimes when computing all and the first answer set only are almost the same. Although this effect occurs with all configurations and is not related to our new approach, we briefly discuss it. Each instance has either one or no answer set. Despite this, computing all answer sets can in principle be slower than computing the first answer set since the reasoner has to determine that there are no further ones. However, in this case, the instances terminate almost immediately after the (only) answer set has been found. Since the only answer set is the saturated one, whose minimality wrt. the reduct has already been established, learning techniques in the ordinary ASP solver seem to help the reasoner to determine that there cannot be further (smaller) answer sets.

Discussion and Summary. As stated above, this paper focuses on external sources who possess a compact complete family of support sets. For the sake of completeness we still discuss also the case where a complete family of support sets is not small. As an extreme case, consider $P = \{p(n+1) \leftarrow \&even[p](\{\}) \cup \{p(i) \leftarrow | 1 \leq i \leq n\}$ for a given integer n, where $\&even[p]()$ is true iff the number of true atoms over p is even. The program has a single answer set $Y = \{p(i) \mid 1 \leq i \leq n\}$ if n is odd, and no answer set if n is even. In any case, $\hat{P}$ has only two candidates which are easily checked in the **traditional** approach, while exponentially many support sets must be generated to represent the semantics of $\&even[p]()$ (one for each

n	all answer sets			first answer set		
	traditional	sup.sets	inlining	traditional	sup.sets	inlining
8	15.45 (0)	4.13 (0)	0.61 (0)	15.42 (0)	4.12 (0)	0.61 (0)
9	62.89 (11)	31.23 (8)	7.72 (0)	62.81 (11)	31.26 (8)	7.64 (0)
10	102.15 (22)	80.65 (24)	36.09 (8)	102.17 (22)	80.57 (24)	36.11 (8)
11	181.35 (55)	89.87 (26)	47.42 (13)	181.41 (55)	89.96 (26)	47.45 (13)
12	222.05 (66)	135.79 (43)	89.43 (25)	222.05 (66)	135.82 (43)	89.36 (25)
13	256.16 (82)	158.63 (50)	110.26 (32)	256.16 (82)	158.71 (51)	110.19 (32)
14	288.93 (96)	189.18 (62)	152.59 (50)	288.94 (96)	189.24 (62)	152.60 (50)
15	284.97 (93)	178.66 (59)	145.46 (47)	284.96 (93)	178.66 (59)	145.42 (47)
16	294.77 (98)	219.03 (72)	191.25 (62)	294.74 (98)	218.98 (72)	191.21 (62)
17	300.00 (100)	219.19 (73)	175.57 (56)	300.00 (100)	219.19 (73)	175.45 (56)
18	300.00 (100)	231.10 (77)	195.14 (63)	300.00 (100)	231.10 (77)	195.13 (63)
19	300.00 (100)	243.12 (81)	220.70 (71)	300.00 (100)	243.11 (81)	220.72 (71)
20	300.00 (100)	237.07 (79)	217.87 (70)	300.00 (100)	237.07 (79)	217.86 (70)

Table 5: Nonexistence of a vertex covering

subset of $\{p(i) \leftarrow\mid 1 \leq i \leq n\}$ with an even number of elements). In such cases, **traditional** might be exponentially faster than **sup.sets** and **inlining**.

However, this is not the case for many realistic types of external sources, where the existence of of a compact family of support sets is often even provable, such as the ones we used in our experiments. The size of the **inlining** encoding is directly linked to the size of the complete family of support sets, and if this size is small then the **inlining** approach is clearly superior to **sup.sets** as it eliminates the compatibility check and minimality check wrt. external sources altogether, while it has only slightly higher initialization overhead. This overhead can be neglected even in cases where there is no further speedup by **inlining**. **Sup.sets** is in turn superior to **traditional** (even with learning technique EBL) as already observed by Eiter et al. (2014). We can therefore conclude that **inlining** is a significant improvement over **sup.sets** and, for the considered types of external sources, also over **traditional**.

5 Equivalence of HEX-Programs

In this section we exploit the technique of external source inlining from Section 3 for another application in the area of program equivalence criteria. In previous works, characterizations of equivalence of (ordinary) answer set programs under programs extensions have been developed. That is, two programs P and Q are considered to be equivalent if $P \cup R$ and $Q \cup R$ have the same answer sets for all programs R of a certain type, which depends on the notion of equivalence. Most importantly, for *strongly equivalent* programs we have that $P \cup R$ and $Q \cup R$ have the same answer sets for any program R (Lifschitz et al. 2001), while *uniformly equivalent* programs guarantee this only if R is a set of facts (Eiter and Fink 2003). Later, these notions were extended to the non-ground case (Eiter et al. 2005). A more fine-grained approach is the one by Woltran (2008), where R can contain rules other than facts, but the sets of atoms which can occur in rule heads and bodies are restricted. Besides efficiency improvements, external source inlining can also be exploited for extending such equivalence notions from ordinary ASP to HEX-programs.

In this section we present such an extension of the notion by Woltran (2008), which subsumes strong and uniform equivalence as special cases. More specifically, we present a notion for equivalence of HEX-programs P and Q under extensions by additional rules R. The possibly added rules are constrained such that their head and body atoms and input atoms to external atoms can only come from fixed sets. Due to the support for external atoms, which can even be nonmonotonic, and the use of the FLP-reduct (Faber et al. 2011) instead of the GL-reduct (Gelfond and Lifschitz 1988) in the semantics of HEX-programs, this result is not immediate. Since well-known ASP extensions (such as programs with aggregates (Faber et al. 2011) and constraint ASP (Gebser et al. 2009; Ostrowski and Schaub 2012)) amount to special cases of HEX-programs, the results are interesting beyond the specific formalism.

We proceed as follows. In the first step (Section 5.1), only the programs P and Q can be HEX-programs, but the added program R must be ordinary. This amounts to a generalization of the results by Woltran (2008) from ordinary ASP to HEX-programs. In the second step (Section 5.2), we allow also the added program R to contain external atoms. For this purpose, we exploit the possibility to inline external atoms.

5.1 Generalizing Equivalence Results

In the following, for sets $\mathcal{H}$ and $\mathcal{B}$ of atoms we let $\mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle} = \{P \text{ is an ASP-program} \mid H(P) \subseteq \mathcal{H}, B^+(P) \cup B^-(P) \subseteq \mathcal{B}\}$ be the set of ordinary programs whose head and body atoms come only from $\mathcal{H}$ and $\mathcal{B}$, respectively. Ordinary ASP-programs P and Q are called $\langle \mathcal{H},\mathcal{B}\rangle$-equivalent, if the answer sets of $P \cup R$ and $Q \cup R$ are the same for all ordinary ASP-programs R which use only head atoms from $\mathcal{H}$ and only body atoms from $\mathcal{B}$, i.e., $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$.

We first lift this result to the case where P and Q are general HEX-programs which possibly contain external atoms, while R remains an ordinary ASP-program. Formally:

Definition 8 *HEX-programs P and Q are* equivalent wrt. a pair $\langle \mathcal{H},\mathcal{B}\rangle$ of sets of atoms*, or* $\langle \mathcal{H},\mathcal{B}\rangle$-equivalent*, denoted $P \equiv_{\langle \mathcal{H},\mathcal{B}\rangle} Q$, if $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$ for all $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$.*

Similarly, we write $P \subseteq_{\langle \mathcal{H},\mathcal{B}\rangle} Q$ if $\mathcal{AS}(P \cup R) \subseteq \mathcal{AS}(Q \cup R)$ for all $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$.

Towards a characterization of equivalence, one can first show that if there is a counterexample R for $P \equiv_{\langle \mathcal{H},\mathcal{B}\rangle} Q$, i.e., an $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$ such that $\mathcal{AS}(P \cup R) \neq \mathcal{AS}(Q \cup R)$, then there is also a simple counterexample in form of a positive program $R' \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$.

Proposition 5 *Let P and Q be HEX-programs, R be an ordinary ASP-program, and Y be an assignment s.t. $Y \in \mathcal{AS}(P \cup R)$ but $Y \notin \mathcal{AS}(Q \cup R)$. Then there is also a positive ordinary ASP-program R' such that $Y \in \mathcal{AS}(P \cup R')$ but $Y \notin \mathcal{AS}(Q \cup R')$ and $B(R') \subseteq B(R)$ and $H(R') \subseteq H(R)$.*

Next, we show that the concepts on equivalence generalize from ordinary ASP to HEX-programs. In the following, for an assignment Y and a set of atoms A we write $Y|_A$ for for the projection $Y \cap A$ of Y to A. Moreover, for sets of atoms X, Y we write $X \leq^{\mathcal{B}}_{\mathcal{H}} Y$ if $X|_{\mathcal{H}} \subseteq Y|_{\mathcal{H}}$ and $X|_{\mathcal{B}} \supseteq Y|_{\mathcal{B}}$. Intuitively, if $X \leq^{\mathcal{B}}_{\mathcal{H}} Y$ then Y satisfies all positive programs from $\mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$ which are also satisfied by X because it satisfies no fewer heads and no more bodies than X. We write $X <^{\mathcal{B}}_{\mathcal{H}} Y$ if $X \leq^{\mathcal{B}}_{\mathcal{H}} Y$ and $X|_{\mathcal{H}\cup\mathcal{B}} \neq Y|_{\mathcal{H}\cup\mathcal{B}}$.

We use the following concept for witnessing that $\mathcal{AS}(P \cup R) \subseteq \mathcal{AS}(Q \cup R)$ does *not* hold.

Definition 9 *A witness for $P \not\subseteq_{\langle \mathcal{H},\mathcal{B}\rangle} Q$ is a pair (X, Y) of assignments with $X \subseteq Y$ such that*[5]*:*

(i) $Y \models P$ and for each $Y' \subsetneq Y$ with $Y' \models fP^Y$ we have $Y'|_{\mathcal{H}} \subsetneq Y|_{\mathcal{H}}$; and

[5] Note that Woltran (2008) called this *a witness for $P \subseteq_{\langle \mathcal{H},\mathcal{B}\rangle} Q$*, but since it is actually a witness for the violation of the containment, we change the terminology.

(ii) if $Y \models Q$ then $X \subsetneq Y$, $X \models fQ^Y$ and for all X' with $X \leq_{\mathcal{H}}^{\mathcal{B}} X' \subsetneq Y$ we have $X' \not\models fP^Y$.

The idea is that a witness represents a counterexample to the containment, where X characterizes a program R such that Y is an answer set of $P \cup R$ but not of $Q \cup R$. One can show that the existence of a witness and the violation of the containment are equivalent.

Because some steps in the according considerations for ordinary ASP depend on the fact that GL-reducts of programs wrt. assignments are positive programs (cf. $\leq_{\mathcal{H}}^{\mathcal{B}}$), it is an interesting result that the following propositions still hold in its generalized form. Because we use FLP-reducts instead, and P and Q might even contain nonmonotonic external atoms, the results do not automatically carry over. However, a closer analysis reveals that the property of being a positive program is only required for the reducts of R but not of P or Q. Since we restricted R to ordinary ASP-programs for now, and Proposition 5 allows us to further restrict it to positive programs, the use of the FLP-reduct does not harm: if R is positive from the beginning, then also its FLP-reduct (wrt. any assignment) is positive.

Proposition 6 *For* HEX*-programs P and Q and sets $\mathcal{H}$ and $\mathcal{B}$ of atoms, there is a program $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B} \rangle}$ with $\mathcal{AS}(P \cup R) \not\subseteq \mathcal{AS}(Q \cup R)$ iff there is a witness for $P \not\subseteq_{\langle \mathcal{H},\mathcal{B} \rangle} Q$.*

While witnesses compare the sets of answer sets of two programs directly, the next concept of $\langle \mathcal{H}, \mathcal{B} \rangle$-models can be used to characterize a single program. In the following, for two sets of atoms $\mathcal{H}$ and $\mathcal{B}$, a pair (X, Y) of assignments is called $\leq_{\mathcal{H}}^{\mathcal{B}}$-maximal for P if $X \models fP^Y$ and for all X' with $X <_{\mathcal{H}}^{\mathcal{B}} X' \subsetneq Y$ we have $X' \not\models fP^Y$.

Definition 10 *Given sets $\mathcal{H}$, $\mathcal{B}$ of atoms, a pair (X, Y) of assignments is an $\langle \mathcal{H}, \mathcal{B} \rangle$-model of a program P if*

(i) $Y \models P$ and for each $Y' \subsetneq Y$ with $Y' \models fP^Y$ we have $Y'|_{\mathcal{H}} \subsetneq Y|_{\mathcal{H}}$; and

(ii) if $X \subsetneq Y$ then there exists an $X' \subsetneq Y$ with $X'|_{\mathcal{H} \cup \mathcal{B}} = X$ such that (X', Y) is $\leq_{\mathcal{H}}^{\mathcal{B}}$-maximal for P.

One can show that $\langle \mathcal{H}, \mathcal{B} \rangle$-equivalence of two programs can be reduced to a comparison of their $\langle \mathcal{H}, \mathcal{B} \rangle$-models. We denote the set of all $\langle \mathcal{H}, \mathcal{B} \rangle$-models of a program P by $\sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$.

Proposition 7 *For sets $\mathcal{H}$ and $\mathcal{B}$ of atoms and* HEX*-programs P and Q, we have $P \equiv_{\langle \mathcal{H},\mathcal{B} \rangle} Q$ iff $\sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P) = \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$.*

We demonstrate the lifted results using three examples.

Example 10 *Consider the programs $P = \{a \leftarrow \&aOrNotB[a, b]()\}$ and $Q = \{a \leftarrow a;\ a \leftarrow not\, b\}$ where $\&aOrNotB[a, b]()$ evaluates to true whenever a is true or b is false, and to false otherwise. Let $\mathcal{H} = \mathcal{B} = \{a, b\}$. We have that $\sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P) = \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q) = \{(\emptyset, \{a\}), (\{a\}, \{a\}), (\{b\}, \{b\}), (\{a\}, \{a, b\}), (\{b\}, \{a, b\}), (\{a, b\}, \{a, b\})\}$, and thus P and Q are $\langle \mathcal{H}, \mathcal{B} \rangle$-equivalent.*

It is easy to see that for any of the $\langle \mathcal{H}, \mathcal{B} \rangle$-models of form (Y, Y), Y is a model both of P and Q, and for any $Y' \not\subseteq Y$ we have $Y'|_{\mathcal{H}} \subsetneq Y|_{\mathcal{H}}$; for the fourth candidate $(\emptyset, \emptyset)$ one can observe that $\emptyset$ is neither a model of P nor of Q.

For the $\langle \mathcal{H}, \mathcal{B} \rangle$-models of form $(\{a\}, \{a, b\})$ resp. $(\{b\}, \{a, b\})$, one can observe that $X' = \{a\}$ resp. $X' = \{b\}$ satisfies Condition (ii) of Definition 10 both for P and Q, while for $(\emptyset, \{a, b\})$ the only candidate for $X' \subsetneq \{a, b\}$ with $X'|_{\mathcal{H} \cup \mathcal{B}} = X$ is $X' = \emptyset$, but $(\emptyset, \{a, b\})$ is neither $\leq_{\mathcal{H}}^{\mathcal{B}}$-maximal for P nor for Q because $\emptyset \not\models fP^{\{a,b\}}$ and $\emptyset \not\models fQ^{\{a,b\}}$.

For unary Y, the only $\langle\mathcal{H},\mathcal{B}\rangle$-model (X, Y) with $X \neq Y$ of P or Q is $(\emptyset, \{b\})$ because for $X' = \emptyset$ we have $\emptyset \models fP^{\{b\}}$ and $\emptyset \models fQ^{\{a\}}$, and $(\emptyset, \{b\})$ is also $\leq^{\mathcal{B}}_{\mathcal{H}}$-maximal for P and for Q. On the other hand, $(\emptyset, \{a\})$ fails to be an $\langle\mathcal{H},\mathcal{B}\rangle$-model because the only candidate for X' is $\emptyset$, but $\emptyset \not\models fP^{\{a\}}$ and $\emptyset \not\models fQ^{\{a\}}$.

Example 11 *Consider the programs $P = \{a \leftarrow \&neg[b]();\ b \leftarrow \&neg[a]();\ a \leftarrow b\}$ and $Q = \{a \vee b;\ a \leftarrow b\}$ where $\&neg[x]()$ evaluates to true whenever x is false and to true otherwise.*

For $\mathcal{H} = \{a, b\}$ and $\mathcal{B} = \{b\}$ we have that $\sigma_{\langle\mathcal{H},\mathcal{B}\rangle}(P) = \sigma_{\langle\mathcal{H},\mathcal{B}\rangle}(Q) = \{(\{a\}, \{a\}), (\{a\}, \{a, b\}), (\{a, b\}, \{a, b\})\}$, and thus the programs are $\langle\mathcal{H},\mathcal{B}\rangle$-equivalent. The most interesting candidate which fails to be an $\langle\mathcal{H},\mathcal{B}\rangle$-model of either progam is $(\emptyset, \{a, b\})$. For P we have that $fP^{\{a,b\}} = \{a \leftarrow b\}$, of which $\emptyset$ is a model, but for $\{a\}$ we have $\emptyset \leq^{\mathcal{B}}_{\mathcal{H}} \{a\} \subsetneq Y$ and $\{a\} \models fP^{\{a,b\}}$, thus $\emptyset$ is not $\leq^{\mathcal{B}}_{\mathcal{H}}$-maximal for P. For Q we have that $fQ^{\{a,b\}} = \{a \vee b; a \leftarrow b\}$, which is unsatisfied under $\emptyset$.

Example 12 *Consider the programs P and Q from Example 11 and $\mathcal{H} = \{a, b\}$ and $\mathcal{B} = \{a, b\}$. We have that $\sigma_{\langle\mathcal{H},\mathcal{B}\rangle}(P) = \{(\{a\}, \{a\}), (\emptyset, \{a, b\}), (\{a\}, \{a, b\}), (\{a, b\}, \{a, b\})\}$. Note that $(\emptyset, \{a, b\})$ is now an $\langle\mathcal{H},\mathcal{B}\rangle$-model of P because $\emptyset$ is a model of $fP^{\{a,b\}} = \{a \leftarrow b\}$ and there is no X' with $\emptyset \leq^{\mathcal{B}}_{\mathcal{H}} X' \subsetneq Y$ with $X' \models fP^{\{a \leftarrow b\}}$ (because now $\emptyset \not\leq^{\mathcal{B}}_{\mathcal{H}} \{a\}$); thus $\emptyset$ is $\leq^{\mathcal{B}}_{\mathcal{H}}$-maximal for P. On the other hand, $\sigma_{\langle\mathcal{H},\mathcal{B}\rangle}(Q) = \{(\{a\}, \{a\}), (\{a\}, \{a, b\}), (\{a, b\}, \{a, b\})\}$. That is, $(\emptyset, \{a, b\})$ is still not an $\langle\mathcal{H},\mathcal{B}\rangle$-model of Q because $\emptyset$ is not a model of $fQ^{\{a,b\}} = \{a \vee b;\ a \leftarrow b\}$. And thus the programs are not $\langle\mathcal{H},\mathcal{B}\rangle$-equivalent.*

Indeed, for $R = \{b \leftarrow a\} \in \mathcal{P}_{\langle\mathcal{H},\mathcal{B}\rangle}$ we have that $Y = \{a, b\}$ is an answer set of $Q \cup R$ but not of $P \cup R$.

5.2 Adding General HEX-Programs

Up to this point we allowed only the addition of ordinary ASP-programs $R \in \mathcal{P}_{\langle\mathcal{H},\mathcal{B}\rangle}$. As a preparation for the addition of general HEX-programs, we show now that if programs P and Q are $\langle\mathcal{H},\mathcal{B}\rangle$-equivalent, then sets $\mathcal{B}$ and $\mathcal{H}$ can be extended by atoms that do not appear in P and Q and the programs are still equivalent wrt. the expanded sets. Intuitively, this allows introducing auxiliary atoms without harming their equivalence. This possibility is needed for our extensions of the results to the case where R can be a general HEX-program.

Expanding Sets $\mathcal{B}$ and $\mathcal{H}$. If programs P and Q are $\langle\mathcal{H},\mathcal{B}\rangle$-equivalent, then they are also $\langle\mathcal{H}',\mathcal{B}'\rangle$-equivalent whenever $\mathcal{H}' \setminus \mathcal{H}$ and $\mathcal{B}' \setminus \mathcal{B}$ contain only atoms which do not appear in P or Q. This is intuitively the case because such atoms cannot interfere with atoms which are already in the program.

Formally, one can show the following result:

Proposition 8 *For sets $\mathcal{H}$ and $\mathcal{B}$ of atoms, HEX-programs P and Q, and an atom a which does not occur in P or Q, the following holds:*

(i) $P \equiv_{\langle\mathcal{H},\mathcal{B}\rangle} Q$ *iff* $P \equiv_{\langle\mathcal{H}\cup\{a\},\mathcal{B}\rangle} Q$*; and*

(ii) $P \equiv_{\langle\mathcal{H},\mathcal{B}\rangle} Q$ *iff* $P \equiv_{\langle\mathcal{H},\mathcal{B}\cup\{a\}\rangle} Q$*.*

By iterative applications of this result we get the desired result:

Corollary 2 *For sets $\mathcal{H}$ and $\mathcal{B}$ of atoms, programs P and Q, and an sets of atoms $\mathcal{H}'$ and $\mathcal{B}'$ which does not occur in P or Q, we have $P \equiv_{\langle\mathcal{H},\mathcal{B}\rangle} Q$ iff $P \equiv_{\langle\mathcal{H}\cup\mathcal{H}',\mathcal{B}\cup\mathcal{B}'\rangle} Q$.*

Addition of General HEX-Programs. In the following, for sets $\mathcal{H}$, $\mathcal{B}$ of atoms we define the set

$$\mathcal{P}^e_{\langle\mathcal{H},\mathcal{B}\rangle} = \left\{ \text{HEX-program } P \;\middle|\; \begin{array}{l} H(P) \subseteq \mathcal{H}, B^+(P) \cup B^-(P) \subseteq \mathcal{B}, \\ \text{only } \mathcal{B} \text{ are input to external atoms} \end{array} \right\}$$

of general HEX-programs whose head atoms come only from $\mathcal{H}$ and whose body atoms and input atoms to external atoms come only from $\mathcal{B}$.[6] We then extend Definition 8 as follows.

Definition 11 HEX-*programs* P *and* Q *are* e-equivalent wrt. a pair $\langle \mathcal{H}, \mathcal{B} \rangle$ of sets of atoms*, or* $\langle \mathcal{H}, \mathcal{B} \rangle^e$-*equivalent, denoted* $P \equiv^e_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$*, if* $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$ *for all* $R \in \mathcal{P}^e_{\langle \mathcal{H}, \mathcal{B} \rangle}$.

Towards a characterization of $\langle \mathcal{H}, \mathcal{B} \rangle^e$-equivalence, we make use of external atom inlining as by Definition 6 without changing the answer sets of a program, cf. Proposition 1.

We start with a technical result which allows for renaming a predicate input parameter $p_i \in \mathbf{p}$ of an external atom $e = \&g[\mathbf{p}](\mathbf{c})$ in a program P to a new predicate q which does not occur in P. This allows us to rename predicates such that inlining does not introduce rules which derive atoms other than auxiliaries, which is advantageous in the following.

The idea of the renaming is to add auxiliary rules which define q such that its extension represents exactly the former atoms over p_i, i.e., each atom $p_i(\mathbf{d})$ is represented by $q(p_i, \mathbf{d})$. Then, external predicate $\&g$ is replaced by a new $\&g'$ whose semantics is adopted to this encoding of the input atoms.

For the formalization of this idea, let $\mathbf{p}|_{p_i \to q}$ be vector $\mathbf{p}$ after replacement of its i-th element p_i by q. Moreover, for an assignment Y let $Y^q = Y \cup \{p_i(\mathbf{d}) \mid q(p_i, \mathbf{d}) \in Y\}$ be the extended assignment which 'extracts' from each atom $q(p_i, \mathbf{d}) \in Y$ the original atom $p_i(\mathbf{d})$. Then one can show:

Lemma 1 *For an external atom* $e = \&g[\mathbf{p}](\mathbf{c})$ *in program* P*,* $p_i \in \mathbf{p}$*, a new predicate* q*, let* $e' = \&g'[\mathbf{p}|_{p_i \to q}](\mathbf{c})$ *s.t.* $f_{\&g'}(Y, \mathbf{p}|_{p_i \to q}, \mathbf{c}) = f_{\&g}(Y^q, \mathbf{p}, \mathbf{c})$ *for all assignments* Y.

For $P' = P|_{e \to e'} \cup \{q(p_i, \mathbf{d}) \leftarrow p_i(\mathbf{d}) \mid p_i(\mathbf{d}) \in A(P)\}$*,* $\mathcal{AS}(P)$ *and* $\mathcal{AS}(P')$ *coincide, modulo atoms* $q(\cdot)$.

We now come to the actual inlining. Observe that Definitions 6 and 7 are *modular* in the sense that inlining external atoms E in a program P affects only the rules of P containing some of E and adds additional rules, but does not change the remaining rules. If $X \triangle Y = (X \setminus Y) \cup (Y \setminus X)$ denotes the symmetric difference between sets X and Y, one can formally show:

Lemma 2 *For a* HEX-*program* P *and a set of (positive or negative) external atoms* E *in* P*, we have* $P \triangle P_{[E]} = \{r \in P \mid$ *none of* E *occur in* $r\}$.

This equips us to turn to our main goal of characterizing equivalence of HEX-programs. If programs P and Q are $\langle \mathcal{H}, \mathcal{B} \rangle$-equivalent, then $P \cup R$ and $Q \cup R$ have the same answer sets for all ordinary ASP-programs $R \in \mathcal{P}_{\langle \mathcal{H}, \mathcal{B} \rangle}$. We will show that equivalence holds in fact even for HEX-programs $R \in \mathcal{P}^e_{\langle \mathcal{H}, \mathcal{B} \rangle}$. To this end, assume that P and Q are $\langle \mathcal{H}, \mathcal{B} \rangle$-equivalent for some $\mathcal{H}$ and $\mathcal{B}$ and let $R \in \mathcal{P}^e_{\langle \mathcal{H}, \mathcal{B} \rangle}$.

We apply a transformation to all (positive or negative) external atoms E in $P \cup R$ and $Q \cup R$ which appear in the R part, *but not in the* P *part or* Q *part*. To this end, observe that if the same external atom e is shared between P and R resp. Q and R, the restriction of the inlining to the R part is still possible by standardizing external atoms in R apart from those in P and Q, e.g., by introducing a copy of the external predicate; we assume in the following that external atoms have been standardized apart as needed. The transformation is then given as follows:

(1) rename their input parameters using Lemma 1; and

(2) subsequently inline them by applying Definition 6.

[6] Input atoms to external atoms must also be in $\mathcal{B}$ as they appear in bodies of our rewriting by Lemma 1 below.

Let R' denote the result after applying the two steps to R. Note that neither of the two steps modifies P or Q: for (1) this is by construction of the modified program in Lemma 1, for (2) this follows from Lemma 2. As observable from Lemma 1 and Definition 6, head atoms $H(R')$ in R' come either from $H(R)$ or are newly introduced auxiliary atoms; the renaming as by Lemma 1 prohibits that $H(R')$ contains input atoms to external atoms in R. Body atoms $B(R')$ in R' come either from $B(R)$, from input atoms to external atoms in R (see rules (2)), or are newly introduced auxiliary atoms. Since $R \in \mathcal{P}^e_{\langle \mathcal{H},\mathcal{B}\rangle}$, this implies that $H(R') \subseteq \mathcal{H} \cup \mathcal{H}'$ and $B(R') \subseteq \mathcal{B} \cup \mathcal{B}'$, where $\mathcal{H}'$ and $\mathcal{B}'$ are newly introduced auxiliary atoms. Since the auxiliary atoms do not occur in P and Q, by Corollary 2 they do not harm equivalence, i.e., $\langle \mathcal{H},\mathcal{B}\rangle$-equivalence implies $\langle \mathcal{H} \cup \mathcal{H}', \mathcal{B} \cup \mathcal{B}'\rangle$-equivalence. Thus, $\langle \mathcal{H},\mathcal{B}\rangle$-equivalence of P and Q implies that $P \cup R'$ and $Q \cup R'$ have the same answer sets.

The claim follows then from the observation that, due to Lemma 1 and soundness and completeness of inlining (cf. Proposition 1), $P \cup R$ and $Q \cup R$ have the same answer sets whenever $P \cup R'$ and $Q \cup R'$ have the same answer sets.

Example 13 *Consider the programs*

$$\begin{aligned}P =&\{a \leftarrow \&neg[b]();\ b \leftarrow \&neg[a]();\ a \leftarrow b\}\\ Q =&\{a \vee b;\ a \leftarrow b\}\end{aligned}$$

and let $\mathcal{H} = \{a, c\}$ and $\mathcal{B} = \{b\}$. Note that $P \equiv_{\langle \mathcal{H},\mathcal{B}\rangle} Q$. To observe this result, recall that we know $P \equiv_{\langle \{a,b\},\{b\}\rangle} Q$ from Example 11, which implies $P \equiv_{\langle \{a\},\{b\}\rangle} Q$. As $c \notin A(P)$, $c \notin A(Q)$, Proposition 8 further implies that $P \equiv_{\langle \{a,c\},\{b\}\rangle} Q$.

Let $R = \{c \leftarrow \&neg[b]()\} \in \mathcal{P}^e_{\langle \mathcal{H},\mathcal{B}\rangle}$. Renaming the input predicate of $\&neg[b]()$ by step (1) yields the program $\{q(b) \leftarrow b;\ c \leftarrow \&neg'[q]()\}$. After step (2) we have:

$$\begin{aligned}R' = \{&q(b) \leftarrow b;\ c \leftarrow x_e;\ x_e \leftarrow \overline{q(b)};\ \overline{x_e} \leftarrow not\, x_e\\ &\overline{q(b)} \leftarrow not\, q(b);\ \overline{q(b)} \leftarrow x_e;\ q(b) \vee \overline{q(b)} \leftarrow x_e\}\end{aligned}$$

Here, rule $q(b) \leftarrow b$ comes from step (1), $c \leftarrow x_e$ represents the rule in R, and the remaining rules from inlining in step (2). Except for new auxiliary atoms, we have that $H(R')$ use only atoms from $\mathcal{H}$ and $B(R')$ only atoms from $B(R')$. One can check that $P \cup R'$ and $Q \cup R'$ have the same (unique) answer set $\{a, c, \overline{x_e}, \overline{q(b)}\}$, which corresponds to the (same) unique answer set $\{a, c\}$ of $P \cup R$ and $Q \cup R$, respectively.

One can then show that equivalence wrt. program extensions that contain external atoms is characterized by the same criterion as extensions with ordinary ASP-programs only.

Proposition 9 *For sets $\mathcal{H}$ and $\mathcal{B}$ of atoms and* HEX*-programs P and Q, we have $P \equiv^e_{\langle \mathcal{H},\mathcal{B}\rangle} Q$ iff $\sigma_{\langle \mathcal{H},\mathcal{B}\rangle}(P) = \sigma_{\langle \mathcal{H},\mathcal{B}\rangle}(Q)$.*

Finally, for the Herbrand base $HB_{\mathcal{C}}(P)$ of all atoms constructible from the predicates in P and the constants $\mathcal{C}$, strong equivalence (Lifschitz et al. 2001) corresponds to the special case of $\langle HB_{\mathcal{C}}(P), HB_{\mathcal{C}}(P)\rangle$-equivalence, and uniform equivalence (Eiter and Fink 2003) corresponds to $\langle HB_{\mathcal{C}}(P), \emptyset\rangle$-equivalence; this follows directly from definition of strong resp. uniform equivalence.

6 Inconsistency of HEX-Programs

We turn now to inconsistency of HEX-programs. Similarly to equivalence, we want to characterize inconsistency wrt. program extensions. Akin to equivalence, sets $\mathcal{H}$ and $\mathcal{B}$ constrain the atoms that may be occur in rule heads, rule bodies and input atoms to external atoms of the added program, respectively. In contrast to equivalence, the criterion naturally concerns only a single program. However, we are still able to derive the criterion from the above results.

Deriving a Criterion for Inconsistency. We formalize our envisaged notion of inconsistency as follows:

Definition 12 *A* HEX*-program P is called* persistently inconsistent *wrt. sets of atoms $\mathcal{H}$ and $\mathcal{B}$, if $P \cup R$ is inconsistent for all $R \in \mathcal{P}^{e}_{\langle \mathcal{H},\mathcal{B} \rangle}$.*

Example 14 *The program $P = \{p \leftarrow \&neg[p]()\}$ is persistently inconsistent wrt. all $\mathcal{H}$ and $\mathcal{B}$ such that $p \notin \mathcal{H}$. This is because any model Y of P, and thus of $P \cup R$ for some $R \in \mathcal{P}^{e}_{\langle \mathcal{H},\mathcal{B} \rangle}$, must set p to true due to the rule $p \leftarrow \&neg[p]()$. However, $Y \setminus \{p\}$ is a model of $f(P \cup R)^Y$ if no rule in R derives p, hence Y is not a subset-minimal model of $f(P \cup R)^Y$.*

We start deriving a criterion by observing that a program $P_\perp$ is persistently inconsistent wrt. any $\mathcal{H}$ and $\mathcal{B}$ whenever it is classically inconsistent. Then $P_\perp \cup R$ does not even have classical models for any $R \in \mathcal{P}^{e}_{\langle \mathcal{H},\mathcal{B} \rangle}$, and thus it cannot have answer sets. For such a $P_\perp$, another program P is persistently inconsistent wrt. $\mathcal{H}$ and $\mathcal{B}$ iff it is $\langle \mathcal{H},\mathcal{B} \rangle^e$-equivalent to $P_\perp$; the latter can by Proposition 7 be checked by comparing their $\langle \mathcal{H},\mathcal{B} \rangle$-models. This allows us to derive the desired criterion in fact as a special case of the one for equivalence.

Classically inconsistent programs do not have $\langle \mathcal{H},\mathcal{B} \rangle$-models due to violation of Property (i) of Definition 10. Therefore, checking for persistent inconsistency works by checking if P does not have $\langle \mathcal{H},\mathcal{B} \rangle$-models either. To this end, it is necessary that each classical model Y of P violates Property (i) of Definition 10, otherwise (Y, Y) (and possibly (X, Y) for some $X \subsetneq Y$) would be $\langle \mathcal{H},\mathcal{B} \rangle$-models of P. Formally:

Proposition 10 *A* HEX*-program P is persistently inconsistent wrt. sets of atoms $\mathcal{H}$ and $\mathcal{B}$ iff for each classical model Y of P there is an $Y' \subsetneq Y$ such that $Y' \models fP^Y$ and $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$.*

Example 15 (cont'd) *For the program P from Example 14 we have for each classical model $Y \supseteq \{p\}$ holds. However, for each such Y we have that $Y' = Y \setminus \{p\}$ is a model of fP^Y, $Y' \subsetneq Y$ and $Y|_{\mathcal{H}} = Y'|_{\mathcal{H}}$, which proves that $P \cup R$ is inconsistent for all $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B} \rangle}$.*

Example 16 *Consider the program $P = \{a \leftarrow \&aOrNotB[a, b](); \leftarrow a\}$. It is persistently inconsistent wrt. all $\mathcal{H}$ and $\mathcal{B}$ such that $b \notin \mathcal{H}$. This is the case because the rule $a \leftarrow \&aOrNotB[a, b]()$ derives a whenever b is false, which violates the constraint $\leftarrow a$. Formally, one can observe that we have $a \notin Y$ and $b \in Y$ for each classical model Y of P. But then $Y' = Y \setminus \{b\}$ is a model of fP^Y, $Y' \subsetneq Y$ and $Y|_{\mathcal{H}} = Y'|_{\mathcal{H}}$.*

The criterion for inconsistency follows therefore as a special case from the criterion for program equivalence.

Applying the Criterion using Unfounded Sets. Proposition 10 formalizes a condition for deciding persistent inconsistency based on models of the program's reduct. However, practical implementations usually do not explicitly generate the reduct, but are often based on *unfounded sets* (Faber 2005). For a model Y of a program P, smaller models $Y' \subsetneq Y$ of the reduct fP^Y and unfounded sets of P wrt. Y correspond to each other one-by-one. This allows us to transform the above decision criterion such that it can be directly checked using unfounded sets.

We use unfounded sets for logic programs as introduced by Faber (2005) for programs with arbitrary aggregates.

Definition 13 (Unfounded Set) *Given a program P and an assignment Y, let U be any set of atoms appearing in P. Then, U is an* unfounded set for P wrt. Y *if, for each $r \in P$ with $H(r) \cap U \neq \emptyset$, at least one of the following holds:*

(i) some literal of $B(r)$ is false wrt. Y; or

(ii) some literal of $B(r)$ is false wrt. $Y \setminus U$; or

(iii) some atom of $H(r) \setminus U$ is true wrt. Y.

Lemma 3 *For a* HEX*-program P and a model Y of P, a set of atoms U is an unfounded set of P wrt. Y iff $Y \setminus U \models fP^Y$.*

By contraposition, the lemma implies that for a model Y of P and a model $Y' \subseteq Y$ of fP^Y we have that $Y \setminus Y'$ is an unfounded set of P wrt. Y. This allows us to restate our decision criterion as follows:

Corollary 3 *A* HEX*-program P is persistently inconsistent wrt. sets of atoms $\mathcal{H}$ and $\mathcal{B}$ iff for each classical model Y of P there is a nonempty unfounded set U of P wrt. Y s.t. $U \cap Y \neq \emptyset$ and $U \cap \mathcal{H} = \emptyset$.*

Example 17 (cont'd) *For the program P from Example 16 we have that $U = \{b\}$ is an unfounded set of P wrt. any classical model Y of P; by assumption $b \notin \mathcal{H}$ we have $U \cap \mathcal{H} = \emptyset$.*

Application. We now want to discuss a specific use-case of the decision criterion for program inconsistency. However, we stress that this section focuses on the study of the criterion, which is interesting by itself, while a detailed realization of the application is beyond its scope and discussed in more detail by Redl (2017a).

The state-of-the-art evaluation approach for HEX-programs makes use of *program splitting* for handling programs with variables. That is, the overall program is partitioned into components which are arranged in an *acyclic graph*. Then, beginning from the components without predecessors, each component is separately grounded and solved, and each answer set is one-by-one added as facts to the successor components. The process is repeated in a recursive manner such that eventually the leaf components will yield the final answer sets, cf. Eiter et al. (2016).

The main reason for program splitting is that *value invention*, i.e., the introduction of constants by external sources which do not occur in the input program, may lead to a grounding bottleneck if evaluated as monolithic program. This is because the grounder needs to evaluate external atoms under all possible inputs in order to ensure that all possible outputs are respected in the grounding, as demonstrated by the following example.

Example 18 *Consider the program*

$$\begin{aligned} P = \{ & r_1\colon \mathit{in}(X) \vee \mathit{out}(X) \leftarrow \mathit{node}(X) \\ & r_2\colon \leftarrow \mathit{in}(X), \mathit{in}(Y), \mathit{edge}(X,Y) \\ & r_3\colon \mathit{size}(S) \leftarrow \&\mathit{count}[\mathit{in}](S) \} \end{aligned}$$

where facts over $\mathit{node}(\cdot)$ and $\mathit{edge}(\cdot)$ define a graph. Then r_1 and r_2 guess an independent set and r_3 computes its size. The grounder must evaluate $\&\mathit{count}$ under all exponentially many possible extensions of in in order to instantiate rule r_3 for all relevant values of variable S.

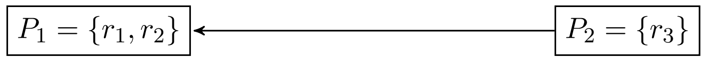


Figure 1: Evaluation of P from Example 18 based on program splitting

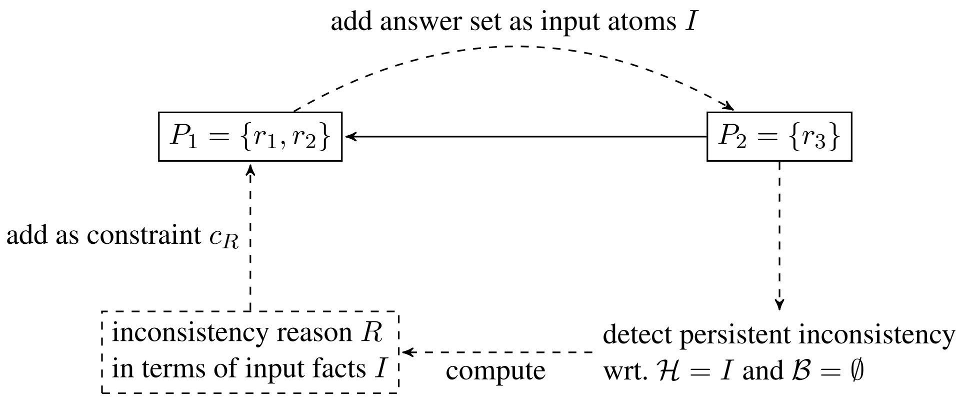


Figure 2: Exploiting persistent inconsistency for search space pruning

In this example, program splitting allows for avoiding unnecessary evaluations. To this end, the program might be split into $P_1 = \{r_1, r_2\}$ and $P_2 = \{r_3\}$ as illustrated in Figure 1. Then the state-of-the-art algorithm grounds and solves P_1, which computes all independent sets, and for each of them P_2 is grounded and solved.

Since the number of independent sets can be exponentially smaller than the set of all node selections, the grounding bottleneck can be avoided. However, program splitting has the disadvantage that nogoods learned from conflict-driven algorithms (Gebser et al. 2012) cannot be effectively propagated through the whole program, but only within a component.

The results from Section 6 can be used to identify a program component as persistently inconsistent wrt. possible input facts from the predecessor component. This information might be used to construct a constraint which describes the reason R for this inconsistency in terms of the input facts, which can be added as constraint c_R to predecessor components in order to eliminate assignments earlier, which would make a successor component inconsistent anyway. The idea is visualized in Figure 2.

For details about the computation of inconsistency reasons, exploiting them for the evaluation and experiments we refer to Redl (2017a).

7 Discussion and Conclusion

Related Work. Our external source inlining approach is related to evaluation approaches for DL-programs (Eiter et al. 2008), i.e., programs with ontologies, cf. Heymans et al. (2010), Xiao and Eiter (2011) and Bajraktari et al. (2017), but it is more general. The former approaches are specific for embedding (certain types of) description logic ontologies. In contrast, ours is generic and can handle arbitrary external sources as long as they are decidable and have finite output for each input (cf. Section 2).

Our rewriting uses the saturation technique and is in this respect also related to the one by Alviano et al. (2015) (cf. also Alviano (2016)), who translated nonmonotonic (cyclic) aggregates to disjunctions. However, an important difference to our approach are that they support only a fixed set of traditional aggregates (such as minimum, maximum, etc) whose semantics is directly exploited in a hard-coded fashion in their rewriting, while our approach is generic and thus more flexible. Moreover, their rewriting still uses simplified (monotonic) aggregates in the resulting rewritten program while we go a step further and eliminate external atoms altogether. This allows the resulting program to be directly forwarded to an ordinary ASP solver, while support for aggregates of any kind or additional compatibility checks of guesses are not required.

Based on this inlining approach, we further provided a characterization of equivalence of HEX-programs. The criteria generalize previous results for ordinary ASP by Woltran (2008). This is a convenient result, but it is not immediate due to possibly nonmonotonic external atoms and the use of the FLP- instead of the GL-reduct. Strong (Lifschitz et al. 2001) and uniform equivalence (Eiter and Fink 2003) are well-knwon and important special cases thereof and carry over as well.

Woltran (2004) also discussed the special cases of *head-relativized* equivalence ($\mathcal{H} = HB_{\mathcal{C}}(P)$ while $\mathcal{B}$ can be freely chosen), and *body-relativized* equivalence ($\mathcal{B} = HB_{\mathcal{C}}(P)$ while $\mathcal{H}$ can be freely chosen). Also the cases where $\mathcal{B} \subseteq \mathcal{H}$ and $\mathcal{H} \subseteq \mathcal{B}$ were analyzed. Corollaries have been derived which simplify the conditions to check for these special cases. They all follow directly from an analogous version of Proposition 9 for plain ASP by substituting $\mathcal{H}$ or $\mathcal{B}$ by a fixed value. Since we established by Proposition 9 that the requirements hold also for HEX-programs, their corollaries, as summarized in Section 5 by Woltran (2008), hold analogously.

The work is also related to the one by Truszczyński (2010), who extended strong equivalence to propositional theories under FLP-semantics. However, the relationship concerns only the use of the FLP-semantics, while the notion of equivalence and the formalism for which the equivalencei is shown are different. In particular, $\langle \mathcal{H}, \mathcal{B} \rangle$-equivalence and external sources were not considered.

Conclusion and Outlook. We presented an approach for external source inlining based on support sets. Due to nonmonotonicity of external atoms, the encoding is not trivial and requires a saturation encoding. We note that the results are interesting beyond HEX-programs since well-known ASP extensions, such as programs with aggregates (Faber et al. 2011) or with specific external atoms such as constraint atoms (Gebser et al. 2009), amount to special cases of HEX, and thus the results are applicable in such cases.

One application of the technique can be found in an alternative evaluation approach, which is intended to be used for external sources which have a compact representation as support sets. Previous approaches had to guess the truth values of external atoms and verify the guesses either by explicit evaluation (as in the traditional approach) or by matching guesses against support sets (as in the approach by Eiter et al. (2014)). Instead, the new inlining-based approach compiles external atoms away altogether such that the program can be entirely evaluated by an ordinary ASP solver. For the considered class of external sources, our experiments show a clear and significant improvement over the previous support-set-based approach by Eiter et al. (2014), which is explained by the fact that the slightly higher initialization costs are exceeded by the significant benefits of avoiding external calls altogether, and for the considered types of external sources also over the traditional approach.

Another application is found in the extension of previous notions of program equivalence from ordinary ASP to HEX-programs. We generalizes such notions from ordinary ASP to HEX-programs. Since this is a theoretical result, compact representation of external sources is not an issue here. From the criterion for program equivalence we derive further criteria for program inconsistency wrt. program extensions, which have applications in context of evaluation algorithms for HEX-programs.

Potential future work includes refinements of the rewriting. Currently, a new auxiliary variable $\overline{a}$ is introduced for all input atoms a of all external atoms. Thus, a quadratic number of auxiliary atoms is required. While the reuse of the auxiliary variables is not always possible, the identification of cases were auxiliary variables can be shared among multiple inlined external atoms is interesting. For the equivalence criteria, future work may also include the extension of the results to non-ground programs, cf. Eiter et al. (2005).

Moreover, currently we do not distinguish between body atoms and input atoms to external atoms when we define which programs are allows to be added. A more fine-grained approach which supports this distinction may allow for identifying programs as equivalent which are not equivalent wrt. to the current notion. Also allowing only external atoms with specific properties, such as monotonicity, may lead to more fine-grained criteria.

Furthermore, a recent alternative notion of equivalence is *rule equivalence* (Bliem and Woltran 2016). Here, not the set of atoms which can occur in the added program is constrained, but the type of the rules. In particular, proper rules may be added, while the addition of facts is limited to certain atoms; generalizing this notion to HEX-programs is a possible starting point for future work.

References

[Alviano 2016] ALVIANO, M. 2016. Evaluating answer set programming with non-convex recursive aggregates. *Fundamentae Informaticae 149,* 1-2, 1–34.

[Alviano et al. 2015] ALVIANO, M., FABER, W., AND GEBSER, M. 2015. Rewriting recursive aggregates in answer set programming: back to monotonicity. *CoRR abs/1507.03923*, 559–573.

[Bajraktari et al. 2017] BAJRAKTARI, L., ORTIZ, M., AND SIMKUS, M. 2017. Clopen knowledge bases: Combining description logics and answer set programming. In *Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017.*, A. Artale, B. Glimm, and R. Kontchakov, Eds. CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org.

[Baumann et al. 2017] BAUMANN, R., DVOŘÁK, W., LINSBICHLER, T., AND WOLTRAN, S. 2017. A general notion of equivalence for abstract argumentation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, C. Sierra, Ed. ijcai.org, 800–806.

[Bliem and Woltran 2016] BLIEM, B. AND WOLTRAN, S. 2016. Equivalence between answer-set programs under (partially) fixed input. In *FoIKS*. Lecture Notes in Computer Science, vol. 9616. Springer, 95–111.

[Calvanese et al. 2007] CALVANESE, D., LEMBO, D., LENZERINI, M., AND ROSATI, R. 2007. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning 39,* 3 (October), 385–429.

[Darwiche and Marquis 2002] DARWICHE, A. AND MARQUIS, P. 2002. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR) 17*, 229–264.

[Eiter and Fink 2003] EITER, T. AND FINK, M. 2003. Uniform equivalence of logic programs under the stable model semantics. In *Logic Programming, 19th International Conference, ICLP 2003, Mumbai, India, December 9-13, 2003, Proceedings*, C. Palamidessi, Ed. Lecture Notes in Computer Science, vol. 2916. Springer, 224–238.

[Eiter et al. 2016] EITER, T., FINK, M., IANNI, G., KRENNWALLNER, T., REDL, C., AND SCHÜLLER, P. 2016. A model building framework for answer set programming with external computations. *TPLP 16,* 4, 418–464.

[Eiter et al. 2012] EITER, T., FINK, M., KRENNWALLNER, T., AND REDL, C. 2012. Conflict-driven ASP solving with external sources. *TPLP 12,* 4-5, 659–679.

[Eiter et al. 2016] EITER, T., FINK, M., KRENNWALLNER, T., AND REDL, C. 2016. Domain expansion for ASP-programs with external sources. *Artificial Intelligence 233*, 84–121.

[Eiter et al. 2014] EITER, T., FINK, M., KRENNWALLNER, T., REDL, C., AND SCHÜLLER, P. 2014. Efficient HEX-program evaluation based on unfounded sets. *Journal of Artificial Intelligence Research 49*, 269–321.

[Eiter et al. 2014] EITER, T., FINK, M., REDL, C., AND STEPANOVA, D. 2014. Exploiting support sets for answer set programs with external evaluations. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, C. E. Brodley and P. Stone, Eds. AAAI Press, 1041–1048.

[Eiter et al. 2014] EITER, T., FINK, M., AND STEPANOVA, D. 2014. Towards practical deletion repair of inconsistent dl-programs. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*. ECAI'14. IOS Press, Amsterdam, The Netherlands, The Netherlands, 285–290.

[Eiter et al. 2005] EITER, T., FINK, M., TOMPITS, H., AND WOLTRAN, S. 2005. Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, M. M. Veloso and S. Kambhampati, Eds. AAAI Press / The MIT Press, 695–700.

[Eiter et al. 2009] EITER, T., IANNI, G., AND KRENNWALLNER, T. 2009. Answer Set Programming: A Primer. In *5th International Reasoning Web Summer School (RW 2009), Brixen/Bressanone, Italy, August 30–September 4, 2009*, S. Tessaris, E. Franconi, T. Eiter, C. Gutierrez, S. Handschuh, M.-C. Rousset, and R. A. Schmidt, Eds. LNCS, vol. 5689. Springer, 40–110.

[Eiter et al. 2008] EITER, T., IANNI, G., LUKASIEWICZ, T., SCHINDLAUER, R., AND TOMPITS, H. 2008. Combining answer set programming with description logics for the semantic web. *Artif. Intell. 172,* 12-13, 1495–1539.

[Faber 2005] FABER, W. 2005. Unfounded sets for disjunctive logic programs with arbitrary aggregates. In *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005), Diamante, Italy, September 5-8, 2005*. Vol. 3662. Springer, 40–52.

[Faber et al. 2011] FABER, W., LEONE, N., AND PFEIFER, G. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence 175,* 1 (January), 278–298.

[Franco and Martin 2009] FRANCO, J. AND MARTIN, J. 2009. *A History of Satisfiability*. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, Chapter 1, 3–74.

[Gebser et al. 2012] GEBSER, M., KAUFMANN, B., AND SCHAUB, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence 187-188*, 52–89.

[Gebser et al. 2009] GEBSER, M., OSTROWSKI, M., AND SCHAUB, T. 2009. Constraint answer set solving. In *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, P. Hill and D. Warren, Eds. Lecture Notes in Computer Science, vol. 5649. Springer-Verlag, 235–249.

[Gelfond and Lifschitz 1988] GELFOND, M. AND LIFSCHITZ, V. 1988. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings of the 5th International Conference and Symposium*, R. Kowalski and K. Bowen, Eds. MIT Press, 1070–1080.

[Gelfond and Lifschitz 1991] GELFOND, M. AND LIFSCHITZ, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing 9,* 3–4, 365–386.

[Heymans et al. 2010] HEYMANS, S., EITER, T., AND XIAO, G. 2010. Tractable reasoning with dl-programs over datalog-rewritable description logics. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, H. Coelho, R. Studer, and M. Wooldridge, Eds. Frontiers in Artificial Intelligence and Applications, vol. 215. IOS Press, 35–40.

[Lembo et al. 2011] LEMBO, D., LENZERINI, M., ROSATI, R., RUZZI, M., AND SAVO, D. F. 2011. *Query Rewriting for Inconsistent DL-Lite Ontologies*. Springer Berlin Heidelberg, Berlin, Heidelberg, 155–169.

[Lifschitz et al. 2001] LIFSCHITZ, V., PEARCE, D., AND VALVERDE, A. 2001. Strongly equivalent logic programs. *ACM Trans. Comput. Logic 2,* 4 (Oct.), 526–541.

[Mayer et al. 2009] MAYER, W., STUMPTNER, M., BETTEX, M., AND FALKNER, A. 2009. On solving complex rack configuration problems using csp methods. In *Proceedings of the Workshop on Configuration at the 21st International Conference on Artificial Intelligence*, M. Stumptner and P. Albert, Eds. Pasadena, CA, USA, 53–60.

[Nieuwenhuis and Oliveras 2005] NIEUWENHUIS, R. AND OLIVERAS, A. 2005. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *In CAV'05 LNCS 3576*. Springer, 321–334.

[Ostrowski and Schaub 2012] OSTROWSKI, M. AND SCHAUB, T. 2012. ASP modulo CSP: The clingcon system. *Theory and Practice of Logic Programming*. To appear.

[Redl 2017a] REDL, C. 2017a. Conflict-driven ASP solving with external sources and program splits. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI 2017), August 19–25, 2017, Melbourne, Australia* (August 19–25, 2017). AAAI Press, 1239–1246.

[Redl 2017b] REDL, C. 2017b. Efficient evaluation of answer set programs with external sources based on external source inlining. In *Proceedings of the Thirty-First AAAI Conference (AAAI 2017), February 4–9, 2016, San Francisco, California, USA* (February 4–9, 2016). AAAI Press.

[Redl 2017c] REDL, C. 2017c. On equivalence and inconsistency of answer set programs with external sources. In *Proceedings of the Thirty-First AAAI Conference (AAAI 2017), February 4–9, 2016, San Francisco, California, USA* (February 4–9, 2016). AAAI Press.

[Truszczyński 2010] TRUSZCZYŃSKI, M. 2010. Reducts of propositional theories, satisfiability relations, and generalizations of semantics of logic programs. *Artificial Intelligence 174,* 16, 1285 – 1306.

[Woltran 2004] WOLTRAN, S. 2004. Characterizations for relativized notions of equivalence in answer set programming. In *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*, J. J. Alferes and J. A. Leite, Eds. Lecture Notes in Computer Science, vol. 3229. Springer, 161–173.

[Woltran 2008] WOLTRAN, S. 2008. A common view on strong, uniform, and other notions of equivalence in answer-set programming. *TPLP 8,* 2, 217–234.

[Xiao and Eiter 2011] XIAO, G. AND EITER, T. 2011. Inline evaluation of hybrid knowledge bases - phd description. In *Web Reasoning and Rule Systems - 5th International Conference, RR 2011, Galway, Ireland, August 29-30, 2011. Proceedings*, S. Rudolph and C. Gutierrez, Eds. Lecture Notes in Computer Science, vol. 6902. Springer, 300–305.

A Proofs

Proposition 1
For all HEX*-programs P, external atoms e in P and a positive complete family of support sets $\mathcal{S}_{\mathbf{T}}(e, P)$ such that $S^+_{\mathbf{T}} \cup \neg S^-_{\mathbf{T}} = I(e, P)$ for all $S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}(e, P)$, the answer sets of P are equivalent to those of $P_{[e]}$, modulo the atoms newly introduced in $P_{[e]}$.*

Proof. ($\Rightarrow$) Let Y be an answer set of P. We show that

$$
\begin{aligned}
Y' = \; & Y \cup \{\overline{a} \mid a \in I(e, P) \setminus Y\} \cup \{\overline{a} \mid a \in I(e, P), Y \models e\} \\
& \cup \{x_e \mid Y \models e\} \cup \{\overline{x_e} \mid Y \not\models e\}
\end{aligned}
$$

is an answer set of $P_{[e]}$.

- We first show that Y' is a model of $P_{[e]}$. If $Y \cup \{\overline{a} \mid a \in I(e, P) \setminus Y\}$ satisfies one of the rule bodies in (1), then $S_{\mathbf{T}} \subseteq Y$ and $(I(e, P) \setminus S^-_{\mathbf{T}}) \cap Y = \emptyset$ (if some $a \in I(e, P) \setminus S_{\mathbf{T}}$ would be in Y, then $\overline{a}$ would not be in $Y \cup \{\overline{a} \mid a \in I(e, P) \setminus Y\}$ and the rule body would not be satisfied) for some $S^+_{\mathbf{T}} \in \mathcal{S}^-_{\mathbf{T}}(e, P)$; this implies $Y \models e$ and, by construction, $x_e \in Y'$. If only Y' but not $Y \cup \{\overline{a} \mid a \in I(e, P) \setminus Y\}$ satisfies one of the rule bodies in (1), then additional atoms of kind $\overline{a}$ must be in Y', which are only added if $Y \models e$; this also implies, by construction, $x_e \in Y'$. Thus we have $x_e \in Y'$ whenever Y' satisfies one of the rule bodies in (1), and thus these rules are all satisfied. We further add $\overline{a}$ whenever $a \notin Y$ or x_e is added (due to $Y \models e$) for all $a \in I(e, P)$, which satisfies rules (2), and we add $\overline{x_e}$ whenever x_e is not added (due to $Y \not\models e$), thus the rule (3) is satisfied. Moreover, the rules in (4) are satisfied because Y is a model of P and the value of x_e under Y' coincides with the value of e under Y by construction.

- Suppose there is a smaller model $Y'_< \subsetneq Y'$ of $fP^{Y'}_{[e]}$ and assume that this $Y'_<$ is subset-minimal. We show that then, for the restriction $Y_<$ of $Y'_<$ to the atoms in P it holds that (i) $Y_<$ is a model of fP^Y and (ii) $Y_< \subsetneq Y$, which contradicts the assumption that Y is an answer set of P.

  (i) Suppose there is a rule $r \in fP^Y$ such that $Y_< \not\models r$. Observe that for $r' = r|_{e \to x_e}$ we have $r' \in fP^{Y'}_{[e]}$ because we have $Y \models B(r)$ (since $r \in fP^Y$) and $Y' \models x_e$ iff $Y \models e$ (by construction of Y'), which implies $Y' \models B(r')$. Moreover, since $Y'_< \models r'$, we either have $Y'_< \models H(r')$ or $Y'_< \not\models B(r')$. In the former case we also have $Y_< \models H(r)$, and thus $Y_< \models r$, because the two assignments resp. rules coincide on ordinary atoms in P; with the same argument $Y_< \models r$ holds also in the latter case if a body atom in $B(r') \setminus \{x_e\}$ is unsatisfied under $Y'_<$. Hence, $Y'_< \models r'$ and $Y_< \not\models r$ is only possible if $e \in B(r)$, $Y'_< \not\models x_e$, and $Y_< \models e$; the latter implies $Y'_< \models e$ as $Y_<$ and $Y'_<$ coincide on $I(e, P)$. Moreover, $Y' \models B(r')$ implies $Y' \models x_e$; by construction of Y' this further implies $\overline{x_e} \notin Y'$.

  Since x_e could not be false in $Y'_<$ if (at least) one of the rules $r_1, \dots, r_n$ in (1) would be in $P^{Y'}_{[e]}$ and had a satisfied body, for each r_i, $1 \leq i \leq n$, one of $Y' \not\models B(r_i)$ (then r_i is not even in $P^{Y'}_{[e]}$) or $Y'_< \not\models B(r_i)$ must hold; but since $Y'_< \subsetneq Y'$ and $B(r_i)$ consists only of positive atoms, the former case in fact implies the latter, thus $Y'_< \not\models B(r_i)$ must hold for all $1 \leq i \leq n$.

  Moreover, we have that $\overline{a} \in Y'_<$ whenever $a \notin Y'_<$ for all $a \in I(e, P)$. This is because $\overline{x_e} \notin Y'$ and thus $a \vee \overline{a} \leftarrow \mathit{not}\, \overline{x_e} \in P^{Y'}_{[e]}$ for all $a \in I(e, P)$ (cf. rules (2)) and $\overline{x_e} \notin Y'_<$; $a \notin Y'_<$ and $\overline{a} \notin Y'_<$ would violate this rule. But then $Y'_<$ does not fulfill any of the cases in which e is true, hence $Y'_< \not\models e$, which contradicts our previous observation, thus the initial assumption that $Y_< \not\models r$ is false and we have $Y_< \models P^Y$.

(ii) Finally, we show that $Y_< \subsetneq Y$, i.e., $Y' \setminus Y'_<$ contains not only atoms from $\{\overline{a} \mid a \in I(e,P)\} \cup \{x_e, \overline{x_e}\}$. We first consider the case $\overline{x_e} \in Y'$. Then $Y' \setminus Y'_<$ cannot contain x_e (because it is not even in Y' by construction) or $\overline{x_e}$ (because it would leave rule (3) unsatisfied). It further cannot contain any $\overline{a}$ because $Y'_<$ is assumed to be subset-minimal and thus contains $\overline{a}$ only if $a \notin Y'$ (and thus $a \notin Y'_<$); this is because $x_e \notin Y'$ and thus all rules in (2) which force $\overline{a}$ to be true, except $\overline{a} \leftarrow \mathit{not}\, a$, are dropped from $fP^{Y'}_{[e]}$; but then removal of any $\overline{a}$ would leave the rule $\overline{a} \leftarrow \mathit{not}\, a$ in (2), which is contained in $fP^{Y'}_{[e]}$, unsatisfied.

In case $x_e \in Y'$, if $Y' \setminus Y'_<$ contains only atoms from $\{\overline{a} \mid a \in I(e,P)\} \cup \{x_e, \overline{x_e}\}$, then it contains x_e (because $\overline{x_e} \notin Y'$ and all $\overline{a}$ for $a \in I(e,P)$ must be true whenever x_e is due to the rules in (2), which are all also in $P^{Y'}_{[e]}$). Moreover, we have that $\overline{a} \in Y'_<$ whenever $a \notin Y'_<$ because $a \vee \overline{a} \leftarrow \mathit{not}\, \overline{x_e} \in fP^{Y'}_{[e]}$ for all $a \in I(e,P)$ (cf. rules in (2)); $a \notin Y'_<$ and $\overline{a} \notin Y'_<$ would violate this rule. But then $Y'_<$ does not fulfill any of the cases in which e is true (otherwise $Y'_<$ would satisfy a body of (1), which would also be satisfied by $Y' \supsetneq Y'_<$, such that the rule would be in $P^{Y'}_{[e]}$ and x_e could not be false in $Y'_<$), hence $Y'_< \not\models e$; since $x_e \in Y'$ implies that $Y' \models e$ we have that $Y' \setminus Y'_<$ must contain at least one of $I(e,P)$ such that the truth values of e can differ under the two assignments, thus it does not only contain atoms from $\{\overline{a} \mid a \in I(e,P)\} \cup \{x_e, \overline{x_e}\}$.

($\Leftarrow$) Let Y' be an answer set of $P_{[e]}$. We show that its restriction Y to ordinary atoms in P is an answer set of P.

- We first show that Y is a model of P. It suffices to show that $Y' \models x_e$ iff $Y' \models e$. The if-direction is obvious as the rules in (1) together with (2) force x_e to be true whenever e is. For the only-if-direction, observe that if $Y' \not\models e$ but $x_e \in Y'$, then $Y' \setminus (\{x_e\} \cup \{\overline{a} \mid a \in Y'\}) \subsetneq Y'$ is a model of $fP^{Y'}_{[a]}$ because it does not satisfy any body in (1), which contradicts the assumption that Y' is an answer set of $P_{[e]}$.

- Suppose there is a smaller model $Y_< \subsetneq Y$ of fP^Y. We show by case distinction that also $fP^{Y'}_{[e]}$ has a smaller model than Y'.

  (a) Case $x_e \in Y'$:
  We show that $Y'_< = Y_< \cup \{\overline{a} \mid a \in I(e,P) \setminus Y_<\} \cup \{\overline{a} \mid a \in I(e,P), Y_< \models e\} \cup \{x_e \mid Y_< \models e\}$ is a model of $fP^{Y'}_{[e]}$ and that $Y'_< \subsetneq Y'$. For the rules in (1), if $Y_< \cup \{\overline{a} \mid a \in I(e,P) \setminus Y_<\}$ satisfies one of their bodies, then we have that $Y_< \models e$ and we set x_e to true, thus the rules are all satisfied. If $Y_< \cup \{\overline{a} \mid a \in I(e,P) \setminus Y_<\}$ does not satisfy one of their bodies but $Y'_<$ does, then the additional atoms in $Y'_<$ can only come from $\{\overline{a} \mid a \in I(e,P), Y_< \models e\}$, which implies $Y_< \models e$ (by construction) and thus $x_e \in Y_<$ also in this case. Hence, the rules in (1) are all satisfied. The construction satisfies also the rules in (2) because we set $\overline{a}$ to true whenever a is false or x_e is true in $Y'_<$ (due to $Y'_< \models e$). Rule (3) is not in $fP^{Y'}_{[e]}$ because $x_e \in Y'$ by assumption. For the rules $P|_{e \to x_e}$ in (4) satisfaction is given because $r \in fP^Y$ iff $r|_{e \to x_e} \in fP^{Y'}_{[e]}$ (since $Y \models e$ iff $Y' \models x_e$), and by construction of $Y'_<$, we set x_e to true iff $Y'_< \models e$.
  Now suppose $Y'_< \not\subsetneq Y'$. We have that $Y_< \subsetneq Y \subseteq Y'$ and that $Y' \setminus Y$ contains no atoms from $\{\overline{a} \mid a \in I(e,P)\} \cup \{x_e, \overline{x_e}\}$. Therefore, $Y'_< \not\subsetneq Y'$ implies that $Y'_<$ adds an atom from this set to

$Y_<$, which is not in Y', i.e., $Y' \setminus Y'_<$ contains an atom from $\{\overline{a} \mid a \in I(e, P)\} \cup \{x_e, \overline{x_e}\}$. But this is impossible since $x_e \in Y'$, thus we also have $\overline{a} \in Y'$ for all $a \in I(e, P)$, while $\overline{x_e} \notin Y'_<$ by construction.

Moreover, $Y'_< \subsetneq Y'$ because they differ in an atom other than $\{\overline{a} \mid a \in I(e, P)\} \cup \{x_e, \overline{x_e}\}$ due to $Y_< \subsetneq Y$.

(b) Case $\overline{x_e} \in Y'$:

We show that $Y'_< = Y_< \cup \{\overline{a} \mid a \in I(e, P) \setminus Y'\} \cup \{\overline{x_e}\}$ is a model of $fP^{Y'}_{[e]}$ and that $Y'_< \subsetneq Y'$.

The rules in (1) are all eliminated from $fP^{Y'}_{[e]}$ because $x_e \notin Y'$ implies $Y' \not\models e$ and thus Y' does not satisfy any body of the rules in (1); this is because due to minimality of Y' and falsehood of x_e, no $\overline{a}$ is set to true if a is already true in Y'. The rules in (2) are satisfied because for every $a \in I(e, P)$ we have that either $\overline{a} \leftarrow \mathit{not}\, a$ is not in $fP^{Y'}_{[e]}$ (if $a \in Y'$) or $\overline{a} \in Y'_<$ by construction (each such $\overline{a}$ is also in Y'). We further have $\overline{x_e} \in Y'_<$ by construction and thus rule (3) is satisfied. For the rules $r' \in f(P|_{e \to x_e})^{Y'}$ in (4), observe that there are corresponding rules $r \in fP^Y$, and that $Y'_<$ coincides with $Y_<$ on atoms other than x_e. If $Y_< \models r$ because $Y_< \models H(r)$ or $Y_< \not\models B(r) \setminus \{e\}$, then this implies $Y'_< \models r'$. If $Y_< \models r$ because $Y_< \not\models e$, then $Y'_< \models r'$ because $Y'_< \not\models x_e$ by construction of $Y'_<$.

Moreover, $Y'_< \subsetneq Y'$ because they differ in an atom other than $\{\overline{a} \mid a \in I(e, P)\} \cup \{x_e, \overline{x_e}\}$ due to $Y_< \subsetneq Y$.

□

Proposition 2
Let X be a set of atoms and P be a HEX*-program such that*

$$
\begin{aligned}
P \supseteq\, & \{r_1 \colon x_e \leftarrow B, b; r_2 \colon x_e \leftarrow B, \overline{b}\} \\
& \cup \{\overline{a} \leftarrow \mathit{not}\, a;\ \overline{a} \leftarrow x_e;\ a \vee \overline{a} \leftarrow \mathit{not}\, \overline{x_e} \mid a \in X\} \\
& \cup \{\overline{x_e} \leftarrow \mathit{not}\, x_e\}
\end{aligned}
$$

where $B \subseteq \{a, \overline{a} \mid a \in X\}$, $b \in X$, and $\overline{x_e}$ occurs only in the rules explicitly shown above. Then P is equivalent to $P' = (P \setminus \{r_1, r_2\}) \cup \{r \colon x_e \leftarrow B\}$.

Proof. We have to show that an assignment Y is an answer set of P iff it is an answer set of P'. It suffices to restrict the discussion to $r_1, r_2 \in P$ and the corresponding rule $r \in P'$ because the other rules in P vs. P' and their reducts P^Y vs. P'^Y wrt. a fixed assignment Y coincide.

($\Rightarrow$) Let Y be an answer set of P. We first show that $Y \models P'$. It suffices show that $Y \models r$. Towards a contradiction, suppose $Y \not\models x_e$ and $Y \models B$. Since we have (at least) one of $b \in Y$ or $\overline{b} \in Y$ (otherwise Y could not satisfy the rule $\overline{b} \leftarrow \mathit{not}\, b \in P$), we also have $Y \not\models r_1$ or $Y \not\models r_2$, which is impossible because Y is an answer set of P.

Thus $Y \models P'$. Towards a contradiction, suppose there is a smaller model $Y_< \subsetneq Y$ of fP'^Y. If $r \notin fP'^Y$ then $Y \not\models B(r)$, which implies that $Y \not\models B(r_1)$ and $Y \not\models B(r_2)$, and thus neither r_1 nor r_2 is in fP^Y. Otherwise, since $Y_< \models fP'^Y$ we have $Y_< \models r$ and thus either $Y_< \models x_e$ or $Y_< \not\models B$. But in both cases also $Y_< \models r_1$ and $Y_< \models r_2$, thus $Y_< \models P^Y$, which contradicts the assumption that Y is an answer set of P.

($\Leftarrow$) Let Y be an answer set of P'. We immediately get $Y \models P$ because $Y \models r$ and r_1 and r_2 are even easier to satisfy than r.

Towards a contradiction, suppose there is a smaller model $Y_< \subsetneq Y$ of fP^Y. If $Y \not\models B$ we have that $r \notin fP'^Y$ and thus $Y_< \models fP'^Y$, which contradicts the assumption that Y is an answer set of P'.

Then $Y \models B$ and we have that $r \in fP'^Y$ and have to show that $Y_< \models r$.

If $Y \models \overline{x_e}$ then $Y \not\models x_e$ (otherwise $Y \setminus \{\overline{x_e}\} \models fP'^Y$, contradicting our assumption that Y is an answer set of P'). Moreover, due to the rule $\overline{b} \leftarrow \mathit{not}\, b$, one of b or $\overline{b}$ must be true in Y. But then Y cannot not satisfy both r_1 and r_2, hence $Y \not\models \overline{x_e}$.

Then $Y \models x_e$ (since $Y \models \overline{x_e} \leftarrow \mathit{not}\, x_e$). If $Y_< \models x_e$ or $Y_< \not\models B$ then also $Y_< \models r$ and we are done. Otherwise, since $Y_< \models fP^Y$, we have (i) either $Y \not\models b$ (thus $r_1 \notin fP^Y$) or $Y_< \not\models b$ (thus $Y_< \models r_1$), where the former case implies the latter since $Y_< \subsetneq Y$, and (ii) either $Y \not\models \overline{b}$ (thus $r_2 \notin fP^Y$) or $Y_< \not\models \overline{b}$ (thus $Y_< \models r_2$), where the former case implies the latter since $Y_< \subsetneq Y$. Thus, we have in any case both $Y_< \not\models b$ and $Y_< \not\models \overline{b}$. But then $Y_< \not\models b \vee \overline{b} \leftarrow \mathit{not}\, \overline{x_e}$, and since this rule is in fP^Y because $Y \not\models \overline{x_e}$, we get $Y_< \not\models fP^Y$. This contradicts our initial assumption that fP^Y has a smaller model than Y, hence Y is an answer set. □

Corollary 1
For all HEX*-programs P, external atoms e in P and a positive complete family of support sets $\mathcal{S}_{\mathbf{T}}(e, P)$, the answer sets of P are equivalent to those of $P_{[e]}$, modulo the atoms newly introduced in program $P_{[e]}$.*

Proof. First apply Proposition 2 and then Proposition 1. □

Proposition 3
For all HEX*-programs P, negated external atoms $\mathit{not}\, e$ in P and a negative complete family of support sets $\mathcal{S}_{\mathbf{F}}(e, P)$, the answer sets of P are equivalent to those of $P_{[\mathit{not}\, e]}$, modulo the atoms newly introduced in program $P_{[\mathit{not}\, e]}$.*

Proof. Using a negative complete family of support sets for defining the auxiliary variable x_e in the rules (5), and replacing $\mathit{not}\, e$ by x_e amounts to the replacement of $\mathit{not}\, e$ by a new external atom e', any applying the rewriting from Definition 6 afterwards. □

Proposition 4
Let $\mathcal{S}_\sigma$ be a positive resp. negative complete family of support sets for some external atom e in a program P, where $\sigma \in \{\mathbf{T}, \mathbf{F}\}$. Then $\mathcal{S}_{\overline{\sigma}} = \{S_{\overline{\sigma}} \in \prod_{S_\sigma \in \mathcal{S}_\sigma} \neg S_\sigma \mid S_{\overline{\sigma}}$ is consistent$\}$ is a negative resp. positive complete family of support sets, where $\overline{\mathbf{T}} = \mathbf{F}$ and $\overline{\mathbf{F}} = \mathbf{T}$.

Proof. We restrict the proof to the case $\sigma = \mathbf{T}$; the case $\sigma = \mathbf{F}$ is symmetric.

If $\mathcal{S}_{\mathbf{T}}$ is a positive complete family of support sets, then the support sets $S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}$ describe the possibilities to satisfy e exhaustively. Thus, in order to falsify e, at least one literal of each $S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}$ must be falsified, i.e., at least one literal in $\neg S_{\mathbf{T}}$ must be satisfied. Thus amounts to the Cartesian product of all sets $\neg S_{\mathbf{T}}$ with $S_{\mathbf{T}} \in \mathcal{S}_{\mathbf{T}}$. □

Proposition 5
Let P and Q be HEX*-programs, R be an ordinary ASP-program, and Y be an assignment s.t. $Y \in \mathcal{AS}(P \cup R)$ but $Y \notin \mathcal{AS}(Q \cup R)$. Then there is also a positive ordinary ASP-program R' such that $Y \in \mathcal{AS}(P \cup R')$ but $Y \notin \mathcal{AS}(Q \cup R')$ and $B(R') \subseteq B(R)$ and $H(R') \subseteq H(R)$.*

Proof. Let P and Q be HEX-programs, R be an ordinary ASP-program, and Y be an assignment such that $Y \in \mathcal{AS}(P \cup R)$ but $Y \notin \mathcal{AS}(Q \cup R)$. We have to show that there is a positive R' such that $Y \in \mathcal{AS}(P \cup R')$ but $Y \notin \mathcal{AS}(Q \cup R')$. As Woltran (2008), we show this in particular for $R' = R^Y$, where $R^Y = \{H(r) \leftarrow B^+(r) \mid r \in R, Y \not\models B^-(r)\}$ is the GL-reduct (Gelfond and Lifschitz 1988), not to be confused with the FLP-reduct which is used in the definition of the HEX-semantics. Obviously we have $B(R') \subseteq B(R)$ and $H(R') \subseteq H(R)$.

- We first show that $Y \in \mathcal{AS}(P \cup R')$. For modelhood, we know that Y is a model of P, thus it suffices to discuss R'. Let $r' \in R'$. Then there is a corresponding rule $r \in R$ such that r' is the only rule in $\{r\}^Y$. We have that $Y \not\models B^-(r)$, otherwise r' would not be in $\{r\}^Y$. But then, since $Y \models r$ (because $Y \models R$ since $Y \in \mathcal{AS}(P \cup R)$ by assumption), we have that $Y \models H(r)$ or $Y \not\models B^+(r)$, which implies that $Y \models r'$.

  It remains to show that there is no $Y' \subsetneq Y$ such that $Y' \models f(P \cup R')^Y$. Towards a contradiction, suppose there is such an Y'; we show that it is also a model of $f(P \cup R)^Y$, which contradicts the assumption that Y is an answer set of $P \cup R$. Obviously we have $Y' \models fP^Y$. Now consider $r \in fR^Y$. Then $Y \models B^+(r)$ and $Y \not\models B^-(r)$. But then $H(r) \leftarrow B^+(r) \in R'$ and $H(r) \leftarrow B^+(r) \in fR'^Y$. Since $Y' \models fR'^Y$, we have that $Y' \models H(r)$ or $Y' \not\models B^+(r)$ and thus $Y' \models r$. Since this holds for all $r \in fR^Y$ this implies $Y' \models f(P \cup R)^Y$, which contradicts the assumption that Y is an answer set of $P \cup R$, thus Y' cannot exist and Y is an answer set of $P \cup R'$.

- We now show that $Y \notin \mathcal{AS}(Q \cup R')$. If $Y \not\models Q \cup R$ then also $Y \not\models Q \cup R'$ because for each $r \in R$ we either have that $Y \models B^-(r)$ (and thus r is not relevant for the inconsistency of $Q \cup R$) or R' contains $H(r) \leftarrow B^+(r)$ instead, which is even harder to satisfy (i.e., is violated whenever r is).

  If $Y \models Q \cup R$ then there is an $Y' \subsetneq Y$ such that $Y' \models f(Q \cup R)^Y$. We show that Y' is also a model of $f(Q \cup R')^Y$. Towards a contradiction, suppose there is an $r' \in f(Q \cup R')^Y$ such that $Y' \not\models r'$. Then r' must be in fR'^Y because if it would be in fQ^Y then Y' could not be a model of $f(Q \cup R)^Y$. Then $Y' \not\models H(r')$ but $Y' \models B^+(r')$. But then there is a rule $r \in fR^Y$ with $H(r) = H(r')$ and $B^+(r) = B^+(r')$ such that $Y' \not\models B^-(r)$ (otherwise $Y \models B^-(r)$ and r' could not be in R' and thus also not in $f(Q \cup R')^Y$). However, then $Y' \not\models r$ and thus $Y' \not\models f(Q \cup R)^Y$, which contradicts our assumption.

□

Proposition 6

For HEX*-programs P and Q and sets $\mathcal{H}$ and $\mathcal{B}$ of atoms, there is a program $R \in \mathcal{P}_{\langle \mathcal{H}, \mathcal{B} \rangle}$ with $\mathcal{AS}(P \cup R) \not\subseteq \mathcal{AS}(Q \cup R)$ iff there is a witness for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$.*

Proof. ($\Rightarrow$) If $\mathcal{AS}(P \cup R) \not\subseteq \mathcal{AS}(Q \cup R)$ for a program R, then there is an assignment Y such that $Y \in \mathcal{AS}(P \cup R)$ but $Y \notin \mathcal{AS}(Q \cup R)$. Due to Proposition 5 we can assume that R is a positive program.

We show that Y satisfies Condition (i) of Definition 9. Since $Y \in \mathcal{AS}(P \cup R)$ we have $Y \models P$. Towards a contradiction, suppose there is an $Y' \subsetneq Y$ such that $Y' \models fP^Y$ and $Y'|_{\mathcal{H}} \not\subsetneq Y|_{\mathcal{H}}$. Then, since $Y' \subseteq Y$, we have $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$. We further have $Y'|_{\mathcal{B}} \subseteq Y|_{\mathcal{B}}$, i.e., $Y \leq^{\mathcal{B}}_{\mathcal{H}} Y'$. Since R is positive, $Y \models R$ implies $Y' \models R$, and since $fR^Y \subseteq R$ this further implies $Y' \models fR^Y$. Since we further have $Y' \models fP^Y$ this gives $Y' \models f(P \cup R)^Y$ and thus Y cannot be an answer set of $P \cup R$, which contradicts our assumption and therefore Condition (i) is satisfied.

We show now that there is an X such that (X, Y) satisfies also Condition (ii), i.e., is a witness as by Definition 9. If $Y \not\models Q$ then Condition (ii) is trivially satisfied for any $X \subseteq Y$ and e.g. (Y, Y) is a witness. Otherwise ($Y \models Q$), note that we have $Y \models R$ since $Y \in \mathcal{AS}(P \cup R)$. Together with the precondition that $Y \notin \mathcal{AS}(Q \cup R)$ this implies that there is an $X \subsetneq Y$ such that $X \models f(Q \cup R)^Y$, which is equivalent to $X \models fQ^Y$ and $X \models fR^Y$. We show that for this X, Condition (ii) is satisfied, hence (X, Y) is a witness. As we already have $X \subsetneq Y$ and $X \models fQ^Y$, it remains to show that for any X' with $X \leq^{\mathcal{B}}_{\mathcal{H}} X' \subsetneq Y$ we have $X' \not\models fP^Y$. If there would be an X' with $X \leq^{\mathcal{B}}_{\mathcal{H}} X' \subsetneq Y$ with $X' \models fP^Y$, then, since we also have $X' \models fR^Y$ (because $X \models fR^Y$ and fR^Y is positive), this implies $X' \models f(P \cup R)^Y$ and contradicts the precondition that $Y \in \mathcal{AS}(P \cup R)$. Thus such an X' cannot exist and Condition (ii) is satisfied by (X, Y).

($\Leftarrow$) Let (X, Y) be a witness for $\mathcal{AS}(P \cup R) \not\subseteq \mathcal{AS}(Q \cup R)$. We make a case distinction: either $Y \not\models Q$ or $Y \models Q$.

- Case $Y \not\models Q$:

  We show for the following $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B} \rangle}$ that $Y \in \mathcal{AS}(P \cup R)$ but $Y \notin \mathcal{AS}(Q \cup R)$:

  $$R = \{a \leftarrow \mid a \in Y|_{\mathcal{H}}\}$$

  Since (X, Y) is a witness, by Property (i) we have $Y \models P$. We further have $Y \models R$, thus $Y \models P \cup R$. Moreover, we obviously have $fR^Y = R$, which contains all atoms from $Y|_{\mathcal{H}}$ as facts. Suppose there is a $Y' \subsetneq Y$ such that $Y' \models f(P \cup R)^Y$; then $Y' \models fP^Y$ and by Property (i) we have $Y'|_{\mathcal{H}} \subsetneq Y|_{\mathcal{H}}$, i.e., at least one atom from $Y|_{\mathcal{H}}$ is unsatisfied under Y'. But then $Y' \not\models fR^Y$ and thus $Y' \not\models f(P \cup R)^Y$, i.e., Y is an answer set of $P \cup R$. On the other hand, $Y \not\models Q$ implies $Y \not\models Q \cup R$ and therefore Y cannot be an answer set of $Q \cup R$.

- Case $Y \models Q$:

  We show for the following $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B} \rangle}$ that $Y \in \mathcal{AS}(P \cup R)$ but $Y \notin \mathcal{AS}(Q \cup R)$:

  $$\begin{aligned} R = &\{a \leftarrow \mid a \in X|_{\mathcal{H}}\} \cup \\ &\{a \leftarrow b \mid a \in (Y \setminus X)|_{\mathcal{H}}, b \in (Y \setminus X)|_{\mathcal{B}}\} \end{aligned}$$

  We first show that $Y \in \mathcal{AS}(P \cup R)$. Since (X, Y) is a witness as by Definition 9, we have $Y \models P$. We further have $Y \models R$ by construction of R because all heads of its rules are in Y.

  Thus it remains to show that it is also a subset-minimal model of $f(P \cup R)^Y$. Towards a contradiction, assume that there is a $Z \subsetneq Y$ such that $Z \models f(P \cup R)^Y$, which is equivalent to $Z \models fP^Y$ and $Z \models fR^Y$, where $fR^Y = R$ (by construction of R). By construction of R, $Z \models R$ implies that $X|_{\mathcal{H}} \subseteq Z|_{\mathcal{H}}$. Property (i) of Definition 9 implies that $Z|_{\mathcal{H}} \subsetneq Y|_{\mathcal{H}}$ and thus $X|_{\mathcal{H}} \subseteq Z|_{\mathcal{H}} \subsetneq Y|_{\mathcal{H}}$. This implies that there is an $a \in (Y \setminus X)|_{\mathcal{H}}$ which is not in $Z|_{\mathcal{H}}$. Since $Y \models Q$, $Z \subsetneq Y$, $Z \models fP^Y$ and $X|_{\mathcal{H}} \subseteq Z|_{\mathcal{H}}$, Property (ii) further implies $Z|_{\mathcal{B}} \not\subseteq X|_{\mathcal{B}}$ (since violating $X \leq^{\mathcal{B}}_{\mathcal{H}} Z$ is the only remaining option to satisfy the property). As we also have $Z|_{\mathcal{B}} \subseteq Y|_{\mathcal{B}}$ (because $Z \subsetneq Y$), there is a $b \in (Y \setminus X)|_{\mathcal{B}}$ which is also in Z. Hence, we have an $a \in (Y \setminus X)|_{\mathcal{H}}$ and a $b \in (Y \setminus X)|_{\mathcal{B}}$ such that only b is also in Z, hence the rule $a \leftarrow b \in R$ (and $a \leftarrow b \in fR^Y$) is violated by Z, thus $Z \not\models fR^Y$ and $Z \not\models f(P \cup R)^Y$, which contradicts our assumption.

  It remains to show that $Y \notin \mathcal{AS}(Q \cup R)$. We already know that $Y \models Q \cup R$ and must show that $f(Q \cup R)^Y$ has a smaller model than Y. Since (X, Y) is a witness, we have $X \subsetneq Y$ and $X \models fQ^Y$ by Property (ii). As $X \models R$ (it satisfies all facts $\{a \leftarrow \mid a \in X|_{\mathcal{H}}\}$ and no other rules of R are applicable as their bodies contain only atoms that are not in X), we get $X \models fR^Y$ and have $X \models f(Q \cup R)^Y$. Therefore $Y \notin \mathcal{AS}(Q \cup R)$.

□

Towards a characterization of equivalence in terms of $\langle \mathcal{H}, \mathcal{B} \rangle$-models we introduce the following lemma.

Lemma 4 *For sets $\mathcal{H}$ and $\mathcal{B}$ of atoms and programs P, Q, $(Y, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P) \setminus \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$ iff there is a witness (X, Y) for $P \subseteq_{\langle \mathcal{H},\mathcal{B} \rangle} Q$ with $X|_{\mathcal{H}} = Y|_{\mathcal{H}}$.*

Proof. ($\Rightarrow$) Since $(Y, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$, Property (i) of Definition 9 holds because Property (i) of Definition 10 is the same and holds. For Property (ii) of Definition 9, $(Y, Y) \notin \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$ implies that either $Y \not\models Q$ or there is a $Y' \subsetneq Y$ such that $Y' \models fQ^Y$ and $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$. In the former case, Property (ii) of Definition 9 holds trivially for all $X \subseteq Y$ and, e.g., (Y, Y) is witness for $P \subseteq_{\langle \mathcal{H},\mathcal{B} \rangle} Q$, for which $Y|_{\mathcal{H}} = Y|_{\mathcal{H}}$ clearly holds. In case $Y \models Q$ we have that there is some $X \subsetneq Y$ with $X \models fQ^Y$ and $X|_{\mathcal{H}} = Y|_{\mathcal{H}}$. In order to show that (X, Y) satisfies Property (ii), it remains to show that for all X' with $X \leq^{\mathcal{B}}_{\mathcal{H}} X' \subsetneq Y$ we have $X' \not\models fP^Y$. If there would be an X' with $X \leq^{\mathcal{B}}_{\mathcal{H}} X' \subsetneq Y$ and $X' \models fP^Y$, then $X|_{\mathcal{H}} = Y|_{\mathcal{H}}$ would imply $X'|_{\mathcal{H}} = Y|_{\mathcal{H}}$, and thus Property (i) of Definition 10 would be violated by (Y, Y) wrt. P, which contradicts the assumption that $(Y, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$.

($\Leftarrow$) For a witness (X, Y) for $P \subseteq_{\langle \mathcal{H},\mathcal{B} \rangle} Q$ with $X|_{\mathcal{H}} = Y|_{\mathcal{H}}$, Property (i) of Definition 9 implies that $(Y, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$ and it remains to show that $(Y, Y) \notin \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$. Since (X, Y) is a witness with $X|_{\mathcal{H}} = Y|_{\mathcal{H}}$, we have either $Y \not\models Q$ or $X \subsetneq Y$ and $X \models fQ^Y$. In the former case (Y, Y) cannot be an $\langle \mathcal{H},\mathcal{B} \rangle$-model of Q due to violation of Property (i) of Definition 10. In the latter case (Y, Y) cannot be an $\langle \mathcal{H},\mathcal{B} \rangle$-model of Q since our assumption $X|_{\mathcal{H}} = Y|_{\mathcal{H}}$ also contradicts Property (i) of Definition 10. $\square$

Proposition 7

For sets $\mathcal{H}$ and $\mathcal{B}$ of atoms and HEX*-programs P and Q, we have $P \equiv_{\langle \mathcal{H},\mathcal{B} \rangle} Q$ iff $\sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P) = \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$.*

Proof. ($\Rightarrow$) We make a proof by contraposition. Wlog. assume there is an $(X, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P) \setminus \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$ (the case $(X, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q) \setminus \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$ is symmetric). We have to show that then $P \equiv_{\langle \mathcal{H},\mathcal{B} \rangle} Q$ does not hold.

Since $(X, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$, we also have $(Y, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$ (cf. Definition 10). If $(Y, Y) \notin \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$ then by Lemma 4 there is a witness (X, Y) for $P \not\subseteq_{\langle \mathcal{H},\mathcal{B} \rangle} Q$ and thus by Proposition 6 there is a program $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B} \rangle}$ with $\mathcal{AS}(P \cup R) \not\subseteq \mathcal{AS}(Q \cup R)$, hence $P \equiv_{\langle \mathcal{H},\mathcal{B} \rangle} Q$ does not hold.

In case $(Y, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$ we have $X \subsetneq Y$ (X and Y cannot be equal because $(X, Y) \notin \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$). We make a case distinction.

- Case 1: There exists an X' with $X \leq^{\mathcal{B}}_{\mathcal{H}} X' \subsetneq Y$ such that $(X', Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$:

  Since $(Y, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$ but $(X, Y) \notin \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$, the latter fails to satisfy Definition 10 due to Property (ii). Then $X <^{\mathcal{B}}_{\mathcal{H}} X'$ must hold (rather than $X|_{\mathcal{H} \cup \mathcal{B}} = X'|_{\mathcal{H} \cup \mathcal{B}}$) because only in this case satisfaction of Property (ii) of Definition 10 wrt. X can differ from satisfaction wrt. X'. Then there is a $Z \subsetneq Y$ with $Z|_{\mathcal{H} \cup \mathcal{B}} = X'$ such that (Z, Y) is $\leq^{\mathcal{B}}_{\mathcal{H}}$-maximal for Q and thus $Z \models fQ^Y$. We show that (Z, Y) is a witness for $P \not\subseteq_{\langle \mathcal{H},\mathcal{B} \rangle} Q$. Since $(Y, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$, Property (i) of Definition 9 holds for (Z, Y). Moreover, we have $Z \models fQ^Y$ and, since $(X, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$, we have by Property (ii) of Definition 10 for all X'' with $X <^{\mathcal{B}}_{\mathcal{H}} X'' \subsetneq Y$ that $X'' \not\models fP^Y$. Since $X <^{\mathcal{B}}_{\mathcal{H}} Z$ (as a consequence of $Z|_{\mathcal{H} \cup \mathcal{B}} = X'$ and $X <^{\mathcal{B}}_{\mathcal{H}} X'$), Property (ii) of Definition 9 holds for Z and thus (Z, Y) is a witness for $P \not\subseteq_{\langle \mathcal{H},\mathcal{B} \rangle} Q$.

- Case 2: For each X' with $X \leq^{\mathcal{B}}_{\mathcal{H}} X' \subsetneq Y$ we have $(X', Y) \notin \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$:

  We already have $(X, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$ and thus there is a $Z \subsetneq X$ with $Z|_{\mathcal{H} \cup \mathcal{B}} = X$ such that $Z \models fP^Y$. We show that (Z, Y) is a witness for the reverse problem $Q \not\subseteq_{\langle \mathcal{H},\mathcal{B} \rangle} P$. Since $(Y, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(Q)$ we have that Property (i) of Definition 9 is satisfied. We have $Y \models P$ (due to Property (i) of Definition 10 wrt. $(X, Y) \in \sigma_{\langle \mathcal{H},\mathcal{B} \rangle}(P)$), thus for satisfaction of Property (ii) of Definition 9 recall that we have $Z \models fP^Y$ and it remains to show that for each X'' with $X \leq^{\mathcal{B}}_{\mathcal{H}} X'' \subsetneq Y$ we have $X'' \not\models fQ^Y$. If there would be such an X'' with $X'' \models fQ^Y$, then there would also a $\leq_{\mathcal{H}}^{\mathcal{B}}$-maximal one X''' and (X''', Y) would be an $\langle \mathcal{H}, \mathcal{B} \rangle$-model of Q, which contradicts our assumption that $(X', Y) \notin \sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(Q)$ for each X' with $X \leq_{\mathcal{H}}^{\mathcal{B}} X' \subsetneq Y$.

($\Leftarrow$) We make a proof by contraposition. Suppose $P \not\equiv_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$, then either $P \subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ or $Q \subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} P$ does not hold; we assume wlog. that $P \subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ does not hold (the other case is symmetric). We have to show that $\sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(P) = \sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(Q)$ does not hold either.

By Proposition 6 there is a witness (X, Y) for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$. Then by Property (i) of Definition 9 we have $Y \models P$ and for all $Y' \subsetneq Y$ with $Y' \models fP^Y$ we have $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$, which implies that $(Y, Y) \in \sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(P)$.

If $(Y, Y) \notin \sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(Q)$, it is proven that $\sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(P) \neq \sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(Q)$.

Otherwise we have $(Y, Y) \in \sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(Q)$. By Property (i) of Definition 9 we then have $Y \models Q$ and by Lemma 4 we have $X|_{\mathcal{H}} \neq Y|_{\mathcal{H}}$ and thus $X \subsetneq Y$. Since (X, Y) is a witness for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ we have $X \models fQ^Y$ and for all X' with $X \leq_{\mathcal{H}}^{\mathcal{B}} X' \subsetneq Y$ we have $X' \not\models fP^Y$. Take an arbitrary pair (Z, Y) of assignments with $Z \subsetneq Y$ for which $X \leq_{\mathcal{H}}^{\mathcal{B}} Z$ holds and which is $\leq_{\mathcal{H}}^{\mathcal{B}}$-maximal for Q (such a pair exists because we already know that $X \models fQ^Y$). Moreover, $(Y, Y) \in \sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(Q)$ implies that Property (i) of Definition 10 holds for $(Z|_{\mathcal{H} \cup \mathcal{B}}, Y)$. Therefore $(Z|_{\mathcal{H} \cup \mathcal{B}}, Y) \in \sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(Q)$.

On the other hand, $(Z|_{\mathcal{H} \cup \mathcal{B}}, Y) \notin \sigma_{\langle \mathcal{H}, \mathcal{B} \rangle}(P)$ because (X, Y) is a witness for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ and therefore for all X' with $X \leq_{\mathcal{H}}^{\mathcal{B}} X' \subsetneq Y$ we have $X' \not\models fP^Y$. Since $X \leq_{\mathcal{H}}^{\mathcal{B}} Z \subsetneq Y$ we also have that $X'' \not\models fP^Y$ for all X'' such that $Z|_{\mathcal{H} \cup \mathcal{B}} \leq_{\mathcal{H}}^{\mathcal{B}} X'' \subsetneq Y$. But then there cannot be an X'' with $X''|_{\mathcal{H} \cup \mathcal{B}} = Z|_{\mathcal{H} \cup \mathcal{B}}$ such that $X'' \models fP^Y$. Therefore Property (ii) of Definition 10 cannot be satisfied due to failure to find a pair (X'', Y) with $X'' \subsetneq Y$ and $X''|_{\mathcal{H} \cup \mathcal{B}} = Z|_{\mathcal{H} \cup \mathcal{B}}$ that is $\leq_{\mathcal{H}}^{\mathcal{B}}$-maximal for P. $\square$

Proposition 8
For sets $\mathcal{H}$ and $\mathcal{B}$ of atoms, HEX*-programs P and Q, and an atom a which does not occur in P or Q, the following holds:*

(i) $P \equiv_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ *iff* $P \equiv_{\langle \mathcal{H} \cup \{a\}, \mathcal{B} \rangle} Q$*; and*

(ii) $P \equiv_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ *iff* $P \equiv_{\langle \mathcal{H}, \mathcal{B} \cup \{a\} \rangle} Q$.

Proof. Property (i) ($\Rightarrow$) We make a proof by contraposition. If $P \equiv_{\langle \mathcal{H} \cup \{a\}, \mathcal{B} \rangle} Q$ does not hold, then either $P \subseteq_{\langle \mathcal{H} \cup \{a\}, \mathcal{B} \rangle} Q$ or $Q \subseteq_{\langle \mathcal{H} \cup \{a\}, \mathcal{B} \rangle} P$; as the two cases are symmetric it suffices to consider the former. If $P \subseteq_{\langle \mathcal{H} \cup \{a\}, \mathcal{B} \rangle} Q$ does not hold then by Proposition 6 there is a witness (X, Y) for $P \not\subseteq_{\langle \mathcal{H} \cup \{a\}, \mathcal{B} \rangle} Q$. We show that we can also construct a witness for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$, which implies by another application of Proposition 6 that $P \subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ and thus $P \equiv_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ do not hold.

In particular, $(X \setminus \{a\}, Y \setminus \{a\})$ is a witness for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$. We show this separately depending on the type of (X, Y).

- If neither X nor Y contains a, then (X, Y) itself is also a witness for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$. Property (i) of Definition 9 holds because we know that $Y \models P$ and for each $Y' \subsetneq Y$ with $Y' \models fP^Y$ we have that $Y'|_{\mathcal{H} \cup \{a\}} \subsetneq Y|_{\mathcal{H} \cup \{a\}}$; the latter implies $Y'|_{\mathcal{H}} \subsetneq Y|_{\mathcal{H}}$ since $a \notin Y$ and thus Y' and Y must differ in an atom from $\mathcal{H}$.

  For Property (ii), if $Y \not\models Q$ we are done. Otherwise we know that $X \subsetneq Y$ and $X \models fQ^Y$ and that for all X' with $X \leq_{\mathcal{H} \cup \{a\}}^{\mathcal{B}} X'$ we have $X' \not\models fP^Y$ (since (X, Y) satisfies Property (ii) wrt. $\mathcal{H} \cup \{a\}$ and $\mathcal{B}$). We have to show that $X' \not\models fP^Y$ holds also for all X' with $X \leq_{\mathcal{H}}^{\mathcal{B}} X' \subsetneq Y$. However, each X' such that $X \leq_{\mathcal{H}}^{\mathcal{B}} X'$ has to satisfy $X'|_{\mathcal{H}} \supseteq X|_{\mathcal{H}}$ and $X'|_{\mathcal{B}} \subseteq X|_{\mathcal{B}}$; the former implies

$X'|_{\mathcal{H}\cup\{a\}} \supseteq X|_{\mathcal{H}\cup\{a\}}$ because $a \notin Y$, $X \subsetneq Y$ and $X' \subsetneq Y$. Then for this X' also $X \leq^{\mathcal{B}}_{\mathcal{H}\cup\{a\}} X'$ holds, and therefore satisfaction of Property (ii) wrt. $\mathcal{H} \cup \{a\}$ and $\mathcal{B}$ implies $X' \not\models fP^Y$. Thus Property (ii) holds also wrt. $\mathcal{H}$ and $\mathcal{B}$.

- If only Y but not X contains a, then $(X, Y \setminus \{a\})$ is also a witness for $P \not\subseteq_{\langle\mathcal{H},\mathcal{B}\rangle} Q$. For Property (i), $Y \models P$ implies $Y \setminus \{a\} \models P$ because a does not occur in P. Now suppose there is a $Y' \subsetneq Y \setminus \{a\}$ such that $Y' \models fP^Y$ and $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$. Then Y' and Y differ in an atom other than a and we have that $Y' \cup \{a\} \subsetneq Y$ and $Y' \cup \{a\}|_{\mathcal{H}\cup\{a\}} = Y|_{\mathcal{H}\cup\{a\}}$; this contradicts the assumption that Property (i) holds wrt. $\mathcal{H} \cup \{a\}$ and $\mathcal{B}$.

  For Property (ii), if $Y \not\models Q$ then also $Y \setminus \{a\} \not\models Q$ because a does not occur in Q and we are done. Otherwise we know that $X \subsetneq Y$ and $X \models fQ^Y$ and that for all X' with $X \leq^{\mathcal{B}}_{\mathcal{H}\cup\{a\}} X'$ we have $X' \not\models fP^Y$ (since (X, Y) satisfies Property (ii) wrt. $\mathcal{H} \cup \{a\}$ and $\mathcal{B}$). In this case, X and Y must in fact differ in more atoms than just a: otherwise $Y \models P$ would imply $X \models fP^Y$ (because a does not occur in P and $fP^Y \subseteq P$); since $X \leq^{\mathcal{B}}_{\mathcal{H}\cup\{a\}} X' \subsetneq Y$ for any X' with $X'|_{\mathcal{H}\cup\mathcal{B}} = X|_{\mathcal{H}\cup\mathcal{B}}$ this would contradict the assumption that Property (ii) holds wrt. $\mathcal{H} \cup \{a\}$ and $\mathcal{B}$. But then $X \subsetneq Y \setminus \{a\}$. Moreover, each X' such that $X \leq^{\mathcal{B}}_{\mathcal{H}} X' \subsetneq Y \setminus \{a\}$ has to satisfy $X'|_{\mathcal{H}} \supseteq X|_{\mathcal{H}}$ and $X'|_{\mathcal{B}} \subseteq X|_{\mathcal{B}}$; the former implies $X'|_{\mathcal{H}\cup\{a\}} \supseteq X|_{\mathcal{H}\cup\{a\}}$ because $a \notin Y \setminus \{a\}$, $X \subsetneq Y$ and $X' \subsetneq Y$. Then for this X' also $X \leq^{\mathcal{B}}_{\mathcal{H}\cup\{a\}} X' \subsetneq Y$ holds, and therefore satisfaction of Property (ii) wrt. $\mathcal{H} \cup \{a\}$ and $\mathcal{B}$ implies $X' \not\models fP^Y$. Thus Property (ii) holds also wrt. $\mathcal{H}$ and $\mathcal{B}$.

- If both X and Y contain a, then $(X \setminus \{a\}, Y \setminus \{a\})$ is also a witness for $P \not\subseteq_{\langle\mathcal{H},\mathcal{B}\rangle} Q$. For Property (i), $Y \models P$ implies $Y \setminus \{a\} \models P$ because a does not occur in P. Now suppose there is a $Y' \subsetneq Y \setminus \{a\}$ such that $Y' \models fP^Y$ and $Y'|_{\mathcal{H}} = (Y \setminus \{a\})|_{\mathcal{H}}$. Then Y' and $Y \setminus \{a\}$ differ in an atom other than a and we have that $Y' \cup \{a\} \subsetneq Y$, $Y' \cup \{a\} \models fP^Y$ (since a does not occur in P) and $Y' \cup \{a\}|_{\mathcal{H}\cup\{a\}} = Y|_{\mathcal{H}\cup\{a\}}$; this contradicts the assumption that Property (i) holds wrt. $\mathcal{H} \cup \{a\}$ and $\mathcal{B}$.

  For Property (ii), if $Y \not\models Q$ then also $Y \setminus \{a\} \not\models Q$ because a does not occur in Q and we are done. Otherwise we know that $X \subsetneq Y$ (and thus $X \setminus \{a\} \subsetneq Y \setminus \{a\}$) and $X \models fQ^Y$ and that for all X' with $X \leq^{\mathcal{B}}_{\mathcal{H}\cup\{a\}} X' \subsetneq Y$ we have $X' \not\models fP^Y$ (since (X, Y) satisfies Property (ii) wrt. $\mathcal{H} \cup \{a\}$ and $\mathcal{B}$). We have to show that $X' \not\models fP^Y$ holds also for all X' with $X \setminus \{a\} \leq^{\mathcal{B}}_{\mathcal{H}} X' \subsetneq Y \setminus \{a\}$. Consider such an X', then $X'|_{\mathcal{H}} \supseteq (X \setminus \{a\})|_{\mathcal{H}}$, $X'|_{\mathcal{B}} \subseteq (X \setminus \{a\})|_{\mathcal{B}}$, and $X' \subsetneq Y \setminus \{a\}$. Now let $X'' = X' \cup \{a\}$. Then $X''|_{\mathcal{H}\cup\{a\}} \supseteq X|_{\mathcal{H}\cup\{a\}}$ because a is added to X'' and the superset relation is already known to hold for all other atoms from $\mathcal{H}$. Moreover, $X''|_{\mathcal{B}} \subseteq X|_{\mathcal{B}}$ still holds because $X'|_{\mathcal{B}} \subseteq X|_{\mathcal{B}}$ and the only element a added to X'' is also in X. Moreover, we still have $X'' \subsetneq Y$ because $a \in Y$ and X' and Y differ in at least one atom other than a due to $X' \subsetneq Y \setminus \{a\}$. These conditions together imply $X \leq^{\mathcal{B}}_{\mathcal{H}\cup\{a\}} X'' \subsetneq Y$, and thus satisfaction of Property (ii) wrt. $\mathcal{H} \cup \{a\}$ and $\mathcal{B}$ implies $X'' \not\models fP^Y$. Since X'' and X' differ only in a, which does not appear in fP^Y, this further implies $X' \not\models fP^Y$. Hence Property (ii) holds also wrt. $\mathcal{H}$ and $\mathcal{B}$.

Property (i) ($\Leftarrow$) Trivial because $P \equiv_{\langle\mathcal{H}\cup\{a\},\mathcal{B}\rangle} Q$ is a stronger condition than $P \equiv_{\langle\mathcal{H},\mathcal{B}\rangle} Q$ since it allows a larger class of programs to be added.

Property (ii) ($\Rightarrow$) We make a proof by contraposition. If $P \equiv_{\langle\mathcal{H},\mathcal{B}\cup\{a\}\rangle} Q$ does not hold, then either $P \not\subseteq_{\langle\mathcal{H},\mathcal{B}\cup\{a\}\rangle} Q$ or $Q \not\subseteq_{\langle\mathcal{H},\mathcal{B}\cup\{a\}\rangle} P$; as the two cases are symmetric it suffices to consider the former.

If $P \subseteq_{\langle \mathcal{H}, \mathcal{B} \cup \{a\} \rangle} Q$ does not hold then by Proposition 6 there is a witness (X, Y) for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \cup \{a\} \rangle} Q$. We show that we can also construct a witness for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$, which implies by another application of Proposition 6 that $P \subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ and thus $P \equiv_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ does not hold.

We show in particular that (X, Y) is also a witness for $P \not\subseteq_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$. Property (i) of Definition 9 is also satisfied wrt. $\mathcal{H}$ and $\mathcal{B}$ (instead of $\mathcal{H}$ and $\mathcal{B} \cup \{a\}$) as this condition is independent of $\mathcal{B}$.

If $Y \not\models Q$ then Property (ii) is also satisfied and we are done. Otherwise we know, that $X \subsetneq Y$, $X \models fQ^Y$ and for all X' with $X \leq_{\mathcal{H}}^{\mathcal{B} \cup \{a\}} X' \subsetneq Y$ we have $X' \not\models fP^Y$. We have to show that $X' \not\models fP^Y$ holds also for all X' with $X \leq_{\mathcal{H}}^{\mathcal{B}} X' \subsetneq Y$. Consider such an X', then $X'|_{\mathcal{H}} \supseteq X|_{\mathcal{H}}$, $X'|_{\mathcal{B}} \subseteq X|_{\mathcal{B}}$ and $X' \subsetneq Y$. Now let $X'' = X' \setminus \{a\}$ if $a \in X'$ and $a \notin X$, and $X'' = X'$ otherwise. We have then $X''|_{\mathcal{B} \cup \{a\}} \subseteq X|_{\mathcal{B} \cup \{a\}}$ because a is removed from X'' whenever it is not in X, and the subset relation is known for all other atoms from $\mathcal{B}$. Moreover, $X''|_{\mathcal{H}} \supseteq X|_{\mathcal{H}}$ still holds because $X'|_{\mathcal{H}} \supseteq X|_{\mathcal{H}}$ and the only element a which might be missing in X'' compared to X is only removed if it is not in X anyway. These conditions together imply $X \leq_{\mathcal{H}}^{\mathcal{B} \cup \{a\}} X' \subsetneq Y$, and thus satisfaction of Property (ii) wrt. $\mathcal{H}$ and $\mathcal{B} \cup \{a\}$ implies $X'' \not\models fP^Y$. Since X'' and X' may only differ in a, which does not appear in fP^Y, this implies $X' \not\models fP^Y$. Hence Property (ii) holds also wrt. $\mathcal{H}$ and $\mathcal{B}$.

Property (ii) ($\Leftarrow$) Trivial because $P \equiv_{\langle \mathcal{H}, \mathcal{B} \cup \{a\} \rangle} Q$ is a stronger condition than $P \equiv_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ since it allows a larger class of programs to be added. $\square$

Corollary 2
For sets $\mathcal{H}$ and $\mathcal{B}$ of atoms, programs P and Q, and an sets of atoms $\mathcal{H}'$ and $\mathcal{B}'$ which does not occur in P or Q, we have $P \equiv_{\langle \mathcal{H}, \mathcal{B} \rangle} Q$ iff $P \equiv_{\langle \mathcal{H} \cup \mathcal{H}', \mathcal{B} \cup \mathcal{B}' \rangle} Q$.

Proof. The claim follows immediately by applying Proposition 8 iteratively to each element in $\mathcal{H}'$ resp. $\mathcal{B}'$. $\square$

Lemma 1
For an external atom $e = \&g[\mathbf{p}](\mathbf{c})$ in program P, $p_i \in \mathbf{p}$, a new predicate q, let $e' = \&g'[\mathbf{p}|_{p_i \to q}](\mathbf{c})$ s.t. $f_{\&g'}(Y, \mathbf{p}|_{p_i \to q}, \mathbf{c}) = f_{\&g}(Y^q, \mathbf{p}, \mathbf{c})$ for all assignments Y.

For $P' = P|_{e \to e'} \cup \{q(p_i, \mathbf{d}) \leftarrow p_i(\mathbf{d}) \mid p_i(\mathbf{d}) \in A(P)\}$, $\mathcal{AS}(P)$ and $\mathcal{AS}(P')$ coincide, modulo atoms $q(\cdot)$.

Proof. ($\Rightarrow$) For an answer set Y of P we show that $Y' = Y \cup \{q(p_i, \mathbf{d}) \mid p_i(\mathbf{d}) \in Y\}$ is an answer set of P'.

Since input parameter q in e' behaves like p_i in e, $Y' \models q(p_i, \mathbf{d})$ iff $Y \models p_i(\mathbf{d})$ for all $p_i(\mathbf{d}) \in A(P)$ by construction, and Y' satisfies all rules $r \in \{q(p_i, \mathbf{d}) \leftarrow p_i(\mathbf{d}) \mid p_i(\mathbf{d}) \in A(P)\}$ by construction, we have that Y' is a model of P'.

Now suppose towards a contradiction that there is a smaller model $Y'_< \subsetneq Y'$ of $fP'^{Y'}$ and let this model be subset-minimal. Then $Y'_< \setminus Y'$ must contain at least one atom other than over q because switching an atom $q(p_i, \mathbf{d})$ to false is only possible if the respective atom $p_i(\mathbf{d})$ is also switched to false, otherwise a rule $r \in \{q(p_i, \mathbf{d}) \leftarrow p_i(\mathbf{d}) \mid p_i(\mathbf{d}) \in A(P)\}$ (which is contained in the reduct $fP'^{Y'}$ because $Y' \models B(r)$) would remain unsatisfied. But then for $Y_< = Y'_< \cap A(P)$ we have that $Y_< \subsetneq Y$. Now consider some $r \in fP^Y$: then there is a respective $r' \in fP'^{Y'}$ with e' in place of e and such that $Y'_< \models r'$. Observe that $p_i(\mathbf{d}) \in Y_<$ implies $q(p_i, \mathbf{d}) \in Y'_<$ (otherwise a rule in $fP'^{Y'}$ remains unsatisfied under $Y'_<$) and that $q(p_i, \mathbf{d}) \in Y'_<$ implies $p_i(\mathbf{d}) \in Y_<$ due to assumed subset-minimality of $Y'_<$ (there is no reason to set $q(p_i, \mathbf{d})$ to true if $p_i(\mathbf{d})$ is false). This gives in summary that $q(p_i, \mathbf{d}) \in Y'_<$ iff $p_i(\mathbf{d}) \in Y_<$ for all atoms $p_i(\mathbf{d}) \in A(P)$. But then we have also $Y_< \models r$ because the only possible difference between r and r' is that r might contain e while r' contains e', but since $q(p_i, \mathbf{d}) \in Y'_<$ iff $p_i(\mathbf{d})$ for all atoms $p_i(\mathbf{d}) \in A(P)$, we have that $Y'_< \models r'$ implies $Y_< \models r$. That is, $Y_< \subsetneq Y$ is a smaller model of fP^Y, which contradicts the assumption that Y is an answer set.

($\Leftarrow$) For an answer set Y' of P' we show that $Y = Y' \cap A(P)$ is an answer set of P. First observe that for any $p_i(\mathbf{d}) \in A(P)$ we have that $q(p_i, \mathbf{d}) \in Y'$ iff $p_i(\mathbf{d}) \in Y$: the if-direction follows from satisfaction of the rules in P' under Y', the only-if direction follows from subset-minimality of Y'.

Then the external atoms e in P behave under Y like the respective e' in P' under Y', which implies that $Y \models P$.

Now suppose towards a contradiction that there is a smaller model $Y_< \subsetneq Y$ of fP^Y. We show that then for $Y'_< = Y_< \cup \{q(p_i, \mathbf{d}) \mid p_i(\mathbf{d}) \in Y_<\}$ we have $Y'_< \models fP'^{Y'}$. But this follows from the observation that $fP'^{Y'}$ consists only of (i) rules that correspond to rules in fP^Y but with e' in place of e, and (ii) the rule $q(p_i, \mathbf{d}) \leftarrow p_i(\mathbf{d})$ for all $p_i(\mathbf{d}) \in Y'$. Satisfaction of (i) follows from the fact that $Y \models e$ iff $Y' \models e'$, satisfaction of (ii) is given by construction of $Y'_<$. Moreover, we have that $Y'_< \subsetneq Y'$: we have $Y_< \subsetneq Y \subseteq Y'$ and all atoms $q(p_i, \mathbf{d})$ added to $Y_<$ are also in Y' because it satisfies the rule $q(p_i, \mathbf{d}) \leftarrow p_i(\mathbf{d}) \in P'$; properness of the subset-relation follows from $Y_< \subsetneq Y$. Therefore we have $Y'_< \subsetneq Y'$ and $Y'_< \models fP'^{Y'}$, which contradicts the assumption that Y' is an answer set of P'. $\square$

Lemma 2
For a HEX*-program P and a set of (positive or negative) external atoms E in P, we have $P \triangle P_{[E]} = \{r \in P \mid$ none of E occur in $r\}$.*

Proof. For a single external atom $e \in E$ observe that all rules $r \in P_{[e]}$, which were constructed by (1)-(3) in Definition 6, contain at least one atom which does not appear in P. Thus these rules can only be in $P_{[e]}$ but not in P and thus not in $P \triangle P_{[e]}$. For the rules $r \in P_{[e]}$ constructed by (4) in Definition 6, note that $r \in P$ iif e does not appear in r. This is further the case iff $r \in P \triangle P_{[e]}$. In summary, $P \triangle P_{[e]}$ contains all and only the rules from P which do not contain e.

By iteration of the argument, one gets the same result for the set E of external atoms. $\square$

Proposition 9
For sets $\mathcal{H}$ and $\mathcal{B}$ of atoms and HEX*-programs P and Q, we have $P \equiv^e_{\langle \mathcal{H},\mathcal{B}\rangle} Q$ iff $\sigma_{\langle \mathcal{H},\mathcal{B}\rangle}(P) = \sigma_{\langle \mathcal{H},\mathcal{B}\rangle}(Q)$.*

Proof. ($\Rightarrow$) If $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$ for all $R \in \mathcal{P}^e_{\langle \mathcal{H},\mathcal{B}\rangle}$, then this holds in particular for all programs $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$ without external atoms. Then by Proposition 7 we have $\sigma_{\langle \mathcal{H},\mathcal{B}\rangle}(P) = \sigma_{\langle \mathcal{H},\mathcal{B}\rangle}(Q)$.

($\Leftarrow$) Suppose $\sigma_{\langle \mathcal{H},\mathcal{B}\rangle}(P) = \sigma_{\langle \mathcal{H},\mathcal{B}\rangle}(Q)$, then by Proposition 7 we have $P \equiv_{\langle \mathcal{H},\mathcal{B}\rangle} Q$ and by Corollary 2 we have $P \equiv_{\langle \mathcal{H}\cup\mathcal{H}',\mathcal{B}\cup\mathcal{B}'\rangle} Q$ for all sets $\mathcal{H}'$, $\mathcal{B}'$ of atoms which do not occur in P or Q. Now consider $R \in \mathcal{P}^e_{\langle \mathcal{H},\mathcal{B}\rangle}$. We have to show that $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$.

Let R' be the ordinary ASP-program after standardizing input atoms to external atoms apart from the atoms in P and Q (using Lemma 1) and subsequent inlining all external atoms in R using Definition 6. Note that R' uses only atoms from $\mathcal{H}$ in its heads, atoms from $\mathcal{B}$ in its bodies, and and newly introduced atoms $A(R') \setminus A(R)$; the latter are selected such that they do not occur in P or Q. We further have that R' is free of external atoms, thus $R' \in \mathcal{P}_{\langle \mathcal{H}\cup\mathcal{H}',\mathcal{B}\cup\mathcal{B}'\rangle}$ for $\mathcal{H}' = \mathcal{B}' = A(R') \setminus A(R)$.

We then have $P \equiv_{\langle \mathcal{H}\cup\mathcal{H}',\mathcal{B}\cup\mathcal{B}'\rangle} Q$ (by Corollary 2, as discussed above). By definition of $\equiv_{\langle \mathcal{H}\cup\mathcal{H}',\mathcal{B}\cup\mathcal{B}'\rangle}$ this gives $\mathcal{AS}(P \cup R') = \mathcal{AS}(Q \cup R')$. Then by Proposition 1 we have that $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$. $\square$

Proposition 10
A HEX*-program P is persistently inconsistent wrt. sets of atoms $\mathcal{H}$ and $\mathcal{B}$ iff for each classical model Y of P there is an $Y' \subsetneq Y$ such that $Y' \models fP^Y$ and $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$.*

Proof. Let $P_\perp$ be a program without classical models (e.g., $\{a \leftarrow; \leftarrow a\}$). Then, by monotonicity of classical logic, $P_\perp \cup R$ is inconsistent (wrt. the HEX-semantics) for all $R \in \mathcal{P}^e_{\langle \mathcal{H},\mathcal{B}\rangle}$, i.e., we have that $\mathcal{AS}(P_\perp \cup R) = \emptyset$ for all $R \in \mathcal{P}^e_{\langle \mathcal{H},\mathcal{B}\rangle}$.

We have to show that $\mathcal{AS}(P \cup R) = \emptyset$ for all $R \in \mathcal{P}^e_{\langle \mathcal{H},\mathcal{B}\rangle}$ iff for each model Y of P there is an $Y' \subsetneq Y$ such that $Y' \models fP^Y$ and $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$. Due to Proposition 1, each program with external atoms may be replaced by an ordinary ASP-program such that the answer sets correspond to each other one-by-one; therefore the former statement holds iff $\mathcal{AS}(P \cup R) = \emptyset$ for all $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$, i.e., it suffices to consider ordinary ASP-programs R. The claim is proven if we can show that $\mathcal{AS}(P \cup R) \subseteq \mathcal{AS}(P_\perp \cup R)$ for all $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$.

This corresponds to deciding $P \subseteq_{\langle \mathcal{H},\mathcal{B}\rangle} P_\perp$. By Proposition 6, $P \subseteq_{\langle \mathcal{H},\mathcal{B}\rangle} P_\perp$ is the case iff *no* witness for $P \not\subseteq_{\langle \mathcal{H},\mathcal{B}\rangle} P_\perp$ exists. Since $P_\perp$ does not have any classical models, each pair (X, Y) of assignments trivially satisfies Condition (ii) because $Y \not\models P_\perp$, thus a pair (X, Y) is not a witness iff it violates Property (i). This condition is violated by (X, Y) iff $Y \not\models P$ or there exists a $Y' \subsetneq Y$ such that $Y' \models fP^Y$ and $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$; this is exactly the stated condition. □

Lemma 3
For a HEX*-program P and a model Y of P, a set of atoms U is an unfounded set of P wrt. Y iff $Y \setminus U \models fP^Y$.*

Proof. ($\Rightarrow$) We have to show that any rule $r \in fP^Y$ is satisfied under $Y \setminus U$. First observe that $Y \models H(r)$ because otherwise we also had $Y \not\models B(r)$ (since Y is a model of P) and thus $r \notin fP^Y$. If $Y \setminus U \models H(r)$ we are done ($Y \setminus U \models r$). Otherwise we have $H(r) \cap U \neq \emptyset$ and thus one of the conditions of Definition 13 holds for r. This cannot be Condition (i) because otherwise we had $r \notin fP^Y$. If it is Condition (ii) then $Y \setminus U \not\models B(r)$ and thus $Y \setminus U \models r$. If it is Condition (iii) then $Y \setminus U \models H(r)$ and thus $Y \setminus U \models r$.

($\Leftarrow$) Let $Y' \subseteq Y$ be a model of fP^Y. We have to show that $U = Y \setminus Y'$ is a unfounded set of P wrt. Y. To this end we need to show that for all $r \in P$ with $H(r) \cap U \neq \emptyset$ one of the conditions of Definition 13 holds. If $r \notin fP^Y$ then $Y \not\models B(r)$ and thus Condition (i) holds. If $r \in fP^Y$ then we either have $Y' \not\models B(r)$ or $Y' \models H(r)$. If $Y' \not\models B(r)$ then $Y \setminus U \not\models B(r)$ because $Y \setminus U = Y'$, i.e., Condition (ii) holds. If $Y' \models H(r)$ then there is an $h \in H(r)$ s.t. $h \in Y$ and $h \in Y'$ and thus $h \notin U$. Then we have $h \in Y \setminus U$ and thus $Y \models h$, i.e., Condition (iii) holds. □

Corollary 3
A HEX*-program P is persistently inconsistent wrt. sets of atoms $\mathcal{H}$ and $\mathcal{B}$ iff for each classical model Y of P there is a nonempty unfounded set U of P wrt. Y s.t. $U \cap Y \neq \emptyset$ and $U \cap \mathcal{H} = \emptyset$.*

Proof. By Proposition 10 we know that $P \cup R$ is inconsistent for all $R \in \mathcal{P}_{\langle \mathcal{H},\mathcal{B}\rangle}$ iff for each model Y of P there is an $Y' \subsetneq Y$ such that $Y' \models fP^Y$ and $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$. Each such model Y' corresponds one-by-one to a nonempty unfounded set $U = Y \setminus Y'$ of P wrt. Y, for which we obviously have $U \cap Y \neq \emptyset$ and $Y'|_{\mathcal{H}} = Y|_{\mathcal{H}}$ iff $U \cap \mathcal{H} = \emptyset$. □