Efficient Evaluation of Answer Set Programs with External Sources Based on External Source Inlining*

Christoph Redl
Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
redl@kr.tuwien.ac.at

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. Traditionally, HEX-programs are evaluated using a rewriting to ordinary ASP programs which guess truth values of external atoms; this yields answer set candidates whose guesses are verified by evaluating the source. Despite the integration of learning techniques into this approach, which reduce the number of candidates and of necessary verification calls, the remaining external calls are still expensive. In this paper we present an alternative 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. To this end, we make use of *support sets*, which describe conditions on input atoms that are sufficient to satisfy an external atom. The approach is implemented in the DLVHEX reasoner. Experiments show a significant performance gain.

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 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. 2014a), 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 programs without external calls.

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, Eiter, and Xiao 2010), 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, Ostrowski, and Schaub 2009). The approach uses *support sets* (cf. e.g. Darwiche and Marquis [2011]), 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 (Eiter et al. 2014b); however, this was only for improving 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 show that this can lead to significant performance improvements.

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 non-monotonicity, we use a *saturation encoding* based on support sets.
- While Section 3 focuses on positive external atoms, Section 4 shows how our approach can be extended to the negative case. To this end we make use of *negative support sets*, which can be gained from positive ones.
- In Section 5 we implement the approach in the DLVHEX system. Experimental evaluation shows a significant speedup, both over traditional evaluation and over a previous approach based on support sets for guess verification.

*This research has been supported by the Austrian Science Fund (FWF) project P27730.
Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

- Section 6 discusses related work and concludes the paper.

For space reasons, proofs are outsourced to the extended version at http://www.kr.tuwien.ac.at/research/projects/inthex/extinlining-ext.pdf.

2 Preliminaries

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

HEX**-Programs.**. We briefly recall HEX-programs, which generalize (disjunctive) logic programs under the answer set semantics (Gelfond and Lifschitz 1991); for more details and background, see Eiter et al. [2005] and Eiter et al. [2014a].

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 $\mathbf{p} = p_1, \ldots, p_k$ is a list of input parameters (predicates or object constants), called *input list*, and $\mathbf{c} = c_1, \ldots, c_l$ are output constants.

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

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

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

For a rule r, its *head* is $H(r) = \{a_1, \ldots, a_k\}$, its *body* is $B(r) = \{b_1, \ldots, b_m, \mathrm{not}\ b_{m+1}, \ldots, \mathrm{not}\ b_n\}$, 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 set of constants $\mathcal{C}$, let $HB_{\mathcal{C}}(\mathcal{P})$ denote the *Herbrand base* containing all atoms constructible from the predicates 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.

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 $1{+}k{+}l$-ary *two-valued (Boolean) oracle function* $f_{\&g}$ that is defined for all possible values of Y, $\mathbf{p}$ and $\mathbf{c}$. Thus, $\&g[\mathbf{p}](\mathbf{c})$ is true relative to Y iff $f_{\&g}(Y, \mathbf{p}, \mathbf{c}) = \mathbf{T}$. Satisfaction of ordinary rules and ASP programs (Gelfond and Lifschitz 1991) is then extended to HEX-rules and programs in the obvious way: 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.* [1]

[1] For ordinary P, these are Gelfond & Lifschitz's answer sets.

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 $fP^{Y_1} = \emptyset$.*

Traditional Evaluation Approach. A HEX-programs P is transformed to ordinary ASP programs 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 encoded by such an answer set may not satisfy P, as $\&g[\mathbf{p}](\mathbf{c})$ under $f_{\&g}$ may differ from the guess $e_{\&g[\mathbf{p}](\mathbf{c})}$. The answer set is merely a *candidate*; if a check against the external source succeeds, it is a *compatible set*:

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.*

Each answer set Y of P is the projection of some compatible set $\hat{Y}$ to $A(P)$, but not vice versa. To discard non-answer sets, minimality wrt. fP^Y is checked (Eiter et al. 2014a).

Example 2 (cont'd)**.** *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 $\hat{Y_1} = \emptyset$ and $\hat{Y_2} = \{p, e_{\&id[p]}\}$. Here Y_1 is a $\subseteq$-minimal model of $fP^{Y_1} = \emptyset$, but Y_2 not of $fP^{Y_2} = P$.*

Evaluation Based on Support Sets. 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 (Eiter et al. 2014b).

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 $\sigma \in \{\mathbf{T}, \mathbf{F}\}$, $S^+_\sigma \subseteq HB_{\mathcal{C}}(\mathcal{P})$, and $S^-_\sigma \subseteq \neg HB_{\mathcal{C}}(\mathcal{P})$ s.t. $A \supseteq S^+_\sigma$ and $A \cap \neg S^-_\sigma = \emptyset$ implies $A \models e$ if $\sigma = \mathbf{T}$ and $A \not\models e$ if $\sigma = \mathbf{F}$ for all assignments A.*

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

Example 3. *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 A with $\{p(a)\} \subseteq A$ but $A \cap \{q(a)\} = \emptyset$ satisfies $\&diff[p, q](a)$.*

Positive support sets $S_{\mathbf{T}}$ for $\&g[\mathbf{p}](\mathbf{c})$ can be added as constraints $\leftarrow S^+_{\mathbf{T}}, \{\mathrm{not}\, a \mid \neg a \in S^-_{\mathbf{T}}\}, \mathrm{not}\ \&g[\mathbf{p}](\mathbf{c})$ to exclude wrongly false guesses (analogously for negative support sets).

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

Definition 5 (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 A with $A \models e$ resp. $A \not\models e$ there is an $S_\sigma \in \mathcal{S}_\sigma$ s.t. $A \supseteq S^+_\sigma$ and $A \cap \neg S^-_\sigma = \emptyset$.*

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 $A \supseteq S^+_\sigma$ and $A \cap \neg S^-_\sigma = \emptyset$ for the current assignment A. 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 $\sigma = \mathbf{F}$. Note that this check is necessary even if all $S_\sigma \in \mathcal{S}_\sigma$ are added as constraints (as described above) because constraints prevent only wrong false guesses for $\sigma = \mathbf{T}$ and wrong true guesses for $\sigma = \mathbf{F}$, but not vice versa. This method is in particular advantageous if the support sets in $\mathcal{S}_\sigma$ are small and few. The approach was lifted to the non-ground level (Eiter et al. 2014b).

3 External Source Inlining

In this section we present a rewriting which compiles HEX-programs into equivalent ordinary ASP programs (modulo auxiliary atoms). 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 them based on their input whenever the original external atom is true, 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 Section 3.1. Afterwards we present a sound and complete encoding based on the saturation technique, cf. Section 3.2. While in the worst case it is exponential due to exponentially many support sets, many realistic external sources are guaranteed to have small support set families. Generally, they tend to be small for sources whose behavior is structured (cf. parameterized complexity), i.e., whose output does not change completely with small changes in the input. We focus on such sources, which include e.g. certain description logics Eiter et al. [2014b] and those we use in the benchmarks.

Constructing support sets depends on the external source. We assume that they are already given and refer to Eiter et al. [2014b] for a discussion of how to create them.

3.1 Observations

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 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 4. *Consider $P = \{a \leftarrow \&true[a]()\}$ where $e = \&true[a]()$ is always true. The program is expected to have the answer set $A = \{a\}$. However, the translated program $P' = \{x_e \leftarrow a;\ x_e \leftarrow \mathrm{not}\, a;\ a \leftarrow x_e\}$ has no answer set because the only candidate is $A' = \{a, x_e\}$ and $fP'^A = \{x_e \leftarrow a;\ a \leftarrow x_e\}$ 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 $\mathrm{not}\, a$ holds, but the rule $x_e \leftarrow \mathrm{not}\, a$, which represents this case, is dropped from the reduct because its body $\mathrm{not}\, a$ is unsatisfied by A'. Hence, although the external atom e holds both under A' and under the smaller model $\emptyset$ of the reduct which dismisses A', this is not detected since the representation of the external atom in the reduct is incomplete. Then the value of x_e and e under a model of the reduct can differ.

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

Example 5. *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 $A = \emptyset$. However, the translated program $P' = \{x_e \vee \bar{x_e};\ \leftarrow a, \mathrm{not}\, x_e;\ \leftarrow \mathrm{not}\, a, x_e;\ a \leftarrow x_e\}$ has not only the intended answer set $\{\bar{x_e}\}$ but also $A' = \{a, x_e\}$ because $fP'^{A'} = \{x_e \vee \bar{x_e};\ a \leftarrow x_e\}$ has no smaller model.*

While the second rewriting attempt from Example 5 works for Example 4, and, conversely, the one applied in Example 4 works for Example 5, 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, Leone, and Pfeifer 2011), such a rewriting must involve disjunctions with head-cycles.

3.2 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. We first show how single positive external atoms can be inlined into a program.

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

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

$$\cup \left\{ \begin{array}{l} \bar{a} \leftarrow \mathrm{not}\, a; \bar{a} \leftarrow x_e \\ a \vee \bar{a} \leftarrow \mathrm{not}\, \bar{x_e} \end{array} \;\middle|\; a \in I(e, P) \right\} \quad (2)$$

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

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

where $\bar{a}$ is a new atom for each a, x_e and $\bar{x_e}$ 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.

Note that the rewriting is only defined for positive external atoms; we discuss the negative case in Section 4.

The rewriting works as follows. The atom x_e represents the former external atom, i.e., that e is true, while $\bar{x_e}$ represents that it is false. The rules in (1) represent all input assignments which satisfy x_e (resp. e). More specifically, each rule in $\{x_e \leftarrow S^+_\mathbf{T} \cup \{\bar{a} \mid \neg a \in S^-_\mathbf{T}\} \mid S \in \mathcal{S}_\mathbf{T}(e, P)\}$ 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 $\bar{a}$ 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 A, all relevant rules in (1), i.e. those which might apply to subsets of A, are contained in the reduct wrt. A (because a could become false

in a smaller model of the reduct); recall that in Example 4 the reason for incorrectness of the rewriting was exactly that these rules were dropped. The derivation of $\bar{a}$ despite a being true is only necessary if x_e is true wrt. A; 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, Ianni, and Krennwallner 2009); the use of disjunctions with head-cycles is in general unavoidable for complexity reasons (unless the polynomial hierarchy collapses). Next, rule (3) enforces $\bar{x_e}$ 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 result we 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.[2] While this might lead to an exponential blowup it simplifies the proof. We show afterwards that rewriting is still correct even if $\mathcal{S}_{\mathbf{T}}(e,P)$ does not fulfill the property.

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 $P_{[e]}$, modulo the atoms newly introduced in $P_{[e]}$.*

The following results shows that the idea still works for arbitrary complete positive families of support sets $\mathcal{S}_{\mathbf{T}}(e,P)$.

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

$$P \supseteq \{r_1\colon x_e \leftarrow B, b; r_2\colon x_e \leftarrow B, \bar{b}\} \cup \{\bar{a} \leftarrow \text{not } a;\ \bar{a} \leftarrow x_e;\ a \vee \bar{a} \leftarrow \text{not } \bar{x_e} \mid a \in X\} \cup \{\bar{x_e} \leftarrow \text{not } x_e\}$$

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

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 $P_{[e]}$, modulo the atoms newly introduced in $P_{[e]}$.*

Example 6. *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:*

$$P_{[e]} = \{x_e \leftarrow a;\ x_e \leftarrow \bar{b}\ \ \bar{a} \leftarrow \text{not } a;\ \bar{a} \leftarrow x_e;\ \bar{b} \leftarrow \text{not } b;\ \bar{b} \leftarrow x_e\ \ a \vee \bar{a} \leftarrow \text{not } \bar{x_e};\ b \vee \bar{b} \leftarrow \text{not } \bar{x_e}\ \ \bar{x_e} \leftarrow \text{not } x_e;\ a \leftarrow x_e\}$$

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

Multiple external atoms can be inlined by iterative application. However, separate auxiliaries must be introduced for atoms that are input to multiple external atoms.

[2] 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.

4 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 $\text{not } e$ because progams P and $P[e]$ are in this case in general *not* equivalent.

Example 7. *Consider $P = \{p \leftarrow \text{not } \&neg[p]()\}$, where $\&neg[p]()$ is true if p is false and vice versa. The only answer set of P is $Y = \emptyset$ but the rewriting (Definition 6) yields*

$$P_{[\&neg[p]()]} = \{x_e \leftarrow \bar{p};\ \bar{p} \leftarrow \text{not } p;\ \bar{p} \leftarrow x_e\ \ p \vee \bar{p} \leftarrow \text{not } \bar{x_e};\ \bar{x_e} \leftarrow \text{not } x_e; p \leftarrow \text{not } x_e\}$$

which has the answer sets $Y'_1 = \{x_e, \bar{p}\}$ and $Y'_2 = \{\bar{x_e}, p\}$, which represent the interpretations $Y_1 = \emptyset$ 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 input atoms of external atoms may support themselves. More precisely, due to rule (3), an external atom is false by default if none of the rules (1) applies. If one of the external atom's input atoms depends on falsehood of the external atom, as in Example 7, then the input atom might be supported by falsehood of the external atom, although this falsehood itself depends on the input atom.

Extending the Rewriting to Negated External Atoms. In order to extend our approach to the inlining of negated external atoms $\text{not } e$ 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 $\text{not } e$ by a positive one e' such that $A \models e'$ iff $A \not\models e$ for all assignments A; 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 $\text{not } e$ in P, let*

$$P_{[\text{not } e]} = \{x_e \leftarrow S^+_{\mathbf{F}} \cup \{\bar{a} \mid \neg a \in S^-_{\mathbf{F}}\} \mid S_{\mathbf{F}} \in \mathcal{S}_{\mathbf{F}}(e,P)\} \quad (5)$$

$$\cup \left\{ \begin{matrix} \bar{a} \leftarrow \text{not } a; \bar{a} \leftarrow x_e \\ a \vee \bar{a} \leftarrow \text{not } \bar{x_e} \end{matrix} \;\middle|\; a \in I(e,P) \right\} \quad (6)$$

$$\cup \{\bar{x_e} \leftarrow \text{not } x_e\} \quad (7)$$

$$\cup\, P|_{\text{not } e \to x_e} \quad (8)$$

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

One can show that this rewriting is sound and complete.

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

Transforming Complete Families of Support Sets. 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 $\mathcal{S}_{\bar{\sigma}} = \{S_{\bar{\sigma}} \in \prod_{S_\sigma \in \mathcal{S}_\sigma} \neg S_\sigma \mid$*

$S_{\bar{\sigma}}$ is consistent} *is a negative resp. positive complete family of support sets, where* $\overline{\mathbf{T}} = \mathbf{F}$ *and* $\overline{\mathbf{F}} = \mathbf{T}$.

However, similarly to a transformation of the formula in 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.

5 Implementation and Evaluation

We implemented external source inlining in the DLVHEX system, which is based on GRINGO and CLASP. External sources are supposed to provide a complete set of (possibly nonground) support sets. The approach from this paper allows for evaluating a HEX-program completely by the backend without any external calls during solving (external calls are only necessary at the beginning for support set learning).

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. [2014a]) obsolete.

Experimental Setup. We compare three evaluation approaches. The **traditional** evaluation algorithm guesses the truth values of external atoms and verifies them by evaluation. During evaluation, conflict-driven learning techniques are applied to learn parts of the external atom's behavior. The approach based on **support sets** (**sup.sets**) learns support sets provided by the external source at the beginning. 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. The new **inlining** approach also learns support sets at the beginning, 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.

We present four benchmarks with 100 randomly generated instances each, which were run on a Linux server with two 12-core AMD 6176 SE CPUs/128GB RAM using a 300 secs timeout. Despite similar benchmarks, the runtimes are not directly comparable to those by Eiter et al. [2014b] because of other solver improvements in the meantime and, for the taxi benchmark, an adopted scenario; however, the trends concerning **sup.sets** and **traditional** are the same. Note that our goal is to show improvements compared to previous HEX-algorithms, but not to compare HEX to other formalisms.

Our hypothesis is that **inlining** outperforms both **traditional** and **sup.sets**. This is because the only significant costs when generating the rewriting come from 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 (as in our benchmarks).[3] On the other hand, with **inlining**, (i) no external calls and (ii) no additional minimality check are needed. Hence, we expect further benefits and negligible additional costs.

[3]If they are not small, **traditional** might be faster than **sup.sets** and **inlining**. Consider $P = \{p(n{+}1) \leftarrow \&even[p]()\} \cup \{p(i) \leftarrow \mid 1 \leq i \leq n\}$ where $\&even[p]()$ is true iff the number of true atoms over p is even. Then $\hat{P}$ has only two candidates which are easily checked, while exponentially many support sets must be generated.

n	all answer sets						first answer set					
	traditional		sup.sets		inlining		traditional		sup.sets		inlining	
6	185.45	(35)	23.57	(1)	11.47	(0)	12.05	(0)	1.09	(0)	0.76	(0)
7	251.68	(81)	83.24	(3)	22.21	(2)	22.25	(2)	3.19	(0)	1.53	(0)
8	266.22	(85)	183.48	(43)	59.54	(11)	61.33	(10)	22.42	(1)	3.10	(0)
9	272.70	(85)	263.01	(85)	86.07	(13)	76.74	(12)	56.57	(12)	6.18	(0)
10	278.26	(83)	275.47	(83)	121.39	(16)	102.86	(12)	98.96	(12)	11.97	(0)
11	292.05	(85)	300.00	(100)	167.00	(45)	158.73	(41)	176.44	(49)	22.52	(0)
12	300.00	(100)	300.00	(100)	180.43	(41)	159.64	(47)	210.52	(51)	40.43	(0)

Table 1: House Configuration Problem

House Problem. An abstraction of configuration problems considers sets of *cabinets*, *rooms*, *objects* and *persons* and assigns 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 (Mayer et al. 2009). 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.

Table 1 shows the results, where numbers in parentheses indicate timeout instances. We have that **sup.sets** clearly outperforms **traditional** when computing all answer sets due to faster candidate checking; **inlining** leads to a further speedup as it eliminates wrong guesses and the checking step altogether, while the additional initialization overhead is negligible. If only one answer set is computed, this initialization overhead even exceeds the benefits of **sup.sets** in some cases because the benefit is limited due to early abortion (as also observed by Eiter et al. [2014b]). However, the further performance boost by **inlining** compensates this drawback of **sup.sets** s.t. it is clearly the most efficient configuration.

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 s.t. adjacent nodes have different colors. We use an encoding which splits the guessing part P_{col} from the checking part P_{check}. The latter 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 P_{check}, extended with facts over predicate inp, delivers an answer set which contains inv. A compact complete family of support sets for $\&query[P_{check}, inp, inv]()$ exists. The size of the instances is the number of nodes n.

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

[4]from http://143.205.174.183/reconcile/tools.

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)	1.63	(0)	1.36	(0)	0.45 (0)	0.45 (0)	0.45 (0)
100	300.00	(100)	8.50	(0)	7.86	(0)	1.99 (0)	1.99 (0)	1.99 (0)
140	300.00	(100)	28.32	(0)	27.56	(0)	6.40 (0)	6.40 (0)	6.43 (0)
180	300.00	(100)	74.88	(0)	73.79	(0)	16.41 (0)	16.45 (0)	16.43 (0)
220	300.00	(100)	152.41	(21)	150.77	(20)	35.19 (0)	35.35 (0)	35.23 (0)

Table 2: Graph Coloring Problem

n	all answer sets						first answer set		
	traditional		sup.sets		inlining		traditional	sup.sets	inlining
4	0.90	(0)	1.49	(0)	0.26	(0)	0.20 (0)	1.49 (0)	0.18 (0)
5	37.09	(3)	45.42	(0)	1.39	(0)	2.90 (0)	45.28 (0)	0.21 (0)
6	225.01	(59)	262.08	(74)	12.40	(0)	63.71 (17)	262.14 (75)	0.26 (0)
7	300.00	(100)	300.00	(100)	186.74	(29)	207.57 (67)	300.00 (100)	0.32 (0)
8	300.00	(100)	300.00	(100)	295.53	(97)	277.10 (92)	300.00 (100)	0.41 (0)
9	300.00	(100)	300.00	(100)	300.00	(100)	297.01 (99)	300.00 (100)	0.52 (0)

Table 3: Driver - Customer Assignment Problem

The results are shown in Table 2. While **sup.sets** already outperforms **traditional**, **inlining** leads to a further small speedup. Compared to the house problem, 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 not harm.

Taxi Assignment. We consider a program with access to an ontology, cf. *DL-atoms* (Eiter et al. 2008), 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 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 3 shows the results. Here, **sup.sets** is counterproductive compared to **traditional** because of the large number of solution candidates. Although we add support sets as constraints (cf. Section 2), positive resp. negative support sets prevent only wrong false resp. true guesses, but not vice versa. Hence, the costs of not learning from external calls exceed the benefit of faster checking. However, since **inlining** prevents wrong guesses, it does not suffer this problem.

LUBM Diamond. 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.

n	all answer sets						first answer set		
	traditional		sup.sets		inlining		traditional	sup.sets	inlining
20	1.08	(0)	0.34	(0)	0.31 (0)		0.34	(0)	0.34 (0)	0.31 (0)
30	27.73	(3)	0.98	(0)	0.34 (0)		5.66	(0)	0.98 (0)	0.34 (0)
40	145.06	(35)	16.68	(2)	0.40 (0)		84.73	(14)	16.74 (2)	0.40 (0)
50	249.78	(76)	80.69	(15)	0.48 (0)		213.45	(60)	80.61 (15)	0.47 (0)
60	285.70	(90)	184.25	(47)	0.57 (0)		265.61	(85)	184.23 (47)	0.57 (0)
70	298.13	(99)	254.00	(74)	0.72 (0)		297.17	(99)	254.06 (73)	0.72 (0)

Table 4: Default Rules over LUBM in $DL\text{-}Lite_{\mathcal{A}}$

Table 4 shows the results. As already observed by Eiter et al. [2014b] and different from the previous benchmark, **sup.sets** outperforms **traditional** due to a smaller number of model candidates, hence fewer wrong guesses occur and the more efficient check compensates the lost possibility to learn from external calls. Again, **inlining** is the most efficient configuration as it does not only prevent wrong guesses but also spares external calls. Thanks to a compact family support sets, the speedup is dramatic.

Experiments Summary. The size of the **inlining** encoding is linked to the size of the complete family of support sets. Although it is exponential in the worst case, many practical source have compact families of support sets. In this case, the **inlining** approach is clearly superior to **sup.sets** (which is superior to **traditional**) as it eliminates the compatibility check and minimality check wrt. external sources altogether, while it has only slightly higher initialization overhead.

6 Discussion and Conclusion

Related Work. Our approach is related to evaluation approaches for DL-programs (Eiter et al. 2008) (programs with ontologies), cf. e.g. Heymans, Eiter, and Xiao [2010], but is more general. The rewriting uses the saturation technique and is related to the one by Alviano, Faber, and Gebser [2015] who translated aggregates to disjunctions. However, they support only a fixed set of aggregates while our approach supports arbitrary sources. Moreover, it eliminates external atoms completely, while Alviano, Faber, and Gebser [2015] still use simplified (monotonic) aggregates in the result.

Conclusion and Outlook. We presented an approach for external source inlining based on support sets. The program can then be evaluated by an ordinary ASP solver and external atom guesses do not need to be verified at all. All our experiments show a clear improvement over the previous approach by Eiter et al. [2014b], which is explained by the fact that the slightly higher initialization costs are exceeded by the significant benefits of avoiding external calls altogether.

Future work may include refinements of the rewriting. Currently, a new auxiliary variable $\bar{a}$ is introduced for all input atoms a of all external atoms. Note that it introduces even a new auxiliary atom for each external atom that uses a as input. 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.

References

Alviano, M.; Faber, W.; and Gebser, M. 2015. Rewriting recursive aggregates in answer set programming: back to monotonicity. *CoRR* abs/1507.03923.

Darwiche, A., and Marquis, P. 2011. A knowledge compilation map. *CoRR* abs/1106.1819.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2005. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In Kaelbling, L. P., and Saffiotti, A., eds., *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, 90–96. Denver, USA: Professional Book Center.

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.

Eiter, T.; Fink, M.; Krennwallner, T.; and Redl, C. 2012. Conflict-driven ASP solving with external sources. *TPLP* 12(4-5):659–679.

Eiter, T.; Fink, M.; Krennwallner, T.; Redl, C.; and Schüller, P. 2014a. Efficient HEX-program evaluation based on unfounded sets. *Journal of Artificial Intelligence Research* 49:269–321.

Eiter, T.; Fink, M.; Redl, C.; and Stepanova, D. 2014b. Exploiting support sets for answer set programs with external evaluations. In Brodley, C. E., and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 1041–1048. AAAI Press.

Eiter, T.; Ianni, G.; and Krennwallner, T. 2009. Answer Set Programming: A Primer. In Tessaris, S.; Franconi, E.; Eiter, T.; Gutierrez, C.; Handschuh, S.; Rousset, M.-C.; and Schmidt, R. A., eds., *5th International Reasoning Web Summer School (RW 2009), Brixen/Bressanone, Italy, August 30–September 4, 2009*, volume 5689 of *LNCS*, 40–110. Springer.

Faber, W.; Leone, N.; and Pfeifer, G. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence* 175(1):278–298.

Gebser, M.; Ostrowski, M.; and Schaub, T. 2009. Constraint answer set solving. In Hill, P., and Warren, D., eds., *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, volume 5649 of *Lecture Notes in Computer Science*, 235–249. Springer-Verlag.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9(3–4):365–386.

Heymans, S.; Eiter, T.; and Xiao, G. 2010. Tractable reasoning with dl-programs over datalog-rewritable description logics. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 35–40. IOS Press.

Mayer, W.; Bettex, M.; Stumptner, M.; and Falkner, A. 2009. On solving complex rack configuration problems using csp methods. In *IJCAI'09 Workshop on Configuration*.