Exploiting Unfounded Sets for HEX-Program Evaluation*

Thomas Eiter, Michael Fink, Thomas Krennwallner, Christoph Redl, and Peter Schüller

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,fink,tkren,redl,ps}@kr.tuwien.ac.at

Abstract. HEX programs extend logic programs with external computations through external atoms, whose answer sets are the minimal models of the Faber-Leone-Pfeifer-reduct. As already reasoning from Horn programs with nonmonotonic external atoms of polynomial complexity is on the second level of the polynomial hierarchy, answer set checking needs special attention; simply computing reducts and searching for smaller models does not scale well. We thus extend an approach based on unfounded sets to HEX and integrate it in a Conflict Driven Clause Learning framework for HEX program evaluation. It reduces the check to a search for unfounded sets, which is more efficiently implemented as a SAT problem. We give a basic encoding for HEX and show optimizations by additional clauses. Experiments show that the new approach significantly decreases runtime.

Keywords: Answer Set Programming, Nonmonotonic Reasoning, Unfounded Sets, FLP Semantics

1 Introduction

Answer Set Programming (ASP) is a declarative programming approach which due to expressive and efficient systems like SMODELS, DLV and CLASP, has been gaining popularity for many applications [2]. Current trends in computing, such as context awareness or distributed systems, raised the need for access to external sources in a program, which, e.g., on the Web ranges from light-weight data access (e.g., XML, RDF, or data bases) to knowledge-intensive formalisms (e.g., description logics).

To cater for this need, HEX-programs [9] extend ASP with so-called external atoms, through which the user can couple any external data source with a logic program. Roughly, such atoms pass information from the program, given by predicate extensions, into an external source which returns output values of an (abstract) function that it computes. This convenient extension has been exploited for many different applications, including querying data and ontologies on the Web, multi-context reasoning, or e-government, to mention a few (cf. [5]). It is highly expressive as recursive data exchange between the logic program and external sources is possible.

The semantics of HEX-programs is defined in terms of answer sets based on the FLP reduct [12]: an interpretation $\mathbf{A}$ is an answer set of a program Π, iff it is a $\subseteq$-minimal

* This research has been supported by the Austrian Science Fund (FWF) project P20840, P20841, P24090, and by the Vienna Science and Technology Fund (WWTF) project ICT08-020.

model of the FLP-reduct $f\Pi^{\mathbf{A}}$ of the program wrt. $\mathbf{A}$, which is the set of all rules whose body is satisfied by $\mathbf{A}$. This semantics is equivalent to the GL-reduct based semantics of ordinary logic programs [14], but has advantages for extensions with nonmonotonic aggregates [12] or the more general external atoms in HEX-programs.

Currently, a HEX-program Π is evaluated in two steps as follows. In step 1, external atoms are viewed as ordinary atoms (*replacement atoms*) and their truth values are guessed by added choice rules. The resulting ordinary ASP program $\hat{\Pi}$ is evaluated by an ordinary ASP solver and each answer set $\hat{\mathbf{A}}$ returned is checked against the external sources, i.e., the guess is verified. After that, the guess for the non-replacement atoms, called $\mathbf{A}$, is known to be a model of Π, but it is yet unknown whether $\mathbf{A}$ is also a subset-minimal model of the reduct $f\Pi^{\mathbf{A}}$. This has to be ensured in step 2 by an *FLP check*. A straightforward method, called *explicit FLP check*, is to compute $f\Pi^{\mathbf{A}}$ and to check whether it has some model smaller than $\mathbf{A}$. However, this approach is not efficient in practice, actually the explicit FLP check often dominates the overall runtime.

This calls for alternative methods to do the FLP check efficiently, which we address in this paper. For ordinary programs, unfounded sets proved to be a fruitful approach [16], which later had been extended to programs with aggregates [11]: an interpretation is an FLP-answer set of some program, if and only if it is unfounded-free, i.e., is disjoint from every unfounded set. Thus to decide whether a candidate is an answer set, one can simply search for unfounded sets, rather than to explicitly construct the reduct and enumerate its models in search for a smaller one.

Starting from this idea, we define unfounded sets for HEX-programs following [11] and explore their efficiency for FLP checking. Briefly, our main contributions are:

- We present an encoding of the unfounded set existence problem to a set of *nogoods*, i.e., constraints that have to be satisfied, and show that the solutions correspond 1-1 with the unfounded sets. They can then be computed using a SAT solver and a post-processing step which checks that the values of replacement atoms comply with the results of the external calls. Furthermore, we consider optimizations which hinge on dependency between external and ordinary atoms, determined in careful analysis. Benchmarks show that this strategy is already more efficient than the explicit FLP check.
- We consider how information gained in the FLP check can be used in generating candidate answer sets in step 1. Recently, adopting a Clause Driven Conflict Learning approach [3], this step has been enhanced by learning [6], in which nogoods describing the external source behavior are learned during the search (called *external behavior learning* or EBL), in order to guide it towards right guesses. We show how step 1 can learn additional nogoods from unfounded sets that avoid the reconstruction of the same or related unfounded sets, yielding further gain.

An experimental evaluation of the above techniques for advanced reasoning applications, including Multi-Context Systems [1, 8], abstract argumentation [4] and UNSAT testing [11], shows that unfounded sets checking combined with learning methods from [6] improves HEX-program evaluation considerably. As unfounded-freeness may be ensured by syntactic criteria in relevant cases (which makes the FLP check obsolete), the new approach enables significant speedup and enlarges the scope of HEX applicability. Proofs of our results are given in an extended version [7].

2 Preliminaries

In this section, we start with some basic definitions, and then introduce HEX-programs.

In accordance with [13, 6], a *(signed) literal* is a positive or a negative formula $\mathbf{T}a$ resp. $\mathbf{F}a$, where a is a ground atom of form $p(c_1, \ldots, c_\ell)$, with predicate p and constants $c_1, \ldots, c_\ell$, abbreviated $p(\mathbf{c})$. For a literal $\sigma = \mathbf{T}a$ or $\sigma = \mathbf{F}a$, let $\overline{\sigma}$ denote its opposite, i.e., $\overline{\mathbf{T}a} = \mathbf{F}a$ and $\overline{\mathbf{F}a} = \mathbf{T}a$. An *assignment* is a consistent set of literals $\mathbf{T}a$ or $\mathbf{F}a$, where $\mathbf{T}a$ expresses that $a \in \mathcal{A}$ and $\mathbf{F}a$ that $a \notin \mathcal{A}$. $\mathcal{A}$ is *complete*, also called an *interpretation*, if no assignment $\mathbf{A}' \supset \mathbf{A}$ exists. We denote by $\mathbf{A}^{\mathbf{T}} = \{a \mid \mathbf{T}a \in \mathbf{A}\}$ and $\mathbf{A}^{\mathbf{F}} = \{a \mid \mathbf{F}a \in \mathbf{A}\}$ the set of atoms that are true, resp. false in $\mathbf{A}$, and by $ext(q, \mathbf{A}) = \{\mathbf{c} \mid \mathbf{T}q(\mathbf{c}) \in \mathbf{A}\}$ the extension of a predicate q. Furthermore, $\mathbf{A}|_q$ is the set of all literals over atoms of form $q(\mathbf{c})$ in $\mathbf{A}$. For a list $\mathbf{q} = q_1, \ldots, q_k$ of predicates we write $p \in \mathbf{q}$ iff $q_i = p$ for some $1 \leq i \leq n$, and let $\mathbf{A}|_{\mathbf{q}} = \bigcup_j \mathbf{A}|_{q_j}$.

A *nogood* is a set $\{L_1, \ldots, L_n\}$ of literals $L_i, 1 \leq i \leq n$. An interpretation $\mathbf{A}$ is a *solution* to a nogood δ (resp. a set Δ of nogoods), iff $\delta \not\subseteq \mathbf{A}$ (resp. $\delta \not\subseteq \mathbf{A}$) for all $\delta \in \Delta$.

HEX-Program Syntax. As introduced in [9], HEX-programs are a generalization of (disjunctive) extended logic programs under the answer set semantics [14]; for details and background see [9]. HEX-programs extend ordinary ASP programs by *external atoms*, which enable a bidirectional interaction between a program and external sources of computation. External atoms have a list of input parameters (constants or predicate names) and a list of output parameters. Informally, to evaluate an external atom, the reasoner passes the constants and extensions of the predicates in the input tuple to the external source associated with the external atom. The external source computes output tuples which are with the output list. Formally, a *ground external atom* is of the form

$$\&g[\mathbf{p}](\mathbf{c}), \tag{1}$$

where $\mathbf{p} = p_1, \ldots, p_k$ are constant input parameters (predicate names or object constants), and $\mathbf{c} = c_1, \ldots, c_l$ are constant output terms.

Ground HEX-programs are then defined similar to ground ordinary ASP programs.

Definition 1 (Ground HEX-programs). *A ground HEX-program consists of rules*

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n \ , \tag{2}$$

where each a_i is an (ordinary) ground atom $p(c_1, \ldots, c_\ell)$ with constants c_i, $1 \leq i \leq \ell$, each b_j is either an ordinary ground atom or a ground external atom, and $k + n > 0$.[1]

The *head* of a rule r is $H(r) = \{a_1, \ldots, a_n\}$ and the *body* is $B(r) = \{b_1, \ldots, b_m,$ not $b_{m+1}, \ldots,$ not $b_n\}$. We call b or not b in a rule body a *default literal*; $B^+(r) = \{b_1, \ldots, b_m\}$ is the *positive body*, $B^-(r) = \{b_{m+1}, \ldots, b_n\}$ is the *negative body*. For a program Π, let $A(\Pi)$ be the set of all ordinary atoms occurring in Π. For a default literal b, let $\mathbf{t}b = \mathbf{T}a$ if $b = a$ for an atom a, and $\mathbf{t}b = \mathbf{F}a$ if $b = \text{not } a$. Conversely, $\mathbf{f}b = \mathbf{F}a$ if $b = a$ and $\mathbf{f}b = \mathbf{T}a$ if $b = \text{not } a$.

We also use non-ground programs. However, as suitable safety conditions allow for using a grounding procedure [10], we limit our investigation to ground programs.

HEX-Program Semantics and Evaluation. The semantics of a ground external atom $\&g[\mathbf{p}](\mathbf{c})$ wrt. an interpretation $\mathbf{A}$ is given by the value of a $1{+}k{+}l$-ary Boolean *oracle*

[1] For simplicity, we do not formally introduce strong negation but view, as customary, classical literals $\neg a$ as new atoms together with a nogood $\{\mathbf{T}a, \mathbf{T}\neg a\}$.

function, denoted by $f_{\&g}$ following [9], that is defined for all possible values of $\mathbf{A}$, $\mathbf{p}$ and $\mathbf{c}$. Thus, $\&g[\mathbf{p}](\mathbf{c})$ is true relative to $\mathbf{A}$ if and only if it holds that $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = 1$. Satisfaction of ordinary rules and ASP programs [14] is then extended to HEX-rules and programs in the obvious way, and the notion of extension $ext(\cdot, \mathbf{A})$ for external predicates $\&g$ with input lists $\mathbf{p}$ is naturally defined by $ext(\&g[\mathbf{p}], \mathbf{A}) = \{\mathbf{c} \mid f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = 1\}$.

An input predicate p of an external predicate with input list $\&g[\mathbf{p}]$ is *monotonic* (*antimonotonic*), iff $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = 1$ implies $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = 1$ ($f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = 0$ implies $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = 0$) for all $\mathbf{A}'$ s.t. $ext(p, \mathbf{A}') \supseteq ext(p, \mathbf{A})$ and $ext(q, \mathbf{A}') = ext(q, \mathbf{A})$ for $q \in \mathbf{p}$ and $q \neq p$. The sublist of all monotonic resp. antimonotonic p is denoted by $m(\&g)$ resp. $a(\&g)$ and the sublist of neither monotonic nor antimonotonic (i.e., *nonmonotonic*) p by $n(\&g)$; we also write $\mathbf{p}_\tau$ for $\tau(\&g)$, $\tau \in \{m, a, n\}$.

Definition 2 (FLP-Reduct [12]). *For an interpretation* $\mathbf{A}$ *over a program* Π*, the* FLP-reduct *of* Π *wrt.* $\mathbf{A}$ *is the set* $f\Pi^{\mathbf{A}} = \{r \in \Pi \mid \mathbf{A} \models b, \textit{for all } b \in B(r)\}$ *of all rules whose body is satisfied under* $\mathbf{A}$.

An assignment $\mathbf{A}_1$ is smaller or equal to another assignment $\mathbf{A}_2$ wrt. a program Π, denoted $\mathbf{A}_1 \leq_\Pi \mathbf{A}_2$ iff $\{\mathbf{T}a \in \mathbf{A}_1 \mid a \in A(\Pi)\} \subseteq \{\mathbf{T}a \in \mathbf{A}_2 \mid a \in A(\Pi)\}$.

Definition 3 (Answer Set). *An answer set of* Π *is a* $\leq_\Pi$*-minimal model* $\mathbf{A}$ *of* $f\Pi^{\mathbf{A}}$.

Since interpretations (thus answer sets) are complete assignments, slightly abusing notation, we uniquely identify them with the set of all positive literals they contain.

Example 1. Consider the program $\Pi = \{p \leftarrow \&id[p]()\}$, where $\&id[p]()$ is true iff p is true. Then Π has the answer set $\mathbf{A}_1 = \emptyset$; indeed it is a $\leq_\Pi$-minimal model of $f\Pi^{\mathbf{A}_1} = \emptyset$.

The answer sets of a HEX-program Π are determined by the DLVHEX solver using a transformation to ordinary ASP programs as follows. Each external atom $\&g[\mathbf{p}](\mathbf{c})$ in Π is replaced by an ordinary ground *external 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 to the program. The answer sets of the resulting *guessing program* $\hat{\Pi}$ are determined by an ordinary ASP solver and projected to non-replacement atoms. However, the resulting assignments are not necessarily models of Π, as the value of $\&g[\mathbf{p}]$ under $f_{\&g}$ can be different from the one of $e_{\&g[\mathbf{p}]}(\mathbf{c})$. Each answer set of $\hat{\Pi}$ is thus merely a *candidate* which must be checked against the external sources. If no discrepancy is found, the model candidate is a *compatible set* of Π. More precisely,

Definition 4 (Compatible Set). *A* compatible set *of a program* Π *is an assignment* $\hat{\mathbf{A}}$

(i) which is an answer set [14] of the guessing program $\hat{\Pi}$*, and*

(ii) $f_{\&g}(\hat{\mathbf{A}}, \mathbf{p}, \mathbf{c}) = 1$ *iff* $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \hat{\mathbf{A}}$ *for all external atoms* $\&g[\mathbf{p}](\mathbf{c})$ *in* Π*, i.e. the guessed values coincide with the actual output under the input from* $\hat{\mathbf{A}}$.

The compatible sets of Π computed by DLVHEX include (modulo $A(\Pi)$) all (FLP) answer sets. For each answer set $\mathbf{A}$ there is a compatible set $\hat{\mathbf{A}}$ such that $\mathbf{A}$ is the restriction of $\hat{\mathbf{A}}$ to non-replacement atoms, but not vice versa. To filter out the compatible sets which are not answer sets, the current evaluation algorithm proceeds as follows. Each compatible set $\mathbf{A}$ is fed to the FLP check, which explicitly constructs $f\Pi^{\mathbf{A}}$. After that, all models of the reduct are enumerated and compared to $\mathbf{A}$. If there is a model which is strictly smaller than $\mathbf{A}$ wrt. Π, then $\mathbf{A}$ is rejected, otherwise $\mathbf{A}$ is an answer set.

Example 2 (cont'd). Reconsider the program $\Pi = \{\, p \leftarrow \&id[p]() \,\}$ from above. Then the corresponding guessing program is $\hat{\Pi} = \{p \leftarrow e_{\&id[p]}(); e_{\&id[p]} \vee ne_{\&id[p]} \leftarrow\}$ and has the answer sets $\mathbf{A}_1 = \emptyset$ and $\mathbf{A}_2 = \{\mathbf{T}p, \mathbf{T}e_{\&id[p]}\}$. While $\mathbf{A}_1$ is also a $\leq_\Pi$-minimal model of $f\Pi^{\mathbf{A}_1} = \emptyset$, A_2 is not a $\leq_\Pi$-minimal model of $f\Pi^{\mathbf{A}_2} = \Pi$.

3 Unfounded Set Detection

It appears that in most current application scenarios there is no smaller model of the reduct $f\Pi^{\mathbf{A}}$, i.e., most assignments $\mathbf{A}$ extracted from compatible sets $\hat{\mathbf{A}}$ pass the FLP check. Moreover, this check is computationally costly: all models of $f\Pi^{\mathbf{A}}$ must be enumerated, along with calls to the external sources to ensure compatibility. Even worse, as we need to search for a smaller *model* and not just for a smaller compatible set, $f\Pi^{\mathbf{A}}$ usually has even more models then the original program. More precisely, the explicit FLP check corresponds to the search for compatible sets of the following program:

$$\begin{aligned} &f\hat{\Pi}^{\hat{\mathbf{A}}} \cup \{\leftarrow a \mid a \text{ is ordinary }, \mathbf{T}a \notin \hat{\mathbf{A}}\} \cup \{a \vee a' \leftarrow \mid \mathbf{T}a \in \hat{\mathbf{A}}\} \\ &\cup \{\leftarrow \text{not } \mathit{smaller}\} \cup \{\mathit{smaller} \leftarrow \text{not } a \mid a \text{ is ordinary }, \mathbf{T}a \in \hat{\mathbf{A}}\}. \end{aligned}$$

It consists of the reduct $f\hat{\Pi}^{\hat{\mathbf{A}}}$ and rules that restrict the search to proper subinterpretations of $\hat{\mathbf{A}}$, where $\mathit{smaller}$ is a new atom. Moreover, as we actually need to search for models and not just compatible sets, rules of the form $a \vee a' \leftarrow$ (where a' is a new atom for each $\mathbf{T}a \in \hat{\mathbf{A}}$) make sure that atoms can be arbitrarily true without having a justifying rule in Π. Because of these guessing rules, the rules in the reduct $f\hat{\Pi}^{\hat{\mathbf{A}}}$—except for the guesses on replacement atoms—can be rewritten to constraints, which is more efficient. Our comparison in Section 5 uses this optimized version of the explicit check, but still demonstrates a significant performance gain by our novel approach.

In this section we present a novel FLP check algorithm based on unfounded sets (UFS). That is, instead of explicitly searching for smaller models of the reduct, we check if the candidate answer set is unfounded-free. For now, the unfounded set-based check is also realized as a post-check, i.e., it is carried out only after the interpretation has been completed. Nevertheless it performs much better than the explicit FLP check. Investigating the effects of doing this check over partial interpretations and interleaving it with the main search for compatible sets is future work. We use unfounded sets for logic programs as introduced in [11] for programs with arbitrary aggregates.

Definition 5 (Unfounded Set). *Given a program Π and an assignment $\mathbf{A}$, let X be any set of ordinary ground atoms appearing in Π. Then, X is an* unfounded set for Π wrt. $\mathbf{A}$ *if, for each rule r having some atoms from X in the head, at least one of the following conditions holds, where $\mathbf{A} \mathbin{\dot\cup} \neg.X = (\mathbf{A} \setminus \{\mathbf{T}a \mid a \in X\}) \cup \{\mathbf{F}a \mid a \in X\}$:*

(i) some literal of $B(r)$ is false wrt. $\mathbf{A}$,
(ii) some literal of $B(r)$ is false wrt. $\mathbf{A} \mathbin{\dot\cup} \neg.X$, or
(iii) some atom of $H(r) \setminus X$ is true wrt. $\mathbf{A}$

Intuitively, an unfounded set is a set of atoms which only cyclically support each other. Answer sets can be characterized in terms of unfounded sets.

Definition 6 (Unfounded-free Interpretations). *An interpretation $\mathbf{A}$ of a program Π is* unfounded-free *iff $\mathbf{A}^{\mathbf{T}} \cap X = \emptyset$, for all unfounded sets X of Π wrt. $\mathbf{A}$.*

Theorem 1 (Characterization of Answer Sets). *A model* $\mathbf{A}$ *of a program* Π *is an answer set iff it is* unfounded-free.

Example 3. Consider the program Π and $\mathbf{A}_2$ from Example 2. Then $X = \{p\}$ is an unfounded set since X intersects with the head of $p \leftarrow \&id[p]()$ and $\mathbf{A} \dot{\cup} \neg.X \not\models \&id[p]()$. Therefore $\mathbf{A}_2$ is not unfounded-free and not an answer set.

3.1 Nogoods for Unfounded Set Search Encoding

We realize the search for unfounded sets using nogoods, i.e., for a given Π and an assignment $\mathbf{A}$ we construct a set of nogoods, such that solutions to this set correspond to (potential) unfounded sets. We then use a SAT solver to search for such unfounded sets.

Our encoding of unfounded set detection, which is related to [3] but respects external atoms, uses a set $\Gamma_\Pi^{\mathbf{A}} = N_\Pi^{\mathbf{A}} \cup O_\Pi^{\mathbf{A}}$, of nogoods where $N_\Pi^{\mathbf{A}}$ contains all *necessary* constraints and $O_\Pi^{\mathbf{A}}$ are *optional* optimization nogoods that prune irrelevant parts of the search space. The idea is that the set of ordinary atoms of a solution to $\Gamma_\Pi^{\mathbf{A}}$ represents a (potential) unfounded set U of Π wrt. $\mathbf{A}$, while the external replacement atoms encode the truth values of the corresponding external atoms under $\mathbf{A} \dot{\cup} \neg.U$.

Let $B_o^+(r)$ be the subset of $B^+(r)$ consisting of all *o*rdinary atoms, and $B_e(r)$ the subset of $B(r)$ consisting of all *e*xternal replacement atoms. Then, the nogood set $\Gamma_\Pi^{\mathbf{A}}$ is built over atoms $A(\Gamma_\Pi^{\mathbf{A}}) = A(\hat{\Pi}) \cup \{h_r, l_r \mid r \in \Pi\}$, where h_r, and l_r are additional atoms for every rule r in Π. The *mandatory part* $N_\Pi^{\mathbf{A}} = \{\{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{A}\}\} \cup \left(\bigcup_{r \in \Pi} R_{r,\mathbf{A}}\right)$ consists of a nogood $\{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{A}\}$, eliminating unfounded sets that do not intersect with true atoms in $\mathbf{A}$, as well as nogoods $R_{r,\mathbf{A}}$ for every $r \in \Pi$. The latter consist of a *head criterion* $H_{r,\mathbf{A}}$ and a *conditional part* $C_{r,\mathbf{A}}$ for each rule, defined by:

– $R_{r,\mathbf{A}} = H_{r,\mathbf{A}} \cup C_{r,\mathbf{A}}$, where

– $H_{r,\mathbf{A}} = \{\{\mathbf{T}h_r\} \cup \{\mathbf{F}h \mid h \in H(r)\}\} \cup \{\{\mathbf{F}h_r, \mathbf{T}h\} \mid h \in H(r)\}$
encodes that h_r is true for a rule r iff some atom of $H(r)$ is in the unfounded set;

– $$C_{r,\mathbf{A}} = \begin{cases} \{\{\mathbf{T}h_r\} \cup & \\ \quad \{\mathbf{F}a \mid a \in B_o^+(r), \mathbf{A} \models a\} \cup \{\mathbf{t}a \mid a \in B_e(\hat{r})\} \cup & \\ \quad \{\mathbf{T}h \mid h \in H(r), \mathbf{A} \models h\}\} & \text{if } \mathbf{A} \models B(r), \\ \{\} & \textit{otherwise} \end{cases}$$
encodes that condition (i), (ii) or (iii) of Definition 5 must hold if h_r is true.

More specifically, for an unfounded set U and a rule r with $H(r) \cap U \neq \emptyset$ (h_r is true) it must not happen that $\mathbf{A} \models B(r)$ (condition (i) fails), no $a \in B_o^+(r)$ with $\mathbf{A} \models a$ is in the unfounded set and all $a \in B_e(\hat{r})$ are true under $\mathbf{A} \dot{\cup} \neg.U$ (condition (ii) fails), and all $h \in H(r)$ with $\mathbf{A} \models h$ are in the unfounded set (condition (iii) fails).

Towards computing unfounded sets, observe that they can be extended to solutions to the set of nogoods $\Gamma_\Pi^{\mathbf{A}}$ over $A(\Gamma_\Pi^{\mathbf{A}})$. Conversely, the solutions to $\Gamma_\Pi^{\mathbf{A}}$ include specific extensions of all unfounded sets, characterized by *induced assignments*: That is, by assigning true to all atoms in U, to all h_r such that $H(r)$ intersects with U, and to all external replacement atoms $e_{\&g[\mathbf{p}]}(\mathbf{c})$ such that $\&g[\mathbf{p}](\mathbf{c})$ is true under $\mathbf{A} \dot{\cup} \neg.U$, and assigning false to all other atoms in $A(\Gamma_\Pi^{\mathbf{A}})$. More formally, we define:

Definition 7 (Induced Assignment of an Unfounded Set). *Let U be an unfounded set of a program Π wrt. assignment $\mathbf{A}$. The* assignment induced by U*, denoted $I(U, \Gamma_\Pi^{\mathbf{A}})$, is*

$$\begin{aligned}I(U,\Gamma_\Pi^{\mathbf{A}}) &= I'(U,\Gamma_\Pi^{\mathbf{A}}) \cup \{\mathbf{F}a \mid a \in A(\Gamma_\Pi^{\mathbf{A}}), \mathbf{T}a \notin I'(U,\Gamma_\Pi^{\mathbf{A}})\}, \textit{ where}\\ I'(U,\Gamma_\Pi^{\mathbf{A}}) &= \{\mathbf{T}a \mid a \in U\} \cup \{\mathbf{T}h_r \mid r \in \Pi, H(r) \cap U \neq \emptyset\} \cup\\ &\quad \{\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \mid e_{\&g[\mathbf{p}]}(\mathbf{c}) \in A(\hat{\Pi}), \mathbf{A} \,\dot\cup\, \neg.U \models \&g[\mathbf{p}](\mathbf{c})\}.\end{aligned}$$

While concrete instances for $O_\Pi^{\mathbf{A}}$ are defined in Section 4, note that for the next result we simply require that the optimization part $O_\Pi^{\mathbf{A}}$ is conservative in the sense that for every unfounded set U of Π wrt. $\mathbf{A}$, it holds that $I(U,\Gamma_\Pi^{\mathbf{A}})$ is a solution to $O_\Pi^{\mathbf{A}}$ as well (which is shown for the different optimizations considered in the next section). Then, the solutions to $\Gamma_\Pi^{\mathbf{A}}$ include all assignments induced by unfounded sets of Π wrt. $\mathbf{A}$ (but not every solution corresponds to such an induced assignment, intuitively because it does not reflect the semantics of external sources).

Proposition 1. *Let U be an unfounded set of a program Π wrt. assignment $\mathbf{A}$ such that $\mathbf{A}^{\mathbf{T}} \cap U \neq \emptyset$. Then $I(U,\Gamma_\Pi^{\mathbf{A}})$ is a solution to $\Gamma_\Pi^{\mathbf{A}}$.*

Corollary 1. *If $\Gamma_\Pi^{\mathbf{A}}$ has no solution, then $U \cap \mathbf{A}^{\mathbf{T}} = \emptyset$ for every unfounded set U of Π.*

The next property allows us to find the unfounded sets of Π wrt. $\mathbf{A}$ among all solutions to $\Gamma_\Pi^{\mathbf{A}}$ by using a postcheck on the external atoms.

Proposition 2. *Let S be a solution to $\Gamma_\Pi^{\mathbf{A}}$ such that*

(a) $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in S$ *and* $\mathbf{A} \not\models \&g[\mathbf{p}](\mathbf{c})$ *implies* $\mathbf{A} \,\dot\cup\, \neg.U \models \&g[\mathbf{p}](\mathbf{c})$*; and*
(b) $\mathbf{F}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in S$ *and* $\mathbf{A} \models \&g[\mathbf{p}](\mathbf{c})$ *implies* $\mathbf{A} \,\dot\cup\, \neg.U \not\models \&g[\mathbf{p}](\mathbf{c})$

where $U = \{a \mid a \in A(\Pi), \mathbf{T}a \in S\}$. Then U is an unfounded set of Π wrt. $\mathbf{A}$.

Informally, the proposition states that true non-replacement atoms in S which also appear in Π form an unfounded set, provided that truth of the external replacement atoms $e_{\&g[\mathbf{p}]}(\mathbf{c})$ in S coincides with the truth of the corresponding $\&g[\mathbf{p}](\mathbf{c})$ under $\mathbf{A} \,\dot\cup\, \neg.U$ (as in Definition 7). However, this check is just required if the truth value of $e_{\&g[\mathbf{p}]}(\mathbf{c})$ in S and of $\&g[\mathbf{p}](\mathbf{c})$ under $\mathbf{A}$ differ. This gives rise to an important optimization for the implementation: external atoms, whose (known) truth value of $\&g[\mathbf{p}](\mathbf{c})$ under $\mathbf{A}$ matches the truth value of $e_{\&g[\mathbf{p}]}(\mathbf{c})$ in S, do not need to be postchecked.

Example 4. Reconsider program $\Pi = \{r_1 : p \leftarrow \&id[p]()\}$ from Ex. 2 and the compatible set $\mathbf{A}_2 = \{\mathbf{T}p, \mathbf{T}e_{\&id[p]}\}$. The nogood set $N_\Pi^{\mathbf{A}_2} = \{\{\mathbf{T}h_{r_1}, \mathbf{F}p\}, \{\mathbf{F}h_{r_1}, \mathbf{T}p\}, \{\mathbf{T}h_{r_1}, \mathbf{T}e_{\&id[p]}(), \mathbf{T}p\}\}$ has solutions $S \supseteq \{\mathbf{T}h_{r_1}, \mathbf{T}p, \mathbf{F}e_{\&id[p]}()\}$, which correspond to the unfounded set $U = \{p\}$. Here, $\mathbf{F}e_{\&id[p]}()$ represents that $\mathbf{A}_2 \dot\cup \neg.U \not\models \&id[p]()$.

Note that due to the premises in Conditions (a) and (b) of Proposition 2, the postcheck is faster if $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in S$ whenever $\mathbf{A} \models \&g[\mathbf{p}](\mathbf{c})$ holds for many external atoms in Π. This can be exploited during the construction of S as follows: If it is not absolutely necessary to set the truth value of $e_{\&g[\mathbf{p}]}(\mathbf{c})$ differently, then carry over the value from $\&g[\mathbf{p}](\mathbf{c})$ under $\mathbf{A}$. Specifically, this is successful if $e_{\&g[\mathbf{p}]}(\mathbf{c})$ does not occur in $\Gamma_\Pi^{\mathbf{A}}$.

4 Optimization and Learning

In this section we first discuss some refinements and optimizations of our encoding of nogoods for UFS search. In particular, we add additional nogoods which prune irrelevant parts of the search space. After that, we propose a strategy for learning nogoods from detected unfounded sets, avoiding that the same unfounded set is generated again later.

4.1 Optimization

The following optimizations turned out to be effective in improving UFS search.

Restricting the UFS Search to Atoms in the Compatible Set. First, not all atoms in a program are relevant for the unfounded set search: atoms that are false under $\mathbf{A}$ can be ignored. Formally one can show the following:

Proposition 3. *If U is an unfounded set of Π wrt. $\mathbf{A}$ and there is an $a \in U$ s.t. $\mathbf{A} \not\models a$, then $U \setminus \{a\}$ is an unfounded set of Π wrt. $\mathbf{A}$.*

Avoiding Guesses of External Replacement Atoms. Second, in some situations the truth value of an external replacement atom b in a solution S to $\Gamma_{\Pi}^{\mathbf{A}}$ is void. That is, both $(S \setminus \{\mathbf{T}b, \mathbf{F}b\}) \cup \{\mathbf{T}b\}$ and $(S \setminus \{\mathbf{T}b, \mathbf{F}b\}) \cup \{\mathbf{F}b\}$ are solutions to $\Gamma_{\Pi}^{\mathbf{A}}$ (which represent the same unfounded set). Then we can set the truth value to an (arbitrary) fixed value instead of inspecting both alternatives. The following provides a sufficient criterion:

Proposition 4. *Let b be an external atom replacement, and let S be a solution to $\Gamma_{\Pi}^{\mathbf{A}}$. If for all rules $r \in \Pi$, such that $\mathbf{A} \models B(r)$ and where $b \in B^{+}(\hat{r})$ or $b \in B^{-}(\hat{r})$, either*

(a) for some $a \in B_{o}^{+}(r)$ such that $\mathbf{A} \models a$, it holds that $\mathbf{T}a \in S$; or
(b) for some $a \in H(r)$ such that $\mathbf{A} \models a$, it holds that $\mathbf{F}a \in S$;

then both $(S \setminus \{\mathbf{T}b, \mathbf{F}b\}) \cup \{\mathbf{T}b\}$ and $(S \setminus \{\mathbf{T}b, \mathbf{F}b\}) \cup \{\mathbf{F}b\}$ are solutions to $\Gamma_{\Pi}^{\mathbf{A}}$.

This property can be utilized by adding the following additional nogoods. Recall that $A(\Gamma_{\Pi}^{\mathbf{A}})$ contains atoms l_r for every $r \in \Pi$. They are intuitively used to encode for a solution S to $\Gamma_{\Pi}^{\mathbf{A}}$, whether the truth values of the external atom replacements in $B(r)$ are relevant, or whether they can be set arbitrarily for r. The following nogoods label relevant rules r, forcing l_r to be false iff one of the preconditions in Proposition 4 holds:

$$\begin{aligned} L_r^{\mathbf{A}} =& \{\{\mathbf{T}l_r, \mathbf{T}a\} \mid a \in B_o^{+}(r), \mathbf{A} \models a\} \cup \{\{\mathbf{T}l_r, \mathbf{F}a\} \mid a \in H(r), \mathbf{A} \models a\} \cup \\ & \{\{\mathbf{F}l_r\} \cup \{\mathbf{F}a \mid a \in B_o^{+}(r), \mathbf{A} \models a\} \cup \{\mathbf{T}a \mid a \in H(r), \mathbf{A} \models a\}\}. \end{aligned}$$

These constraints exclusively enforce $\mathbf{T}l_r$ or $\mathbf{F}l_r$. Hence, the truth value of l_r deterministically depends on the other atoms, i.e., the nogoods do not cause additional guessing.

By Prop. 4 we can set the truth value of an external replacement atom b arbitrarily, if l_r is false for all r such that $b \in B^{+}(\hat{r})$ or $b \in B^{-}(\hat{r})$. As mentioned after Prop. 2, it is advantageous to set the truth value of $e_{\&g[\mathbf{p}]}(\mathbf{c})$ to the one of $\&g[\mathbf{p}](\mathbf{c})$ under $\mathbf{A}$, because this can reduce the number of external atoms that must be checked. The following nogoods enforce a coherent interpretation of the external replacement atoms:

$$\begin{aligned} F_r^{\mathbf{A}} =& \{\{\mathbf{F}l_r \mid b \in B^{+}(\hat{r}) \cup B^{-}(\hat{r})\} \cup \{\mathbf{F}b\} \mid b \in B_e(\hat{r}), \mathbf{A} \models b\} \cup \\ & \{\{\mathbf{F}l_r \mid b \in B^{+}(\hat{r}) \cup B^{-}(\hat{r})\} \cup \{\mathbf{T}b\} \mid b \in B_e(\hat{r}), \mathbf{A} \not\models b\} \end{aligned}$$

In summary, our optimization part therefore is given by $O_{\Pi}^{\mathbf{A}} = \bigcup_{r \in \Pi} L_r^{\mathbf{A}} \cup F_r^{\mathbf{A}}$.

Example 5. Consider the program $\Pi = \{r_1 : p \leftarrow \&id[p]().,r_2 : q \leftarrow \&id[q]().\}$, and the compatible set $\mathbf{A} = \{\mathbf{T}p, \mathbf{F}q, \mathbf{T}e_{\&id[p]}(), \mathbf{F}e_{\&id[q]}()\}$. Then, $N_{\Pi}^{\mathbf{A}}$ has solutions $S_1 \supseteq \{\mathbf{T}h_{r_1}, \mathbf{T}p, \mathbf{F}e_{\&id[p]}(), \mathbf{F}h_{r_2}, \mathbf{F}q, \mathbf{F}e_{\&id[q]}()\}$ and $S_2 \supseteq \{\mathbf{T}h_{r_1}, \mathbf{T}p, \mathbf{F}e_{\&id[p]}(), \mathbf{F}h_{r_2}, \mathbf{F}q, \mathbf{T}e_{\&id[q]}()\}$ (which represent the same unfounded set $U = \{p\}$). Here, the optimization part for r_2, $L_{r_2}^{\mathbf{A}} \cup F_{r_2}^{\mathbf{A}} = \{\{\mathbf{T}l_{r_2}, \mathbf{F}q\}, \{\mathbf{F}l_{r_2}, \mathbf{T}q\}, \{\mathbf{F}l_{r_2}, \mathbf{T}e_{\&id[q]}()\}\}$, eliminates solutions S_2 for $\Gamma_{\Pi}^{\mathbf{A}}$. This is beneficial as for solutions S_1 the postcheck is easier ($e_{\&id[q]}()$ in S_1 and $\&id[q]()$ have the same truth value under $\mathbf{A}$).

Exchanging Nogoods between UFS and Main Search. The third optimization allows for the exchange of learned knowledge about external atoms between the UFS check and the main search for compatible sets. For this purpose, we first define nogoods which correctly describe the input-output relationship of external atoms.

Definition 8. *A nogood of the form $N = \{\mathbf{T}t_1, \dots, \mathbf{T}t_n, \mathbf{F}f_1, \dots, \mathbf{F}f_m, \circ e_{\&g[\mathbf{p}]}(\mathbf{c})\}$, where $\circ$ is $\mathbf{T}$ or $\mathbf{F}$, is a* valid input-output-relationship*, iff for all assignments $\mathbf{A}$, $\mathbf{T}t_i \in \mathbf{A}$, for $1 \leq i \leq n$, and $\mathbf{F}f_i \in \mathbf{A}$, for $1 \leq i \leq m$, implies $\mathbf{A} \models \&g[\mathbf{p}](\mathbf{c})$ if $\circ = \mathbf{F}$, and $\mathbf{A} \not\models \&g[\mathbf{p}](\mathbf{c})$ if $\circ = \mathbf{T}$.*

Let N be a nogood which is a valid input-output-relationship learned during the *main search*, i.e., for compatible sets of $\hat{\Pi}$, and let $\bar{\circ} = \mathbf{T}$ if $\circ = \mathbf{F}$, resp. $\bar{\circ} = \mathbf{F}$ if $\circ = \mathbf{T}$.

Definition 9 (Nogood Transformation $\mathcal{T}$). *For a valid input-output relationship N and an assignment $\mathbf{A}$, the nogood transformation $\mathcal{T}$ is defined as*

$$\mathcal{T}(N, \mathbf{A}) = \begin{cases} \emptyset & \textit{if } \mathbf{F}t_i \in \mathbf{A} \textit{ for some } 1 \leq i \leq n, \\ \{\{\mathbf{F}t_1, \dots, \mathbf{F}t_n\} \cup \{\circ e_{\&g[\mathbf{p}]}(\mathbf{c})\}\} \cup & \\ \quad \{\mathbf{T}f_i \mid 1 \leq i \leq m, \mathbf{A} \models f_i\} & \textit{otherwise}. \end{cases}$$

The next result states that $\mathcal{T}(N, \mathbf{A})$ can be considered, for all valid input-output relationships N under all assignments $\mathbf{A}$, without losing unfounded sets.

Proposition 5. *Let N be a valid input-output relationship, and let U be an unfounded set wrt. Π and $\mathbf{A}$. Then $I(U, \Gamma_{\Pi}^{\mathbf{A}})$ is a solution to $\mathcal{T}(N, \mathbf{A})$.*

Hence, all valid input-output relationships for external atoms which are learned during the search for compatible sets, can be reused (applying the above transformation) for the unfounded set check. Moreover, during the evaluation of external atoms in the postcheck for candidate unfounded sets (solutions to $\Gamma_{\Pi}^{\mathbf{A}}$), further valid input-output relationships might be learned. These can in turn be used by further unfounded set checks.

Example 6 (Set Partitioning). Consider the program Π

$$\begin{aligned} sel(a) &\leftarrow domain(a), \&diff[domain, nsel](a) \\ nsel(a) &\leftarrow domain(a), \&diff[domain, sel](a) \\ domain(a) &\leftarrow \end{aligned}$$

where $\&diff[p, q](X)$ computes the set of all elements X which are in the extension of p but not in the extension of q. Informally, this program implements a choice from $sel(a)$ and $nsel(a)$. Consider the compatible set $\mathbf{A}^{\mathbf{T}} = \{domain(a), sel(a), e_{\&diff[nsel]}(a)\}$. Suppose the main search has learned the input-output relationship $N = \{\mathbf{T}domain(a), \mathbf{F}nsel(a), \mathbf{F}e_{\&diff[nsel]}(a)\}$. Then the transformed nogood is $\mathcal{T}(N,\mathbf{A})=\{\{\mathbf{F}domain(a), \mathbf{F}e_{\&diff[nsel]}(a)\}\}$, which intuitively encodes that, if $domain(a)$ is *not* in the unfounded set U, then $e_{\&diff[nsel]}(a)$ is true under $\mathbf{A} \dot{\cup} \neg.U$. This is clear because $e_{\&diff[nsel]}(a)$ is true under $\mathbf{A}$ and it can only change its truth value if $domain(a)$ becomes false.

However, it is important that this optimization *cannot* be simultaneously used with the previous one as this can result in contradictions due to (transformed) learned nogoods. Consequently, the previous optimization has been disabled in running our experiments.

4.2 Learning Nogoods from Unfounded Sets

Until now only detecting unfounded sets has been considered. A strategy to *learn* from detected unfounded sets for the main search for compatible sets is missing. Here we develop such a strategy and call it *unfounded set learning* (UFL).

Example 7. Consider the program $\Pi = \{\, p \leftarrow \&id[p]();\ x_1 \vee x_2 \vee \cdots \vee x_k \leftarrow \}$. As we know from Example 3, $\{p\}$ is a UFS wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}e_{\&id}()\}$, regarding just the first rule. However, the same is true for any $\mathbf{A}' \supset \mathbf{A}$ regarding Π, i.e., p must never be true.

The program in Example 7 has many compatible sets, and half of them (all where p is true) will fail the UFS check for the same reason. We thus develop a strategy for generating additional nogoods to guide the further search for compatible sets in a way, such that the same unfounded sets are not reconsidered.

For an unfounded set U of Π wrt. $\mathbf{A}$ we define the following set of learned nogoods: $L(U, \Pi, \mathbf{A}) = \{\{\sigma_0, \sigma_1, \ldots, \sigma_j\} \mid \sigma_0 \in \{\mathbf{T}a \mid a \in U\}, \sigma_i \in H_i \textit{ for all } 1 \leq i \leq j)\}$, where $H_i = \{\mathbf{T}h \in H(r_i) \mid h \notin U, \mathbf{A} \models h\} \cup \{\mathbf{F}b \in B_o^+(r_i) \mid \mathbf{A} \not\models b\}$ and $\{r_1, \ldots, r_j\} = \{r \in \Pi \mid H(r) \cap U \neq \emptyset, U \cap B_o^+(r) = \emptyset\}$ is the set of external rules of Π wrt. U, i.e., all rules which do not depend on U.

Formally we can show that adding this set of nogoods is correct:

Proposition 6. *If U is an unfounded set of Π wrt. $\mathbf{A}$, then every answer set of Π is a solution to the nogoods in $L(U, \Pi, \mathbf{A})$.*

Example 8. Consider the program Π from Example 7 and suppose we have found the unfounded set $U = \{p\}$ wrt. interpretation $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}x_1\} \cup \{\mathbf{F}a_i \mid 1 < i \leq k\}$. Then the learned nogood $L_2(U, \mathbf{A}, \Pi) = \{\mathbf{T}p\}$ immediately guides the search to the part of the search tree where p is false, i.e., roughly half of the guesses are avoided.

We also considered a different learning strategy based on the models of $f\Pi^{\mathbf{A}}$ rather than the unfounded set U itself, hinging on the observation (cf. [12]) that for every unfounded set U, $\mathbf{A} \dot{\cup} \neg.U$ is a model of $f\Pi^{\mathbf{A}}$ (hence $U \neq \emptyset$ refutes $\mathbf{A}$ as a minimal model of $f\Pi^{\mathbf{A}}$). However, this strategy appeared to be inferior to the one above.

5 Implementation and Evaluation

For implementing our technique, we integrated CLASP into our prototype system DLVHEX; we use CLASP as an ASP solver for computing compatible sets and as a SAT solver for solving the nogood set of the UFS check. We evaluated the implementation on a Linux server with two 12-core AMD 6176 SE CPUs with 128GB RAM.

Table 1 summarizes our benchmark results (plain stands for disabling EBL and UFL). We can see a clear improvement both for synthetic and for application instances,[2] due to the UFS check and EBL. Moreover, a closer analysis shows that the UFS check in some cases not only decreases the runtime but also the numbers of enumerated candidates (UFS candidates resp. model candidates of the FLP reduct) and of external atom evaluations.

[2] Detailed instance information: `http://www.kr.tuwien.ac.at/staff/ps/unfoundedsets/`

Set Partitioning. This benchmark extends the program from Ex. 6 by the additional constraint $\leftarrow sel(X), sel(Y), sel(Z), X \neq Y, X \neq Z, Y \neq Z$ and varies the size of $domain$. Here we see a big advantage of the UFS check over the explicit check, both for computing all answer sets and for finding the first one. A closer investigation shows that the improvement is mainly due to the optimizations described in Sec. 4 which make the UFS check investigate significantly fewer candidates than the explicit FLP check. Furthermore the UFS check requires fewer external computations.

Multi-Context Systems (MCSs). MCSs [1] are a formalism for interlinking knowledge based systems; in [8], *inconsistency explanations (IEs)* for an MCS were defined. This benchmark computes the IEs, which correspond 1-1 to answer sets of an encoding rich in cycles through external atoms (which evaluate local knowledge base semantics). We use random instances of different topologies created with an available benchmark generator.

For most instances, we observed that the number of candidates for smaller models of the FLP reduct equals the one of unfounded set candidates. This is intuitive as each unfounded set corresponds to a smaller model; the optimization techniques do not prune the search space in this case. However, as we stop the enumeration as soon as a smaller model resp. an unfounded set is found, depending on the specific program and solver heuristics, the explicit and the UFS check may consider different numbers of interpretations. This explains why the UFS check is sometimes slightly slower than the explicit check. However, it always has a smaller delay between different UFS candidates, which sometimes makes it faster even if it visits more candidates.

The effects of external behavior learning [6] and of unfounded set learning is clearly evident in the MCS benchmarks: the UFS check profits more from EBL than the explicit check, further adding to its advantage. By activating UFL (not possible in the explicit check) we gain another significant speedup.

Intuitively, consistent and inconsistent MCSs are dual, as for each candidate the explicit resp. UFS check fails, i.e., stops early, vs. for some (or many) candidates the check succeeds (stops late). However, the mixed results do not permit us to draw solid conclusions on the computational relationship of the evaluation methods.

Note that MCS topologies are bound to certain system sizes, and the difficulty of the instances varies among topologies; thus larger instances may have shorter runtimes.

Abstract Argumentation. In this benchmark we compute ideal set extensions for randomized instances of abstract argumentation frameworks [4] of different sizes.

Table 1c shows average runtimes, each accumulated over 10 benchmark instances. In these instances, few unfounded sets exist, hence both the explicit and the UFS check often enumerate all candidates before they stop the search. As with MCS, the numbers of reduct model candidates and UFS candidates is in most cases equal, but the UFS check again enumerates its candidates faster; this explains the observed speedup.

Different from MCS, external learning prunes the search space only very little for these benchmarks, which can be explained by the structure of the encoding. Also UFL does not help much here, as few unfounded sets exist.

UNSAT. We also experimented with an encoding of the propositional UNSAT problem based on aggregates (not shown in figures), which was used in [11] to show the hardness result for the UFS decision problem. Here the optimizations discussed in Section 4 prune

Table 1: Benchmark Results (— indicates timeout (300s) of resp. instances)

(a) Inconsistent MCSs

#contexts	compute all answer sets					finding first answer set				
	explicit check		UFS check			explicit check		UFS check		
	plain	+EBL	plain	+EBL	+UFL	plain	+EBL	plain	+EBL	+UFL
3	9.08	6.11	6.29	2.77	0.85	4.01	2.53	3.41	1.31	0.57
4	89.71	36.28	80.81	12.63	5.27	53.59	16.99	49.56	6.09	1.07
5	270.10	234.98	268.90	174.23	18.87	208.62	93.29	224.01	32.85	3.90
6	236.02	203.13	235.55	179.24	65.49	201.84	200.06	201.24	166.04	28.34
7	276.94	241.27	267.82	231.08	208.47	241.09	78.72	240.72	66.56	16.41
8	286.61	153.41	282.96	116.89	69.69	201.10	108.29	210.61	103.11	30.98
9	—	208.92	—	191.46	175.26	240.75	112.08	229.14	76.56	44.73
10	—	—	—	289.87	289.95	—	125.18	—	75.24	27.05

(b) Consistent MCSs

#contexts	(no answer sets)				
	explicit check		UFS check		
	plain	+EBL	plain	+EBL	+UFL
3	8.61	4.68	7.31	2.44	0.50
4	86.55	48.53	80.31	25.98	1.89
5	188.05	142.61	188.10	94.45	4.62
6	209.34	155.81	207.14	152.32	14.39
7	263.98	227.99	264.00	218.94	49.42
8	293.64	209.41	286.38	189.86	124.23
9	—	281.98	—	260.01	190.56
10	—	274.76	—	247.67	219.83

(c) Argumentation (plain)

#args	all answer sets		first answer set	
	Explicit	UFS	Explicit	UFS
5	1.47	1.13	0.70	0.62
6	4.57	2.90	1.52	1.27
7	19.99	10.50	3.64	2.77
8	80.63	39.01	9.46	6.94
9	142.95	80.66	30.12	20.97
10	240.46	122.81	107.14	63.50

(d) Set Partitioning

	n	3	4	5	6	7	8	9	10	11	12	13	···	20
all answers	explicit	0.2	1.2	10.9	94.3	—	—	—	—	—	—	—	—	—
	+EBL	0.1	0.5	4.3	34.8	266.1	—	—	—	—	—	—	—	—
	UFS	0.1	0.1	0.2	0.3	0.8	1.8	4.5	11.9	32.4	92.1	273.9	—	—
	+EBL	0.1	0.1	0.1	0.1	0.2	0.2	0.3	0.4	0.6	0.8	1.2	···	11.1
first answer	explicit	0.1	0.2	0.7	4.3	26.1	163.1	—	—	—	—	—	—	—
	+EBL	0.1	0.2	0.8	4.9	31.1	192.0	—	—	—	—	—	—	—
	UFS	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	···	0.5
	+EBL	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	···	0.3

the search space (as in the set partitioning benchmark), which makes the UFS check enumerate fewer candidates, involving also fewer external atom calls.

6 Discussion and Conclusion

Related to our work is [15], which reduces stable model checking for disjunctive logic programs to unsatisfiability testing of CNFs, which like answer set checking from FLP-reducts is co-NP-complete [12]. The difference between ordinary disjunctive programs and FLP programs with external atoms is that co-NP-hardness holds already for Horn programs with nonmonotonic external atoms that are decidable in polynomial time. For computationally harder external atoms, the complexity increases relative to an oracle for the external function (see [12]). The approach of [15] is extended to conflict-driven learning and unfounded set checking in [3]. Here, two CLASP [13] instances generate and check answer set candidates. As model checking may become computationally harder in our setting, the results there do not carry over immediately.

We presented a new algorithm for deciding whether a model $\mathbf{A}$ of a HEX-program Π is a subset-minimal model of its FLP-reduct $f\Pi^{\mathbf{A}}$, adopting the notion of unfounded set in [11]. We realized unfounded set (UFS) checking by an encoding as a SAT instance, which produces candidate unfounded sets. Subsequently a (rather simple) postcheck decides whether there is indeed an unfounded set. Experiments have shown that this check is much more efficient than the explicit minimality check. We showed how to learn from identified unfounded sets, by deriving nogoods which guide future search in model generation and help avoiding to rediscover unfounded sets.

Our ongoing work includes interleaving UFS checks with the model generation process, i.e., on incomplete interpretations. If it is clear that a partial interpretation can never become an answer set, one can backtrack earlier; this may pay off for certain classes of instances. Furthermore, we investigate sufficient conditions to simplify the UFS check, aim at syntactic properties that are easy to check. Of particular interest are relevant program classes for which the UFS check can be skipped; this holds e.g. for programs without cyclic information flow through external atoms.

Another issue for future work is to study heuristics for guiding the search for an unfounded set. Currently, our implementation applies the same strategies as for the model generation task. Our experimental comparison with the explicit FLP check in terms of candidate sets considered, however, suggests that there might be room for improvement by employing specific choices. Developing appropriate such heuristics, and validating their effectiveness on candidate set enumeration remains to be explored.

References

1. Brewka, G., Eiter, T.: Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems. In: AAAI'07. pp. 385–390. AAAI Press (2007)
2. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM 54(12), 92–103 (2011)
3. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-driven disjunctive answer set solving. In: KR'08. pp. 422–432. AAAI Press (2008)
4. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77(2), 321–357 (1995)
5. Eiter, T., Fink, M., Ianni, G., Krennwallner, T., Schüller, P.: Pushing efficient evaluation of HEX programs by modular decomposition. In: LPNMR'11. pp. 93–106 (2011)
6. Eiter, T., Fink, M., Krennwallner, T., Redl, C.: Conflict-driven ASP solving with external sources. Theor. Pract. Log. Prog. (2012), to appear
7. Eiter, T., Fink, M., Krennwallner, T., Redl, C., Schüller, P.: Improving HEX-Program Evaluation based on Unfounded Sets. Tech. Rep. INFSYS RR-1843-12-08. TU Wien (2012)
8. Eiter, T., Fink, M., Schüller, P., Weinzierl, A.: Finding explanations of inconsistency in Multi-Context Systems. In: KR'10. pp. 329–339. AAAI Press (2010)
9. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In: IJCAI'05. pp. 90–96. Professional Book Center (2005)
10. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning. In: ESWC'06. pp. 273–287. (2006)
11. Faber, W.: Unfounded sets for disjunctive logic programs with arbitrary aggregates. In: LPNMR'05. pp. 40–52. Springer (2005)
12. Faber, W., Leone, N., Pfeifer, G.: Semantics and complexity of recursive aggregates in answer set programming. Artif. Intell. 175(1), 278–298 (2011)
13. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. 187–188, 52–89 (2012)
14. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generat. Comput. 9(3–4), 365–386 (1991)
15. Koch, C., Leone, N., Pfeifer, G.: Enhancing disjunctive logic programming systems by SAT checkers. Artif. Intell. 151(1–2), 177–212 (2003)
16. Leone, N., Rullo, P., Scarcello, F.: Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation. Inform. Comput. 135(2), 69–112 (1997)