Journal of Artificial Intelligence Research 49 (2014) 269–321 Submitted 09/13; published 02/14

Efficient HEX-Program Evaluation Based on Unfounded Sets

Thomas Eiter EITER@KR.TUWIEN.AC.AT
Michael Fink FINK@KR.TUWIEN.AC.AT
Thomas Krennwallner TKREN@KR.TUWIEN.AC.AT
Christoph Redl REDL@KR.TUWIEN.AC.AT
Institut für Informationssysteme
Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria

Peter Schüller PETERSCHUELLER@SABANCIUNIV.EDU
Faculty of Engineering and Natural Sciences
Sabanci University
Orhanli, Tuzla, 34956 Istanbul, Turkey

Abstract

HEX-programs extend logic programs under the answer set semantics with external computations through external atoms. As reasoning from ground Horn programs with nonmonotonic external atoms of polynomial complexity is already on the second level of the polynomial hierarchy, minimality checking of answer set candidates needs special attention. To this end, we present an approach based on *unfounded sets* as a generalization of related techniques for ASP programs. The unfounded set detection is expressed as a propositional SAT problem, for which we provide two different encodings and optimizations to them. We then integrate our approach into a previously developed evaluation framework for HEX-programs, which is enriched by additional learning techniques that aim at avoiding the reconstruction of the same or related unfounded sets. Furthermore, we provide a syntactic criterion that allows one to skip the minimality check in many cases. An experimental evaluation shows that the new approach significantly decreases runtime.

1. Introduction

Answer Set Programming (ASP) is a declarative problem solving approach. Due to expressive extensions and efficient systems like SMODELS (Simons, Niemelä, & Soininen, 2002), DLV (Leone, Pfeifer, Faber, Eiter, Gottlob, Perri, & Scarcello, 2006) and CLASP (Gebser, Kaufmann, & Schaub, 2012), it has been gaining popularity for many applications (Brewka, Eiter, & Truszczyński, 2011). However, current trends in computing, such as context awareness or distributed systems, raised the need for access to external sources in a program. For instance, external sources on the Web range from light-weight data access (e.g., XML, RDF, or data bases) to knowledge-intensive formalisms (e.g., OWL ontologies).

To cater for this need, Eiter, Ianni, Schindlauer, and Tompits (2005) defined HEX-programs as an extension of ASP with so-called external atoms, through which the user can couple any external information source with a logic program. Roughly, such atoms pass information, given by predicate extensions, from the program to an external source which returns output values of an (abstract) function that it computes. For example, a rule $nb(X, Y) \leftarrow \&neighbor[{'map'}, X](Y)$ may informally import for a point X on a map that is stored in the file map (in a particular data format), each point Y in the neighborhood of X into the predicate nb. Such convenient external access has been exploited for many applications, including querying data and ontologies (Eiter et al., 2008b; Hoehndorf et al., 2007; Marano et al., 2010), e-government (Zirtiloğlu & Yolum, 2008), fuzzy answer set programming (Nieuwenborgh, Cock, & Vermeir, 2007a), multi-context reasoning (Brewka & Eiter, 2007; Eiter et al., 2012b), (Nieuwenborgh et al., 2007b; Basol et al., 2010). The formalism is highly expressive as recursive data exchange between the rules and external sources is possible.

The semantics of HEX-programs is model-based and given by answer sets following the approach of Faber, Leone, and Pfeifer (2011), which extends the answer set semantics of logic programs (Gelfond & Lifschitz, 1991) to logic programs with aggregates; the Faber et al. approach (known as the FLP semantics) preserves the property that answer sets have, in the spirit of the closed world assumption, smallest positive information content, which is formally captured by a minimality condition on models.

The current approach for the evaluation of HEX-programs (Eiter et al., 2006a, 2011) is to rewrite a given HEX-program to an answer set program by (i) eliminating external atoms in favor of auxiliary atoms using so called *replacement atoms*, and (ii) introducing auxiliary rules such that the answer sets of the HEX-program Π correspond to a subset of the answers sets of the resulting program $\hat{\Pi}$ in which the auxiliary atoms faithfully represent the values of the external atoms; this compatibility condition of an answer set of $\hat{\Pi}$ is tested in a postcheck.

For computing the answer sets of a (disjunctive) logic program P like $\hat{\Pi}$, different methods have been proposed. An immediate one is to implement the definition of an answer set and test whether a given interpretation is a minimal model of the so called reduct of the program P wrt. the interpretation; to this end, a suitable candidate answer set might be guessed or generated by heuristics. This approach was essentially adopted for the solvers GNT (Janhunen et al., 2006) and CMODELS (Lierler, 2005), which use for this test a logic program, respectively a SAT encoding. A different approach was presented by Leone et al. (1997) based on the notion of *unfounded set* (Van Gelder, Ross, & Schlipf, 1991), which they extended from normal (non-disjunctive) to disjunctive logic programs. Intuitively, a set U of atoms is unfounded wrt. to a model of a program P, if switching all atoms in U to false does not lead to violated rules; the answer sets of P are then its models that are unfounded-free, i.e., the models disjoint from all respective unfounded sets. For checking (un)foundedness of a given candidate answer set, Koch et al. (2003) presented a SAT encoding. Drescher et al. (2008) later exploited findings of them and Leone et al. to extend the technique of conflict-driven clause learning used by the CLASP solver to disjunctive logic programs.

In all the quoted works, however, access to external sources was not an issue, and thus they cannot be deployed to HEX-programs. In fact, in addition to the compatibility check of an answer set of the replacement program $\hat{\Pi}$, the current HEX evaluation must in a second step test the minimality of the interpretation induced for the HEX-program Π wrt. the program reduct. This method, which we refer to as the *explicit FLP check*, turns out to be less efficient in practice, and it often dominates the total runtime; thus a more efficient method is desirable.

Motivated by this and the seminal approach of Leone et al., we consider in this paper the use of unfounded sets as an alternative to the explicit FLP check for HEX-programs, which we refer to as the *unfounded set check*. To this end, we extend the notion of unfounded sets for disjunctive logic programs to HEX-programs, following the lines of Faber (2005), where unfounded sets for logic programs with aggregates were defined, and consider their use in combination with clause learning techniques. Our main contributions are summarized as follows:

- We present a basic encoding of unfounded set existence to a set of *nogoods*, i.e., constraints that have to be satisfied, and we show that its solutions are in 1-1 correspondence with the unfounded sets. The latter can thus be computed running a SAT solver, followed by a post-processing step which checks that the values of replacement atoms are compatible with the external call results. Benchmarks show that this strategy is already more efficient than the explicit FLP check.

- We then present an advanced encoding of unfounded set existence that is reusable for any interpretation. Compared to our first encoding, it is conceptually more involved and has (slightly) higher initialization cost, but it has the advantage that it can be reused for all unfounded set checks and needs no separate initialization for each check. Our benchmarks show that the advanced encoding is superior to the first one for many practical problems.

- Next, we consider optimizations which hinge on dependencies between external and ordinary atoms that are determined in careful analysis. These optimizations can be integrated into our encodings by adding further nogoods which restrict the search space to relevant parts.

- We consider how to exploit information gained in the unfounded set check of a candidate answer set in answer set candidate generation, i.e., in the evaluation of the program $\hat{\Pi}$. Adopting a Conflict Driven Clause Learning approach (Drescher et al., 2008), this step has been recently enhanced by *external behavior learning* (Eiter et al., 2012a), in which nogoods describing the external source behavior are learned during the search to guide model generation towards proper guesses. We show how to learn in the candidate generation step additional nogoods from unfounded sets that avoid the reconstruction of the same or related unfounded sets, yielding further gain.

- We present a *syntactic decision criterion* that can be used to decide whether a program possibly has unfounded sets. If the result of this check is negative, then the computationally expensive search for unfounded sets can be skipped entirely. The criterion is based on atom dependency and, loosely speaking, says that there are no cyclic dependencies of ground atoms through external atoms. This property can be efficiently checked for a given ground HEX-program using standard methods. In fact it applies to a range of applications, in particular for input-stratified programs, where external sources are accessed in a workflow to produce input for the next stage of computation. However, advanced applications of HEX-programs can have cycles through external atoms, e.g., in natural encodings of problems on multi-context systems (Brewka & Eiter, 2007) or abstract argumentation systems (Dung, 1995), for which the FLP check can not be simply skipped.

- In further elaboration, we then consider a *program decomposition* based on the dependency graph that is induced by a program Π (note that exploiting syntactic modularization of unfounded sets can be traced back to Leone et al., 1997; Koch et al., 2003). We show that Π has some unfounded set wrt. a candidate answer set $\mathbf{A}$ exactly if some of the components Π_C in its decomposition has some unfounded set wrt. $\mathbf{A}$; as computing the decomposition can be realized efficiently and does not incur large overhead, we can apply the decision criterion for skipping the FLP check efficiently on a finer-grained level, and the search for unfounded sets can be guided to relevant program parts.

- An experimental evaluation on advanced reasoning applications shows that unfounded sets checking combined with learning methods of Eiter et al. (2012a) improves HEX-program evaluation considerably, sometimes drastically. More specifically, the benchmark applications include reasoning tasks in Multi-Context Systems (Brewka & Eiter, 2007; Eiter et al., 2012b), abstract argumentation (Dung, 1995), terminological default reasoning over description logic knowledge bases (Baader & Hollunder, 1995), and conformant planning (Goldman & Boddy, 1996; Smith & Weld, 1998). The experiments have been carried out with DLVHEX version 2.3.0, a prototype solver for HEX-programs, which was extended to support the techniques developed in this paper. The decomposition approach can yield a considerable gain, as it appears e.g. for the HEX-encoding of a Dung-style argumentation semantics (Dung, 1995) and DL-programs (Eiter et al., 2008a). On the other hand, for the terminological default reasoning benchmark, our syntactic criterion lets us conclude that the FLP check is obsolete. In conclusion, the new approach enables significant speedup and thus enlarges the scope of HEX applicability.

1.1 Organization

The rest of this paper is organized as follows. The next section provides preliminaries on HEX-programs and their evaluation via answer sets of a transformed ASP program without external atoms. In Section 3, we define unfounded sets and present a basic and a uniform encoding of unfounded set search using nogoods. Section 4 considers refinements and optimizations of the encodings, as well as external behavior learning to prevent reconstruction of unfounded sets. In Section 5, we give a syntactic decision criterion to avoid the FLP check and a program decomposition method exploiting it. Experimental results of a prototype implementation are reported in Section 6. In Section 7, we consider related work and extensions of our approach. In Section 8, we conclude and point out issues for further research.

2. Preliminaries

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

In accordance with Gebser et al. (2012) and Eiter et al. (2012a), 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 $p(c_1, \dots, c_\ell)$, with predicate p and constants $c_1, \dots, c_\ell$, 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* $\mathbf{A}$ over a (finite) set of atoms A is a consistent set of signed literals $\mathbf{T}a$ or $\mathbf{F}a$, where $\mathbf{T}a$ expresses that $a \in A$ is true and $\mathbf{F}a$ that it is false; $\mathbf{A}$ 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 in $\mathbf{A}$. Furthermore, $\mathbf{A}|_q$ is the set of all literals over atoms with predicate q in $\mathbf{A}$. For a list $\mathbf{q} = q_1, \dots, q_k$ of predicates, we write $p \in \mathbf{q}$ iff $q_i = p$ for some $1 \leq i \leq k$, and let $\mathbf{A}|_{\mathbf{q}} = \bigcup_j \mathbf{A}|_{q_j}$.

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

Example 1 The interpretation $\mathbf{A} = \{\mathbf{T}a, \mathbf{F}b, \mathbf{T}c\}$ is a solution to the nogood $\{\mathbf{T}a, \mathbf{T}b, \mathbf{T}c\}$ but not to $\{\mathbf{T}a, \mathbf{F}b, \mathbf{T}c\}$.

2.1 HEX-Programs

Next, we recall syntax and semantics of HEX-programs.

2.1.1 HEX-PROGRAM SYNTAX

As introduced by Eiter et al. (2005), HEX-programs are a generalization of (disjunctive) extended logic programs under the answer set semantics (Gelfond & Lifschitz, 1991). HEX-programs extend 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 matched with the output list. Syntactically, a *ground external atom* is of the form

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

where $\&g$ is an *external predicate*, $\mathbf{p} = p_1, \dots, p_k$ are the *input list* consisting of predicate names or object constants, and $\mathbf{c} = c_1, \dots, c_l$ are the *output list* consisting of constant terms. Predicates in the input list are sometimes called *input predicates*.

A *default literal* is a formula b or $\text{not } b$, where b is a *ground ordinary atom* of form $p(c_1, \dots, c_\ell)$ with constants c_i, $1 \leq i \leq \ell$, or and external atom. For every set S of ordinary and external atoms, we let $\text{not } S = \{\text{not } b \mid b \in S\}$.

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

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

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

where each a_i is an ordinary ground atom, each b_j is either an ordinary ground atom or a ground external atom, and $k + n > 0$.[1]

For a rule r, the *head* is $H(r) = \{a_1, \dots, a_k\}$ and the *body* is $B(r) = B^+(r) \cup \text{not } B^-(r)$, where $B^+(r) = \{b_1, \dots, b_m\}$ is the *positive body*, $B^-(r) = \{b_{m+1}, \dots, b_n\}$ is the *negative body*. If $B(r) = \emptyset$, then r is a *fact*, and we omit $\leftarrow$. For a program Π, let $A(\Pi)$ be the set of all ordinary atoms and $EA(\Pi)$ be the set of all external 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 $b = \text{not } a$. Furthermore, $\mathbf{f}b = \mathbf{F}a$ if $b = a$ and $\mathbf{f}b = \mathbf{T}a$ if $b = \text{not } a$.

We call a rule r a *constraint*, if $B(r) = \emptyset$.

Example 2 For rule $r = a \vee b \leftarrow c, \text{not } d$ we have $H(r) = \{a, b\}$, $B^+(r) = \{c\}$ and $B^-(r) = \{d\}$. We further have $\mathbf{t}c = \mathbf{T}c$, $\mathbf{f}c = \mathbf{F}c$, $\mathbf{t} \text{ not } d = \mathbf{F}d$ and $\mathbf{f} \text{ not } d = \mathbf{T}d$.

We will also consider non-ground programs (i.e., with variables allowed in place of object constants) in our examples. In particular, external atoms $\&g[\mathbf{X}](\mathbf{Y})$ may contain variables in their input list $\mathbf{X}$ and output list $\mathbf{Y}$. For such programs, suitable safety conditions allow for using a *grounding procedure* which transforms the program to a variable-free program with the same answer sets (Eiter, Ianni, Schindlauer, & Tompits, 2006). However, we limit our formal investigation to ground programs.

1. For simplicity, we do not formally introduce strong negation but view, as customary, classical literals $\neg a$ as new atoms together with a constraint which disallows that a and $\neg a$ are simultaneously true.

Example 3 The program

$$\begin{aligned} domain(a); \quad domain(b) \\ sel(X) &\leftarrow domain(X), \text{not } nsel(X) \\ nsel(X) &\leftarrow domain(X), \text{not } sel(X) \end{aligned}$$

encodes the problem of partitioning two domain elements a and b into two sets sel and $nsel$.

2.1.2 HEX-PROGRAM SEMANTICS

An ordinary ground atom a is *true* relative to assignment $\mathbf{A}$, denoted $\mathbf{A} \models a$, if $\mathbf{T}a \in \mathbf{A}$ and *false* otherwise. A default-negated ground atom $\text{not}\, a$ is *true* relative to assignment $\mathbf{A}$, denoted $\mathbf{A} \models \text{not}\, a$, if $\mathbf{F}a \in \mathbf{A}$ and *false* otherwise.

The semantics of a ground external atom of form (1) wrt. an interpretation $\mathbf{A}$ is given by the value of a $1{+}k{+}l$-ary Boolean *oracle function* $f_{\&g}$ that is defined for all possible values of $\mathbf{A}$, $\mathbf{p}$ and $\mathbf{c}$, such that $\&g[\mathbf{p}](\mathbf{c})$ is true relative to $\mathbf{A}$, denoted $\mathbf{A} \models \&g[\mathbf{p}](\mathbf{c})$, if and only if $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = 1$.

Example 4 (Set Partitioning) Consider the program Π

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

where for predicates p and q, $\&\mathit{diff}[p, q](X)$ 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 $nsel(a)$.

Satisfaction of ordinary rules and ASP programs (Gelfond & Lifschitz, 1991) is then extended to HEX-rules and programs in the obvious way: a rule r is satisfied by assignment $\mathbf{A}$, denoted $\mathbf{A} \models r$, iff $\mathbf{A} \models h$ for some $h \in H(r)$, or $\mathbf{A} \not\models b$ for some $b \in B^+(r)$, or $\mathbf{A} \models b$ for some $b \in B^-(r)$. A program Π is satisfied by assignment $\mathbf{A}$ iff $\mathbf{A} \models r$ for all $r \in \Pi$. An interpretation $\mathbf{A}$ is a *model* of a program Π, denoted $\mathbf{A} \models \Pi$, iff $\mathbf{A} \models r$ for all $r \in \Pi$.

The notion of extension $ext(\cdot, \mathbf{A})$ for external predicates $\&g$ with input lists $\mathbf{p}$ is naturally defined by $ext(\&g[\mathbf{p}], \mathbf{A}) = \{\mathbf{c} \mid f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = 1\}$.

Definition 2 (FLP-Reduct (Faber et al., 2011)) *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 by* $\mathbf{A}$.

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* $f\Pi^{\mathbf{A}}$.

Example 5 Consider the program Π:

$$\begin{aligned} p &\leftarrow \&id[q]() \\ q &\leftarrow p \end{aligned}$$

where $f_{\&id}(\mathbf{A}, p) = 1$ iff $\mathbf{T}p \in \mathbf{A}$ is true. Then Π has the answer set $\mathbf{A}_1 = \emptyset$; indeed it is a $\leq_\Pi$-minimal model of $f\Pi^{\mathbf{A}_1} = \emptyset$. Note that $\mathbf{A}_2 = \{\mathbf{T}p, \mathbf{T}q\}$ is not an answer set of Π, as it is not a minimal model of $f\Pi^{\mathbf{A}_2} = \Pi$.

2.2 Evaluation of HEX-Programs

A possible way to determine the answer sets of a HEX-program Π is to use a transformation to an ASP program without external atoms whose answer sets encompass all answer sets of Π. We now describe such a transformation. Each external atom $a = \&g[\mathbf{p}](\mathbf{c})$ in a rule $r \in \Pi$ is replaced by an ordinary ground *replacement atom* $\hat{a} = e_{\&g[\mathbf{p}]}(\mathbf{c})$ (resulting in a rule $\hat{r}$), and an additional 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 ASP solver and projected to non-replacement atoms. However, the resulting assignments might be spurious answer sets of Π, as the values of $\&g[\mathbf{p}]$ and $e_{\&g[\mathbf{p}]}(\mathbf{c})$ relative to an interpretation may not coincide. 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* $\hat{\mathbf{A}}$ *such that*

(i) $\hat{\mathbf{A}}$ *is an answer set (Gelfond & Lifschitz, 1991) 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 values of the oracle functions.*

A subset of the compatible sets of Π represents the answer sets of Π, where each answer set $\mathbf{A}$ of Π is given by the restriction of a unique compatible set $\hat{\mathbf{A}}$ to the non-replacement atoms. More formally, an answer set $\mathbf{A}$ of a program Π corresponds to the compatible set

$$\begin{aligned}\kappa(\Pi, \mathbf{A}) = \mathbf{A} &\cup \{\mathbf{T}e_a, \mathbf{F}ne_a \mid a \text{ is an external atom in } \Pi, \mathbf{A} \models e\} \\ &\cup \{\mathbf{F}e_a, \mathbf{T}ne_a \mid a \text{ is an external atom in } \Pi, \mathbf{A} \not\models e\} \ .\end{aligned}$$

To filter out those compatible sets that do not yield answer sets, each compatible set $\hat{\mathbf{A}}$ has to be checked against models of the FLP reduct. To be more specific, a procedure called *explicit FLP check* constructs the reduct $f\Pi^{\mathbf{A}}$ and checks whether it has a model $\mathbf{A}'$ smaller than $\mathbf{A}$; if such an $\mathbf{A}'$ is found, it rejects $\mathbf{A}$, otherwise outputs $\mathbf{A}$ as an answer set.

The explicit FLP check rewrites the HEX-program to an ASP program without external atoms and amounts to the search for answer sets of the following program, in which the truth values of all replacement atoms coincide with the according oracle function values:

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

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.

Proposition 1 *Let* $\mathbf{A}$ *be an interpretation extracted from a compatible set* $\hat{\mathbf{A}}$ *of program* Π*. Then the program* $\mathit{Check}(\Pi, \mathbf{A})$ *has an answer set* $\mathbf{A}'$ *such that* $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = 1$ *iff* $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \mathbf{A}'$ *for all external atoms* $\&g[\mathbf{p}](\mathbf{c})$ *in* Π*, if and only if* $\mathbf{A}$ *is not an answer set of* Π.

Because of the guessing rules, we can rewrite all rules in $f\hat{\Pi}^{\hat{\mathbf{A}}}$ except the guesses on replacement atoms to *constraints* as follows:

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

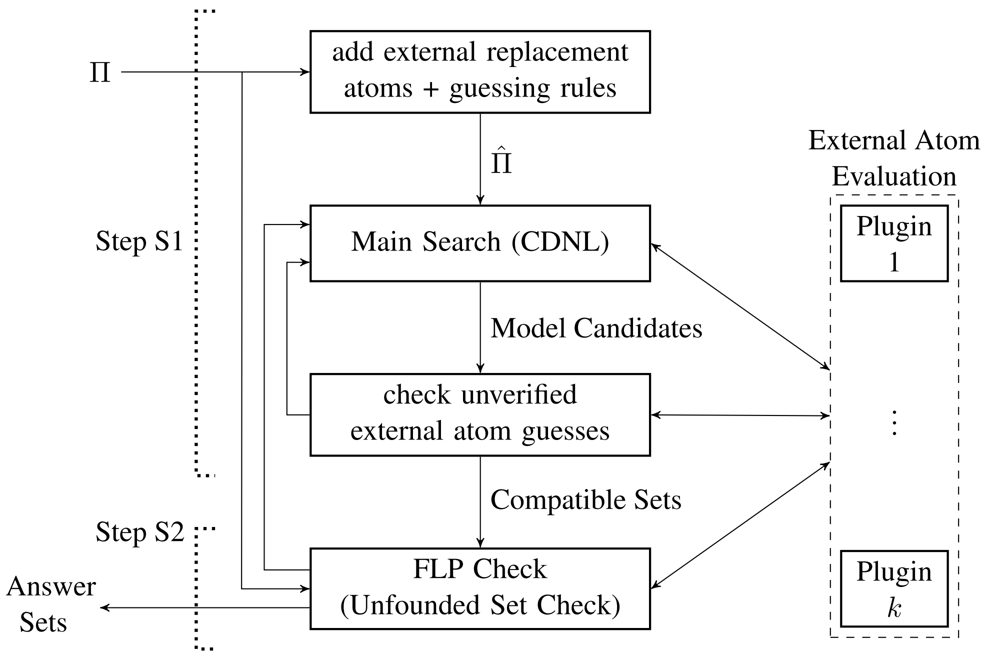

Figure 1: Overview of the framework for evaluating HEX-programs.

where $\bar{f}\hat{\Pi}^{\hat{\mathbf{A}}}$ denotes the FLP reduct of $\hat{\Pi}$ wrt. interpretation $\hat{\mathbf{A}}$ with each rule of form (2) except guessing rules for replacement atoms being rewritten to

$$\leftarrow \text{not}\, a_1, \ldots, \text{not}\, a_k, b_1, \ldots, b_m, \text{not}\ b_{m+1}, \ldots, \text{not}\ b_n \ .$$

Proposition 2 *Let* $\mathbf{A}$ *be an interpretation extracted from a compatible set* $\hat{\mathbf{A}}$ *of program* Π*. Then the program* $\mathit{CheckOptimized}(\Pi, \mathbf{A})$ *has an answer set* $\mathbf{A}'$ *such that* $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = 1$ *if and only if* $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \mathbf{A}'$ *for all external atoms* $\&g[\mathbf{p}](\mathbf{c})$ *in* Π*, if and only if* $\mathbf{A}$ *is not an answer set of* Π*.*

This program is more efficient for evaluation. Our comparison in Section 6 uses this optimized version of the explicit check, but still demonstrates a significant performance gain by our novel approach.

Example 6 (cont'd) Reconsider the program $\Pi = \{\, p \leftarrow \&id[q](); q \leftarrow p \,\}$ from above. Then the corresponding guessing program is $\hat{\Pi} = \{p \leftarrow e_{\&id[q]}(); q \leftarrow p; e_{\&id[q]}() \vee ne_{\&id[q]}() \leftarrow\}$ and yields the compatible sets $\hat{\mathbf{A}}_1 = \emptyset$ and $\hat{\mathbf{A}}_2 = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}e_{\&id[p]}\}$. While $\mathbf{A}_1 = \emptyset$ is also a $\leq_\Pi$-minimal model of $f\Pi^{\mathbf{A}_1} = \emptyset$, $\mathbf{A}_2 = \{\mathbf{T}p, \mathbf{T}q\}$ is not a $\leq_\Pi$-minimal model of $f\Pi^{\mathbf{A}_2} = \Pi$. Indeed, the program

$$\begin{aligned}\mathit{Check}(\Pi, \mathbf{A}_2) = \hat{\Pi} \cup \{p \vee p' \leftarrow; q \vee q' \leftarrow; e_{\&id[q]}() \vee e'_{\&id[q]}() \leftarrow\} \cup \{\leftarrow \text{not}\ \mathit{smaller}\} \\ \cup \{\mathit{smaller} \leftarrow \text{not}\, p\} \cup \{\mathit{smaller} \leftarrow \text{not}\, q; \mathit{smaller} \leftarrow \text{not}\, e_{\&id[q]}()\}\end{aligned}$$

has the answer set $\mathbf{A}' = \left\{\mathbf{F}p, \mathbf{T}p', \mathbf{F}q, \mathbf{T}q', \mathbf{F}e_{\&id[q]}(), \mathbf{T}ne_{\&id[q]}(), \mathbf{T}e'_{\&id[q]}(), \mathbf{T}\mathit{smaller}\right\}$ and $f_{\&id}(\mathbf{A}', q, \epsilon) = 0$ and $\mathbf{F}e_{\&id[q]}() \in \mathbf{A}'$.

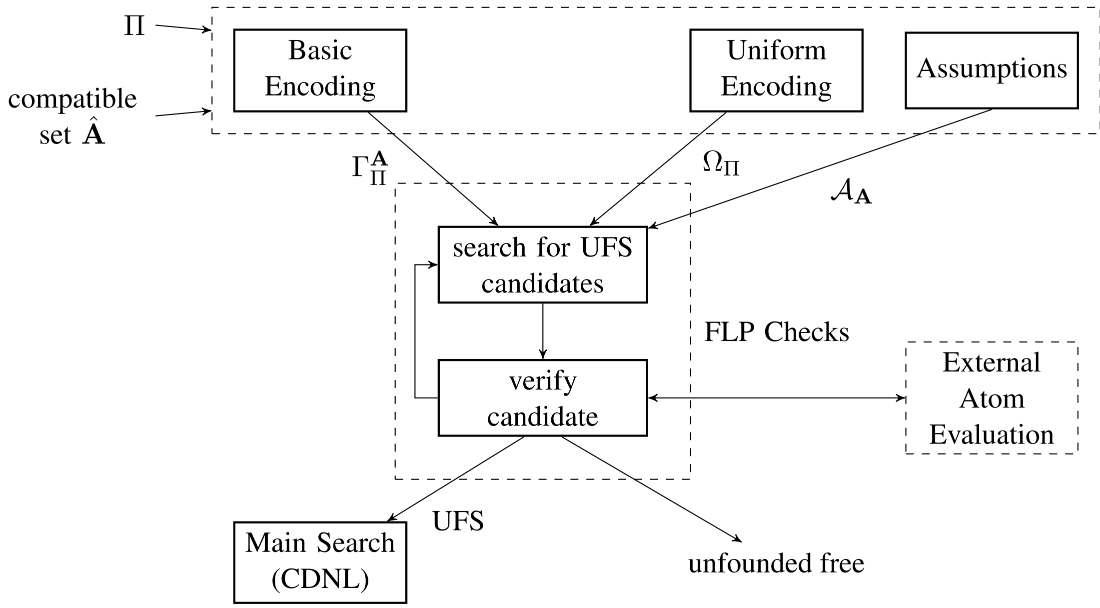

Figure 2: FLP Check based on Unfounded Sets

The complete prodedure of computing answer sets of HEX-programs has been described by Eiter et al. (2012a) and is shown as a block diagram in Figure 1; the first version introducing this approach was DLVHEX version 2.1.0. Step S1 enumerates compatible sets; we reuse this part of the evaluation process from prior work and do not modify it. Step S2 checks whether a compatible set is indeed a HEX answer set. The improvement of the efficiency of S2 is the focus of this work. S1 transforms the input program Π into $\hat{\Pi}$ by introducing replacement atoms and guesses for replacement atoms. The main search enumerates model candidates, i.e., answer sets of $\hat{\Pi}$. (Depending on heuristics, this search can evaluate external atoms for guiding the search.) Model candidates are verified against the semantics of external atoms. If this check fails, the main search continues to enumerate model candidates; if the check succeeds, then the model candidate is a compatible set. S2 checks whether a compatible set is an answer set of Π. Previous work realized this step using an explicit FLP check. In this work we propose two alternatives to carry out this FLP check based on unfounded sets.

Figure 2 depicts a block diagram of the FLP checks proposed in this work, called basic and uniform encoding. Both encodings are SAT theories. The basic encoding builds on the program Π and the compatible set $\hat{\mathbf{A}}$. The uniform encoding is based only on Π (hence we can reuse it for all compatible sets), however it requires us to set *solver assumptions* based on $\hat{\mathbf{A}}$. The encoding (and assumptions) are used to search for unfounded set (UFS) candidates (by a SAT solver). A UFS candidate has to be verified against the values of external atoms (these are guessed in the UFS encoding). If all UFS candidates fail the external atom check, or if there are no UFS candidates, then $\hat{\mathbf{A}}$ is an unfounded-free compatible set and hence an answer set of Π. Otherwise the FLP check found an unfounded set wrt. $\hat{\mathbf{A}}$ and the main search continues looking for a new model candidate.

We next describe our encodings for UFS-checking.

3. Unfounded Set Detection

As described in Section 2.2, the minimality check, also called the *explicit FLP check*, is computationally costly and involves much overhead: all models of $\mathit{Check}(\Pi, \mathbf{A})$ must be enumerated, and calls to the external sources to test compatibility must be made. Even worse, as we need to search for a smaller *model* and not just for a smaller compatible set, $\mathit{Check}(\Pi, \mathbf{A})$ usually (to our experience) has even more models than the original program. Moreover, it appears that in many current application scenarios there is no smaller model of the reduct $f\Pi^{\mathbf{A}}$, i.e., most assignments extracted from compatible sets $\hat{\mathbf{A}}$ pass the FLP check. There are two possible reliefs: developing a cheaper minimality check or to avoid the minimality check if possible. While this section targets at the former idea, the latter one is addressed in Section 5.

To this end, we present a novel FLP check algorithm based on unfounded sets (UFS). Instead of explicitly searching for smaller models of the reduct, we check whether the candidate answer set is unfounded-free. To this end, we use unfounded sets for HEX-programs akin to those by Faber (2005) 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 $\mathbf{A} \dot{\cup} \neg.X = (\mathbf{A} \setminus \{\mathbf{T}a \mid a \in X\}) \cup \{\mathbf{F}a \mid a \in X\}$:*

(i) some literal of $B(r)$ is false wrt. $\mathbf{A}$,

(ii) some literal of $B(r)$ is false wrt. $\mathbf{A} \dot{\cup} \neg.X$, 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 circularly support each other; by assigning all of them false, no violation of any rule will be introduced. As for answer sets, their minimality enforces now that no subset of the atoms that are true in an answer set can form an unfounded set; in fact, answer sets can be characterized in terms of unfounded sets, using the following notion.

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 every unfounded set X of Π wrt. $\mathbf{A}$.*

The following result is a generalization of a respective result for ordinary (disjunctive) logic programs (Leone et al., 1997) and logic programs with aggregates (Faber, 2005).

Theorem 3 (Characterization of Answer Sets) *A model $\mathbf{A}$ of a HEX-program Π is an answer set of Π iff it is* unfounded-free.

Example 7 Consider the program Π and $\mathbf{A}_1$ from Example 6. Trivially, $\mathbf{A}_1$ is unfounded-free, and thus $\mathbf{A}_1$ is an answer set of Π. On the other hand, the set $X = \{p, q\}$ is an unfounded set w.r.t. $\mathbf{A}_2$, since X intersects with the head of $p \leftarrow \&id[q]()$ and $\mathbf{A} \dot{\cup} \neg.X \not\models \&id[q]()$. Therefore $\mathbf{A}_2$ is not unfounded-free and not an answer set of Π.

3.1 Basic Encoding of the Unfounded Set Search

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.

More specifically, our encoding of unfounded set detection uses a set

$$\Gamma_{\Pi}^{\mathbf{A}} = N_{\Gamma,\Pi}^{\mathbf{A}} \cup O_{\Gamma,\Pi}^{\mathbf{A}} \ ,$$

of nogoods where $N_{\Gamma,\Pi}^{\mathbf{A}}$ contains all *necessary* constraints and the set $O_{\Gamma,\Pi}^{\mathbf{A}}$ are *optional* optimization nogoods that prune irrelevant parts of the search space; the latter is related to the one of Drescher et al. (2008) but respects external atoms. 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 replacement atoms encode the truth values of the corresponding external atoms under $\mathbf{A} \dot{\cup} \neg.U$.

For a rule r, let $B_o^+(r) \subseteq B^+(r)$ consist of all *o*rdinary atoms, and let $B_e(r) \subseteq B(r)$ consist of all *e*xternal replacement atoms. Then, $\Gamma_{\Pi}^{\mathbf{A}}$ 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 new atoms for every rule r in Π. The *necessary part*

$$N_{\Gamma,\Pi}^{\mathbf{A}} = \{\{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{A}\}\} \cup \left(\bigcup_{r\in\Pi} R_{r,\mathbf{A}}\right)$$

of $\Gamma_{\Pi}^{\mathbf{A}}$ consists of a nogood $\{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{A}\}$, which eliminates unfounded sets that do not intersect with true atoms in $\mathbf{A}$, and of nogoods $R_{r,\mathbf{A}} = H_{r,\mathbf{A}} \cup C_{r,\mathbf{A}}$ for every $r \in \Pi$ where

- $H_{r,\mathbf{A}} = \{\{\mathbf{T}h_r\} \cup \{\mathbf{F}h \mid h \in H(r)\}\} \cup \{\{\mathbf{F}h_r, \mathbf{T}h\} \mid h \in H(r)\}$, called the *head criterion*, encodes that h_r is true for a rule r iff some atom of $H(r)$ is in the unfounded set; and
- $$C_{r,\mathbf{A}} = \begin{cases} \{\{\mathbf{T}h_r\} \cup \\ \quad \{\mathbf{F}a \mid a \in B_o^+(r), \mathbf{A} \models a\} \cup \{\mathbf{t}a \mid a \in B_e(\hat{r})\} \cup \\ \quad \{\mathbf{T}h \mid h \in H(r), \mathbf{A} \models h\}\} & \textit{if } \mathbf{A} \models B(r), \\ \{\} & \textit{otherwise}, \end{cases}$$
  called the *conditional part* $C_{r,\mathbf{A}}$, 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). Concrete instances for $O_{\Pi}^{\Gamma,\mathbf{A}}$ are defined in Section 4.

Example 8 Consider $\Pi = \{r_1 : p \leftarrow \&id[p]()\}$ and the compatible set $\hat{\mathbf{A}} = \{\mathbf{T}p, \mathbf{T}e_{\&id[p]}\}$. The nogood set $N_{\Gamma,\Pi}^{\mathbf{A}_2}$ is $\{\{\mathbf{T}h_{r_1}, \mathbf{F}p\}, \{\mathbf{F}h_{r_1}, \mathbf{T}p\}, \{\mathbf{T}h_{r_1}, \mathbf{T}e_{\&id[p]}(), \mathbf{T}p\}\}$.

Towards computing unfounded sets, observe that every unfounded set can be extended to a solution to the nogood set $\Gamma_{\Pi}^{\mathbf{A}}$ over $A(\Gamma_{\Pi}^{\mathbf{A}})$. Conversely, the solutions to $\Gamma_{\Pi}^{\mathbf{A}}$ include specific extensions of the unfounded sets, given for each unfounded set U by assigning true to all atoms in U, to all h_r such that $H(r)$ intersects with U, and to all 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,

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

$$I_\Gamma(U, \Gamma_{\Pi}^{\mathbf{A}}, \Pi, \mathbf{A}) = I_\Gamma^0(U, \Pi, \mathbf{A}) \cup \{\mathbf{F}a \mid a \in A(\Gamma_{\Pi}^{\mathbf{A}}), \mathbf{T}a \notin I_\Gamma^0(U, \Pi, \mathbf{A})\} \ ,$$

where

$$I^0_\Gamma(U,\Pi,\mathbf{A}) = \{\mathbf{T}a \mid a \in U\} \cup \{\mathbf{T}h_r \mid r \in \Pi, H(r) \cap U \neq \emptyset\} \cup \{\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \mid \&g[\mathbf{p}](\mathbf{c}) \in EA(\Pi),\ \mathbf{A} \dot{\cup} \neg.U \models \&g[\mathbf{p}](\mathbf{c})\}\ .$$

We call a set N of nogoods *conservative*, if it holds for every unfounded set U of Π wrt. $\mathbf{A}$ that $I_\Gamma(U, \Gamma^{\mathbf{A}}_\Pi, \Pi, \mathbf{A})$ is a solution to N. We then show that the solutions to $\Gamma^{\mathbf{A}}_\Pi$ include all assignments induced by unfounded sets of Π wrt. $\mathbf{A}$, assuming that $O^{\Gamma,\mathbf{A}}_\Pi$ is conservative.

Proposition 4 *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 $I_\Gamma(U, \Gamma^{\mathbf{A}}_\Pi, \Pi, \mathbf{A})$ is a solution to $\Gamma^{\mathbf{A}}_\Pi$.*

Note that the converse does not hold, i.e., not every solution corresponds to some induced assignment; intuitively this is because it does not reflect the semantics of external sources. Regardless of this we immediately obtain from Proposition 4 a useful test for unfounded-freeness.

Corollary 5 *If $\Gamma^{\mathbf{A}}_\Pi$ has no solution, then $U \cap \mathbf{A}^{\mathbf{T}} = \emptyset$ for every unfounded set U of Π.*

Using the following result, we can find the unfounded sets of Π wrt. $\mathbf{A}$ among all solutions to $\Gamma^{\mathbf{A}}_\Pi$ by using a postcheck on the external atoms.

Theorem 6 *Let S be a solution to $\Gamma^{\mathbf{A}}_\Pi$ 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 the non-replacement atoms in S that are true and also appear in Π form an unfounded set, provided that truth of the 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 values 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.

It follows immediately from Definition 7 that this postcheck does not eliminate unfounded sets, as formalized by the following proposition.

Proposition 7 *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 $I_\Gamma(U, \Gamma^{\mathbf{A}}_\Pi, \Pi, \mathbf{A})$ fulfills Conditions (a) and (b) of Theorem 6.*

Example 9 Reconsider program $\Pi = \{r_1 : p \leftarrow \&id[p]()\}$ from Example 8 and the compatible set $\hat{\mathbf{A}}_2 = \{\mathbf{T}p, \mathbf{T}e_{\&id[p]}\}$. The nogood set $N^{\mathbf{A}_2}_{\Gamma,\Pi} = \{\{\mathbf{T}h_{r_1}, \mathbf{F}p\}, \{\mathbf{F}h_{r_1}, \mathbf{T}p\}, \{\mathbf{T}h_{r_1}, \mathbf{T}e_{\&id[p]}(), \mathbf{T}p\}\}$ 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 Theorem 6, 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 $\Gamma^{\mathbf{A}}_\Pi$.

3.2 Uniform Encoding of the Unfounded Set Search

The encoding $\Gamma_{\Pi}^{\mathbf{A}}$ presented in the previous subsection has the disadvantage that it depends on the current assignment $\mathbf{A}$. Therefore it needs to be generated separately for every unfounded set check if the assignment changed (which is very likely). As this causes significant overhead, we present now an advanced encoding which can be reused for any assignment. We introduce some additional variables which represent the truth values of the atoms in the current assignment. Prior to an unfounded set check, the current assignment is injected by setting the values of these variables to fixed values, which can be done using *assumptions* as supported by modern SAT solvers such as CLASP. Changing assumptions is much easier than changing the encoding, which leads to an additional speedup in some cases, especially for programs which need many unfounded set checks.

Our advanced encoding uses a set Ω_{Π} of nogoods. As before, the idea is that the set of ordinary atoms of a solution to Ω_{Π} represents a (potential) unfounded set U of Π wrt. some assignment $\mathbf{A}$, while the replacement atoms encode the truth values of the corresponding external atoms under $\mathbf{A} \dot{\cup} \neg.U$. The encoding Ω_{Π} is conceptually more complex than $\Gamma_{\Pi}^{\mathbf{A}}$; the initialization is computationally (slightly) more costly, hence the advantages of our new encoding become visible for instances with many compatible sets (thus many unfounded set checks), while it might be counterproductive for small instances.

The nogood set Ω_{Π} is built over the atoms of $\hat{\Pi}$ and and further fresh atoms not occurring in $\hat{\Pi}$: h_r and l_r, for every rule r in Π, $a_{\mathbf{A}}$ for every ordinary atom $a \in A(\hat{\Pi})$ (i.e. ordinary atoms in Π and replacement atom auxiliaries), and $a_{\mathbf{A}\dot{\cup}\neg.U}$, $a_{\mathbf{A}\wedge U}$, $a_{\overline{\mathbf{A}}\vee U}$ for every ordinary atom $a \in A(\Pi)$. The *auxiliary atoms* $a_{\mathbf{A}}$, $a_{\mathbf{A}\dot{\cup}\neg.U}$, $a_{\mathbf{A}\wedge U}$, $a_{\overline{\mathbf{A}}\vee U}$ are used to make the encoding usable for any assignment $\mathbf{A}$. Only during the unfounded set check with respect to a certain assignment, we will temporarily add assumptions to the solver which force certain truth values of the atoms $a_{\mathbf{A}}$ for all $a \in A(\hat{\Pi})$ depending on the current assignment $\mathbf{A}$. Intuitively, $a_{\mathbf{A}}$ represents the truth value of a in $\mathbf{A}$ and $a_{\mathbf{A}\dot{\cup}\neg.U}$ of a in $\mathbf{A} \dot{\cup} \neg.U$ (where U is the current unfounded set), $a_{\mathbf{A}\wedge U}$ represents that a is true in $\mathbf{A}$ and is contained in U, and $a_{\overline{\mathbf{A}}\vee U}$ represents that a is false in $\mathbf{A}$ or it is contained in U.

To this end, a *set of assumptions* is a consistent set $\mathcal{A}$ of signed literals. A solution $\mathbf{A}$ to a nogood δ resp. a set Δ of nogoods *satisfies* $\mathcal{A}$, if $\mathcal{A} \subseteq \mathbf{A}$. That is, assumptions fix the truth value of some atoms. Modern ASP and SAT solvers support assumptions natively, and they can be easily undone without a complete reset of the reasoner and recreating the whole problem instance. This is an essential feature for efficiently implementing our improved encoding.

Our encoding Ω_{Π} is then

$$\Omega_{\Pi} = N_{\Omega,\Pi} \cup O_{\Omega,\Pi} \quad ,$$

where $N_{\Omega,\Pi} = \{\{\mathbf{F}a \mid a \in A(\Pi)\}\} \cup \bigcup_{a \in A(\Pi)} D_a \cup \bigcup_{r \in \Pi}(H_r \cup C_r)$ is the necessary part and

- $\{\mathbf{F}a \mid a \in A(\Pi)\}$ encodes that we search for a nonempty unfounded set;
- $$D_a = \begin{cases} \{\{\mathbf{F}a_{\mathbf{A}\wedge U}, \mathbf{T}a_{\mathbf{A}}, \mathbf{T}a\}, \{\mathbf{T}a_{\mathbf{A}\wedge U}, \mathbf{F}a_{\mathbf{A}}\}, \{\mathbf{T}a_{\mathbf{A}\wedge U}, \mathbf{F}a\}\} \cup \\ \{\{\mathbf{F}a_{\overline{\mathbf{A}}\vee U}, \mathbf{F}a_{\mathbf{A}}\}, \{\mathbf{F}a_{\overline{\mathbf{A}}\vee U}, \mathbf{T}a\}, \{\mathbf{T}a_{\overline{\mathbf{A}}\vee U}, \mathbf{T}a_{\mathbf{A}}, \mathbf{F}a\}\} \cup \\ \{\{\mathbf{T}a_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{F}a_{\mathbf{A}}\}, \{\mathbf{T}a_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{T}a\}, \{\mathbf{F}a_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{T}a_{\mathbf{A}}, \mathbf{F}a\}\} \end{cases}$$
  encodes that $a_{\mathbf{A}\wedge U}$ is true iff $a_{\mathbf{A}}$ and a are both true, $a_{\overline{\mathbf{A}}\vee U}$ is true iff $a_{\mathbf{A}}$ is false or a is true, and $a_{\mathbf{A}\dot{\cup}\neg.U}$ is true iff $a_{\mathbf{A}}$ is true and a is false;
- $H_r = \{\{\mathbf{T}h_r\} \cup \{\mathbf{F}h \mid h \in H(r)\}\} \cup \{\{\mathbf{F}h_r, \mathbf{T}h\} \mid h \in H(r)\}$
  encodes that h_r is true for a rule r iff some atom of $H(r)$ is in the unfounded set; and

- $$C_r = \begin{cases} \{\{\mathbf{T}h_r\} \cup & \\ \quad \{\mathbf{T}a_{\mathbf{A}} \mid a \in B^+(\hat{r})\} \cup \{\mathbf{F}a_{\mathbf{A}} \mid a \in B^-(\hat{r})\} \cup & (i) \\ \quad \{\mathbf{F}a_{\mathbf{A}\wedge U} \mid a \in B_o^+(r)\} \cup \{\mathbf{t}a \mid a \in B_e(\hat{r})\} \cup & (ii) \\ \quad \{\mathbf{T}h_{\overline{\mathbf{A}}\vee U} \mid h \in H(r)\}\} & (iii) \end{cases}$$

  encodes that if h_r is true, then one of (i), (ii) or (iii) in Definition 5 must hold.

  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).

Example 10 For $\Pi = \{r_1 : p \leftarrow \&id[p]()\}$ in Example 6, the constructed nogood set is

$$\begin{aligned} \Omega_\Pi \quad = \quad & \{\{\mathbf{F}p\}, \{\mathbf{F}p_{\mathbf{A}\wedge U}, \mathbf{T}p_{\mathbf{A}}, \mathbf{T}p\}, \{\mathbf{T}p_{\mathbf{A}\wedge U}, \mathbf{F}p_{\mathbf{A}}\}, \{\mathbf{T}p_{\mathbf{A}\wedge U}, \mathbf{F}p\}, \\ & \{\mathbf{F}p_{\bar{\mathbf{A}}\vee U}, \mathbf{F}p\}, \{\mathbf{F}p_{\bar{\mathbf{A}}\vee U}, \mathbf{T}p\}, \{\mathbf{T}p_{\bar{\mathbf{A}}\vee U}, \mathbf{T}p_{\mathbf{A}}, \mathbf{F}p\}, \{\mathbf{T}p_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{F}p_{\mathbf{A}}\}, \\ & \{\mathbf{T}p_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{T}p\}, \{\mathbf{F}p_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{T}p_{\mathbf{A}}, \mathbf{F}p\}, \\ & \{\mathbf{T}h_{r_1}, \mathbf{F}p\}, \{\mathbf{F}h_{r_1}, \mathbf{T}p\}, \{\mathbf{T}h_{r_1}, \mathbf{T}e_{\&id[p]}()_{\mathbf{A}}, \mathbf{T}e_{\&id[p]}(), \mathbf{T}p_{\bar{\mathbf{A}}\vee U}\}\} \ . \end{aligned}$$

Towards computing unfounded sets, observe that every unfounded set can be extended to a solution to the set of nogoods Ω_Π over $A(\Omega_\Pi)$. Conversely, the solutions to Ω_Π include specific extensions of all unfounded sets, which are again 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 replacement atoms $e_{\&g[\mathbf{p}]}(\mathbf{c})$ such that $\&g[\mathbf{p}](\mathbf{c})$ is true under $\mathbf{A} \dot{\cup} \neg.U$, appropriate truth values to the auxiliary atoms according to their intuitive meaning, and assigning false to all other atoms in $A(\Omega_\Pi)$. More formally, this leads us to the following assignment:

Definition 8 (Induced Assignment of an Unfounded Set wrt. $\mathbf{\Omega_\Pi}$) *Let U be an unfounded set of a program Π wrt. assignment $\mathbf{A}$. The* assignment induced by U wrt. Ω_Π*, denoted $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$, is*

$$I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A}) = I^0_\Omega(U, \Pi, \mathbf{A}) \ \cup \{\mathbf{F}a \mid a \in A(\Omega_\Pi), \mathbf{T}a \notin I^0_\Omega(U, \Pi, \mathbf{A})\} \ ,$$

where

$$\begin{aligned} I^0_\Omega(U, \Pi, \mathbf{A}) \ = \ & \{\mathbf{T}a \mid a \in U\} \cup \{\mathbf{T}h_r \mid r \in \Pi, H(r) \cap U \neq \emptyset\} \cup \\ & \{\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \mid \&g[\mathbf{p}](\mathbf{c}) \in EA(\Pi), \ \mathbf{A} \dot{\cup} \neg.U \models \&g[\mathbf{p}](\mathbf{c})\} \cup \\ & \{\mathbf{T}a_{\mathbf{A}} \mid a \in A(\Pi), \mathbf{T}a \in \mathbf{A}\} \cup \{\mathbf{T}\hat{a}_{\mathbf{A}} \mid a \in EA(\Pi), \mathbf{A} \models a\} \cup \\ & \{\mathbf{T}a_{\mathbf{A}\wedge U} \mid a \in A(\Pi), \mathbf{T}a \in \mathbf{A}, a \in U\} \cup \\ & \{\mathbf{T}a_{\mathbf{A}\dot{\cup}\neg.U} \mid a \in A(\Pi), \mathbf{T}a \in \mathbf{A}, a \notin U\} \cup \\ & \{\mathbf{T}a_{\overline{\mathbf{A}}\vee U} \mid a \in A(\Pi), \mathbf{F}a \in \mathbf{A} \text{ or } a \in U\} \ . \end{aligned}$$

If we adopt for an assignment $\mathbf{A}$ the assumption set

$$\begin{aligned} \mathcal{A}_{\mathbf{A}} \ = \ & \{\mathbf{T}a_{\mathbf{A}} \mid a \in A(\Pi), \mathbf{T}a \in \mathbf{A}\} \cup \{\mathbf{F}a_{\mathbf{A}} \mid a \in A(\Pi), \mathbf{F}a \in \mathbf{A}\} \cup \\ & \{\mathbf{T}\hat{a}_{\mathbf{A}} \mid a \in EA(\Pi), \mathbf{A} \models a\} \cup \{\mathbf{F}\hat{a}_{\mathbf{A}} \mid a \in EA(\Pi), \mathbf{A} \not\models a\} \ , \end{aligned}$$

then all assignments induced by unfounded sets of Π wrt. $\mathbf{A}$ are solutions to Ω_Π wrt. $\mathcal{A}_{\mathbf{A}}$ (but not conversely, because intuitively the latter do not reflect the semantics of external sources).

As before, we call a set of nogoods N *conservative*, if $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ is a solution to N for every unfounded set U of Π wrt. $\mathbf{A}$. Under this property, those interpretations are solutions of the whole nogood set which comply with the assumptions from $\mathbf{A}$.

Proposition 8 *Let U be an unfounded set of a program Π wrt. assignment $\mathbf{A}$ such that $\mathbf{A}^{\mathbf{T}} \cap U \neq \emptyset$. If $O_{\Omega,\Pi}$ is conservative, then $I_{\Omega}(U, \Omega_{\Pi}, \Pi, \mathbf{A})$ is a solution to Ω_{Π} that satisfies $\mathcal{A}_{\mathbf{A}}$.*

Corollary 9 *If Ω_{Π} has no solution which satisfies $\mathcal{A}_{\mathbf{A}}$, then $U \cap \mathbf{A}^{\mathbf{T}} = \emptyset$ for every unfounded set U of Π (assuming $O_{\Omega,\Pi}$ is conservative).*

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

Theorem 10 *Let S be a solution to Ω_{Π} (with conservative $O_{\Omega,\Pi}$) that satisfies $\mathcal{A}_{\mathbf{A}}$ such that*

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

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

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

As for $\Gamma_{\Pi}^{\mathbf{A}}$, the proposition states that the non-replacement atoms in S that are true and appear in Π form an unfounded set, provided that each replacement atom $e_{\&g[\mathbf{p}]}(\mathbf{c})$ in S has the same truth value as $\&g[\mathbf{p}](\mathbf{c})$ under $\mathbf{A} \mathbin{\dot{\cup}} \neg.U$ (as in Definition 8). Again, this check is just required if the truth value of $e_{\&g[\mathbf{p}]}(\mathbf{c})$ in S is different from the one of $\&g[\mathbf{p}](\mathbf{c})$ under $\mathbf{A}$.

Similarly as for the encoding Γ, it follows immediately from Definition 8 that this postcheck does not eliminate unfounded sets, as formalized by the following proposition.

Proposition 11 *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 $I_{\Omega}(U, \Omega_{\Pi}, \Pi, \mathbf{A})$ fulfills Conditions (a) and (b) of Theorem 10.*

Example 11 Reconsider program $\Pi = \{r_1 : p \leftarrow \&id[p]()\}$ from Example 6 and the compatible set $\mathbf{A}_2 = \{\mathbf{T}p, \mathbf{T}e_{\&id[p]}\}$. The nogood set

$$\begin{aligned}\Omega_{\Pi} \quad = \quad & \{\{\mathbf{F}p\}, \{\mathbf{F}p_{\mathbf{A}\wedge U}, \mathbf{T}p_{\mathbf{A}}, \mathbf{T}p\}, \{\mathbf{T}p_{\mathbf{A}\wedge U}, \mathbf{F}p_{\mathbf{A}}\}, \{\mathbf{T}p_{\mathbf{A}\wedge U}, \mathbf{F}p\}, \\ & \{\mathbf{F}p_{\bar{\mathbf{A}}\vee U}, \mathbf{F}p\}, \{\mathbf{F}p_{\bar{\mathbf{A}}\vee U}, \mathbf{T}p\}, \{\mathbf{T}p_{\bar{\mathbf{A}}\vee U}, \mathbf{T}p_{\mathbf{A}}, \mathbf{F}p\}, \{\mathbf{T}p_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{F}p_{\mathbf{A}}\}, \\ & \{\mathbf{T}p_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{T}p\}, \{\mathbf{F}p_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{T}p_{\mathbf{A}}, \mathbf{F}p\}, \\ & \{\mathbf{T}h_{r_1}, \mathbf{F}p\}, \{\mathbf{F}h_{r_1}, \mathbf{T}p\}, \{\mathbf{T}h_{r_1}, \mathbf{T}e_{\&id[p]}()_{\mathbf{A}}, \mathbf{T}e_{\&id[p]}(), \mathbf{T}p_{\bar{\mathbf{A}}\vee U}\}\}\end{aligned}$$

with assumptions $\mathcal{A}_{\mathbf{A}_2} = \{\mathbf{T}p_{\mathbf{A}}\}$ has solutions $S \supseteq \{\mathbf{T}h_{r_1}, \mathbf{T}p, \mathbf{T}p_{\mathbf{A}}, \mathbf{F}e_{\&id[p]}, \mathbf{T}p_{\mathbf{A}\wedge U}, \mathbf{T}p_{\bar{\mathbf{A}}\vee U}, \mathbf{F}p_{\mathbf{A}\dot{\cup}\neg.U}\}$, which correspond to the unfounded set $U = \{p\}$. Here, $\mathbf{F}e_{\&id[p]}()$ represents that $\mathbf{A}_2 \mathbin{\dot{\cup}} \neg.U \not\models \&id[p]()$.

We will see in Section 6 that the encoding Ω_{Π} is superior to $\Gamma_{\Pi}^{\mathbf{A}}$ for many practically relevant programs. The effect becomes especially visible if they need many unfounded set checks, which intuitively is the case when many answer sets exist; here reusability of the encoding is very beneficial, while for small programs with few answer sets, the incurred overhead does not lead to savings.

4. Optimization and Learning

In this section we first discuss some refinements and optimizations of our nogood encodings for UFS search. In particular, we present nogoods which prune irrelevant parts of the search space; they can be integrated into both encodings $\Gamma_{\Pi}^{\mathbf{A}}$ and Ω_{Π} under suitable adjustments. After that, we propose a strategy for learning nogoods from detected unfounded sets, avoiding that the same unfounded set is generated later again.

4.1 Optimization

We present now three optimizations which turned out to be effective in improving UFS search, where the second and the third exclude each other, i.e., they can not be used simultaneously.

4.1.1 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.

Proposition 12 *Suppose U is an unfounded set of Π wrt. an interpretation $\mathbf{A}$ such that $\mathbf{A} \not\models a$ for some $a \in U$. Then $U \setminus \{a\}$ is an unfounded set of Π wrt. $\mathbf{A}$.*

The nogoods for this optimization are simple. In the encoding $\Gamma_{\Pi}^{\mathbf{A}}$, we add the conservative nogood $\{\mathbf{T}a\}$ for each $a \in A(\Pi)$ with $\mathbf{A} \not\models a$ to the optimization part $O_{\Gamma,\Pi}^{\mathbf{A}}$ and in the encoding Ω_{Π} the conservative nogood $\{\mathbf{F}a_{\mathbf{A}}, \mathbf{T}a\}$ for each $a \in A(\Pi)$ to the optimization part $O_{\Omega,\Pi}$.

4.1.2 Avoiding Guesses of Replacement Atoms

In some situations, the truth value of a replacement atom b in a solution S to $\Gamma_{\Pi}^{\mathbf{A}}$ resp. Ω_{Π} with assumptions $\mathcal{A}_{\mathbf{A}}$ is irrelevant. That is, both $S_{\mathbf{T}b} = (S \setminus \{\mathbf{T}b, \mathbf{F}b\}) \cup \{\mathbf{T}b\}$ and $S_{\mathbf{F}b} = (S \setminus \{\mathbf{T}b, \mathbf{F}b\}) \cup \{\mathbf{F}b\}$ are solutions to $\Gamma_{\Pi}^{\mathbf{A}}$ resp. Ω_{Π} that satisfy $\mathcal{A}_{\mathbf{A}}$, and they represent the same unfounded set. We then can set the truth value of b to an (arbitrary) fixed value instead of inspecting both alternatives. The next proposition states a sufficient criterion for this irrelevance.

Proposition 13 *Let b be a replacement atom, and let S be a solution to $\Gamma_{\Pi}^{\mathbf{A}}$ resp. Ω_{Π} satisfying $\mathcal{A}_{\mathbf{A}}$. If for every rule $r \in \Pi$ such that $b \in B^+(\hat{r}) \cup B^-(\hat{r})$ and $\mathbf{A} \models B(r)$, 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 $S_{\mathbf{T}b}$ and $S_{\mathbf{F}b}$ are solutions to $\Gamma_{\Pi}^{\mathbf{A}}$ resp. Ω_{Π} that satisfy $\mathcal{A}_{\mathbf{A}}$.

This property can be utilized by adding conservative nogoods. Recall that $A(\Gamma_{\Pi}^{\mathbf{A}})$ and $A(\Omega_{\Pi})$ contain atoms l_r for every $r \in \Pi$. They intuitively serve to encode for a solution S to $\Gamma_{\Pi}^{\mathbf{A}}$ resp. Ω_{Π} with assumptions $\mathcal{A}_{\mathbf{A}}$ whether the truth values of the replacement atoms in $B(r)$ are relevant or can be set arbitrarily. The following nogoods label relevant rules r, forcing l_r to be false iff some of the conditions in Proposition 13 holds. For the encoding $\Gamma_{\Pi}^{\mathbf{A}}$, we add to $O_{\Gamma,\Pi}^{\mathbf{A}}$ for each rule r:

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

For the encoding Ω_{Π}, we add to $O_{\Omega,\Pi}$ for each rule r:

$$\begin{aligned} L_{\Omega,r} = & \{\{\mathbf{T}l_r, \mathbf{T}a, \mathbf{T}a_{\mathbf{A}}\} \mid a \in B_o^+(r)\} \cup \{\{\mathbf{T}l_r, \mathbf{F}a, \mathbf{T}a_{\mathbf{A}}\} \mid a \in H(r)\} \cup \\ & \{\{\mathbf{F}l_r\} \cup \{\mathbf{F}a_{\mathbf{A}\wedge U} \mid a \in B_o^+(r)\} \cup \{\mathbf{T}a_{\bar{\mathbf{A}}\vee U} \mid a \in H(r)\}\} \ . \end{aligned}$$

These constraints exclusively enforce either $\mathbf{T}l_r$ 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 Proposition 13 we can set the truth value of a replacement atom b arbitrarily, if l_r is false for all r such that $b \in B^+(\hat{r})$ or $b \in B^-(\hat{r})$. However, it must be ensured that changing the truth value of replacement atoms does not harm the satisfaction of the conditions in Theorem 6 (resp. Theorem 10).

As mentioned after Theorem 6, it is beneficial 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. Importantly, this also relaxes the antecedence of the conditions in Theorem 6 (resp. Theorem 10), and guarantees that they are not harmed. The following nogoods enforce a coherent interpretation of the replacement atoms.

For the encoding $\Gamma_\Pi^\mathbf{A}$ we add to $O_{\Gamma,\Pi}^\mathbf{A}$ for each rule r:

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

while for the encoding Ω_Π we add to $O_{\Gamma,\Pi}$ for each rule r:

$$\begin{aligned} F_{\Omega,r} = &\big\{\{\mathbf{F}l_r \mid b \in B^+(\hat{r}) \cup B^-(\hat{r})\} \cup \{\mathbf{T}b_\mathbf{A}, \mathbf{F}b\} \mid b \in B_e(\hat{r})\big\} \cup \\ &\big\{\{\mathbf{F}l_r \mid b \in B^+(\hat{r}) \cup B^-(\hat{r})\} \cup \{\mathbf{F}b_\mathbf{A}, \mathbf{T}b\} \mid b \in B_e(\hat{r})\big\} \quad . \end{aligned}$$

In summary, the encoding $\Gamma_\Pi^\mathbf{A}$ has the optimization part $O_{\Gamma,\Pi}^\mathbf{A} = \bigcup_{r\in\Pi} L_{\Gamma,r}^\mathbf{A} \cup F_{\Gamma,r}^\mathbf{A}$ and the encoding Ω_Π the optimization part $O_{\Omega,\Pi} = \bigcup_{r\in\Pi} L_{\Omega,r} \cup F_{\Omega,r}$.

We give now an example for this optimization using our encoding Γ.

Example 12 Consider the program $\Pi = \{r_1\colon p \leftarrow \&id[p]();r_2\colon q \leftarrow \&id[q]()\}$, and the compatible set $\hat{\mathbf{A}} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}e_{\&id[p]}(), \mathbf{T}e_{\&id[q]}()\}$. Then the necessary part of encoding $\Gamma_\Pi^\mathbf{A}$ 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 $\mathbf{A}$).

Note that if this optimization is not used, then for all rules r the atom l_r is in fact not needed and thus unconstrained. To avoid an exponential increase of the number of UFS candidates, these atoms should then be set to a fixed value.

4.1.3 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 9 *A nogood of the form* $N = \{\mathbf{T}t_1, \dots, \mathbf{T}t_n, \mathbf{F}f_1, \dots, \mathbf{F}f_m, \sigma e_{\&g[\mathbf{p}]}(\mathbf{c})\}$*, where* σ *is* $\mathbf{T}$ *or* $\mathbf{F}$*, is a* valid input-output-relationship*, if for every assignment* $\mathbf{A}$ *such that* $N \setminus \{\sigma e_{\&g[\mathbf{p}]}(\mathbf{c})\} \subseteq \mathbf{A}$ *it holds that* $\mathbf{A} \models \&g[\mathbf{p}](\mathbf{c})$ *if* $\sigma = \mathbf{F}$*, and* $\mathbf{A} \not\models \&g[\mathbf{p}](\mathbf{c})$ *if* $\sigma = \mathbf{T}$.

Here, the signed literals with atoms t_i, $1 \leq i \leq n$, resp. f_j, $1 \leq j \leq m$, reflect the relevant true resp. false atoms in the interpretation $\mathbf{A}$, built over predicates which occur in the input list $\mathbf{p}$. Techniques for learning such nogoods have been described by Eiter et al. (2012a) and exploit properties of external sources (such as monotonicity and functionality) to restrict the size of N.

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 $\bar{\mathbf{F}} = \mathbf{T}$ and $\bar{\mathbf{T}} = \mathbf{F}$.

Definition 10 (Nogood Transformation $\mathcal{T}_\Gamma$) *For a valid input-output relationship* $N = \{\mathbf{T}t_1, \dots, \mathbf{T}t_n, \mathbf{F}f_1, \dots, \mathbf{F}f_m, \sigma e_{\&g[\mathbf{p}]}(\mathbf{c})\}$ *and an assignment* $\mathbf{A}$*, the nogood transformation* $\mathcal{T}_\Gamma$ *is defined as*

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

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

Proposition 14 *Let* N *be a valid input-output relationship, and let* U *be an unfounded set wrt.* Π *and* $\mathbf{A}$*. If* $O^{\mathbf{A}}_{\Gamma,\Pi}$ *contains only conservative nogoods, then* $I_\Gamma(U, \Gamma^{\mathbf{A}}_\Pi, \Pi, \mathbf{A})$ *is a solution to* $\mathcal{T}_\Gamma(N, \mathbf{A})$ *(i.e., also nogoods* $\mathcal{T}_\Gamma(N, \mathbf{A})$ *are conservative).*

Hence, all valid input-output relationships for external atoms that are learned during the search for compatible sets can be reused (applying the above transformation) for the UFS check. Moreover, during the evaluation of external atoms in the postcheck for candidate unfounded sets (i.e., solutions to $\Gamma^{\mathbf{A}}_\Pi$), further valid input-output relationships might be learned. They can in turn be used by (further) unfounded set checks (in transformed form) but also in the search for compatible sets.

Example 13 (Set Partitioning) For the program Π from Example 4, consider the compatible set $\hat{\mathbf{A}} = \{\mathbf{T}domain(a), \mathbf{T}sel(a), \mathbf{T}e_{\&diff[nsel]}(a)\}$. 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 $a\mathcal{T}_\Gamma(N,\mathbf{A}) = \{\{\mathbf{F}domain(a), \mathbf{F}e_{\&diff[nsel]}(a)\}\}$; it 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 holds because $e_{\&diff[nsel]}(a)$ is true under $\mathbf{A}$ and can only change its truth value if $domain(a)$ becomes false.

This learning technique can be adopted for our encoding Ω_Π as follows.

Definition 11 (Nogood Transformation $\mathcal{T}_\Omega$) *For a valid input-output relationship* N*, the nogood transformation* $\mathcal{T}_\Omega$ *is defined as*

$$\mathcal{T}_\Omega(N) = \{\{\sigma e_{\&g[\mathbf{p}]}(\mathbf{c})\} \cup \{\mathbf{T}t_{1\mathbf{A}}, \mathbf{F}t_1, \dots, \mathbf{T}t_{n\mathbf{A}}, \mathbf{F}t_n, \mathbf{F}f_{1\mathbf{A}\dot\cup\neg.U}, \dots, \mathbf{F}f_{m\mathbf{A}\dot\cup\neg.U}\}\} \ .$$

Compared to $\mathcal{T}_\Gamma(N, \mathbf{A})$, the main difference is that $\mathcal{T}_\Omega(N)$ is reusable for every assignment, similar to the definition of our unfounded set detection problem Ω_Π.

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

Proposition 15 *Let* N *be a valid input-output relationship, and let* U *be an unfounded set wrt.* Π *and* $\mathbf{A}$*. If* $O_{\Omega,\Pi}$ *contains only conservative nogoods, then* $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ *is a solution to* $\mathcal{T}_\Omega(N)$ *(i.e., also nogoods* $\mathcal{T}_\Omega(N)$ *are conservative).*

Hence, also with encoding Ω_Π all valid input-output relationships for external atoms that are learned during the search for compatible sets can be reused and vice versa.

Example 14 (cont'd) Reconsider the program Π from Example 4 and the compatible set $\hat{\mathbf{A}} = \{\mathbf{T}domain(a), \mathbf{T}sel(a), \mathbf{T}e_{\&diff[domain,nsel]}(a)\}$. Suppose the main search has learned the input-output relationship $N = \{\mathbf{T}domain(a),\ \mathbf{F}nsel(a), \mathbf{F}e_{\&diff[domain,nsel]}(a)\}$. The transformed nogood is

$$\mathcal{T}_{\Omega}(N){=}\{\{\mathbf{T}domain(a)_{\mathbf{A}}, \mathbf{F}domain(a), \mathbf{F}nsel(a)_{\mathbf{A}\dot{\cup}\neg.U}, \mathbf{F}e_{\&diff[domain,nsel]}(a)_{\mathbf{A}\dot{\cup}\neg.U}\}\} \quad ;$$

it intuitively encodes that if $domain(a)$ is true in the assignment $\mathbf{A}$ but *not* in the unfounded set U, and if $nsel(a)$ is false in $\mathbf{A} \dot{\cup} \neg.U$, then $e_{\&diff[domain,nsel]}(a)$ is true under $\mathbf{A} \dot{\cup} \neg.U$. This holds as $e_{\&diff[domain,nsel]}(a)$ is true under $\mathbf{A}$ and can only change its truth value if $domain(a)$ gets false.

The nogood exchange also benefits from our advanced encoding. With our previous encoding $\Gamma_{\Pi}^{\mathbf{A}}$, we needed to build the SAT instance from scratch for every unfounded set check. Thus, nogoods learned in the main search for compatible sets need to be transformed and added to the UFS detection problem for every check (otherwise they are lost). With our new encoding Ω_{Π}, this is done only once because learned nogoods are kept for multiple unfounded set checks. This also allows us to make use of advanced forgetting heuristics in SAT solvers more effective.

Finally, an important note is that the optimizations presented in Section 4.1.2 and 4.1.3 *can not* be used simultaneously (differently from the optimizations in Section 4.1.1 and 4.1.2 resp. 4.1.1 and 4.1.3), as this can result in contradictions due to (transformed) learned nogoods. We thus disabled the optimization for avoiding guesses of replacement atoms (Section 4.1.2) in our experiments.

4.2 Learning Nogoods from Unfounded Sets

Until now we have considered merely detecting unfounded sets. A strategy to *learn* from detected unfounded sets for the main search for compatible sets is missing. We next develop such a strategy which we call *unfounded set learning* (UFL).

Example 15 Consider the program $\Pi = \{\, p \leftarrow \&id[p]();\ x_1 \vee x_2 \vee \cdots \vee x_k \leftarrow \}$. As we know from Example 7, $U = \{p\}$ is a UFS of the subprogram $\Pi' = \{\, p \leftarrow \&id[p]() \,\}$ wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}e_{\&id}()\}$. The same is true for Π and moreover for every $\mathbf{A}' \supset \mathbf{A}$; i.e., p must never be true.

The program in Example 15 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 search for compatible sets in a way such that the same unfounded sets are not reconsidered. We present two such strategies, but will focus on the first one because our experiments have shown that the first one is superior for all our instances.

4.2.1 UFS-BASED LEARNING

For an unfounded set U of Π wrt. $\mathbf{A}$ we define a set $L_1(U, \Pi, \mathbf{A})$ of learned nogoods as follows. Suppose that $r_1, \ldots, r_j$ are all rules r in Π such that $H(r) \cap U \neq \emptyset$ and $U \cap B_o^+(r) = \emptyset$, i.e., the set of all "external" rules of Π wrt. U (rules which do not directly depend positively on U). Then

$$L_1(U, \Pi, \mathbf{A}) = \{\{\sigma_0, \sigma_1, \ldots, \sigma_j\} \mid \sigma_0 \in \{\mathbf{T}a \mid a \in U\}, \sigma_i \in H_i \, for\ all\ 1 \leq i \leq j)\} \quad ,$$

where $H_i = \{\mathbf{T}h \mid h \in H(r_i) \setminus U,\ \mathbf{A} \models h\} \cup \{\mathbf{F}b \mid b \in B_o^+(r_i),\ \mathbf{A} \not\models b\}$. Formally we can show that adding this set of nogoods is correct, i.e., does not prune answer sets:

Proposition 16 *If U is an unfounded set of Π wrt. $\mathbf{A}$, then every answer set of Π is a solution to the nogoods in $L_1(U, \Pi, \mathbf{A})$.*

Example 16 Consider the program Π from Example 15 and suppose we have found the unfounded set $U = \{p\}$ wrt. the interpretation $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}x_1\} \cup \{\mathbf{F}x_i \mid 1 < i \leq k\}$. Then the learned nogood $L_1(U, \mathbf{A}, \Pi) = \{\mathbf{T}p\}$ immediately guides the search to the part of the search tree where p is false, i.e., roughly half of the guesses are avoided.

4.2.2 Reduct-Based Learning

A different learning strategy is based on the models of $f\Pi^{\mathbf{A}}$ rather than the unfounded set U itself, hinging on the observation 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 $f\Pi^{\mathbf{A}}$. This was noted by Faber (2005) for aggregates.

We exploit this to construct nogoods from a nonempty U wrt. a model $\mathbf{A}$ as follows. The interpretation $\mathbf{A} \dot{\cup} \neg.U$ is not only a model of $f\Pi^{\mathbf{A}}$, but of *all* programs $\Pi' \subseteq f\Pi^{\mathbf{A}}$. Hence, if an assignment $\mathbf{A}'$ falsifies *at least* the rules of Π which $\mathbf{A}$ falsifies, and $\mathbf{A}'^{\mathbf{T}} \supset (\mathbf{A} \dot{\cup} \neg.U)^{\mathbf{T}}$, then $\mathbf{A}'$ is not an answer set of Π. This yields the following nogood set $L_2(U, \Pi, \mathbf{A})$. Suppose $r_1, \ldots, r_n$ are all rules r of Π which are *not* in its FLP-reduct wrt. $\mathbf{A}$ (i.e., $\mathbf{A} \not\models B(r_i)$. Then

$$
\begin{aligned}
L_2(U, \Pi, \mathbf{A}) = \{\{\mathbf{T}a \mid a \in (\mathbf{A} \dot{\cup} \neg.U)^{\mathbf{T}}\} \cup \{\sigma_0, \sigma_1, \ldots, \sigma_n\} \\
\mid \sigma_0 \in \{\mathbf{T}a \mid a \in U\}, \sigma_i \in H_i \text{ for all } 1 \leq i \leq n\} \quad ,
\end{aligned}
$$

where $H_i = \{\mathbf{t}a \mid a \in B(\hat{r}), \hat{\mathbf{A}} \not\models a\}$, $1 \leq i \leq n$. That is, each nogood consists of the true-part of the smaller model $\mathbf{A} \dot{\cup} \neg.U$ of the reduct $f\Pi^{\mathbf{A}}$, an unfounded atom σ_0 (i.e. true in $\mathbf{A}$ but not in $\mathbf{A} \dot{\cup} \neg.U$), and a false body literal σ_i ($1 \leq i \leq n$) for each rule of Π with unsatisfied body wrt. $\mathbf{A}$.

Example 17 Let $\Pi = \{p \leftarrow \&id[p](); q \leftarrow \&id[q]()\}$, where $\&id[a]()$ evaluates to true iff a is true. Suppose $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q\}$. Then $U = \{\mathbf{T}p, \mathbf{T}q\}$ is an unfounded set wrt. $\mathbf{A}$. In the above construction rule we have $\mathbf{A} \dot{\cup} \neg.U = \{\}$, $\sigma_0 \in \{\mathbf{T}p, \mathbf{T}q\}$ and $n = 0$ (because both rule bodies are satisfied wrt. $\mathbf{A}$). The learned nogoods are $L_2(U, \Pi, \mathbf{A}) = \{\{\mathbf{T}p\}, \{\mathbf{T}q\}\}$.

In Example 17, the learned nogoods will immediately guide the search to the interpretation $\{\mathbf{F}p, \mathbf{F}q\}$, which is the only one which becomes an answer set. Formally, we can show:

Proposition 17 *If U is an unfounded set of Π wrt. $\mathbf{A}$ and $\mathbf{A} \models \Pi$, then each answer set of Π is a solution to all nogoods in $L_2(U, \Pi, \mathbf{A})$.*

However, $L_2(U, \Pi, \mathbf{A})$ appeared to be clearly inferior to $L_1(U, \Pi, \mathbf{A})$ from Section 4.2.1. Informally, its nogoods overfit the detected unfounded set and do not generalize well to other ones.

5. Deciding the Necessity of the Minimality Check

Although the minimality check based on unfounded sets is more efficient than the explicit minimality check, the computational costs are still high. Moreover, during the evaluation of $\hat{\Pi}$ for computing the compatible set $\hat{\mathbf{A}}$, the ASP solver has already made an unfounded set check, and we can safely assume that it is founded from the perspective of the ASP solver. Hence, all remaining unfounded sets which were not discovered by the ASP solver must involve external sources, as their behavior is not fully captured by the ASP solver.

In this section we pursue these ideas and give a decision criterion for deciding whether a further UFS check is necessary. We eventually define a class of programs which needs no additional UFS check. Intuitively, we show that every unfounded set that is not already detected during the construction of $\hat{\mathbf{A}}$ contains input atoms of external atoms involved in cycles. If the program has no such input atom, then the UFS check is superfluous. Afterwards, we show how to apply this criterion, which holds in practically relevant cases, to program components; this often yields additional speedup. However, there are also cases where the UFS check can not be skipped; e.g., recursive URL retrieval from a web resource (which requires cyclic use of an external atom).

5.1 Basic Decision Criterion

We start with a definition of atom dependency.

Definition 12 (Atom Dependency) *For a ground program Π, and ground atoms $p(\mathbf{c})$ and $q(\mathbf{d})$, we say that*

(i) $p(\mathbf{c})$ depends *on $q(\mathbf{d})$, denoted $p(\mathbf{c}) \rightarrow q(\mathbf{d})$, if for some rule $r \in \Pi$ we have $p(\mathbf{c}) \in H(r)$ and $q(\mathbf{d}) \in B(r)$;*

(ii) $p(\mathbf{c})$ depends externally *on $q(\mathbf{d})$, denoted $p(\mathbf{c}) \rightarrow_e q(\mathbf{d})$, if some rule $r \in \Pi$ and external atom $\&g[q_1, \ldots, q_n](\mathbf{e}) \in B^+(r) \cup B^-(r)$ exist such that $p(\mathbf{c}) \in H(r)$ and $q \in \{q_1, \ldots, q_n\}$.*

In the following, we consider *dependency graphs* $G^R_\Pi = (V, E)$ for a ground program Π, whose vertices V are the ground atoms and whose edges E are given by a binary relation R over ground atoms ($E = R$). We call $p(\mathbf{c}) \rightarrow q(\mathbf{d})$ also an *ordinary edge* and $p(\mathbf{c}) \rightarrow_e q(\mathbf{d})$ an *e-edge*.

We establish a lemma that allows us to restrict our attention to the "core" of an unfounded set, i.e., its most essential part; we can disregard atoms in a cut of G^R_Π, which is defined as follows.

Definition 13 (Cut) *Let U be an unfounded set of Π wrt. $\mathbf{A}$. A set of atoms $C \subseteq U$ is a* cut *of G^R_Π, if*

(i) $b \rightarrow_e a \notin G^R_\Pi$, for all $a \in C$ and $b \in U$ (C has no incoming e-edge from U),

(ii) $b \rightarrow a \notin G^R_\Pi$ and $a \rightarrow b \notin G^R_\Pi$, for all $a \in C$ and $b \in U \setminus C$ (there are no ordinary edges between C and $U \setminus C$).

We first prove that cuts can be removed from unfounded sets and the resulting set is still an unfounded set.

Lemma 18 (Unfounded Set Reduction Lemma) *Let U be an unfounded set of Π wrt. a complete assignment $\mathbf{A}$, and let C be a cut of G^R_Π. Then, $Y = U \setminus C$ is an unfounded set of Π wrt. $\mathbf{A}$.*

Example 18 Consider the following program:

$$\Pi = \{r \leftarrow \&id[r](); \quad p \leftarrow \&id[r](); \quad p \leftarrow q; \quad q \leftarrow p\} \ .$$

Then we have $p \rightarrow q$, $q \rightarrow p$, $r \rightarrow_e r$ and $p \rightarrow_e r$. Π has the unfounded set $U = \{p, q, r\}$ wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$. Observe that $C = \{p, q\}$ is a cut of G^R_Π, and therefore $U \setminus C = \{r\}$ is an unfounded set of Π wrt. $\mathbf{A}$.

Next we prove that intuitively, for each unfounded set U of Π, either the input to some external atom is unfounded itself, or U is already detected when $\hat{\Pi}$ is evaluated.

Lemma 19 (EA-Input Unfoundedness) *Let U be an unfounded set of Π wrt. an assignment $\mathbf{A}$. If G^R_Π has no edge $x \rightarrow_e y$ such that $x, y \in U$, then U is an unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$.*

Example 19 Reconsider the program Π from Example 18. Then the unfounded set $U' = \{p, q\}$ wrt. $\mathbf{A}' = \{\mathbf{T}p, \mathbf{T}q, \mathbf{F}r\}$ is already detected when

$$\hat{\Pi} = \{\, e_{\&id[r]}() \vee ne_{\&id[r]}() \leftarrow \ ; \quad r \leftarrow e_{\&id[r]}(); \quad p \leftarrow e_{\&id[r]}(); \quad p \leftarrow q; \quad q \leftarrow p \,\}$$

is evaluated by the ASP solver because no edges $p \rightarrow_e q$ and $q \rightarrow_e p$ exist. In contrast, the unfounded set $U'' = \{p, q, r\}$ wrt. $\mathbf{A}'' = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$ is *not* detected by the ASP solver because $p, r \in U''$ and $p \rightarrow_e r$.

Thus, the unfounded sets of Π wrt. $\mathbf{A}$ that are not recognized during the evaluation of $\hat{\Pi}$ have cyclic dependencies over input atoms of some external atom. Programs with acyclic dependencies do not need additional UFS checks.

Recall that a *cycle* wrt. a binary relation R is a sequence $C = c_0, c_1, \ldots, c_n, c_{n+1}$ of elements, $n \geq 0$, such that $(c_i, c_{i+1}) \in R$ for all $0 \leq i \leq n$ and $c_0 = c_{n+1}$. A set S *contains a cycle* wrt. R, if there is a cycle $C = c_0, c_1, \ldots, c_n, c_{n+1}$ wrt. R such that $c_i \in S$ for all $0 \leq i \leq n+1$.

Informally, the next proposition states that each unfounded set of Π wrt. $\mathbf{A}$ which contains no cycle through the input atoms to some external atom corresponds to some unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$, i.e., the unfoundedness is already detected when $\hat{\Pi}$ is evaluated.

Let $\rightarrow^d = \rightarrow \cup \leftarrow \cup \rightarrow_e$, where $\leftarrow$ is the inverse of $\rightarrow$ (i.e. $\leftarrow = \{(x, y) \mid (y, x) \in \rightarrow\}$). A cycle $c_0, c_1, \ldots, c_n, c_{n+1}$ under $\rightarrow^d$ is called an *e-cycle*, if it contains an e-edge, i.e., $c_i \rightarrow_e c_{i+1}$ for some $0 \leq i \leq n$.

Theorem 20 (Relevance of e-cycles) *Let $U \neq \emptyset$ be an unfounded set of Π wrt. an interpretation $\mathbf{A}$ which does not contain any e-cycle under $\rightarrow^d$. Then $\hat{\Pi}$ has a nonempty unfounded set wrt. $\hat{\mathbf{A}}$.*

Corollary 21 *If a program Π has no e-cycle under $\rightarrow^d$ and $\hat{\Pi}$ has no unfounded set wrt. an interpretation $\hat{\mathbf{A}}$, then $\mathbf{A}$ is unfounded-free for Π.*

This corollary can be used to increase the performance of an evaluation algorithm as follows: if there is no cycle under $\rightarrow^d$ containing e-edges, then an explicit unfounded set check is not necessary because the unfounded set check during the evaluation of $\hat{\Pi}$ is sufficient. Note that this test can be done efficiently (in fact in linear time, similar to deciding stratifiability of an ordinary logic program). Moreover, in practice one can abstract from $\rightarrow^d$ by using analogous relations on the level of predicate symbols instead of atoms. Clearly, if there is no e-cycle in the predicate dependency graph, then there can also be no e-cycle in the atom dependency graph. Hence, the predicate dependency graph can be safely used to decide whether the unfounded set check can be skipped.

Example 20 All example programs so far need an UFS check, but the program $\Pi = \{out(X) \leftarrow \&\mathit{diff}[set_1, set_2](X)\} \cup F$, where diff computes the set difference of unary predicates set_1 and set_2 and F is any set of facts, needs no UFS check as there is no e-cycle under $\rightarrow^d$. Also the program $\Pi = \{str(Z) \leftarrow dom(Z), str(X), str(Y), \text{not } \&concat[X, Y](Z)\}$ (where $\&concat$ takes two constants and computes their string concatenation) needs no UFS check; there is a cycle over an external atom, but no e-cycle under $\rightarrow^d$.

Unfortunately, the converse of Theorem 20 does not hold, that is, $\hat{\Pi}$ may fail to be unfounded-free wrt. $\hat{\mathbf{A}}$ but no unfounded set of Π wrt. $\mathbf{A}$ contains an e-cycle; thus, the condition in Corollary 21 is not necessary for unfounded-freeness of Π wrt. $\mathbf{A}$. However, the following generalization of Theorem 20 allows us to conclude that if $\hat{\Pi}$ is unfounded-free wrt. $\hat{\mathbf{A}}$, then every unfounded set U of Π wrt. $\mathbf{A}$ must contain an atom that provides input to an external atom on a cycle under $\rightarrow^d$.

Definition 14 (Cyclic Input Atoms) *For a program Π, an atom a is a* cyclic input atom*, if some edge $b \rightarrow_e a$ with a path from a to b under $\rightarrow^d$ exists.*

Let $CA(\Pi)$ denote the set of all cyclic input atoms of program Π.

Theorem 22 (Unfoundedness of Cyclic Input Atom) *Let $U \neq \emptyset$ be an unfounded set of Π wrt. $\mathbf{A}$ such that $U \cap CA(\Pi) = \emptyset$. Then, $\hat{\Pi}$ has a nonempty unfounded set wrt. $\hat{\mathbf{A}}$.*

As a consequence of Theorem 22, we can add the nogood $\{\mathbf{F}a \mid a \in CA(\Pi)\}$ to $\Gamma_{\Pi}^{\mathbf{A}}$. Again using predicate symbols instead of atoms reduces the overhead of the dependency graph.

Example 21 Reconsider Π in Example 18. Then $U = \{p, q\}$ is an unfounded set wrt. $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{F}r\}$; as U is disjoint from $CA(\Pi) = \{r\}$, it is detected during the evaluation of $\hat{\Pi}$.

5.2 Program Decomposition

The usefulness of the decision criterion can be increased by decomposing the program into components, such that the criterion can be applied componentwise. This allows us to restrict the UFS check to components with e-cycles, while e-cycle-free components can be ignored.

Let $\mathcal{C}$ be a partitioning of the ordinary atoms $A(\Pi)$ of Π into subset-maximal strongly connected components under $\rightarrow \cup \rightarrow_e$. We define for each partition $C \in \mathcal{C}$ the subprogram Π_C *associated with* C as $\Pi_C = \{r \in \Pi \mid H(r) \cap C \neq \emptyset\}$.

We next show that if a program has an unfounded set U wrt. $\mathbf{A}$, then $U \cap C$ is an unfounded set wrt. $\mathbf{A}$ for the subprogram of some strongly connected component C.

Theorem 23 *Let $U \neq \emptyset$ be an unfounded set of Π wrt. $\mathbf{A}$. Then, for some $C \in \mathcal{C}$ it holds that $U \cap C$ is a nonempty unfounded set of Π_C wrt. $\mathbf{A}$.*

Note that constraints (i.e., rules with empty head) do not harm this proposition. Each constraint r of kind $\leftarrow B(r)$ can be rewritten to $p \leftarrow B(r), \mathrm{not}\, p$ for a new atom p, and $C = \{p\}$ is a strongly connected component with $\Pi_C = \{r\}$, which does not contain an e-cycle. Thus, for the rewritten constraints the according subprograms Π_C can be ignored anyways.

This proposition states that a search for unfounded sets can be done independently for the subprograms Π_C for all $C \in \mathcal{C}$. If there is an unfounded set of Π wrt. an assignment, then there is also an unfounded set of at least one program component wrt. this assignment. We know by Corollary 21 that programs Π without e-cycles can only contain unfounded sets that are already detected when $\hat{\Pi}$ is solved. If we apply Theorem 23 to the subprograms Π_C, we can safely ignore e-cycle-free program components.

Example 22 Reconsider the program Π from Example 18. Then $\mathcal{C}$ contains the components $C_1 = \{p, q\}$ and $C_2 = \{r\}$ and we have $\Pi_{C_1} = \{p \leftarrow \&id[r](); p \leftarrow q; q \leftarrow p\}$ and $\Pi_{C_2} = \{r \leftarrow \&id[r]()\}$. By Theorem 23, each unfounded set of Π wrt. some assignment $\mathbf{A}$ gives rise to an unfounded set of either Π_{C_1} or Π_{C_2}. E.g., consider $U = \{p, q, r\}$ and $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$; then $U \cap \{r\} = \{r\}$ is an unfounded set of Π_{C_2} wrt. $\mathbf{A}$. As Π_{C_1} has no e-cycles, we conclude from Corollary 21 that all unfounded sets of Π_{C_1} are already detected when $\hat{\Pi}$ (resp., $\hat{\Pi}_{C_1}$) is evaluated. Hence, only Π_{C_2} needs an additional UFS check. Indeed, the only unfounded set of $\hat{\Pi}$ that is not detected when $\hat{\Pi}$ is evaluated is $\{r\}$, which is unfounded wrt. each interpretation $\mathbf{A} \supseteq \{\mathbf{T}r\}$ for Π_{C_2} and Π.

Finally, we show that splitting, i.e., the component-wise check for foundedness, does not lead to spurious unfounded sets.

Proposition 24 *If U is an unfounded set of Π_C wrt. $\mathbf{A}$ such that $U \subseteq C$, then U is an unfounded set of Π wrt. $\mathbf{A}$.*

The results can be generalized to subprograms that are larger than strongly connected components; however, we leave a detailed study of this for future work.

6. 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.

In our experiments, we will also use *external behavior learning (EBL)* as developed by Eiter et al. (2012a). The basic idea is to learn additional nogoods from evaluations of external atoms, which capture (parts of) the behavior of external sources. Thus, these nogoods eliminate model candidates which violate the known semantics of external atoms.

Regarding a concrete setting, there is a large number of combinations of EBL and the techniques presented in this paper. Indeed, we may either activate or deactivate external behavior learning and use either the explicit or the UFS-based minimality check. In the latter case, we can further use unfounded set learning (UFL), the decision criterion for skipping the unfounded set check can be exploited or ignored, and program decomposition might be used or not. Moreover, we can choose between the encodings Γ and Ω. In total, these are 34 different settings.

However, we will restrict our discussion to some interesting configurations. In general, we will activate the developed features stepwise such that in our tables the efficiency increases from left to right. We will start with the traditional algorithm based on an explicit minimality check without any learning techniques of Eiter et al. (2012a) and from this paper (i.e., only conflict-driven learning inside CLASP is used). In the next step we will add external behavior learning, while UFL is not possible with the explicit check. Then we switch from the explicit minimality check to the UFS-based check without learning and without exploiting the decision criterion and program decomposition. Nevertheless, this naive kind of UFS-based minimality checking is often more efficient than the explicit minimality check with EBL. In the next step, we add the decision criterion and program decomposition. In the following, *monolithic* (*mol*) means that both the decision criterion and the program decomposition are off, and *modular* (*mod*) that they are on. Next we add EBL and UFL to the UFS-based minimality check, and finally we switch the encoding from Γ to Ω (including EBL, UFL and modular decomposition). However, we might skip some of the steps for specific benchmarks and argue why they are uninteresting in the respective cases. Detailed instance information and results with all combinations of parameters are available.[2]

2. http://www.kr.tuwien.ac.at/research/projects/hexhex/ufs

Briefly, our results show a clear improvement, for both synthetic and application instances, by the UFS check and EBL. Moreover, a closer analysis shows that the UFS check decreases in some cases not only the runtime but also the number of enumerated candidates (UFS resp. model candidates of the FLP reduct) and the number of external atom evaluations.

We evaluated the implementation on a Linux server with two 12-core AMD 6176 SE CPUs with 128GB RAM running DLVHEX version 2.3.0. The evaluated techniques were configured using commandline arguments. To the best of our knowledge, DLVHEX is the only implementation of the HEX semantics. In each test run the CPU usage was limited to two CPU cores, running a Condor load distribution system which ensures robust runtimes (i.e., multiple runs of the same instance have negligible deviations). The timeout was uniformly set to 300 seconds for each instance; for each parameter value, the average runtime over all instances is printed where timeouts, whose number is shown in parentheses, are fully taken into account.

6.1 Detailed Benchmark Description and Experimental Results

We give now a detailed description of the benchmarks used in our experiments, and present the results of our experimental evaluation.

6.1.1 SET PARTITIONING

This benchmark extends the program from Example 4 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 $domain$. The results are shown in Table 1. Here we see a big advantage of the UFS check over the explicit check, for both computing all answer sets and finding the first one. A closer investigation shows that the improvement is mainly due to the optimizations in Section 4, which make the UFS check investigate significantly fewer candidates than the explicit FLP check. Furthermore the UFS check requires fewer external computations.

Both the explicit and the UFS-based minimality check benefit from EBL if we compute all answer sets, but the results show that the UFS-based check benefits more. In contrast, UFL (not shown in the table) does not lead to a further speedup as no unfounded sets are found in this program.

The decision criterion and program decomposition improve the runtime slightly for small instances. However, for large instances the decision criterion cannot avoid the UFS check in most components of the program because of its cyclic structure. Thus a single UFS check over the whole program is replaced by multiple UFS checks over individual program components, which involves more overhead that becomes visible when computing all answer sets.

If we compute only one answer set, then EBL turns out to be counter-productive. This is because learning is involved with additional overhead, while we cannot profit much from the learned knowledge if we abort after the first answer set, hence the costs exceed the benefit.

Using the encoding Ω instead of Γ increases the efficiency in this case, because there is not only a large number of answer sets but also a large number of answer set candidates. Thus, a reusable encoding is very beneficial, even if we compute only one answer set.

Since in the evaluation of this program no unfounded sets are encountered, it is obvious that additional unfounded set checks over partial interpretations increase the overhead at no benefit; hence we do not discuss respective results.

	all answer sets						first answer set					
domain	explicit	+EBL	UFS Γ mol	UFS Γ mod	+EBL	Ω	explicit	+EBL	UFS Γ mol	UFS Γ mod	+EBL	Ω
5 (1)	300.00 (1)	300.00 (1)	0.33 (0)	0.32 (0)	0.09 (0)	0.07 (0)	54.02 (0)	53.80 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)
6 (1)	300.00 (1)	300.00 (1)	0.77 (0)	0.81 (0)	0.12 (0)	0.10 (0)	300.00 (1)	300.00 (1)	0.04 (0)	0.05 (0)	0.06 (0)	0.06 (0)
7 (1)	300.00 (1)	300.00 (1)	1.73 (0)	1.78 (0)	0.20 (0)	0.13 (0)	300.00 (1)	300.00 (1)	0.06 (0)	0.06 (0)	0.06 (0)	0.07 (0)
8 (1)	300.00 (1)	300.00 (1)	4.35 (0)	4.17 (0)	0.31 (0)	0.16 (0)	300.00 (1)	300.00 (1)	0.07 (0)	0.06 (0)	0.07 (0)	0.07 (0)
9 (1)	300.00 (1)	300.00 (1)	10.42 (0)	10.21 (0)	0.47 (0)	0.23 (0)	300.00 (1)	300.00 (1)	0.08 (0)	0.07 (0)	0.08 (0)	0.09 (0)
10 (1)	300.00 (1)	300.00 (1)	26.31 (0)	25.13 (0)	0.53 (0)	0.29 (0)	300.00 (1)	300.00 (1)	0.09 (0)	0.09 (0)	0.11 (0)	0.12 (0)
15 (1)	300.00 (1)	300.00 (1)	300.00 (1)	300.00 (1)	2.83 (0)	0.79 (0)	300.00 (1)	300.00 (1)	0.19 (0)	0.15 (0)	0.27 (0)	0.26 (0)
20 (1)	300.00 (1)	300.00 (1)	300.00 (1)	300.00 (1)	12.98 (0)	1.95 (0)	300.00 (1)	300.00 (1)	0.38 (0)	0.29 (0)	0.57 (0)	0.57 (0)
25 (1)	300.00 (1)	300.00 (1)	300.00 (1)	300.00 (1)	45.18 (0)	4.11 (0)	300.00 (1)	300.00 (1)	0.70 (0)	0.47 (0)	1.09 (0)	1.08 (0)

Table 1: Set Partitioning Benchmark Results

Note that the results are not comparable to those by Eiter et al. (2012a), because previous work focused on the computation of subset-minimal compatible sets and did not perform a minimality check wrt. the reduct, i.e., the semantics was different.

6.1.2 NONMONOTONIC MULTI-CONTEXT SYSTEMS

Nonmonotonic Multi-Context-Systems (MCSs) (Brewka & Eiter, 2007) are a generic formalism for aligning knowledge bases called *contexts*, which emerged from an approach by Ghidini and Giunchiglia (2001). The contexts are interlinked via bridge rules which enable belief exchange across contexts; the MCS semantics requires that local belief sets are compliant with the bridge rules. Such compliance can be impossible to achieve; that is, the MCS is inconsistent. To understand the reasons for the latter, Eiter et al. (2012b) defined *inconsistency explanations (IEs)* for a MCS, which can be computed with a HEX-program encoding. This encoding is based on *Saturation*, which is a general technique for solving Σ_2^p problems in disjunctive answer set programming (cf., Leone et al., 2006). Intuitively, a quantified Boolean formula (QBF) of the form $\exists\mathbf{X}\forall\mathbf{Y}\Phi(\mathbf{X},\mathbf{Y})$ is evaluated using this technique as follows. Disjunctions are used to guess whether a variable v is true or false. A 'spoil' atom is made true whenever Φ evaluates to true given truth assignments to $\mathbf{X}$ and $\mathbf{Y}$. Finally whenever 'spoil' is true, all literals over $\mathbf{Y}$ are set to true; this creates a unique assignment of the respective atoms. Given a guess on $\mathbf{X}$, its unique 'spoil' extension is an answer set if and only if all guesses of truth assignments $\mathbf{Y}$ make the spoil atom true and *saturate* the guess to become the unique extension—this holds due to the minimality condition on answer sets of the reduct (see Definition 3).

We use the HEX-encoding for computing IEs as a benchmark, as the saturation is rich in cycles through external atoms and disjunctive rule heads. External atoms in this benchmark evaluate semantics of contexts in the MCS (i.e., the local belief sets or models).

We use random instances of different MCS topologies, i.e., connection graphs of contexts, created with our MCS benchmark generator.[3] Note that the topologies are by their structure bound to certain system sizes (number of contexts), and that the difficulty of the instances varies among topologies; thus larger instances may have shorter runtimes. Our instances have up to 10 contexts, each consisting of a randomly generated consistent normal answer set program.

3. Described at http://www.kr.tuwien.ac.at/research/systems/dmcs/experiments.html, online available at https://dmcs.svn.sourceforge.net/svnroot/dmcs/dmcs/trunk

#ctx	explicit	+EBL	UFS Γ mol	UFS Γ mod	+EBL	+UFL	Ω
3 (6)	4.78 (0)	3.97 (0)	2.96 (0)	2.97 (0)	1.65 (0)	0.08 (0)	0.08 (0)
4 (10)	51.90 (1)	45.91 (1)	48.71 (1)	48.59 (1)	23.48 (0)	0.10 (0)	0.11 (0)
5 (8)	149.53 (3)	137.95 (3)	150.80 (3)	150.64 (3)	94.45 (1)	0.10 (0)	0.12 (0)
6 (6)	159.41 (3)	154.69 (3)	157.62 (3)	157.72 (3)	151.89 (3)	0.12 (0)	0.15 (0)
7 (12)	231.23 (9)	227.45 (9)	234.74 (9)	234.63 (9)	216.75 (8)	0.17 (0)	0.20 (0)
8 (5)	244.39 (4)	204.92 (3)	246.42 (4)	246.34 (4)	190.60 (3)	0.17 (0)	0.21 (0)
9 (8)	300.00 (8)	278.44 (7)	300.00 (8)	300.00 (8)	264.65 (6)	0.22 (0)	0.24 (0)
10 (11)	300.00 (11)	268.78 (9)	300.00 (11)	300.00 (11)	247.16 (8)	0.25 (0)	0.31 (0)

Table 2: Consistent MCSs Benchmark Results

#ctx	explicit	+EBL	all answer sets UFS Γ mol	UFS Γ mod	+EBL	+UFL	Ω
3 (9)	3.29 (0)	2.70 (0)	2.44 (0)	2.34 (0)	1.09 (0)	0.14 (0)	0.14 (0)
4 (14)	41.57 (1)	17.94 (0)	37.04 (1)	37.03 (1)	6.05 (0)	2.71 (0)	0.61 (0)
5 (11)	154.55 (5)	148.11 (5)	154.17 (5)	153.94 (5)	108.87 (2)	3.65 (0)	1.28 (0)
6 (18)	130.90 (7)	102.57 (6)	128.26 (7)	128.12 (7)	87.75 (4)	10.61 (0)	1.55 (0)
7 (13)	166.14 (5)	118.04 (5)	157.67 (5)	157.06 (5)	107.50 (4)	84.08 (2)	29.47 (0)
8 (6)	261.96 (5)	143.75 (2)	262.95 (5)	263.00 (5)	118.36 (2)	55.86 (1)	51.13 (1)
9 (14)	286.74 (13)	206.10 (9)	287.10 (12)	287.32 (12)	189.48 (8)	124.34 (5)	130.56 (6)
10 (12)	300.00 (12)	300.00 (12)	300.00 (12)	300.00 (12)	290.18 (11)	290.69 (11)	277.05 (11)

#ctx	explicit	+EBL	first answer set UFS Γ mol	UFS Γ mod	+EBL	+UFL	Ω
3 (9)	0.09 (0)	0.09 (0)	0.08 (0)	0.08 (0)	0.08 (0)	0.08 (0)	0.09 (0)
4 (14)	0.13 (0)	0.14 (0)	0.11 (0)	0.12 (0)	0.12 (0)	0.11 (0)	0.13 (0)
5 (11)	0.16 (0)	0.17 (0)	0.14 (0)	0.14 (0)	0.14 (0)	0.14 (0)	0.16 (0)
6 (18)	0.18 (0)	0.19 (0)	0.16 (0)	0.16 (0)	0.15 (0)	0.15 (0)	0.18 (0)
7 (13)	0.19 (0)	0.17 (0)	0.17 (0)	0.17 (0)	0.15 (0)	0.15 (0)	0.17 (0)
8 (6)	0.23 (0)	0.20 (0)	0.21 (0)	0.20 (0)	0.17 (0)	0.17 (0)	0.19 (0)
9 (14)	0.32 (0)	0.27 (0)	0.28 (0)	0.28 (0)	0.22 (0)	0.23 (0)	0.28 (0)
10 (12)	0.44 (0)	0.33 (0)	0.39 (0)	0.39 (0)	0.29 (0)	0.29 (0)	0.34 (0)

Table 3: Inconsistent MCSs Benchmark Results

The number of candidates for smaller models of the FLP reduct equals the number of unfounded set candidates as each unfounded set corresponds to a smaller model. However, as we stop the enumeration as soon as a smaller model respectively an unfounded set is found, the explicit and the UFS check may consider depending on the specific program and solver heuristics different numbers of interpretations. This explains why the UFS check is sometimes slightly slower than the explicit check. However, the delay between different UFS candidates was always smaller, which sometimes makes it faster even if it visits more candidates.

The results for consistent and inconsistent MCSs are shown in Table 2 and 3, respectively, where the number of instances of of each system size is given in parentheses. 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. Nonetheless, we can see that the UFS check based on Ω was often much faster than the explicit check (up to three orders of magnitude).

As consistent MCSs have no IEs and hence no answer sets, we need not distinguish for them between computing one or all answer sets. The effects of external behavior learning (Eiter et al., 2012a) and of unfounded set learning are clearly evident in the MCS benchmarks, for both computing the first and all answer sets. The UFS check profits more from EBL than the explicit check, further adding to its advantage. By activating UFL (which is not possible in the explicit check) we gain another significant speedup.

We now discuss the effects of the syntactic decision criterion and program decomposition. Due to saturation, the encoding contains cycles where nearly all cycles in the HEX-program contain at least one external atom. Therefore, the decision criterion can reduce the set of atoms, for which the UFS check needs to be performed, only by the atoms that are defined in the EDB. This does not yield significant efficiency improvements. However, the benchmark results for MCS instances confirm that the syntactic check is cheap and does not hurt performance. Over all 186 instances, the total runtime with decision criterion and program decomposition was 11,695 seconds compared to 11,702 seconds without, and the number of instance timeouts was the same.

If we use encoding Ω instead of Γ, we can observe another significant speedup for computing all IEs of inconsistent MCSs. This is because there usually exist many answer sets (often thousands), and thus a reusable encoding is very beneficial. In contrast, if we compute only the first answer set or the MCS is consistent (no answer set exists), then the check with the more involved encoding Ω is slightly slower; its reusability does not pay off if we abort after the first answer set.

In summary, we can observe that the encoding Ω leads to a significant performance gain over encoding Γ, while the decision criterion and decomposition do not help. In our next benchmark we will observe opposite effects.

6.1.3 ABSTRACT ARGUMENTATION

In this benchmark we compute ideal set extensions (Dung, Mancarella, & Toni, 2007) for randomly generated instances of abstract argumentation frameworks (AFs) (Dung, 1995) of different sizes. The problem of checking whether a given set of arguments is an ideal set of an AF is co-NP-complete (Dunne, 2009). In this benchmark we use a HEX encoding that mirrors this complexity: it guesses such a set and checks its ideality using the Saturation technique involving an external atom (see Appendix A.1).

Table 4 shows the results for different numbers of arguments, where each entry is the average of 30 benchmark instances. We compare the following configurations for both computing all and the first answer set.

In the first column we do an explicit minimality check without learning techniques. The second column shows that learning (EBL) leads to almost the same runtime results. This can be explained by the structure of the encoding, which does not allow for effectively reusing learned nogoods.

In the third column, we perform an UFS-based minimality check using our encoding Γ, but without applying the decision criterion and decomposition. We can observe that this is already a significant improvements compared to the explicit minimality check, illustrating the effectiveness of our new approach. Similar as in the MCS benchmark, the number of reduct model candidates is equal to the number of UFS candidates in most cases, but the UFS check again enumerates its candidates faster; this explains the observed speedup.

#args	all answer sets					
	explicit	+EBL	UFS Γ mol	UFS Γ mod	+EBL +UFL	Ω
1 (30)	0.06 (0)	0.06 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)
2 (30)	0.08 (0)	0.07 (0)	0.06 (0)	0.06 (0)	0.06 (0)	0.07 (0)
3 (30)	0.11 (0)	0.10 (0)	0.08 (0)	0.08 (0)	0.08 (0)	0.09 (0)
4 (30)	0.19 (0)	0.19 (0)	0.14 (0)	0.12 (0)	0.12 (0)	0.13 (0)
5 (30)	0.32 (0)	0.32 (0)	0.26 (0)	0.18 (0)	0.18 (0)	0.19 (0)
6 (30)	0.71 (0)	0.72 (0)	0.55 (0)	0.33 (0)	0.33 (0)	0.36 (0)
7 (30)	1.58 (0)	1.66 (0)	1.16 (0)	0.52 (0)	0.51 (0)	0.56 (0)
8 (30)	4.75 (0)	5.04 (0)	3.06 (0)	1.09 (0)	1.08 (0)	1.15 (0)
9 (30)	14.02 (0)	14.97 (0)	8.65 (0)	1.86 (0)	1.84 (0)	1.95 (0)
10 (30)	41.10 (0)	44.38 (0)	24.53 (0)	4.73 (0)	4.58 (0)	4.79 (0)
11 (30)	129.35 (1)	139.80 (2)	51.39 (0)	9.34 (0)	9.34 (0)	9.48 (0)
12 (30)	250.16 (12)	258.82 (17)	119.44 (0)	12.49 (0)	12.38 (0)	12.39 (0)
13 (30)	294.91 (27)	296.67 (27)	274.65 (19)	24.26 (0)	24.33 (0)	24.44 (0)
14 (30)	290.01 (29)	290.01 (29)	290.00 (29)	51.38 (3)	51.65 (3)	51.98 (3)
15 (30)	290.01 (29)	290.01 (29)	290.00 (29)	79.93 (3)	78.00 (3)	78.19 (3)
16 (30)	300.00 (30)	300.00 (30)	300.00 (30)	80.10 (4)	77.91 (4)	77.95 (4)
17 (30)	300.00 (30)	300.00 (30)	300.00 (30)	81.90 (5)	77.04 (5)	76.85 (5)
18 (30)	300.00 (30)	300.00 (30)	300.00 (30)	127.43 (8)	126.57 (8)	125.91 (8)
19 (30)	300.00 (30)	300.00 (30)	280.39 (28)	173.16 (13)	148.13 (10)	147.62 (10)
20 (30)	300.00 (30)	300.00 (30)	278.20 (27)	167.72 (12)	167.02 (12)	166.07 (12)

#args	first answer set					
	explicit	+EBL	UFS Γ mol	UFS Γ mod	+EBL +UFL	Ω
1 (30)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)
2 (30)	0.07 (0)	0.07 (0)	0.06 (0)	0.06 (0)	0.06 (0)	0.06 (0)
3 (30)	0.09 (0)	0.09 (0)	0.08 (0)	0.08 (0)	0.07 (0)	0.08 (0)
4 (30)	0.14 (0)	0.14 (0)	0.12 (0)	0.10 (0)	0.10 (0)	0.12 (0)
5 (30)	0.22 (0)	0.22 (0)	0.21 (0)	0.15 (0)	0.15 (0)	0.17 (0)
6 (30)	0.46 (0)	0.47 (0)	0.42 (0)	0.27 (0)	0.27 (0)	0.29 (0)
7 (30)	0.76 (0)	0.79 (0)	0.68 (0)	0.37 (0)	0.37 (0)	0.40 (0)
8 (30)	2.34 (0)	2.44 (0)	1.98 (0)	0.89 (0)	0.90 (0)	0.94 (0)
9 (30)	7.35 (0)	7.82 (0)	5.76 (0)	1.36 (0)	1.28 (0)	1.34 (0)
10 (30)	19.47 (0)	21.05 (0)	15.37 (0)	3.54 (0)	3.53 (0)	3.68 (0)
11 (30)	63.39 (1)	67.39 (1)	26.30 (0)	4.61 (0)	4.66 (0)	4.69 (0)
12 (30)	119.65 (4)	126.18 (4)	60.88 (0)	6.11 (0)	6.11 (0)	6.13 (0)
13 (30)	197.04 (14)	201.27 (15)	149.25 (3)	16.34 (0)	16.49 (0)	16.50 (0)
14 (30)	227.27 (22)	227.72 (22)	218.00 (17)	41.28 (2)	41.68 (2)	41.76 (2)
15 (30)	260.02 (26)	260.02 (26)	260.01 (26)	40.92 (2)	41.38 (2)	41.62 (2)
16 (30)	230.04 (23)	230.04 (23)	230.02 (23)	40.63 (3)	40.69 (3)	40.84 (3)
17 (30)	250.03 (25)	250.03 (25)	250.01 (25)	35.24 (2)	35.60 (2)	35.57 (2)
18 (30)	270.02 (27)	270.02 (27)	270.01 (27)	74.89 (5)	75.47 (5)	75.10 (5)
19 (30)	230.06 (23)	230.06 (23)	211.12 (21)	66.58 (4)	67.03 (4)	67.04 (4)
20 (30)	220.07 (22)	220.07 (22)	200.29 (20)	81.81 (5)	82.33 (5)	82.45 (5)

Table 4: Argumentation Benchmark Results

When we enable the decision criterion and program decomposition, we can observe a further speedup. This is because cycles in argumentation instances usually involve only small parts of the overall program; thus the UFS search can be significantly simplified by excluding large program parts. We further have observed that program decomposition without the decision criterion is counter-productive (not shown in the table), because a single UFS search over the whole program is replaced by many UFS searches over program components (without the decision criterion, no such check is excluded). This incurs more overhead.

In the fifth column we enable EBL and UFL, which leads to a small speedup in some cases. However, as already mentioned above, no effective reuse of learned nogoods is possible.

Switching the encoding from Γ to Ω leads to a small speedup in some cases, but is also counterproductive in others. This is because the programs in this benchmark have usually only very few compatible sets, and only few unfounded set checks need to be performed. Hence the lower initialization overhead of the encoding Ω does not influence the runtime dramatically. On the other hand, the higher complexity of the encoding Ω increases the runtime of small instances.

6.1.4 DEFAULT REASONING OVER DESCRIPTION LOGICS

A prominent instance of HEX-programs are DL-programs, which combine description logic ontologies with rules; they result by using a special external atom that is available as DL-plugin in DLVHEX. Via DL-programs, we obtain an encoding of terminological default reasoning over description logic ontologies in the approach of Baader and Hollunder (1995) into HEX-programs, in which defaults require cyclic dependencies over external atoms. However, as all such dependencies involve default negated atoms, we have no cycles according to Definition 12, which respects only positive dependencies. Hence, the decision criterion comes to the conclusion that no UFS check is required.

We used variants of the benchmarks presented by Eiter et al. (2012a), which query wines from an ontology and classify them as red or white wines, where a wine is assumed to be white unless the ontology explicitly entails the opposite. In this scenario, the decision criterion eliminates all unfounded set checks. However, as there is only one compatible set per instance, there would be only one unfounded set check anyway, hence the speedup due to the decision criterion is not significant. But the effect of the decision criterion can be increased by slightly modifying the scenario such that there are multiple compatible sets. This can be done, for instance, by nondeterministic default classifications, e.g., if a wine is not Italian, then it is either French or Spanish by default. Our experiments have shown that with a small number of compatible sets, the performance enhancement due to the decision criterion is marginal, but increases with larger numbers of compatible sets. For instance, for 243 compatible sets (and thus 243 unfounded set checks) we could observe a speedup from 13.59 to 12.19 seconds.

6.1.5 CONFORMANT PLANNING

In classical AI planning, a planning domain contains a description of actions with their preconditions and effects in the world. Finding a plan means to find a sequence of actions that reaches from a given initial state a state fulfilling a given goal condition. Conformant planning (Goldman & Boddy, 1996; Smith & Weld, 1998) is the same problem but where the initial state is only partially specified and/or the domain is nondeterministic, such that by executing the plan we reach the goal regardless of the action outcomes and the actual initial state.

We here experimented on a very simple conformant planning domain: two robots with a limited sensor range patrol an area, in which is an object at an unknown initial location. The goal is to find a sequence of movements of the two robots such that they detect the object in all cases. For experiments we used an encoding which realizes conformant planning using Saturation (see above) and contains an external atom for computing whether the patrol robots detect object (cf. Appendix A.2). In general, deciding the existence of a short (polynomial length bounded) conformant plan is Σ_3^p-

map size	plan length	all answer sets						
		explicit	UFS Γ mol	UFS Γ mod	+EBL	+UFL	Ω -EBL-UFL	Ω +EBL+UFL
3×4 (10)	1	7.10 (0)	0.12 (0)	0.11 (0)	0.11 (0)	0.12 (0)	0.12 (0)	0.14 (0)
4×4 (10)	1	10.66 (0)	0.16 (0)	0.15 (0)	0.15 (0)	0.15 (0)	0.15 (0)	0.18 (0)
5×4 (10)	1	10.69 (0)	0.15 (0)	0.15 (0)	0.14 (0)	0.14 (0)	0.13 (0)	0.15 (0)
6×4 (10)	2	206.45 (2)	1.98 (0)	1.38 (0)	1.67 (0)	1.69 (0)	1.09 (0)	1.35 (0)
7×4 (10)	2	258.82 (5)	2.85 (0)	1.79 (0)	2.44 (0)	2.43 (0)	1.50 (0)	1.84 (0)
8×4 (10)	3	300.00 (10)	36.80 (0)	16.41 (0)	40.94 (0)	40.99 (0)	10.42 (0)	13.88 (0)
9×4 (10)	3	300.00 (10)	43.20 (0)	19.53 (0)	78.11 (0)	77.10 (0)	13.91 (0)	19.62 (0)
10×4 (10)	4	300.00 (10)	300.00 (10)	274.53 (5)	300.00 (10)	300.00 (10)	203.70 (2)	252.31 (5)
11×4 (10)	4	300.00 (10)	299.76 (9)	239.61 (5)	300.00 (10)	300.00 (10)	174.86 (2)	209.41 (3)
12×4 (10)	5	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)
13×4 (10)	5	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)
14×4 (10)	6	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)
15×4 (10)	6	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)
16×4 (10)	7	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)

map size	plan length	first answer set						
		explicit	UFS Γ mol	UFS Γ mod	+EBL	+UFL	Ω -EBL-UFL	Ω +EBL+UFL
3×4 (10)	1	0.89 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.06 (0)	0.06 (0)
4×4 (10)	1	1.36 (0)	0.06 (0)	0.05 (0)	0.05 (0)	0.06 (0)	0.06 (0)	0.06 (0)
5×4 (10)	1	2.23 (0)	0.06 (0)	0.07 (0)	0.06 (0)	0.06 (0)	0.07 (0)	0.07 (0)
6×4 (10)	2	7.21 (0)	0.22 (0)	0.15 (0)	0.14 (0)	0.14 (0)	0.12 (0)	0.13 (0)
7×4 (10)	2	17.39 (0)	0.34 (0)	0.22 (0)	0.21 (0)	0.20 (0)	0.17 (0)	0.18 (0)
8×4 (10)	3	139.26 (1)	6.07 (0)	2.73 (0)	2.73 (0)	2.69 (0)	1.45 (0)	1.78 (0)
9×4 (10)	3	150.50 (3)	3.24 (0)	1.47 (0)	1.69 (0)	1.70 (0)	0.89 (0)	1.16 (0)
10×4 (10)	4	255.89 (7)	92.19 (2)	47.58 (0)	82.84 (2)	82.52 (2)	24.23 (0)	31.36 (0)
11×4 (10)	4	300.00 (10)	97.11 (2)	39.99 (0)	84.08 (1)	83.85 (1)	19.53 (0)	25.85 (0)
12×4 (10)	5	287.76 (9)	198.75 (5)	143.52 (4)	184.81 (5)	184.78 (5)	131.46 (4)	136.64 (4)
13×4 (10)	5	300.00 (10)	287.07 (9)	211.97 (5)	277.79 (9)	277.71 (9)	165.64 (4)	185.84 (4)
14×4 (10)	6	300.00 (10)	300.00 (10)	244.33 (7)	300.00 (10)	300.00 (10)	213.89 (5)	232.85 (6)
15×4 (10)	6	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	285.36 (9)	296.10 (9)
16×4 (10)	7	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)

Table 5: Conformant Planning Benchmark Results

complete, see Turner (2002), but if action executability is decidable in polynomial time, the problem is in Σ_2^p; our example domain enjoys the property.

Our results are displayed in Table 5, which shows averages over 10 instances per size. The instances consist of $n{\times}4$ grids with $n \in \{3,\ldots,16\}$, the plan length required for finding a solution increases with larger instance sizes. (This is because the number of robots does not increase while the two robots must still cover the whole area.) Instances were generated by randomly placing robots in opposite quarters of the map.

As expected we observe that the explicit FLP check performs worst, followed by the monolithic UFS check with Γ encoding, and the modular UFS Γ encoding; the UFS Ω encoding (without external behavior nor unfounded set learning) performs best. External behavior learning (EBL) and unfounded set learning (UFL) do not improve the performance, on the contrary it increases the run times significantly for the modular UFS Γ check and slightly for the Ω check. EBL does not change times significantly for the explicit check, therefore we omit results for explicit +EBL.

By looking into profiling information and at the domain we found the reasons: (a) the external atoms depend on a large part of the interpretation (locations of all robots) so EBL cannot cut away significant portions of the search space; (b) evaluating the external atom takes a negligible amount of time, so beneficial effects of EBL will be outweighed by is computational overhead. For UFL we observed that the benchmark instances contain only few unfounded sets (pruning less than half of the answer set candidates) and thus UFL cannot improve performance given the overhead it incurs. We conclude that in some scenarios, using EBL and UFL can reduce efficiency.

As in the Ω check the UFS check encoding is constructed only once, the overhead for EBL and UFL observed with the Γ encoding no longer has such a big impact. Nevertheless Ω without learning slightly outperforms Ω with UFL and EBL. An analysis of the number of UFS checks and the number of external atom evaluations (not shown in table) revealed that UFL and EBL decreases (i) the number of unfounded set checks needed and (ii) the number of external atoms evaluated (in one $9{\times}4$ instance, from 5132 to 3341). Thus if external computations would be more costly, the positive effects of UFL and EBL on Ω would outweigh their computational overhead and it would be beneficial to activate UFL and EBL in this domain.

Interestingly, for small map sizes we see that with the Ω encoding, the $3{\times}4$ instances actually seem to be harder to solve than the larger $4{\times}4$ and $5{\times}4$ instances. This is because all these instances require plan length 1 to find a solution; so the larger instances are more constrained than the smaller instances because there robots have less freedom to move around while still detecting all objects. Hence, for $5{\times}4$ maps the solver finds solutions faster than for $4{\times}4$ maps.

The conclusion from this benchmark is that depending on the computational task in external atoms, UFL and EBL can be beneficial or harmful for efficiency of reasoning.

6.2 Summary

Our experiments have shown that the learning technique EBL developed by Eiter et al. (2012a) and the techniques introduced in this paper lead to significant performance improvements in most cases, with few exceptions for specific benchmarks. The effects of external behavior learning (EBL) are clearly evident both for the explicit minimality check and for the unfounded set-based check, but are even more prominent with the latter. Independently of whether EBL is used or not, unfounded set checking pushes the efficiency of HEX-program evaluation compared to explicit minimality checking. Moreover, it allows for learning of additional nogoods, which is also advantageous in most of our benchmarks. Regarding the two problem encodings, the benchmarks show that the UFS check is usually faster with the Ω encoding than with the Γ encoding, however the former UFS check involves more initialization overhead, which might be counterproductive for small programs.

The decision criterion may lead to an additional speedup and does not introduce notable overhead, thus it can always be activated. Finally, program decomposition often leads to an additional performance gain, but should only be used in combination with the decision criterion because otherwise a single UFS check is replaced by multiple UFS checks, which involves more overhead.

7. Discussion

We now discuss related work and outline some possible starting points for extensions.

7.1 Related Work

Constraint answer set solving (Gebser, Ostrowski, & Schaub, 2009) can be seen as a special case of HEX-programs. It extends ASP by dedicated constraint atoms in rule bodies (comparisons of numeric constraint variables) that allow for a bidirectional exchange of information between the logic program and the constraint solver. While the constraint solver is a concrete instance of an external source, HEX-programs support arbitrary external sources. The idea of *external behavior learning* (EBL), introduced by Eiter et al. (2012a) is further related to the work of Drescher and Walsh (2012) since both approaches generate nogoods on-the-fly.

Besides grounding, one of the main differences between ASP and SAT solving is *foundedness*, i.e., the truth of each atom in an answer set is justified by a non-circular derivation from rules and facts. To account for circularity, the notion of unfounded set has been introduced by van Gelder et al. (1991) for defining the well-founded semantics of logic programs with negation, by constructing the least fixpoint of a monotone operator on partial assignments; the total fixpoints of this operator correspond to the stable models of the logic program. This actually allows to give a characterization of the stable models in terms of unfounded sets. In fact, unfounded set checking turned out to be a fruitful model-based approach in ASP solving, which has an equally successful syntactic counterpart known as *loop formulas* (Lin & Zhao, 2004; Lee & Lifschitz, 2003; Lee, 2005). Different kinds of unfounded set checks with different complexities have been developed for various program classes.

The computation of answer sets by a model generate and test approach, which is pursued by many ASP solvers, requires that a form of minimality check or unfounded set check is carried out already for ordinary logic programs (i.e., in absence of external atoms). However, for normal program this test is tractable and it is frequently realized using *source pointers* (Simons et al., 2002). Intuitively, the reasoner stores for each atom a pointer to a rule which possibly supports this atom. The list of source pointers is updated during propagation. If at some point there is no supporting rule for an atom, then it can conclude that this atom must be false.

In the context of ASP, the notion of unfounded set has been explicitly formulated and extended to disjunctive logic programs by Leone et al. (1997), who proved that the stable models of a disjunctive logic program are its models that are unfounded-free. This results was the basis for the architecture of the DLV solver, which generates answer set candidates that are checked for unfounded-freeness. This test, which like answer set checking for disjunctive answer set programs, is co-NP-complete (Faber, 2005), has been reduced by Koch et al. (2003) to unsatisfiability testing of a SAT instance. This approach has been later extended to conflict-driven learning and unfounded set checking by Drescher et al. (2008), where two instances for the CLASP solver, an ASP instance and a SAT instance, are used to generate and check answer set candidates, respectively. In parallel to our work, this technique was recently refined by exploiting assumptions such that the encoding of the unfounded set search does not need to be adopted to the current assignment (Gebser, Kaufmann, & Schaub, 2013). This is related to our uniform encoding of the unfounded set search, but still restricted to disjunctive ASP *without* external sources. For HEX-programs, the unfounded set search needs to respect the semantics of external atoms and is thus a more general problem. Alviano, Calimeri, Faber, Leone, and Perri (2011) consider normal logic programs with monotone and antimonotone aggregate atoms, and defined unfounded sets for such programs. Based on this, they extended the well-founded semantics of Van Gelder et al. (1991), which—although closely related—is weaker than the general FLP semantics (Faber et al., 2011).

We now considered unfounded set checking in the presence of external sources and for FLP answer sets, for which the results by Drescher et al. (2008) do not immediately carry over. Indeed, already for ground Horn programs, the presence of nonmonotonic external atoms that are decidable in polynomial time makes unfoundedness checking intractable (more precisely, co-NP-complete), such that deciding the existence of an FLP answer set is a Σ_2^p-complete problem in the ground case.

For computationally harder external atoms, the complexity may increase relative to an oracle for the external function (see Faber, 2005). However, the results from this paper do still apply in such cases.

Drescher et al. (2008) employed also a splitting technique, which goes back to the work of Leone et al. (1997); it is related to our program decomposition, but works for ASP programs without external sources only. Note that our notion of splitting is different from the well-known splitting sets by Lifschitz and Turner (1994), as we consider only positive dependencies for ordinary atoms. While we consider e-cycles, which are specific for HEX-programs, the interest of Drescher et al. (2008) was with head-cycles, which may arise with disjunctive rule heads. In fact, our approach may be regarded as an extension of the one of Drescher et al., since the evaluation of $\hat{\Pi}$ follows their principles of performing UFS checks in case of head-cycles.

For the evaluation of the FLP semantics, an unfounded set check or explicit FLP check is instrumental. we just mention that other semantics of HEX-programs may not involve such a step which is intractable in general (as follows from Leone et al., 2006, already for ground Horn programs with nonmonotonic external atoms that are decidable in polynomial time). For instance, Shen (2011) and Shen and Wang (2011) present a well-justified semantics where unfounded set checking is essentially replaced by a fixpoint iteration which, intuitively, tests if a model candidate reproduces itself but excludes circular justifications. However, the complexity of answer set computation does not decrease by this approach in general, and in particular deciding well-justified answer set existence for ground Horn programs with nonmonotonic external atoms that are decidable in polynomial time is Σ_2^p-complete, and thus as hard as deciding the existence of answer sets as in Definition 3.

7.2 Extensions

We have designed our unfounded set check as a postcondition test; that is like the explicit FLP check, it is carried out only after a complete assignment has been generated as an answer set candidate. However, in certain cases it might be obvious that a partial interpretation (in which some truth values are open) can be extended to an answer set, because the existence of an unfounded set is guaranteed for any extension to a complete assignment. One can then backtrack earlier, which intuitively leads to a saving for certain classes of instances.

Exploring this idea, we have generalized our framework with a control component which decides, based on a heuristics, when an unfounded set check is carried out; in the standard setting, this is whenever an assignment in the model generation is complete (i.e., a complete assignment is given).

A UFS check on partial assignments, that is sound with respect to any extension to a complete assignment, is possible if the ASP solver has finished unit propagation over a maximal subset of the program such that the interpretation is already complete on it, and all guessed values of external atom replacements are correct. We thus used this criterion, which is easy to test, for a greedy heuristics to issue UFS checks in our prototype system.

However, in contrast to our initial expectation, we found that for all our benchmarks the UFS check wrt. partial assignments was not productive. A closer look reveals that this is essentially because nogood learning from unfounded sets (UFL) effectively avoids the reconstruction of the same unfounded set anyway. It therefore rarely happens that UFS checking wrt. a partial interpretation identifies an unfounded set earlier than UFS checking wrt. complete interpretations. Therefore, we believe that UFS checking wrt. a partial interpretation rarely identifies an unfounded set earlier than UFS checking wrt. complete assignments. As UFS checking for HEX-programs involves the evalu-

#args	all answer sets Ω +EBL+UFL	Ω partial (periodic) +EBL+UFL	Ω partial (max) +EBL+UFL	first answer set Ω +EBL+UFL	Ω partial (periodic) +EBL+UFL	Ω partial (max) +EBL+UFL
5 (1)	0.07 (0)	0.09 (0)	0.11 (0)	0.05 (0)	0.06 (0)	0.07 (0)
6 (1)	0.10 (0)	0.13 (0)	0.15 (0)	0.06 (0)	0.07 (0)	0.09 (0)
7 (1)	0.13 (0)	0.15 (0)	0.19 (0)	0.07 (0)	0.08 (0)	0.11 (0)
8 (1)	0.18 (0)	0.20 (0)	0.26 (0)	0.08 (0)	0.10 (0)	0.14 (0)
9 (1)	0.24 (0)	0.26 (0)	0.35 (0)	0.09 (0)	0.12 (0)	0.17 (0)
10 (1)	0.29 (0)	0.33 (0)	0.47 (0)	0.11 (0)	0.14 (0)	0.21 (0)
15 (1)	0.80 (0)	0.96 (0)	1.61 (0)	0.24 (0)	0.38 (0)	0.73 (0)
20 (1)	1.96 (0)	2.46 (0)	4.92 (0)	0.51 (0)	0.97 (0)	2.30 (0)
25 (1)	4.15 (0)	5.52 (0)	11.25 (0)	0.97 (0)	1.98 (0)	4.50 (0)

Table 6: Set Partitioning with UFS Checking over Partial Assignments

#ctx	Ω +EBL+UFL	Ω partial (periodically) +EBL+UFL	Ω partial (max) +EBL+UFL
3 (6)	0.08 (0)	0.09 (0)	0.10 (0)
4 (10)	0.11 (0)	0.11 (0)	0.12 (0)
5 (8)	0.12 (0)	0.12 (0)	0.13 (0)
6 (6)	0.15 (0)	0.15 (0)	0.16 (0)
7 (12)	0.20 (0)	0.20 (0)	0.21 (0)
8 (5)	0.21 (0)	0.21 (0)	0.22 (0)
9 (8)	0.24 (0)	0.24 (0)	0.27 (0)
10 (11)	0.31 (0)	0.31 (0)	0.32 (0)

Table 7: Consistent MCSs Benchmark Results with UFS Checking over Partial Assignments

#ctx	all answer sets Ω +EBL+UFL	Ω partial (periodic) +EBL+UFL	Ω partial (max) +EBL+UFL	first answer set Ω +EBL+UFL	Ω partial (periodic) +EBL+UFL	Ω partial (max) +EBL+UFL
3 (9)	0.14 (0)	0.13 (0)	0.16 (0)	0.09 (0)	0.09 (0)	0.10 (0)
4 (14)	0.61 (0)	0.64 (0)	0.88 (0)	0.13 (0)	0.13 (0)	0.14 (0)
5 (11)	1.28 (0)	1.36 (0)	1.81 (0)	0.16 (0)	0.16 (0)	0.17 (0)
6 (18)	1.55 (0)	1.67 (0)	2.49 (0)	0.18 (0)	0.18 (0)	0.18 (0)
7 (13)	29.47 (0)	31.54 (0)	44.90 (1)	0.17 (0)	0.17 (0)	0.18 (0)
8 (6)	51.13 (1)	51.22 (1)	51.66 (1)	0.19 (0)	0.20 (0)	0.21 (0)
9 (14)	130.56 (6)	130.99 (6)	133.84 (6)	0.28 (0)	0.27 (0)	0.28 (0)
10 (12)	277.05 (11)	277.20 (11)	278.21 (11)	0.34 (0)	0.35 (0)	0.36 (0)

Table 8: Inconsistent MCSs Benchmark Results with UFS Checking over Partial Assignments

ation of external sources and compatibility testing, this easily leads to costs that are higher than the potential savings. A more detailed analysis requires further studies; since the results do not seem to be promising, we leave this for possible future work.

Table 6–10 show the benchmark results if UFS checking wrt. partial assignments is enabled when computing all or the first answer set only.

The first column shows the runtime with UFS checking wrt. complete interpretations only, using encoding Ω, EBL and UFL (equivalent to the last column in the tables in Section 6). The second column shows the results with UFS checking wrt. partial assignments, using a heuristics which performs the UFS check periodically (*periodic*). The third column shows the runtimes if the UFS check is always performed, if no other propagation technique can derive further truth values (*max*).

#args	all answer sets Ω +EBL+UFL	Ω partial (periodic) +EBL+UFL	Ω partial (max) +EBL+UFL	first answer set Ω +EBL+UFL	Ω partial (periodic) +EBL+UFL	Ω partial (max) +EBL+UFL
1 (30)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)	0.05 (0)
2 (30)	0.07 (0)	0.06 (0)	0.07 (0)	0.06 (0)	0.07 (0)	0.07 (0)
3 (30)	0.09 (0)	0.09 (0)	0.10 (0)	0.08 (0)	0.08 (0)	0.09 (0)
4 (30)	0.13 (0)	0.14 (0)	0.16 (0)	0.12 (0)	0.12 (0)	0.14 (0)
5 (30)	0.19 (0)	0.20 (0)	0.22 (0)	0.17 (0)	0.16 (0)	0.18 (0)
6 (30)	0.36 (0)	0.36 (0)	0.39 (0)	0.29 (0)	0.29 (0)	0.31 (0)
7 (30)	0.56 (0)	0.56 (0)	0.59 (0)	0.40 (0)	0.40 (0)	0.42 (0)
8 (30)	1.15 (0)	1.15 (0)	1.19 (0)	0.94 (0)	0.94 (0)	0.96 (0)
9 (30)	1.95 (0)	1.94 (0)	2.01 (0)	1.34 (0)	1.35 (0)	1.39 (0)
10 (30)	4.79 (0)	4.80 (0)	4.96 (0)	3.68 (0)	3.67 (0)	3.75 (0)
11 (30)	9.48 (0)	9.49 (0)	9.71 (0)	4.69 (0)	4.71 (0)	4.74 (0)
12 (30)	12.39 (0)	12.42 (0)	12.79 (0)	6.13 (0)	6.11 (0)	6.23 (0)
13 (30)	24.44 (0)	24.45 (0)	25.32 (0)	16.50 (0)	16.46 (0)	16.80 (0)
14 (30)	51.98 (3)	52.03 (3)	52.57 (3)	41.76 (2)	41.80 (3)	41.98 (3)
15 (30)	78.19 (3)	78.14 (3)	79.81 (3)	41.62 (2)	41.53 (2)	42.02 (2)
16 (30)	77.95 (4)	77.99 (4)	79.52 (4)	40.84 (3)	40.79 (3)	41.04 (3)
17 (30)	76.85 (5)	76.86 (5)	77.82 (5)	35.57 (2)	35.53 (2)	35.58 (2)
18 (30)	125.91 (8)	126.17 (8)	128.83 (8)	75.10 (5)	75.32 (5)	75.37 (5)
19 (30)	147.62 (10)	147.51 (10)	149.62 (10)	67.04 (4)	66.88 (4)	67.59 (4)
20 (30)	166.07 (12)	165.96 (12)	168.53 (12)	82.45 (5)	82.27 (5)	82.90 (5)

Table 9: Argumentation with UFS Checking over Partial Assignments

It can be observed that UFS checking wrt. partial assignments does not lead to a further speedup in any case. Quite the contrary, some instances have significantly higher runtimes with more frequent unfounded set checks. This is best visible in the set partitioning benchmark (Table 6), when computing all explanations for inconsistent MCSs with 5, 6 or 7 contexts (Table 9), and when computing all answer sets in the conformant planning benchmark (Table 10). In the set partitioning benchmark the effects are especially significant, which is as expected because every compatible set is unfounded-free. Thus, additional UFS checks are always counterproductive. In the consistent multi-context systems, reasoning is fast anyway, thus the frequency of UFS checking has no significant impact (Table 7). In the argumentation benchmark we can also observe a slight slowdown by more frequent UFS checking, although it is less dramatic than in the other benchmarks because the other propagation methods are applicable more frequently and thus fewer UFS checks are performed even with setting *max* (Table 9).

On the other hand, for ASP solving (where no such extra costs incur), UFS checks over partial interpretations may still be beneficial, as reported by Gebser et al. (2013). In conclusion, UFS checks on partial assignments of HEX-programs will require tailored heuristics that not only take the structure of the program, but also domain-specific knowledge into account, which remains for future work.

8. Conclusion

HEX-programs are an expressive extension of non-monotonic logic programs with access to external information via external atoms; supported by a plugin architecture they can be fruitfully deployed for a range of applications. External atoms however make the efficient evaluation of HEX-programs a challenging task, and in particular to compute answer sets of a HEX-program Π, which are the models $\mathbf{A}$ of Π that are subset-minimal models of its FLP-reduct $f\Pi^{\mathbf{A}}$ (which keeps all rules whose

	#ctx	all answer sets Ω +EBL+UFL	Ω partial (periodic) +EBL+UFL	Ω partial (max) +EBL+UFL	first answer set Ω +EBL+UFL	Ω partial (periodic) +EBL+UFL	Ω partial (max) +EBL+UFL
3×4 (10)	1	0.14 (0)	0.14 (0)	0.16 (0)	0.06 (0)	0.06 (0)	0.08 (0)
4×4 (10)	1	0.18 (0)	0.17 (0)	0.20 (0)	0.06 (0)	0.06 (0)	0.08 (0)
5×4 (10)	1	0.15 (0)	0.15 (0)	0.18 (0)	0.07 (0)	0.07 (0)	0.09 (0)
6×4 (10)	2	1.35 (0)	1.35 (0)	1.48 (0)	0.13 (0)	0.13 (0)	0.15 (0)
7×4 (10)	2	1.84 (0)	1.83 (0)	2.03 (0)	0.18 (0)	0.18 (0)	0.21 (0)
8×4 (10)	3	13.88 (0)	14.23 (0)	17.27 (0)	1.78 (0)	1.86 (0)	2.36 (0)
9×4 (10)	3	19.62 (0)	19.96 (0)	23.75 (0)	1.16 (0)	1.18 (0)	1.42 (0)
10×4 (10)	4	252.31 (5)	257.18 (5)	289.20 (7)	31.36 (0)	33.40 (0)	49.36 (0)
11×4 (10)	4	209.41 (3)	214.72 (3)	244.84 (5)	25.85 (0)	27.15 (0)	37.64 (0)
12×4 (10)	5	300.00 (10)	300.00 (10)	300.00 (10)	136.64 (4)	137.22 (4)	142.74 (4)
13×4 (10)	5	300.00 (10)	300.00 (10)	300.00 (10)	185.84 (4)	188.73 (4)	209.44 (4)
14×4 (10)	6	300.00 (10)	300.00 (10)	300.00 (10)	232.85 (6)	235.06 (7)	243.73 (7)
15×4 (10)	6	300.00 (10)	300.00 (10)	300.00 (10)	296.10 (9)	297.42 (9)	300.00 (10)
16×4 (10)	7	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)	300.00 (10)

Table 10: Conformant Planning with UFS Checking over Partial Assignments

bodies are satisfied). To improve on this expensive test (which was customary in implementations so far), we have presented an alternative test based on unfounded sets that we obtain by adapting the notion of unfounded set for aggregates by Faber (2005) to external atoms. Also Alviano et al. (2011) use a related notion of unfounded sets for programs with aggregates, but restrict the discussion to monotonic and antimonotonic aggregates. We have realized unfounded set (UFS) checking by a transformation to SAT solving, where the satisfying assignments of a constructed CNF generate candidate unfounded sets, which in turn are subject to a (rather simple) postcheck that takes external atom evaluation into account.

In particular, we have provided two SAT encodings for UFS checking: the conceptually simple encoding $\Gamma_{\Pi}^{\mathbf{A}}$, which needs initialization for every UFS check, and the advanced encoding Ω_{Π}, which can be reused for all UFS checks. To further boost performance, we have shown how to learn from unfounded sets by deriving nogoods, i.e., constraints (possibly involving also external atoms) which guide future search in model generation and help to avoid that unfounded sets are regenerated. In further elaboration, we have refined the basic approach by suitable program splitting, such that the UFS check can be carried out independently on program components, cutting down the complexity. Furthermore, we have presented a syntactic criterion that allows us to decide efficiently whether the UFS check can be safely skipped for a component or the whole program, exploiting that the answer set candidates from the model search have only special unfounded sets that involve cyclic input to external atoms; for HEX-programs in simple applications, this is usually not the case.

The experimental evaluation of our new approach, which considered different combinations of the techniques and comprised problems from various domains including multi-context systems, abstract argumentation, default reasoning over ontologies, and conformant planning, where HEX-programs serve for easy-cut declarative problem solving, has shown that it is more efficient than the traditional minimal model check; it can lead to exponential gains and yields often drastic performance improvements, while it is not slower (except in very few cases by a marginal amount). Furthermore, the reusable encoding Ω_{Π} turned out to be beneficial for programs that require many unfounded set checks, which includes all programs that have many answer sets.

8.1 Future work

An issue for further improvement of the approach in this paper are other heuristics for UFS checking over partial assignments. While the natural heuristics that we considered are counterproductive, others might lead to additional speedup in some cases. However, this may require to incorporate domain-specific knowledge about external atoms; currently this is only done in the learning algorithms but not in the heuristics. In the same line, one might consider developing a heuristics for dynamically choosing between the UFS search encodings that we presented, and to study heuristics for guiding the unfounded set search, i.e., for variable selection by the SAT solver. Currently, our implementation applies the same heuristics for the unfounded set search as in the model generation process. However, our experimental comparison with the explicit minimality check in terms of the considered candidate sets suggests that there is room for improvement by employing suitable choices. Developing appropriate heuristics and validating their effectiveness on candidate set enumeration remains to be explored. Finally, an obvious issue are other criteria that allow to skip the UFS check or simplify it while they can be efficiently tested.

Acknowledgments

Preliminary results of this work have been presented at the 13th European Conference on Logics in Artificial Intelligence (JELIA 2012), September 26-28, 2012, Toulouse, France (Eiter, Fink, Krennwallner, Redl, & Schüller, 2012c), and the 5th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2012), September 4, 2012, Budapest, Hungary (Eiter, Fink, Krennwallner, Redl, & Schüller, 2012b). We are grateful to the anonymous reviewers for their helpful and constructive comments. This work was supported by the Austrian Science Fund (FWF) via the projects P20840, P20841, P24090, and by the Vienna Science and Technology Fund (WWTF) project ICT08-020. Peter Schüller was supported by TUBITAK Fellowship 2216.

Appendix A. Benchmark Encodings

In this appendix, we give details to some of the benchmark encodings (those which are not described in the references). We note that these encodings have not been developed and tuned for good performance and serve merely for an experimental comparison of the various FLP check realizations. Benchmark encodings and HEX-plugins are publicly available at https://github.com/hexhex/benchmarks.

A.1 Abstract Argumentation

The *Abstract Argumentation* benchmark results in Section 6.1 were obtained using the following encoding, which is derived from encodings for admissible and preferred set extensions of an argumentation framework (A, att) described by Egly, Gaggl, and Woltran (2010).

Input instances of this benchmark are defined over a set A of arguments encoded as facts $arg(a)$ for each $a \in A$ and a set att of attacks between arguments, encoded as facts $att(a, b)$ for some $(a, b) \in A \times A$. The encoding consists of the following rules where $x, y, z \in A$; very similar encodings are explained in detail by Egly et al. (2010) (but without the use of external atoms).

We define defeat from attacks.

$$defeat(x, y) \leftarrow att(x, y).$$

We guess a set $S \subseteq A$ using predicates in_S and out_S.

$$in_S(x) \leftarrow \text{not } out_S(x), arg(x). \qquad out_S(x) \leftarrow \text{not } in_S(x), arg(x).$$

We require that all arguments in S are conflict-free and defended from S.

$$\begin{aligned} &\leftarrow in_S(x), in_S(y), defeat(x,y).\\ defeated(x) &\leftarrow in_S(y), defeat(y,x).\\ notDefended(x) &\leftarrow defeat(y,x), \text{not } defeated(y).\\ &\leftarrow in_S(x), notDefended(x). \end{aligned}$$

For saturation we define a linear order on arguments, including infimum and supremum.

$$\begin{aligned} lt(x,y) &\leftarrow arg(x), arg(y). && (x < y)\\ nsucc(x,z) &\leftarrow lt(x,y), lt(y,z).\\ succ(x,y) &\leftarrow lt(x,y), \text{not } nsucc(x,y).\\ ninf(x) &\leftarrow lt(y,x). & nsup(x) &\leftarrow lt(x,y).\\ inf(x) &\leftarrow \text{not } ninf(x), arg(x). & sup(x) &\leftarrow \text{not } nsup(x), arg(x). \end{aligned}$$

We perform a guess over a set $T \subseteq A$ using a disjunction.

$$in_T(x) \vee out_T(x) \leftarrow arg(x).$$

We check each argument of T whether it is in S and spoil the answer set if $S \subseteq T$.

$$\begin{aligned} sInT_{upto}(y) &\leftarrow inf(y), in_S(y), in_T(y).\\ sInT_{upto}(y) &\leftarrow inf(y), out_S(y).\\ sInT_{upto}(y) &\leftarrow succ(z,y), in_S(y), in_T(y), sInT_{upto}(z).\\ sInT_{upto}(y) &\leftarrow succ(z,y), out_S(y), sInT_{upto}(z).\\ sInT &\leftarrow sup(y), sInT_{upto}(y).\\ spoil &\leftarrow sInT. \end{aligned}$$

We also spoil the answer set if T is not a preferred extension, determined by an external atom with the semantic function $f_{\&argSemExt}$ such that $f_{\&argSemExt}(\mathbf{A}, pref, arg, att, in_T, unused, spoil) = 1$ iff $\mathbf{F}spoil \in \mathbf{A}$ or the extension of predicate in_T is a preferred set extension of the argumentation framework specified by the extension of predicates arg and att. Internally, the external atom uses another ASP program to compute the semantics. This check is performed using an ASP encoding for preferred extensions from Egly et al. (2010).

$$\begin{aligned} tIsNotPref &\leftarrow \&argSemExt[pref, arg, att, in_T, unused, spoil]().\\ spoil &\leftarrow tIsNotPref. \end{aligned}$$

Note that the parameters $pref$ and $unused$ support more general functionalities of $f_{\&argSemExt}$ which are not relevant for this benchmark. We create a unique answer set whenever $spoil$ is true and require that only spoiled answer sets are returned.

$$\begin{aligned} in_T(x) &\leftarrow spoil, arg(x). & out_T(x) &\leftarrow spoil, arg(x).\\ sInT &\leftarrow spoil. & tIsNotPref &\leftarrow spoil. & &\leftarrow \text{not } spoil. \end{aligned}$$

Given an instance encoded as above, an answer set to the above program exists iff there exists an ideal set extension of the given argumentation framework.

A.2 Conformant Planning

The *Conformant Planning* benchmark results in Section 6.1 were obtained using the following encoding.

Input instances of this benchmark are defined over a set R of robots, sets X and Y of valid x and y coordinates of the environment, and a maximum plan length l; an instance contains for each robot $r \in R$ the initial position (x, y) as facts $robo_X(r, x, 0)$ and $robo_Y(r, y, 0)$. The encoding consists of the following rules where, unless stated otherwise, $0 \leq t < l$, $r \in R$, $x \in X$, $y \in Y$.

For each robot we generate four possible moves in the environment.

$$
\begin{aligned}
&move(r,x,y+1,t) \vee \overline{move}(r,x,y+1,t) \leftarrow robo_X(r,x,t), robo_Y(r,y,t). && (y+1 \in Y)\\
&move(r,x,y-1,t) \vee \overline{move}(r,x,y-1,t) \leftarrow robo_X(r,x,t), robo_Y(r,y,t). && (y-1 \in Y)\\
&move(r,x+1,y,t) \vee \overline{move}(r,x+1,y,t) \leftarrow robo_X(r,x,t), robo_Y(r,y,t). && (x+1 \in X)\\
&move(r,x-1,y,t) \vee \overline{move}(r,x-1,y,t) \leftarrow robo_X(r,x,t), robo_Y(r,y,t). && (x-1 \in X)
\end{aligned}
$$

We disallow moving to multiple locations and standing still (the latter is not strictly necessary but we obtained experimental results that way).

$$
\begin{aligned}
&\leftarrow move(r,x_1,y_1,t), move(r,x_1,y_2,t). && (x_1,x_2 \in X,\ x_1 < x_2,\ y_1,y_2 \in Y)\\
&\leftarrow move(r,x,y_1,t), move(r,x,y_2,t). && (y_1,y_2 \in Y,\ y_1 < y_2)\\
move_{\exists}(r,t) &\leftarrow move(r,x,y,t). &&\\
&\leftarrow \text{not } move_{\exists}(r,t). &&
\end{aligned}
$$

The effect of moving is a deterministic change of location.

$$
robo_X(r,x,t+1) \leftarrow move(r,x,y,t). \qquad robo_Y(r,y,t+1) \leftarrow move(r,x,y,t).
$$

For saturation we guess the position of the object.

$$
obj_X(x) \vee \overline{obj}_X(x) \leftarrow . \qquad obj_X(y) \vee \overline{obj}_Y(y) \leftarrow .
$$

We spoil the answer set if the object is at multiple locations.

$$
\begin{aligned}
spoil &\leftarrow obj_X(x_1), obj_X(x_2). && (x_1,x_2 \in X,\ x_1 < x_2)\\
spoil &\leftarrow obj_Y(y_1), obj_Y(y_2). && (y_1,y_2 \in Y,\ y_1 < y_2)
\end{aligned}
$$

We spoil the answer set if the object is at no location.

$$
\begin{aligned}
objectHasNoXUpTo(1) &\leftarrow \overline{obj}_X(1). &&\\
objectHasNoXUpTo(x) &\leftarrow objectHasNoXUpTo(x-1), \overline{obj}_X(x). && (x-1 \in X)\\
spoil &\leftarrow objectHasNoXUpTo(x_{max}). && (x_{max} = max(X))\\
objectHasNoYUpTo(1) &\leftarrow \overline{obj}_Y(1). &&\\
objectHasNoYUpTo(y) &\leftarrow objectHasNoYUpTo(y-1), \overline{obj}_Y(y). && (y-1 \in Y)\\
spoil &\leftarrow objectHasNoYUpTo(y_{max}). && (y_{max} = max(Y))
\end{aligned}
$$

We spoil the answer set if the object is sensed, which is determined by an external atom with the semantic function $f_{\&sense}$ such that $f_{\&sense}(\mathbf{A}, robo_X, robo_Y, obj_X, obj_Y, range, spoil) = 1$ iff

$\mathbf{T}spoil \in \mathbf{A}$ or the predicates $robo_X$, $robo_Y$, obj_X, obj_Y represent in $\mathbf{A}$ a state where the robot has a distance less than $range$ to the object, i.e., the robot can detect the object. The implementation of this external atom was realized in C++ and consists of verifying $range \leq \sqrt{\Delta_x^2 + \Delta_y^2}$ and bookkeeping code to extract Δ_x and Δ_y from $\mathbf{A}$.

$$spoil \leftarrow \&sense[robo_X, robo_Y, obj_X, obj_Y, range, spoil]().$$

We create a unique answer set whenever $spoil$ is true and require that only spoiled answer sets are returned.

$$\begin{aligned} obj_X(x) &\leftarrow spoil. & \overline{obj}_X(x) &\leftarrow spoil. \\ obj_Y(x) &\leftarrow spoil. & \overline{obj}_Y(x) &\leftarrow spoil. \\ objectHasNoXUpTo(x) &\leftarrow spoil. & objectHasNoYUpTo(y) &\leftarrow spoil. \\ &\leftarrow \text{not } spoil. \end{aligned}$$

Given an instance encoded as above, an answer set to the above program exists iff there exists a sequence of movements that ensures to detect the object no matter where it is located. Furthermore the movements required to detect the object, i.e., the conformant plan, is encoded in the answer set in the extension of the $move$ predicate.

Appendix B. Proofs

Proof of Proposition 1. ($\Rightarrow$) Let $\mathbf{A}'$ be an answer set of $Check(\Pi, \mathbf{A})$ such that $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = 1$ iff $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \mathbf{A}'$ for all external atoms $\&g[\mathbf{p}](\mathbf{c})$ in Π.

Since $\hat{\mathbf{A}}$ is a compatible set of Π, $f_{\&g}(\mathbf{A}, \mathbf{p}, \mathbf{c}) = 1$ iff $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \hat{\mathbf{A}}$ for all external atoms $\&g[\mathbf{p}](\mathbf{c})$ in Π. Thus, $f\hat{\Pi}^{\hat{\mathbf{A}}}$ is the same as $f\Pi^{\mathbf{A}}$ with replacement atoms in place of external atoms, and with additional guessing rules for replacement atoms. Since $\mathbf{A}'$ is a model of $Check(\Pi, \mathbf{A})$ it is also a model of $f\hat{\Pi}^{\hat{\mathbf{A}}}$. Let $\mathbf{A}'' = \{\mathbf{T}a \in \mathbf{A}' \mid a \in A(\Pi)\} \cup \{\mathbf{F}a \in \mathbf{A}' \mid a \in A(\Pi)\}$. Since $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = 1$ iff $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \mathbf{A}'$ for all external atoms $\&g[\mathbf{p}](\mathbf{c})$ in Π by assumption, $\mathbf{A}''$ is a model of $f\Pi^{\mathbf{A}}$.

Since $\mathbf{A}'$ is an answer set of $Check(\Pi, \mathbf{A})$, and $\leftarrow a \in Check(\Pi, \mathbf{A})$ for all $a \in A(\Pi)$ with $\mathbf{T}a \notin \hat{\mathbf{A}}$ (and thus $\mathbf{T}a \notin \mathbf{A}$), $\{\mathbf{T}a \in \mathbf{A}'' \mid a \in A(\Pi)\} \subseteq \mathbf{A}$. Finally, due to $\{\leftarrow \text{not } smaller\} \cup \{smaller \leftarrow \text{not } a \mid a \in A(\Pi), \mathbf{T}a \in \hat{\mathbf{A}}\} \in Check(\Pi, \mathbf{A})$, there is at least one $a \in A(\Pi)$ s.t. $\mathbf{T}a \in \hat{\mathbf{A}}$ (and thus also $\mathbf{T}a \in \mathbf{A}$), but $\mathbf{F}a \in \mathbf{A}'$ (and thus also $\mathbf{F}a \in \mathbf{A}''$). Therefore $\{\mathbf{T}a \in \mathbf{A}'' \mid a \in A(\Pi)\} \subsetneq \mathbf{A}$ is a model of Π, and thus $\mathbf{A}$ is not an answer set of Π.

($\Leftarrow$) If $\mathbf{A}$ is not an answer set of Π, then there is a model $\mathbf{A}''$ of $f\Pi^{\mathbf{A}}$ which is smaller in the positive part, i.e., $\{\mathbf{T}a \in \mathbf{A}''\} \subsetneq \{\mathbf{T}a \in \mathbf{A}\}$. Let

$$\mathbf{A}' = \kappa(\Pi, \mathbf{A}'') \cup \{\mathbf{T}a' \mid \mathbf{T}a \in \hat{\mathbf{A}}, \mathbf{F}a \in \mathbf{A}''\} \cup \{\mathbf{F}a' \mid \mathbf{T}a \in \hat{\mathbf{A}}, \mathbf{T}a \in \mathbf{A}''\} \cup \{\mathbf{T}smaller\}.$$

We show that $\mathbf{A}'$ is an answer set of $Check(\Pi, \mathbf{A})$ such that $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = 1$ iff $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \mathbf{A}'$ for all external atoms $\&g[\mathbf{p}](\mathbf{c})$ in Π.

Since $\mathbf{A}$ has been extracted from a compatible set $\hat{\mathbf{A}}$ of Π, $f\hat{\Pi}^{\hat{\mathbf{A}}}$ is the same as $f\Pi^{\mathbf{A}}$ with replacement atoms in place of external atoms, and with additional guessing rules for replacement atoms. Since $\mathbf{A}''$ is a model of $f\Pi^{\mathbf{A}}$, and the truth values of all replacement atoms in $\mathbf{A}'$ coincide with the oracle functions by definition of $\kappa(\Pi, \mathbf{A}'')$, and set exactly one of e_a or ne_a for each external atom a in Π to true (and thus satisfy the guessing rules for replacement atoms), $\mathbf{A}'$ is a model of $f\hat{\Pi}^{\hat{\mathbf{A}}}$. Since $\{\mathbf{T}a \in \mathbf{A}''\} \subsetneq \{\mathbf{T}a \in \mathbf{A}\}$ and thus also $\{\mathbf{T}a \in \mathbf{A}' \mid a \in A(\Pi)\} \subsetneq \{\mathbf{T}a \in \mathbf{A} \mid a \in A(\Pi)\}$, the corresponding constraint $\leftarrow a$ in $\mathit{Check}(\Pi, \mathbf{A})$ is not violated. Moreover, for each a with $\mathbf{T}a \in \hat{\mathbf{A}}$ we have either $\mathbf{T}a \in \mathbf{A}'$ or $\mathbf{T}a' \in \mathbf{A}'$, thus the corresponding rule $a \vee a' \leftarrow$ in $\mathit{Check}(\Pi, \mathbf{A})$ is satisfied. Finally, since $\mathbf{T}\mathit{smaller} \in \mathbf{A}'$, the rules $\{\mathit{smaller} \leftarrow \mathrm{not}\, a \mid a \in A(\Pi), \mathbf{T}a \in \hat{\mathbf{A}}\}$ are satisfied and the constraint $\leftarrow \mathrm{not}\, \mathit{smaller}$ does not fire. Thus $\mathbf{A}'$ is a model of $\mathit{Check}(\Pi, \mathbf{A})$.

We show now that $\mathbf{A}'$ is also a subset-minimal model of $f\mathit{Check}(\Pi, \mathbf{A})^{\mathbf{A}'}$. Observe that

$$f\mathit{Check}(\Pi, \mathbf{A})^{\mathbf{A}'} = f\hat{\Pi}^{\hat{\mathbf{A}}} \cup \{a \vee a' \leftarrow \mid \mathbf{T}a \in \hat{\mathbf{A}}\} \\ \cup \{\mathit{smaller} \leftarrow \mathrm{not}\, a \mid a \in A(\Pi), \mathbf{T}a \in \hat{\mathbf{A}}\}.$$

However, if any atom $a \in A(f\mathit{Check}(\Pi, \mathbf{A})^{\mathbf{A}'})$ with $\mathbf{T}a \in \mathbf{A}'$ is changed to false, then the interpretation is not a model anymore because the corresponding rule $a \vee a' \leftarrow$ is violated, as only one of a and a' is true in $\mathbf{A}'$ by definition; thus no interpretation with smaller positive part than $\mathbf{A}'$ can be a model of $f\mathit{Check}(\Pi, \mathbf{A})^{\mathbf{A}'}$. Hence $\mathbf{A}'$ is an answer set of $\mathit{Check}(\Pi, \mathbf{A})$.

Finally, $f_{\&g}(\mathbf{A}', \mathbf{p}, \mathbf{c}) = 1$ iff $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \mathbf{A}'$ for all external atoms $\&g[\mathbf{p}](\mathbf{c})$ in Π by definition of $\kappa(\Pi, \mathbf{A}'')$. □

Proof of Proposition 2. The result follows from Proposition 1 and the fact that the programs $\mathit{Check}(\Pi, \mathbf{A})$ and $\mathit{CheckOptimized}(\Pi, \mathbf{A})$ have the same answer sets. Indeed, by construction the programs have the same models; consequently, every answer set $\mathbf{A}'$ of $\mathit{Check}(\Pi, \mathbf{A})$ (which is a model of $\mathit{Check}(\Pi, \mathbf{A})$) is a model of $f\mathit{CheckOptimized}(\Pi, \mathbf{A})^{\mathbf{A}'}$. As $\mathbf{A}'$ is a minimal model of $f\mathit{Check}(\Pi, \mathbf{A})^{\mathbf{A}'}$, due to the guessing rules $a \vee a' \leftarrow$, for $\mathbf{T}a \in \hat{\mathbf{A}}$, and $e_a \vee ne_a \leftarrow$, for all external atoms a in $\mathit{Check}(\Pi, \mathbf{A})$, we have either $\{\mathbf{T}a, \mathbf{F}a'\} \subseteq \mathbf{A}'$ or $\{\mathbf{F}a, \mathbf{T}a'\} \subseteq \mathbf{A}'$ (but not both) and either $\{\mathbf{T}e_a, \mathbf{F}ne_a\} \subseteq \mathbf{A}'$ or $\{\mathbf{F}e_a, \mathbf{T}ne_a\} \subseteq \mathbf{A}'$ (but not both); furthermore, due to the constraints $\leftarrow a$ for every $a \in A(\Pi)$ such that $\mathbf{T}a \notin \hat{\mathbf{A}}$, we have $\mathbf{F}a \in \mathbf{A}'$ for each such a. As the same guessing rules and constraints are also in $\mathit{CheckOptimized}(\Pi, \mathbf{A})$, in every model $\mathbf{A}''$ of $\mathit{CheckOptimized}(\Pi, \mathbf{A})^{\mathbf{A}'}$ such that $\mathbf{A}'' \leq_{\mathit{CheckOptimized}(\Pi, \mathbf{A})} \mathbf{A}'$ all atoms a, a', e_a, and ne_a must thus have the same value as in $\mathbf{A}'$; consequently, also $\mathit{smaller}$ must have the same value as in $\mathbf{A}'$. It follows that $\mathbf{A}'$ is a minimal model of $f\mathit{CheckOptimized}(\Pi, \mathbf{A})^{\mathbf{A}'}$, i.e., $\mathbf{A}'$ is an answer set of $\mathit{CheckOptimized}(\Pi, \mathbf{A})$. The argument that every answer set of $\mathit{CheckOptimized}(\Pi, \mathbf{A})$ is an answer set of $\mathit{Check}(\Pi, \mathbf{A})$ is analogous. □

Proof of Theorem 3. The argument that proves Corollary 3 by Faber (2005) can be used mutatis mutandi to prove this statement, with external atoms in place of aggregates. □

Proof of Proposition 4. We proceed by contraposition and show that if $I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$ is not a solution to $\Gamma_\Pi^{\mathbf{A}}$, then U cannot be an unfounded set such that $\mathbf{A}^{\mathbf{T}} \cap U \neq \emptyset$.

First observe that the nogoods in $H_{r,\mathbf{A}}$ demand $\mathbf{T}h_r$ to be true for a rule $r \in \Pi$ in a solution to $\Gamma_\Pi^{\mathbf{A}}$ if and only if some head atom $h \in H(r)$ of this rule is in U. As the truth values of h_r and all $h \in H(r)$ are defined in $I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$ exactly to this criterion, no nogood from $H_{r,\mathbf{A}}$ can be involved in a contradiction. Furthermore, the nogood $\{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{A}\} \in N_{\Gamma,\Pi}^{\mathbf{A}}$ is violated by $I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$ only if $\mathbf{A}^{\mathbf{T}} \cap U = \emptyset$; hence, if $I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$ is not a solution to $\Gamma_\Pi^{\mathbf{A}}$ and

$\mathbf{A}^{\mathbf{T}} \cap U \neq \emptyset$, as $O^{\mathbf{A}}_{\Gamma,\Pi}$ is conservative, then for some rule $r \in \Pi$ the nogood in $C_{r,\mathbf{A}}$ must be violated. That is, we know the following: $\mathbf{T}h_r \in I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$ (and therefore $H(r) \cap U \neq \emptyset$), $\mathbf{F}a \in I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$ for all $a \in B^+_o(r)$, $\mathbf{t}a \in I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$ for all $a \in B_e(\hat{r})$, and $\mathbf{T}h \in I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$ for all $h \in H(r)$ with $\mathbf{A} \models h$. Moreover, we have $C_{r,\mathbf{A}} \neq \emptyset$. We now show that this implies that none of the conditions (i)–(iii) of Definition 5 holds for r wrt. U and $\mathbf{A}$, which means that U is not an unfounded set.

Condition (i) does not hold for r because $\mathbf{A} \models B(r)$ (otherwise $C_{r,\mathbf{A}} = \emptyset$).

Condition (ii) does not hold for r. Suppose to the contrary that it holds. Then there must be some $b \in B(r)$ s.t. $\mathbf{A} \dot{\cup} \neg.U \not\models b$. Because $C_{r,\mathbf{A}} \neq \emptyset$, we know that $\mathbf{A} \models b$. We make a case distinction on the type of b:

- If b is a positive default literal from $A(\Pi)$, then $\mathbf{F}b \in I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$ and therefore $b \notin U$. Consequently $\mathbf{A} \dot{\cup} \neg.U \models b$. Contradiction.
- If b is a negative default literal over $A(\Pi)$, then $\mathbf{A} \models b$ implies $\mathbf{A} \dot{\cup} \neg.U \models b$. Contradiction.
- If b is a positive or default-negated replacement atom, then $\mathbf{t}b \in I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$. But this implies, by definition of $I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$, that $\mathbf{A} \dot{\cup} \neg.U \models b$. Contradiction.

Condition (iii) does not hold for r because $\mathbf{T}h \in I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$ and thus, by definition of $I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$, $h \in U$ for all $h \in H(r)$ with $\mathbf{A} \models h$. Thus $\mathbf{A} \not\models a$ for all $a \in H(r) \setminus U$. □

Proof of Theorem 6. Suppose U is not an unfounded set. Then there is an $r \in \Pi$ s.t. $H(r) \cap U \neq \emptyset$ and none of the conditions (i)–(iii) in Definition 5 is satisfied. We show now that S cannot be a solution to $\Gamma^{\mathbf{A}}_{\Pi}$ that satisfies conditions (a) and (b), which proves the result.

Because Condition (i) does not hold, there is a nogood of form

$$\{\{\mathbf{T}h_r\} \cup \{\mathbf{F}a \mid a \in B^+_o(r), \mathbf{A} \models a\} \cup \{\mathbf{t}a \mid a \in B_e(\hat{r})\} \cup \{\mathbf{T}h \mid h \in H(r), \mathbf{A} \models h\}\}$$

in $\Gamma^{\mathbf{A}}_{\Pi}$.

We now show that S contains all signed literals of this nogood, i.e., the nogood is violated by the assignment S.

Since $H(r) \cap U \neq \emptyset$, $\mathbf{T}h_r \in S$ (otherwise a nogood in $H^{\mathbf{A}}_r$ is violated). As U is not an unfounded set, Condition (ii) in Definition 5 does not hold. Consider any $a \in B^+_o(r)$ s.t. $\mathbf{A} \models a$. Then $a \notin U$, otherwise $A \dot{\cup} \neg.U \not\models a$ and we have a contradiction with the assumption that Condition (ii) is unsatisfied. But then $\mathbf{F}a \in S$ (because S is complete and would imply $a \in U$ otherwise).

Now consider any $\&g[\mathbf{p}](\mathbf{c}) \in EA(r)$. Then $\mathbf{A} \dot{\cup} \neg.U \models \&g[\mathbf{p}](\mathbf{c})$ (as (ii) is violated). If $\mathbf{A} \not\models \&g[\mathbf{p}](\mathbf{c})$, then Condition (i) would be satisfied, hence $\mathbf{A} \models \&g[\mathbf{p}](\mathbf{c})$. But then $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in S$, otherwise $\mathbf{A} \dot{\cup} \neg.U \not\models \&g[\mathbf{p}](\mathbf{c})$ by precondition (b) of this proposition. Next consider all $not\, \&g[\mathbf{p}](\mathbf{c}) \in B_e(r)$. Then $\mathbf{A} \dot{\cup} \neg.U \not\models \&g[\mathbf{p}](\mathbf{c})$ (as (ii) is violated). If $\mathbf{A} \models \&g[\mathbf{p}](\mathbf{c})$, then Condition (i) would be satisfied, hence $\mathbf{A} \not\models \&g[\mathbf{p}](\mathbf{c})$. But then $\mathbf{F}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in S$, otherwise $\mathbf{A} \dot{\cup} \neg.U \models \&g[\mathbf{p}](\mathbf{c})$ by precondition (a) of this proposition. Therefore, we have $\mathbf{t}a \in S$ for all $a \in B_e(\hat{r})$.

Finally, because Condition (iii) in Definition 5 does not hold, $h \in U$ and therefore also $\mathbf{T}h \in S$ for all $h \in H(r)$ with $\mathbf{A} \models a$.

This concludes the proof that S cannot be a solution to $\Gamma^{\mathbf{A}}_{\Pi}$ satisfying (a) and (b), if U is not an unfounded set. □

Proof of Proposition 7. Let $S = I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$. If for an external atom $\&g[\mathbf{y}](\mathbf{x})$ in Π we have $\mathbf{T}e_{\&g[\mathbf{y}]}(\mathbf{x}) \in S$, then by definition of $I_\Gamma(U, \Gamma^{\mathbf{A}}_{\Pi}, \Pi, \mathbf{A})$ we have $\mathbf{A} \dot{\cup} \neg.U \models \&g[\mathbf{y}](\mathbf{x})$

(satisfying (a)). If for an external atom $\&g[\mathbf{y}](\mathbf{x})$ in Π we have $\mathbf{F}e_{\&g[\mathbf{y}]}(\mathbf{x}) \in \mathbf{S}$, then by definition of $I_\Gamma(U, \Gamma^{\mathbf{A}}_\Pi, \Pi, \mathbf{A})$ we have $\mathbf{A} \dot\cup \neg.U \not\models \&g[\mathbf{y}](\mathbf{x})$ (satisfying (b)). □

Proof of Proposition 8. We proceed by contraposition and show that if $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ is not a solution to Ω_Π with assumptions $\mathcal{A}_{\mathbf{A}}$, then U cannot be an unfounded set such that $\mathbf{A}^{\mathbf{T}} \cap U \neq \emptyset$.

First observe that the nogoods in H_r demand $\mathbf{T}h_r$ to be true for a rule $r \in \Pi$ if and only if some head atom $h \in H(r)$ of this rule is in U. Moreover, the nogoods in D_a for each $a \in A(\Pi)$ force $a_{\mathbf{A}\dot\cup\neg.U}$ to true if and only if $\mathbf{T}a \in \mathbf{A}$ and $a \notin U$, which is equivalent to $\mathbf{T}a \in \mathbf{A} \dot\cup \neg.U$; $a_{\mathbf{A}\wedge U}$ to true if and only if $\mathbf{T}a \in \mathbf{A}$ and $a \in U$; and $a_{\overline{\mathbf{A}}\vee U}$ to true if and only if $\mathbf{F}a \in \mathbf{A}$ or $a \in U$. As the truth values of h_r for each $r \in \Pi$, and $a_{\mathbf{A}\dot\cup\neg.U}$ and $a_{\mathbf{A}\wedge U}$ and $a_{\overline{\mathbf{A}}\vee U}$ for each $a \in A(\Pi)$ in $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ are defined exactly to these conditions, a contradiction must involve C_r for some $r \in \Pi$. Furthermore, the nogood $\{\mathbf{F}a \mid a \in A(\Pi)\}$ is violated by $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ only if $A(\Pi) \cap U = \emptyset$, and thus $\mathbf{A}^{\mathbf{T}} \cap U \neq \emptyset$; hence, if $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ is not a solution to Ω_Π such that $\mathbf{A}^{\mathbf{T}} \cap U \neq \emptyset$, since $O_{\Omega,\Pi}$ is conservative, for some rule $r \in \Pi$ the nogood in C_r must be violated. That is, we know the following:

(I) $\mathbf{T}h_r \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ (and therefore $H(r) \cap U \neq \emptyset$),

(II) $\mathbf{T}a_{\mathbf{A}} \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ for all $a \in B^+(\hat{r})$ and $\mathbf{F}a_{\mathbf{A}} \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ for all $a \in B^-(\hat{r})$,

(III) $\mathbf{F}a_{\mathbf{A}\wedge U} \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ for all $a \in B^+_o(r)$, $\mathbf{t}a \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ for all $a \in B_e(\hat{r})$, and

(IV) $\mathbf{T}h_{\overline{\mathbf{A}}\vee U} \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ for all $h \in H(r)$.

We now show that this implies that none of the conditions of Definition 5 holds for r wrt. U and $\mathbf{A}$, which means that U is not an unfounded set (h_r is true in $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$, which implies $H(r) \cap U \neq \emptyset$x).

Condition (i) does not hold for r because of (II), which by our assumptions $\mathcal{A}_{\mathbf{A}}$ implies $\mathbf{A} \models B(r)$. Condition (ii) does not hold for r. Suppose to the contrary that it holds. Then there must be some $b \in B(r)$ s.t. $\mathbf{A} \dot\cup \neg.U \not\models b$. Since Condition (i) is already known to be violated, we can assume that $\mathbf{A} \models b$. We make a case distinction on the type of b:

- If b is a positive default literal from $A(\Pi)$, then $b \in U$ (otherwise $\mathbf{A} \dot\cup \neg.U \models b$). But then we have by definition of $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ that $\mathbf{T}b_{\mathbf{A}\wedge U} \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$, which contradicts (III).
- If b is a negative default literal over $A(\Pi)$, then $\mathbf{A} \models b$ implies $\mathbf{A} \dot\cup \neg.U \models b$. Contradiction.
- If b is a positive or default-negated replacement atom, then $\mathbf{t}b \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$. But this implies, by definition of $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$, that $\mathbf{A} \dot\cup \neg.U \models b$. Contradiction.

Condition (iii) does not hold for r because $\mathbf{T}h_{\overline{\mathbf{A}}\vee U} \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ and thus, by definition of $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$, $h \in U$ for all $h \in H(r)$ with $\mathbf{A} \models h$. Thus $\mathbf{A} \not\models a$ for all $a \in H(r) \setminus U$. □

Proof of Theorem 10. Suppose U is not an unfounded set. Then some $r \in \Pi$ exists such that $H(r) \cap U \neq \emptyset$ and none of the conditions (i)–(iii) in Definition 5 is satisfied. We show that then S, assuming that $\mathcal{A}_{\mathbf{A}}$ is satisfied and (a) and (b) hold, cannot be a solution to Ω_Π.

Due to rule r, Ω_Π (more specifically, $N_{\Omega,\Pi}$) contains a nogood N of form

$$\begin{aligned} N = \{\ & \{\mathbf{T}h_r\} \cup \\ & \{\mathbf{T}a_{\mathbf{A}} \mid a \in B^+(\hat{r})\} \cup \{\mathbf{F}a_{\mathbf{A}} \mid a \in B^-(\hat{r})\} \cup \\ & \{\mathbf{F}a_{\mathbf{A}\wedge U} \mid a \in B^+_o(r)\} \cup \{\mathbf{t}a \mid a \in B_e(\hat{r})\} \cup \\ & \{\mathbf{T}h_{\overline{\mathbf{A}}\vee U} \mid h \in H(r)\}\ \}. \end{aligned}$$

We show that S contains all signed literals of N, i.e., N is violated by S. As $H(r) \cap U \neq \emptyset$, $\mathbf{T}h_r \in S$ (otherwise a nogood in H_r is violated). Furthermore, as U is not an unfounded set, Condition (i) in Definition 5 does not hold for U wrt. $\mathbf{A}$, and hence $\mathbf{A} \models B(r)$. The assumptions $\mathcal{A}_{\mathbf{A}}$ thus enforce that $\mathbf{T}a_{\mathbf{A}} \in S$ for all $a \in B^+(\hat{r})$ and $\mathbf{F}a_{\mathbf{A}} \in S$ for all $a \in B^-(\hat{r})$.

Consider next an arbitrary $a \in B_o^+(r)$. As Condition (ii) in Definition 5 does not hold for U wrt. $\mathbf{A}$, we have $\mathbf{A} \models a$ and $a \notin U$; the latter implies by definition of U that $\mathbf{F}a \in S$. From the nogood $\{\mathbf{T}a_{\mathbf{A}\wedge U}, \mathbf{F}a\}$ we conclude $\mathbf{F}a_{\mathbf{A}\wedge U} \in S$.

We next show that for every $a \in B_e(\hat{r})$, it holds that $\mathbf{t}a \in S$. Indeed, let $\&g[\mathbf{p}](\mathbf{c}) \in EA(r)$. We have $\mathbf{A} \dot{\cup} \neg.U \models \&g[\mathbf{p}](\mathbf{c})$ (as Condition (ii) is violated). Furthermore, $\mathbf{A} \models \&g[\mathbf{p}](\mathbf{c})$, as $\mathbf{A} \not\models \&g[\mathbf{p}](\mathbf{c})$ would imply that Condition (i) is satisfied. Thus from Condition (b) of the hypothesis, it follows that $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in S$. Similarly, let $\mathrm{not}\ \&g[\mathbf{p}](\mathbf{c}) \in B_e(r)$. Then $\mathbf{A} \dot{\cup} \neg.U \not\models \&g[\mathbf{p}](\mathbf{c})$ (as Condition (ii) is violated) and $\mathbf{A} \not\models \&g[\mathbf{p}](\mathbf{c})$ as $\mathbf{A} \models \&g[\mathbf{p}](\mathbf{c})$ would satisfy Condition (i). Thus from Condition (a) of the hypothesis, it follows that $\mathbf{F}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in S$. Thus we have $\mathbf{t}a \in S$ for all $a \in B_e(\hat{r})$.

Finally, because Condition (iii) in Definition 5 does not hold for U wrt. $\mathbf{A}$, we have $h \in U$ and therefore also $\mathbf{T}h \in S$ for all $h \in H(r)$ with $\mathbf{A} \models a$. That is, for each $h \in H(r)$, either $\mathbf{F}h_{\mathbf{A}} \in S$ or $\mathbf{T}h \in S$. But by the nogoods $\{\mathbf{F}h_{\overline{\mathbf{A}}\vee U}, \mathbf{F}h_{\mathbf{A}}\}, \{\mathbf{F}h_{\overline{\mathbf{A}}\vee U}, \mathbf{T}h\} \in D_h$, in both cases we have $\mathbf{T}h_{\overline{\mathbf{A}}\vee U} \in S$.

Hence, $N \subseteq S$ holds, i.e., N is violated by S, and thus S is not a solution of Ω_Π. This completes the proof of the result. □

Proof of Proposition 11. Let $S = I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$. If for an external atom $\&g[\mathbf{y}](\mathbf{x})$ in Π we have $\mathbf{T}e_{\&g[\mathbf{y}]}(\mathbf{x}) \in S$, then by definition of $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ we have $\mathbf{A} \dot{\cup} \neg.U \models \&g[\mathbf{y}](\mathbf{x})$ (satisfying (a)). If for an external atom $\&g[\mathbf{y}](\mathbf{x})$ in Π we have $\mathbf{F}e_{\&g[\mathbf{y}]}(\mathbf{x}) \in \mathbf{S}$, then by definition of $I_\Omega(U, \Omega_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$ we have $\mathbf{A} \dot{\cup} \neg.U \not\models \&g[\mathbf{y}](\mathbf{x})$ (satisfying (b)). □

Proof of Proposition 12. To show that $U \setminus \{a\}$ is an unfounded set wrt. $\mathbf{A}$, consider $r \in \Pi$ such that $H(r) \cap (U \setminus \{a\}) \neq \emptyset$. We have to show that one of the conditions (i)–(iii) of Definition 5 holds for r. By hypothesis, U is an unfounded set of Π wrt. $\mathbf{A}$, and $H(r) \cap (U \setminus \{a\}) \neq \emptyset$ implies $H(r) \cap U \neq \emptyset$. If Condition (i) holds for U, it also holds wrt. $U \setminus \{a\}$ because this condition depends only on r and $\mathbf{A}$. Also if Condition (ii) holds for U, it also holds wrt. $U \setminus \{a\}$ because $\mathbf{A} \dot{\cup} \neg.U$ is equivalent to $\mathbf{A} \dot{\cup} \neg.(U \setminus \{a\})$ since $\mathbf{A} \not\models a$. Finally, if Condition (iii) holds for U, then some atom $b \in H(r) \setminus U$ exists such that $\mathbf{A} \models b$. As $\mathbf{A} \not\models a$, it follows $a \neq b$ and hence condition (iii) holds for $U \setminus \{a\}$. This proves the result. □

Proof of Proposition 13. Suppose that changing the truth value of b in S turns the solution to a counterexample S_b of $\Gamma_\Pi^{\mathbf{A}}$ (resp. Ω_Π). That is, S_b must violate some nogood $N \in \Gamma_\Pi^{\mathbf{A}}$ resp. ($N \in \Omega_\Pi$) containing b, i.e., either $\mathbf{T}b \in N$ or $\mathbf{F}b \in N$.

For the encoding $\Gamma_\Pi^{\mathbf{A}}$, the nogood N corresponds to a rule with $b \in B^+(\hat{r})$ or $b \in B^-(\hat{r})$ and $\mathbf{A} \models B(r)$, and N contains also the signed literals (1) $\mathbf{F}a$ for all $a \in B_o^+$ with $\mathbf{A} \models a$ and (2) $\mathbf{T}a$ for all $a \in H(r)$ with $\mathbf{A} \models a$. By hypothesis, we have either (a) $\mathbf{T}a \in S$ for some $a \in B_o^+(r)$ with $\mathbf{A} \models a$, or (b) $\mathbf{F}a$ for some $a \in H(r)$ with $\mathbf{A} \models a$. But then the nogood cannot be violated, because (a) contradicts one of (1) and (b) contradicts one of (2).

For the encoding Ω_Π, the nogood N corresponds to a rule r with $b \in B^+(\hat{r})$ or $b \in B^-(\hat{r})$. The nogood contains also the signed literals (1) $\mathbf{T}a_{\mathbf{A}}$ for all $a \in B^+(\hat{r})$ and $\mathbf{F}a_{\mathbf{A}}$ for all $a \in B^-(\hat{r})$, (2) $\mathbf{F}a_{\mathbf{A}\wedge U}$ for all $a \in B_o^+$, and (3) $\mathbf{T}h_{\bar{\mathbf{A}}\vee U}$ for all $h \in H(r)$. Because of (1) and since $\mathbf{A}$ is a solution to $\mathcal{A}_{\mathbf{A}}$, we have $\mathbf{A} \models B(r)$. By hypothesis, we have either (a) $\mathbf{T}a \in S$ for some $a \in B_o^+(r)$ with $\mathbf{A} \models a$, or (b) $\mathbf{F}a$ for some $a \in H(r)$ with $\mathbf{A} \models a$. But then the nogood N cannot be violated, because (a) contradicts part (2) by definition of $\mathcal{A}_{\mathbf{A}}$ and $a_{\mathbf{A} \wedge U}$, and (b) contradicts part (3) by definition of $\mathcal{A}_{\mathbf{A}}$ and $h_{\bar{\mathbf{A}} \vee U}$. □

Proof of Proposition 14. If $\mathcal{T}_\Gamma(N, \mathbf{A}) = \emptyset$ then the proposition trivially holds. Otherwise $\mathcal{T}_\Gamma(N, \mathbf{A}) = \{C\}$ and we know that $\mathbf{T}t_i \in \mathbf{A}$ for all $1 \leq i \leq n$. Suppose C is violated by $I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$. Then $\mathbf{F}t_i \in I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$ and therefore $t_i \notin U$ for all $1 \leq i \leq n$, and $\mathbf{T}f_i \in I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$ for all $1 \leq i \leq m$ with $\mathbf{A} \models f_i$, and $\sigma e_{\&g[\mathbf{p}]}(\mathbf{c}) \in I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$.

But then $\mathbf{A} \dot\cup \neg.U \models t_i$ for all $1 \leq i \leq n$ and $\mathbf{A} \dot\cup \neg.U \not\models f_i$ for all $1 \leq i \leq m$. Because the nogood N is a valid input-output-relationship, this implies $\bar\sigma \&g[\mathbf{p}](\mathbf{c}) \in \mathbf{A} \dot\cup \neg.U$. By definition of $I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$, we then have $\bar\sigma e_{\&g[\mathbf{p}]}(\mathbf{c}) \in I_\Gamma(U, \Gamma_\Pi^{\mathbf{A}}, \Pi, \mathbf{A})$, which contradicts the assumption that $\mathcal{T}_\Gamma(N, \mathbf{A})$ is violated. □

Proof of Proposition 15. We know $\mathcal{T}_\Omega(N) = \{C\}$. Suppose $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ violates C. Then $\mathbf{T}t_{i\mathbf{A}} \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ and therefore $\mathbf{T}t_i \in \mathbf{A}$ and $\mathbf{F}t_i \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ for all $1 \leq i \leq n$; $\mathbf{F}f_{i\mathbf{A}\dot\cup\neg.U} \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$ and therefore either $\mathbf{F}f_i \in \mathbf{A}$ or $f_i \in U$, for all $1 \leq i \leq m$; and $\sigma e_{\&g[\mathbf{p}]}(\mathbf{c}) \in I(U, \Omega_\Pi, \mathbf{A})$.

But then, by definition of $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$, $\mathbf{T}t_i \in \mathbf{A}$ and $t_i \notin U$ for all $1 \leq i \leq n$, hence $\mathbf{A} \dot\cup \neg.U \models t_i$ for all $1 \leq i \leq n$. Moreover, $\mathbf{A} \dot\cup \neg.U \not\models f_i$ for all $1 \leq i \leq m$. Because nogood N is a valid input-output-relationship, this implies $\bar\sigma \&g[\mathbf{p}](\mathbf{c}) \in \mathbf{A} \dot\cup \neg.U$. On the other hand, by definition of $I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$, we have $\bar\sigma e_{\&g[\mathbf{p}]}(\mathbf{c}) \in I_\Omega(U, \Omega_\Pi, \Pi, \mathbf{A})$; this contradicts the assumption that the nogood C is violated. □

Proof of Proposition 16. Suppose some answer set $\mathbf{A}'$ of Π exists which is not a solution to $L_1(U, \Pi, \mathbf{A})$, i.e., it violates some nogood $N = \{\sigma_0, \sigma_1, \ldots, \sigma_n\} \in L_1(U, \Pi, \mathbf{A})$. We show that in this case U is an unfounded set of Π wrt. $\mathbf{A}'$ such that $U \cap \mathbf{A}' \neq \emptyset$; this means that $\mathbf{A}'$ is not-unfounded-free, which contradicts that $\mathbf{A}'$ is an answer set of Π.

Let $r \in \Pi$ be a rule such that $H(r) \cap U \neq \emptyset$. We have to show that one of the conditions (i)–(iii) of Definition 5 holds.

If $B_o^+(r) \cap U \neq \emptyset$, then Condition (ii) holds. Hence we assume in the following that $B_o^+(r) \cap U = \emptyset$, which means that r is an external rule of Π wrt. U. But then for some $\sigma_i \in N$ with $1 \leq i \leq n$ we have either (1) $\sigma_i = \mathbf{T}h$ for some $h \in H(r)$ with $h \notin U$ and $\mathbf{A} \models h$, or (2) $\sigma_i = \mathbf{F}b$ for some $b \in B_o^+(r)$ with $\mathbf{A} \not\models b$. Since $\mathbf{A}'$ violates N by assumption, we have $\sigma_i \in \mathbf{A}'$. In Case (1) the Condition (iii) is satisfied, while in Case (2) then Condition (i) is satisfied.

Moreover, by definition of L_1 some $a \in U$ exists such that $\mathbf{T}a \in \mathbf{A}'$, i.e., $\mathbf{A}'$ intersects with U. This proves the result. □

Proof of Proposition 17. Towards a contradiction, suppose some answer set $\mathbf{A}'$ of Π is not a solution to $L_2(U, \Pi, \mathbf{A})$. Let $N = \{\mathbf{T}a \mid a \in \mathbf{A} \dot\cup \neg.U\} \cup \{\sigma_0, \sigma_1, \ldots, \sigma_n\} \in L_2(U, \Pi, \mathbf{A})$ be a violated nogood. Because $\sigma_i \in \mathbf{A}'$ for all $1 \leq i \leq n$, we know that $\mathbf{A}'$ falsifies (at least) the bodies of rules in Π that are falsified by $\mathbf{A}$; consequently, $f\Pi^{\mathbf{A}'} \subseteq f\Pi^{\mathbf{A}}$. From $\mathbf{A} \models \Pi$ and the hypothesis that U is an unfounded set it follows that $\mathbf{A} \dot\cup \neg.U \models f\Pi^{\mathbf{A}}$; hence, also $\mathbf{A} \dot\cup \neg.U \models f\Pi^{\mathbf{A}'}$. Moreover, $\mathbf{T}a \in \mathbf{A}'$ for all $a \in \mathbf{A} \dot\cup \neg.U$, and therefore $\mathbf{A}'^{\mathbf{T}} \supseteq (\mathbf{A} \dot\cup \neg.U)^{\mathbf{T}}$. Because $\sigma_0 \in \mathbf{A}'$, we conclude $\mathbf{A}'^{\mathbf{T}} \supsetneq (\mathbf{A} \dot\cup \neg.U)^{\mathbf{T}}$, i.e., $\mathbf{A}'$ is not a subset-minimal model of $\Pi^{\mathbf{A}'}$. Consequently, $\mathbf{A}'$ is not an answer set of Π, which is a contradiction. □

Proof of Lemma 18. If $Y = \emptyset$, then the result holds trivially. Otherwise, let $r \in \Pi$ with $H(r) \cap Y \neq \emptyset$. We show that one of the conditions (i)–(iii) in Definition 5 holds. Observe that $H(r) \cap U \neq \emptyset$ because $U \supseteq Y$. Since U is an unfounded set of Π wrt. $\mathbf{A}$, either

(i) $\mathbf{A} \not\models b$ for some $b \in B(r)$; or

(ii) $\mathbf{A} \dot{\cup} \neg.U \not\models b$ for some $b \in B(r)$; or

(iii) $\mathbf{A} \models h$ for some $h \in H(r) \setminus U$.

In Case (i), the condition also holds wrt. Y. In case (ii), let $a \in H(r)$ such that $a \in Y$, and $b \in B(r)$ such that $\mathbf{A} \dot{\cup} \neg.U \not\models b$. We make a case distinction: either b is an ordinary literal or an external one.

If b is an ordinary default-negated atom $\mathrm{not}\, c$, then $\mathbf{A} \dot{\cup} \neg.U \not\models b$ implies $\mathbf{T}c \in \mathbf{A}$ and $c \notin U$, and therefore also $\mathbf{A} \dot{\cup} \neg.Y \not\models b$. So assume b is an ordinary atom. If $b \notin U$ then $\mathbf{A} \not\models b$ and Case (i) applies, so assume $b \in U$. Because $a \in H(r)$ and $b \in B(r)$, we have $a \to b$ and therefore either $a, b \in C$ or $a, b \in Y$ (because there are no ordinary edges between C and Y). But by assumption $a \in Y$, and therefore $b \in Y$, hence $\mathbf{A} \dot{\cup} \neg.Y \not\models b$.

If b is an external literal, then there is no $q \in U$ with $a \to_e q$ and $q \notin Y$. Otherwise, this would imply $q \in C$ and C would have an incoming e-edge, which contradicts the assumption that C is a cut of G^R_Π. Hence, for all $q \in U$ with $a \to_e q$, also $q \in Y$, and therefore the truth value of b under $\mathbf{A} \dot{\cup} \neg.U$ and $\mathbf{A} \dot{\cup} \neg.Y$ is the same. Hence $\mathbf{A} \dot{\cup} \neg.Y \not\models b$.

In Case (iii), also $\mathbf{A} \models h$ for some $h \in H(r) \setminus Y$ because $Y \subseteq U$ and therefore $H(r) \setminus Y \supseteq H(r) \setminus U$. □

Proof of Lemma 19. If $U = \emptyset$, then the result holds trivially. Otherwise, suppose $\hat{r} \in \hat{\Pi}$ and $a \in H(\hat{r}) \cap U$. Observe that $\hat{r}$ cannot be an external atom guessing rule because U contains only ordinary atoms. We show that one of the conditions in Definition 5 holds for $\hat{r}$ wrt. $\hat{\mathbf{A}}$.

Because $\hat{r}$ is no external atom guessing rule, there is a corresponding rule $r \in \Pi$ containing external atoms in place of replacement atoms. Because U is an unfounded set of Π and $H(r) = H(\hat{r})$, either:

(i) $\mathbf{A} \not\models b$ for some $b \in B(r)$; or

(ii) $\mathbf{A} \dot{\cup} \neg.U \not\models b$ for some $b \in B(r)$; or

(iii) $\mathbf{A} \models h$ for some $h \in H(r) \setminus U$

In Case (i), let $b \in B(r)$ such that $\mathbf{A} \not\models b$ and $\hat{b}$ the corresponding literal in $B(\hat{b})$ (which is the same if b is ordinary and the corresponding replacement literal if b is external). Then also $\hat{\mathbf{A}} \not\models \hat{b}$ because $\hat{\mathbf{A}}$ is compatible.

In Case (ii), we make a case distinction: either b is ordinary or external.

If b is ordinary, then $b \in B(\hat{r})$ and $\hat{\mathbf{A}} \dot{\cup} \neg.U \not\models b$ holds because $\mathbf{A}$ and $\hat{\mathbf{A}}$ are equivalent for ordinary atoms.

If b is an external atom or default-negated external atom, then no atom $p(\mathbf{c}) \in U$ is input to it, i.e., p is not a predicate input parameter of b; otherwise we had $a \to_e p(\mathbf{c})$, contradicting our assumption that U has no internal e-edges. But then $\mathbf{A} \dot{\cup} \neg.U$ implies $\mathbf{A} \not\models b$ because the truth value of b under $\mathbf{A} \dot{\cup} \neg.U$ and $\mathbf{A}$ is the same. Therefore we can apply Case (i).

In Case (iii), also $\hat{\mathbf{A}} \models h$ for some $h \in H(\hat{r}) \setminus U$ because $H(r) = H(\hat{r})$ contains only ordinary atoms and $\mathbf{A}$ is equivalent to $\hat{\mathbf{A}}$ for ordinary atoms. $\square$

Proof of Theorem 20. We define the *reachable set* $R(a)$ from some atom a as

$$R(a) = \{b \mid (a, b) \in \{\rightarrow \cup \leftarrow\}^*\},$$

i.e., the set of atoms $b \in U$ reachable from a using edges from $\rightarrow \cup \leftarrow$ only but no e-edges.

We first assume that U contains at least one e-edge, i.e., there are $x, y \in U$ such that $x \rightarrow_e y$. Now we show that there is a $u \in U$ with outgoing e-edge (i.e., $u \rightarrow_e v$ for some $v \in U$), but such that $R(u)$ has no incoming e-edges from U (i.e., for all $v \in R(u)$ and $b \in U$, $b \not\rightarrow_e v$ holds). Suppose to the contrary that for all a with outgoing e-edges, the reachable set $R(a)$ has an incoming e-edge from U. We now construct an e-cycle under $\rightarrow^d$, which contradicts our assumption. Start with an arbitrary node $c_0 \in U$ that has an outgoing e-edge, let p_0 be the (possibly empty) path (under $\rightarrow \cup \leftarrow$) from c_0 to the node $d_0 \in R(c_0)$ such that d_0 has an incoming e-edge, i.e., there is a c_1 such that $c_1 \rightarrow_e d_0$; note that $c_1 \notin R(c_0)$.[4] By assumption, also some node d_1 in $R(c_1)$ has an incoming e-edge (from some node $c_2 \notin R(c_1)$). Let p_1 be the path from c_1 to d_1, etc. By iteration we can construct the concatenation of the paths $p_0, (d_0, c_1), p_1, (d_1, c_2), p_2, \ldots, p_i, (d_i, c_{i+1}), \ldots$, where the p_i from c_i to d_i are the paths within reachable sets, and the (d_i, c_{i+1}) are the e-edges between reachable sets. However, as U is finite some nodes on this path must be equal, i.e., a subsequence of the constructed sequence represents an e-cycle (in reverse order).

This proves that u is a node with outgoing e-edge but such that $R(u)$ has no incoming e-edges. We next show that $R(u)$ is a cut of G^R_Π. Condition (i) is immediately satisfied by definition of u. Condition (ii) is shown as follows. Let $u' \in R(u)$ and $v' \in U \setminus R(u)$. We have to show that $u' \not\rightarrow v'$ and $v' \not\rightarrow u'$. Suppose, towards a contradiction, that $u' \rightarrow v'$. Because $u' \in R(u)$, there is a path from u to u' under $\rightarrow \cup \leftarrow$. But if $u' \rightarrow v'$, then there would also be a path from u to v' under $\rightarrow \cup \leftarrow$ and v' would be in $R(u)$, a contradiction. Analogously, $v' \rightarrow u'$ would also imply that there is a path from u to v' because there is a path from u to u', again a contradiction.

Therefore, $R(u) \subseteq U$ is a cut of G^R_Π, and by Lemma 18, it follows that $U \setminus R(u)$ is an unfounded set. Observe that $U \setminus R(u)$ contains one e-edge less than U because u has an outgoing e-edge and that $U \setminus R(u) \neq \emptyset$; indeed, by assumption some $w \in U$ exists such that $u \rightarrow_e w$, and clearly $w \notin R(u)$. By iterating this argument, the number of e-edges in the unfounded set can be reduced to zero in a nonempty core. Eventually Lemma 19 applies, proving that the remaining set is an unfounded set of $\hat{\Pi}$. $\square$

Proof of Corollary 21. Towards a contradiction, suppose an unfounded set U of Π wrt. $\mathbf{A}$ exists. Then U contains no e-cycle because there is no e-cycle under $\rightarrow^d$. By Theorem 20 there is an unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$, which contradicts our assumption that $\hat{\Pi}$ has no unfounded set wrt. $\mathbf{A}$. $\square$

Proof of Theorem 22. If U contains no cyclic input atoms, then all cycles under $\rightarrow^d$ containing e-edges in the atom dependency graph of Π are broken, i.e., U does not contain an e-cycle under $\rightarrow^d$. Then by Theorem 20 there exists a nonempty unfounded set of $\hat{\Pi}$ wrt. $\hat{\mathbf{A}}$. $\square$

4. Whenever $x \rightarrow_e y$ for $x, y \in U$, then there is no path from x to y under $\rightarrow \cup \leftarrow$, because otherwise we would have an e-cycle under $\rightarrow^d$.

Proof of Theorem 23. Let U be a nonempty unfounded set of Π wrt. $\mathbf{A}$. Because $\mathcal{C}$ is a decomposition of $A(\Pi)$ into strongly connected components, the *component dependency graph*

$$\langle \mathcal{C}, \{(C_1, C_2) \mid C_1, C_2 \in \mathcal{C}, \exists a_1 \in C_1, a_2 \in C_2 : (a_1, a_2) \in \rightarrow \cup \rightarrow_e\}\rangle$$

is acyclic. Following the hierarchical component dependency graph from the nodes without predecessor components downwards, we can find a "first" component which has a nonempty intersection with U, i.e., there exists a component $C \in \mathcal{C}$ such that $C \cap U \neq \emptyset$ but $C' \cap U = \emptyset$ for all transitive predecessor components C' of C.

We show that $U \cap C$ is an unfounded set of Π_C wrt. $\mathbf{A}$. Let $r \in \Pi_C$ be a rule such that $H(r) \cap (U \cap C) \neq \emptyset$. We have to show that one of the conditions (i)-(iii) of Definition 5 holds for r wrt. $\mathbf{A}$ and $U \cap C$.

Because U is an unfounded set of Π wrt. $\mathbf{A}$ we know that one of the conditions (i)–(iii) holds for r wrt. $\mathbf{A}$ and U. In case of Condition (i), then it trivially holds also wrt. $\mathbf{A}$ and $U \cap C$ because this condition depends only on the assignment $\mathbf{A}$, but not on the unfounded set U; in case of Condition (iii), it clearly holds because $H(r) \setminus U \neq \emptyset$ is included in $H(r) \setminus (U \cap C)$.

In case of Condition (ii), we have that $\mathbf{A} \dot\cup \neg.U \not\models b$ for some (ordinary or external) body literal $b \in B(r)$. We show next that the truth value of all literals in $B(r)$ is the same under $\mathbf{A} \dot\cup \neg.U$ and $\mathbf{A} \dot\cup \neg.(U \cap C)$, which proves that Condition (ii) holds also wrt. $\mathbf{A}$ and $U \cap C$.

If $b = \mathrm{not}\, a$ for some ordinary atom a, then $\mathbf{T}a \in \mathbf{A}$ and $a \notin U$ and consequently $a \notin U \cap C$, hence $\mathbf{A} \dot\cup \neg.(U \cap C) \not\models b$. If b is an ordinary atom, then either $\mathbf{F}b \in \mathbf{A}$, which implies immediately that $\mathbf{A} \dot\cup \neg.(U \cap C) \not\models b$, or $b \in U$. But in the latter case b is either in a predecessor component C' of C or in C itself (since $h \rightarrow b$ for all $h \in H(r)$). But since $U \cap C' = \emptyset$ for all predecessor components of C, we know $b \in C$ and therefore $b \in (U \cap C)$, which implies $\mathbf{A} \dot\cup \neg.(U \cap C) \not\models b$.

If b is a positive or default-negated external atom, then all input atoms a to b are either in a predecessor component C' of C or in C itself (since $h \rightarrow_e a$ for all $h \in H(r)$). We show with a similar argument as before that the truth value of each input atom a is the same under $\mathbf{A} \dot\cup \neg.U$ and $\mathbf{A} \dot\cup \neg.(U \cap C)$: if $\mathbf{A} \dot\cup \neg.U \models a$, then $\mathbf{T}a \in \mathbf{A}$ and $a \notin U$, hence $a \notin (U \cap C)$ and therefore $\mathbf{A} \dot\cup \neg.(U \cap C) \models a$. If $\mathbf{A} \dot\cup \neg.U \not\models a$, then either $\mathbf{F}a \in \mathbf{A}$, which immediately implies $\mathbf{A} \dot\cup \neg.(U \cap C) \not\models a$, or $a \in U$. But in the latter case a must be in C because $U \cap C' = \emptyset$ for all predecessor components C' of C. Therefore $a \in (U \cap C)$ and consequently $\mathbf{A} \dot\cup \neg.(U \cap C) \not\models a$. Because all input atoms a have the same truth value under $\mathbf{A} \dot\cup \neg.U$ and $\mathbf{A} \dot\cup \neg.(U \cap C)$, the same holds also for the positive or default-negated external atom b itself. $\Box$

Proof of Proposition 24. If $U = \emptyset$, then the result holds trivially. By definition of Π_C, we have $H(r) \cap C = \emptyset$ for all $r \in \Pi \setminus \Pi_C$. By hypothesis we have $U \subseteq C$. But then $H(r) \cap U = \emptyset$ for all $r \in \Pi \setminus \Pi_C$ and U is an unfounded set of Π wrt. $\mathbf{A}$. $\Box$

References

Alviano, M., Calimeri, F., Faber, W., Leone, N., & Perri, S. (2011). Unfounded Sets and Well-Founded Semantics of Answer Set Programs with Aggregates. *Journal of Artificial Intelligence Research*, *42*, 487–527.

Baader, F., & Hollunder, B. (1995). Embedding Defaults into Terminological Knowledge Representation Formalisms. *Journal of Automated Reasoning*, *14*(1), 149–180.

Basol, S., Erdem, O., Fink, M., & Ianni, G. (2010). HEX Programs with Action Atoms. In Hermenegildo, M., & Schaub, T. (Eds.), *Technical Communications of the 26th International Conference on Logic Programming (ICLP'10)*, Vol. 7 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 24–33, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Brewka, G., & Eiter, T. (2007). Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems. In Holte, R. C., & Howe, A. (Eds.), *22nd AAAI Conference on Artificial Intelligence (AAAI'07)*, pp. 385–390. AAAI Press.

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

Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., & Schaub, T. (2008). Conflict-driven disjunctive answer set solving. In Brewka, G., & Lang, J. (Eds.), *11th International Conference Principles of Knowledge Representation and Reasoning (KR 2008), Sydney, Australia, September 16-19, 2008*, pp. 422–432. AAAI Press.

Drescher, C., & Walsh, T. (2012). Answer set solving with lazy nogood generation. In Dovier, A., & Costa, V. S. (Eds.), *Technical Communications of the 28th International Conference on Logic Programming (ICLP 2012)*, Vol. 17 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 188–200. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, *77*(2), 321–357.

Dung, P., Mancarella, P., & Toni, F. (2007). Computing ideal sceptical argumentation. *Artificial Intelligence*, *171*, 642–674.

Dunne, P. E. (2009). The computational complexity of ideal semantics. *Artificial Intelligence*, *173*(18), 1559–1591.

Egly, U., Gaggl, S. A., & Woltran, S. (2010). Answer-set programming encodings for argumentation frameworks. *Argument and Computation*, *1*(2), 147–177.

Eiter, T., Fink, M., Ianni, G., Krennwallner, T., & Schüller, P. (2011). Pushing Efficient Evaluation of HEX Programs by Modular Decomposition. In Delgrande, J., & Faber, W. (Eds.), *11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, Vol. 6645 of *LNAI*, pp. 93–106. Springer.

Eiter, T., Fink, M., Krennwallner, T., & Redl, C. (2012a). Conflict-driven ASP Solving with External Sources. *Theory and Practice of Logic Programming*, *12*(4-5), 659–679.

Eiter, T., Fink, M., Krennwallner, T., Redl, C., & Schüller, P. (2012b). Eliminating Unfounded Set Checking for HEX-Programs. In Fink, M., & Lierler, Y. (Eds.), *5th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2012), September 4, 2012, Budapest, Hungary*, pp. 83–97.

Eiter, T., Fink, M., Krennwallner, T., Redl, C., & Schüller, P. (2012c). Exploiting Unfounded Sets for HEX-Program Evaluation. In del Cerro, L. F., Herzig, A., & Mengin, J. (Eds.), *13th European Conference on Logics in Artificial Intelligence (JELIA 2012), September 26-28, 2012, Toulouse, France*, Vol. 7519 of *LNCS*, pp. 160–175. Springer.

Eiter, T., Fink, M., Schüller, P., & Weinzierl, A. (2012b). Finding explanations of inconsistency in nonmonotonic multi-context systems. Tech. rep. INFSYS RR-1843-12-09, INFSYS RR-1843-03-08, Inst. für Informationssysteme, TU Wien. Preliminary version in Proc. 12th International Conference on Knowledge Representation and Reasoning (KR 2010), pp. 329–339, AAAI Press, 2010.

Eiter, T., Ianni, G., Krennwallner, T., & Schindlauer, R. (2008a). Exploiting conjunctive queries in description logic programs. *Annals of Mathematics and Artificial Intelligence*, *53*(1–4), 115–152.

Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2008b). Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, *172*(12-13), 1495–1539.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2005). A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pp. 90–96. Professional Book Center.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2006a). dlvhex: A Prover for Semantic-Web Reasoning under the Answer-Set Semantics. In *Proceedings of the ICLP'06 Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALP-SWS2006)*, pp. 33–39. CEUR WS.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2006). Effective Integration of Declarative Rules with External Evaluations for Semantic-Web Reasoning. In Sure, Y., & Domingue, J. (Eds.), *3rd European Conference on Semantic Web (ESWC'06)*, Vol. 4011 of *LNCS*, pp. 273–287. Springer.

Faber, W. (2005). Unfounded sets for disjunctive logic programs with arbitrary aggregates. In Baral, C., Greco, G., Leone, N., & Terracina, G. (Eds.), *8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, Vol. 3662, pp. 40–52. Springer.

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

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

Gebser, M., Kaufmann, B., & Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, *187–188*, 52–89.

Gebser, M., Kaufmann, B., & Schaub, T. (2013). Advanced conflict-driven disjunctive answer set solving. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pp. 912–918. AAAI Press.

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

Ghidini, C., & Giunchiglia, F. (2001). Local models semantics, or contextual reasoning=locality+compatibility. *Artificial Intelligence*, *127*(2), 221–259.

Goldman, R., & Boddy, M. (1996). Expressive Planning and Explicit Knowledge. In Drabble, B. (Ed.), *3rd International Conference on Artificial Intelligence Planning Systems (AIPS'96)*, pp. 110–117. AAAI Press.

Hoehndorf, R., Loebe, F., Kelso, J., & Herre, H. (2007). Representing default knowledge in biomedical ontologies: application to the integration of anatomy and phenotype ontologies. *BMC Bioinformatics*, *8*, 377.

Janhunen, T., Niemelä, I., Seipel, D., Simons, P., & You, J.-H. (2006). Unfolding partiality and disjunctions in stable model semantics. *ACM Trans. Comput. Log.*, *7*(1), 1–37.

Koch, C., Leone, N., & Pfeifer, G. (2003). Enhancing disjunctive logic programming systems by SAT checkers. *Artificial Intelligence*, *151*(1–2), 177–212.

Lee, J. (2005). A model-theoretic counterpart of loop formulas. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pp. 503–508. Professional Book Center.

Lee, J., & Lifschitz, V. (2003). Loop Formulas for Disjunctive Logic Programs. In Palamidessi, C. (Ed.), *19th International Conference on Logic Programming (ICLP'03)*, Vol. 2916 of *LNCS*, pp. 451–465. Springer.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., & Scarcello, F. (2006). The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, *7*(3), 499–562.

Leone, N., Rullo, P., & Scarcello, F. (1997). Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics, and Computation. *Information and Computation*, *135*(2), 69–112.

Lierler, Y. (2005). cmodels - SAT-Based Disjunctive Answer Set Solver. In Baral, C., Greco, G., Leone, N., & Terracina, G. (Eds.), *8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005)*, Vol. 3662 of *Lecture Notes in Computer Science*, pp. 447–451. Springer.

Lifschitz, V., & Turner, H. (1994). Splitting a logic program. In Hentenryck, P. V. (Ed.), *11th International Conference on Logic Programming (ICLP'94)*, pp. 23–37. MIT Press.

Lin, F., & Zhao, Y. (2004). ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, *157*(1–2), 115–137.

Marano, M., Obermeier, P., & Polleres, A. (2010). Processing RIF and OWL2RL within DLVHEX. In Hitzler, P., & Lukasiewicz, T. (Eds.), *4th International Conference on Web Reasoning and Rule Systems (RR'10)*, Vol. 6333 of *LNCS*, pp. 244–250. Springer.

Nieuwenborgh, D. V., Cock, M. D., & Vermeir, D. (2007a). Computing fuzzy answer sets using dlvhex. In Dahl, V., & Niemelä, I. (Eds.), *23rd International Conference on Logic Programming (ICLP'07)*, Vol. 4670 of *LNCS*, pp. 449–450. Springer.

Nieuwenborgh, D. V., Eiter, T., & Vermeir, D. (2007b). Conditional planning with external functions. In Baral, C., Brewka, G., & Schlipf, J. S. (Eds.), *9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, Vol. 4483 of *LNCS*, pp. 214–227. Springer.

Shen, Y.-D. (2011). Well-supported semantics for description logic programs. In Walsh, T. (Ed.), *22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pp. 1081–1086. AAAI Press.

Shen, Y.-D., & Wang, K. (2011). Extending logic programs with description logic expressions for the semantic web. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N. F., & Blomqvist, E. (Eds.), *10th International Semantic Web Conference (ISWC'11)*, Vol. 7031 of *LNCS*, pp. 633–648. Springer.

Simons, P., Niemelä, I., & Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, *138*(1-2), 181–234.

Smith, D. E., & Weld, D. S. (1998). Conformant Graphplan. In Mostow, J., Rich, C., & Buchanan, B. (Eds.), *15th National Conference on Artificial Intelligence (AAAI'98)*, pp. 889–896. AAAI Press / The MIT Press.

Turner, H. (2002). Polynomial-length planning spans the polynomial hierarchy. In Flesca, S., Greco, S., Leone, N., & Ianni, G. (Eds.), *European Conference on Logics in Artificial Intelligence (JELIA'02)*, Vol. 2424 of *LNCS*, pp. 111–124. Springer.

Van Gelder, A., Ross, K. A., & Schlipf, J. S. (1991). The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, *38*(3), 619–649.

Zakraoui, J., & Zagler, W. L. (2011). A logical approach to web user interface adaptation. In Holzinger, A., & Simonic, K.-M. (Eds.), *7th Conference of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society (USAB'11)*, Vol. 7058 of *LNCS*, pp. 645–656. Springer.

Zirtiloğlu, H., & Yolum, P. (2008). Ranking semantic information for e-government: complaints management. In *1st International Workshop on Ontology-supported Business Intelligence (OBI'08)*, No. 5 in OBI'08, p. 7. ACM.