Explaining Inconsistency in Answer Set Programs and Extensions[⋆]

Christoph Redl

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

Abstract. Answer Set Programming (ASP) is a well-known problem solving approach based on nonmonotonic logic programs. HEX-programs extend ASP with *external atoms* for accessing arbitrary external information. In this paper we study inconsistent ASP- and HEX-programs, i.e., programs which do not possess answer sets, and introduce a novel notion of *inconsistency reasons* for characterizing their inconsistency depending on the input facts. This problem is mainly motivated by upcoming applications for optimizations of the evaluation algorithms for HEX-programs. Further applications can be found in ASP debugging. We then analyze the complexity of reasoning problems related to the computation of such inconsistency reasons. Finally, we present a meta-programming encoding in disjunctive ASP which computes inconsistency reasons for given normal logic programs, and a basic procedural algorithm for computing inconsistency reasons for general HEX-programs.

1 Introduction

Answer-Set Programming (ASP) is a declarative programming paradigm based on nonmonotonic programs and a multi-model semantics [11]. The problem at hand is encoded as an ASP-program, which consists of rules, such that its models, called *answer sets*, correspond to the solutions to the original problem. HEX-programs are an extension of ASP with external sources such as description logic ontologies and Web resources. So-called external atoms pass information from the logic program (given by predicate extensions and constants), to an external source, which in turn returns values to the program. Notably, *value invention* allows for domain expansion, i.e., external sources might return values which do not appear in the program. For instance, the external atom $\&synonym[car](X)$ might be used to find the synonyms X of car, e.g. $automobile$. Also recursive data exchange between the program and external sources is supported, which leads to high expressiveness of the formalism.

Inconsistent programs are programs which do not possess any answer sets. A natural question about such a program is *why* it is inconsistent. In this paper, we address this question by characterizing inconsistency in terms of sets of atoms occurring in the *input facts (EDB; extensional database)* of the program. Such a characterization is motivated by programs whose *proper rules (IDB; intensional database)* are fixed, but which are

[⋆] This research has been supported by the Austrian Science Fund (FWF) project P27730.

evaluated with many different sets of facts. This typically occurs when the proper rules encode the general problem, while the current instance is specified as facts. It is then interesting to identify sets of instances which lead to inconsistency.

A concrete application of inconsistency reasons can be found in the evaluation algorithm for HEX-programs [3]. Here, in order to handle value invention, the overall program is partitioned into multiple *program components* which are arranged in an acyclic *evaluation graph*. Roughly, the program is evaluated by recursively computing answer sets of a program component and adding them as facts to the successor components until the final answer sets of the leaf units are computed; they correspond to the answer sets of the overall program. Then, an explanation for the inconsistency of a lower program component in terms of its input facts can be exploited for skipping evaluations which are known not to yield any answer sets. Other applications can be found in ASP debugging by guiding the user to find reasons why a particular instance or class of instances is inconsistent.

Previous related work in context of answer set program debugging focused on explaining why a *particular interpretation* fails to be an answer set, while we aim at explaining why the overall program is inconsistent. Moreover, previous debugging approaches also focus on explanations which support the *human user* to find errors in the program. Such reasons can be, for instance, in terms of minimal sets of constraints which need to be removed in order to regain consistency, in terms of odd loops (i.e., cycles of mutually depending atoms which involve negation and are of odd length), or in terms of unsupported atoms, see e.g. [1] and [10] for some approaches. To this end, one usually assumes that a single fixed program is given whose undesired behavior needs to be explained. In contrast, we consider a program whose input facts are subject to change and identify classes of instances which lead to inconsistency, which can help users to get better understanding of the structure of the problem.

Inconsistency management has also been studied for more specialized formalisms such as for multi-context systems (MCSs) [5] and for DL-programs [6]. However, the notions of inconsistencies are not directly comparable to ours as they refer to specific elements of the respective formalism (such as bridge rules of MCSs in the former case and the Abox of ontologies in the latter), which do not exist in general logic programs.

In this paper we focus on the study of inconsistency reasons, including the development of a suitable definition, characterizing them, their complexity properties, and techniques for computing them. The results are intended to serve as foundation for the aforementioned applications in future work.

In more detail, our contributions and the organization of this paper is as follows:

- In Section 2 we present the necessary preliminaries on ASP and HEX-programs.
- In Section 3 we define the novel notion of *inconsistency reasons (IRs)* for HEX-programs wrt. a set of atoms. We then provide a characterization of IRs depending on the models and unfounded sets of the program at hand.
- In Section 4 we analyze the complexity of reasoning tasks related to the computation of inconsistency reasons.
- In Section 5 we show how inconsistency reasons can be computed for normal logic programs resp. general HEX-programs using a meta-programming technique resp. a procedural algorithm.

– In Section 6 we discuss envisaged applications, related work and differences to our approach in more detail.
– In Section 7 we conclude and give an outlook on future work.

2 Preliminaries

In this section we recapitulate the syntax and semantics of HEX-programs, which generalize (disjunctive) logic programs under the answer set semantics [11]; for more details and background we refer to [3]. For simplicity, we restrict the formal discussion to ground (variable-free) programs.

A ground atom a is of the form $p(c_1, \dots, c_\ell)$ with predicate symbol p and constant symbols $c_1, \dots, c_\ell$ from a finite set $\mathcal{C}$, abbreviated as $p(\mathbf{c})$; we write $c \in \mathbf{c}$ if $c = c_i$ for some $1 \leq i \leq \ell$. An *assignment* A over the (finite) set $\mathcal{A}$ of atoms is a set $A \subseteq \mathcal{A}$; here $a \in A$ expresses that a is true, also denoted $A \models a$, and $a \notin A$ that a is false, also denoted $A \not\models a$. An assignment satisfies a set S of atoms, denoted $A \models S$, if $A \models a$ for some $a \in S$; it does not satisfy S, denoted $A \not\models S$, otherwise.

HEX-Program Syntax. HEX-programs extend ordinary ASP-programs by *external atoms*, which enable a bidirectional interaction between a HEX-program and external sources of computation. A *ground external atom* is of the form $\&g[\mathbf{p}](\mathbf{c})$, where $\mathbf{p} = p_1, \dots, p_k$ is a list of input parameters (predicate names or object constants), called *input list*, and $\mathbf{c} = c_1, \dots, c_l$ are constant output terms.[1] For such an external atom e with a predicate parameter p, all atoms of form $p(c_1, \dots, c_\ell)$, which appear in the program, are called *input atoms* of the external atom e.

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

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

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

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

In the following, we call a program *ordinary* if it does not contain external atoms, i.e., if it is a standard ASP-program. Moreover, a rule as by Definition 1 is called *normal* if $k = 1$ and a program is called *normal* if all its rules are normal. A rule $\leftarrow b_1, \dots, b_m, \text{not } b_{m+1}, \dots, \text{not } b_n$ (i.e., with $k = 0$) is called a *constraint* and can be seen as normal rule $f \leftarrow b_1, \dots, b_m, \text{not } b_{m+1}, \dots, \text{not } b_n, \text{not } f$ where f is a new atom which does not appear elsewhere in the program.

HEX-Program Semantics. In the following, assignments are over the set $\mathcal{A}(P)$ of ordinary atoms that occur in the program P at hand. The semantics of a ground external atom $\&g[\mathbf{p}](\mathbf{c})$ wrt. an assignment A is given by the value of a $1{+}k{+}l$-ary *two-valued (Boolean) oracle function* $f_{\&g}$ that is defined for all possible values of A, $\mathbf{p}$ and $\mathbf{c}$. Thus,

[1] The distinction is mainly relevant for nonground programs, which we disregard in this paper.

$\&g[\mathbf{p}](\mathbf{c})$ is true relative to A if $f_{\&g}(A, \mathbf{p}, \mathbf{c}) = \mathbf{T}$ and false otherwise. Satisfaction of ordinary rules and ASP-programs [11] is then extended to HEX-rules and -programs as follows. An assignment A satisfies an atom a, denoted $A \models a$, if $a \in A$, and it does not satisfy it, denoted $A \not\models a$, otherwise. It satisfies a default-negated atom $\text{not}\, a$, denoted $A \models \text{not}\, a$, if $A \not\models a$, and it does not satisfy it, denoted $A \not\models \text{not}\, a$, otherwise. A rule r is satisfied under assignment A, denoted $A \models r$, if $A \models a$ for some $a \in H(r)$ or $A \not\models a$ for some $a \in B(r)$. A HEX-program P is satisfied under assignment A, denoted $A \models P$, if $A \models r$ for all $r \in P$.

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

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

Example 1. Consider the HEX-program $P = \{p \leftarrow \&id[p]()\}$, where $\&id[p]()$ is true iff p is true. Then P has the answer set $A_1 = \emptyset$; indeed it is a subset-minimal model of $fP^{A_1} = \emptyset$.

For a given HEX-program P we let $\mathcal{AS}(P)$ denote the set of all answer sets of P.

3 Explaining Inconsistency of HEX-Programs

In this section we consider programs P that are extended by a set of (input) atoms $I \subseteq \mathcal{D}$ from a given domain $\mathcal{D}$, which are added as facts. More precisely, for a given set $I \subseteq \mathcal{D}$, we consider $P \cup \mathit{facts}(I)$, where $\mathit{facts}(I) = \{a \leftarrow \mid a \in I\}$ is the representation of I transformed to facts. This is in spirit of the typical usage of (ASP- and HEX-)programs, where proper rules (IDB; intensional database) encode the problem at hand and are fixed, while facts (EDB; extensional database) may be subject to change as they are used to specify a concrete instance. Our goal is to express the reasons for inconsistency of HEX-program $P \cup \mathit{facts}(I)$ in terms of I. That is, we want to find a sufficient criterion wrt. I which guarantees that $P \cup \mathit{facts}(I)$ does not possess any answer sets.

Formalizing Inconsistency Reasons. Inspired by inconsistency explanations for multi-context systems [5], we propose to make such a reason dependent on atoms from $\mathcal{D}$ which *must occur* resp. *must not occur* in I such that $P \cup \mathit{facts}(I)$ is inconsistent, while the remaining atoms from $\mathcal{D}$ might either occur or not occur in I without influencing (in)consistency. We formalize this idea as follows:

Definition 3 (Inconsistency Reason (IR)). *Let P be a* HEX-*program and $\mathcal{D}$ be a domain of atoms. An* inconsistency reason (IR) *of P wrt. $\mathcal{D}$ is a pair $R = (R^+, R^-)$ of sets of atoms $R^+ \subseteq \mathcal{D}$ and $R^- \subseteq \mathcal{D}$ with $R^+ \cap R^- = \emptyset$ s.t. $P \cup \mathit{facts}(I)$ is inconsistent for all $I \subseteq \mathcal{D}$ with $R^+ \subseteq I$ and $R^- \cap I = \emptyset$.*

Here, R^+ resp. R^- define the sets of atoms which must be present resp. absent in I such that inconsistency of $P \cup \mathit{facts}(I)$ is inconsistent, while atoms from $\mathcal{D}$ which are neither in R^+ nor in R^- might be arbitrarily added or not without affecting inconsistency.

[2] For ordinary Π, these are Gelfond & Lifschitz's answer sets.

Example 2. An IR of the program $P = \{\leftarrow a, \mathrm{not}\, c;\ d \leftarrow b.\}$ wrt. $\mathcal{D} = \{a, b, c\}$ is $R = (\{a\}, \{c\})$ because $P \cup \mathit{facts}(I)$ is inconsistent for all $I \subseteq \mathcal{D}$ whenever $a \in I$ and $c \notin I$, while b can be in I or not without affecting (in)consistency.

In general there are multiple IRs, some of which might not be minimal. For instance, the program $\{\leftarrow a; \leftarrow b\}$ has inconsistency reasons $R_1 = (\{a\}, \emptyset)$, $R_2 = (\{b\}, \emptyset))$ and $R_3 = (\{a, b\}, \emptyset)$ wrt. $\mathcal{D} = \{a, b\}$. On the other hand, a program P might not have any IR at all if $P \cup \mathit{facts}(I)$ is consistent for all $I \subseteq \mathcal{D}$. This is the case, for instance, for the empty HEX-program $P = \emptyset$. However, one can show that there is always at least one IR if $P \cup \mathit{facts}(I)$ is inconsistent for some $I \subseteq \mathcal{D}$.

Proposition 1. *For all* HEX*-programs P and domains $\mathcal{D}$ such that $P \cup \mathit{facts}(I)$ is inconsistent for some set $I \subseteq \mathcal{D}$ of atoms, then there is an IR of P wrt. $\mathcal{D}$.*

Proof. Take some I such that $P \cup \mathit{facts}(I)$ is inconsistent and consider $R = (R^+, R^-)$ with $R^+ = I$ and $R^- = \mathcal{D} \setminus I$. Then the only I' with $R^+ \subseteq I'$ and $R^- \cap I' = \emptyset$ is I itself. But $P \cup \mathit{facts}(I)$ is inconsistent by assumption. $\square$

Characterizing Inconsistency Reasons. Next, we present an alternative characterization of IRs based on the models of a HEX-program P and unfounded sets of P wrt. these models. To this end we use the following definition and result from [9]:

Definition 4 (Unfounded Set). *Given a* HEX*-program P and an assignment A, let U be any set of atoms appearing in P. Then, U is an* unfounded set *for P* wrt. *A if, for each rule $r \in P$ with $H(r) \cap U \neq \emptyset$, at least one of the following conditions holds:*

(i) some literal of $B(r)$ is false wrt. A; or
(ii) some literal of $B(r)$ is false wrt. $A \setminus U$; or
(iii) some atom of $H(r) \setminus U$ is true wrt. A.

A model M of a HEX-program P is called *unfounded-free* if it does not intersect with an unfounded set of P wrt. M. One can then show [9]:

Proposition 2. *A model M of a* HEX*-program P is an answer set of P iff it is unfounded-free.*

In the following, for sets of atoms $\mathcal{H}$ and $\mathcal{B}$, let $\mathcal{P}^e_{\langle \mathcal{H}, \mathcal{B} \rangle}$ denote the set of all HEX-programs whose head atoms come only from $\mathcal{H}$ and whose body atoms and external atom input atoms come only from $\mathcal{B}$. We then use the following result from [12].

Proposition 3. *Let P be a* HEX*-program. Then $P \cup R$ is inconsistent for all $R \in \mathcal{P}^e_{\langle \mathcal{H}, \mathcal{B} \rangle}$ iff for each classical model A of P there is a nonempty unfounded set U of P wrt. A such that $U \cap A \neq \emptyset$ and $U \cap \mathcal{H} = \emptyset$.*

Intuitively, the result states that a program is inconsistent, iff each (classical) model is dismissed as answer set because it is not unfounded-free. Our concept of an inconsistency reason amounts to a special case of this result. More precisely, (R^+, R^-) for sets of atoms $R^+ \subseteq \mathcal{D}$ and $R^- \subseteq \mathcal{D}$ is an IR iff the program is inconsistent for all additions of facts that contain all atoms from R^+ but none from R^-; this can be expressed by applying the previous result for $\mathcal{B} = \emptyset$ and an appropriate selection of $\mathcal{H}$.

Lemma 1. *Let P be a* HEX*-program and $\mathcal{D}$ be a domain of atoms. A pair (R^+, R^-) of sets of atoms $R^+ \subseteq \mathcal{D}$ and $R^- \subseteq \mathcal{D}$ with $R^+ \cap R^- = \emptyset$ is an IR of a* HEX*-program P wrt. $\mathcal{D}$ iff $P \cup \mathit{facts}(R^+) \cup R$ is inconsistent for all $R \in \mathcal{P}^e_{\langle \mathcal{D} \setminus R^-, \emptyset \rangle}$.*

Proof. The claim follows basically from the observation that the sets of programs allowed to be added on both sides are the same. In more detail:

($\Rightarrow$) Let (R^+, R^-) be an IR of P with $R^+ \subseteq \mathcal{D}$ and $R^- \subseteq \mathcal{D}$ with $R^+ \cap R^- = \emptyset$. Let $R \in \mathcal{P}^e_{\langle \mathcal{D} \setminus R^-, \emptyset \rangle}$. We have to show that $P \cup \mathit{facts}(R^+) \cup R$ is inconsistent. Take $I = R^+ \cup \{a \mid a \leftarrow \in R\}$; then $R^+ \subseteq I$ and $R^- \cap I = \emptyset$ and thus by our precondition that (R^+, R^-) is an IR we have that $P \cup \mathit{facts}(I)$, which is equivalent to $P \cup \mathit{facts}(R^+) \cup R$, is inconsistent.

($\Leftarrow$) Suppose $P \cup \mathit{facts}(R^+) \cup R$ is inconsistent for all $R \in \mathcal{P}^e_{\langle \mathcal{D} \setminus R^-, \emptyset \rangle}$. Let $I \subseteq \mathcal{D}$ such that $R^+ \subseteq I$ and $R^- \cap I = \emptyset$. We have to show that $P \cup \mathit{facts}(I)$ is inconsistent. Take $R = \{a \leftarrow \mid a \in I \setminus R^+\}$ and observe that $R \in \mathcal{P}^e_{\langle \mathcal{D} \setminus R^-, \emptyset \rangle}$. By our precondition we have that $P \cup \mathit{facts}(R^+) \cup R$ is inconsistent. The observation that the latter is equivalent to $P \cup \mathit{facts}(I)$ proves the claim. $\square$

Now Proposition 3 and Lemma 1 in combination allow for characterizing IRs in terms of models and unfounded sets.

Proposition 4. *Let P be a ground* HEX*-program and $\mathcal{D}$ be a domain. Then a pair of sets of atoms (R^+, R^-) with $R^+ \subseteq \mathcal{D}$, $R^- \subseteq \mathcal{D}$ and $R^+ \cap R^- = \emptyset$ is an IR of P iff for all classical models M of P either (i) $R^+ \not\subseteq M$ or (ii) there is a nonempty unfounded set U of P wrt. M such that $U \cap M \neq \emptyset$ and $U \cap (\mathcal{D} \setminus R^-) = \emptyset$.*

Proof. ($\Rightarrow$) Let (R^+, R^-) be an IR of P wrt. $\mathcal{D}$. Consider a classical model M of P. We show that one of (i) or (ii) is satisfied.

If M is not a classical model of $P \cup \mathit{facts}(R^+)$, then, since M is a model of P, we have that $R^+ \not\subseteq M$ and thus Condition (i) is satisfied.

In case that M is a classical model of $P \cup \mathit{facts}(R^+)$, first observe that, since (R^+, R^-) is an IR, by Lemma 1 $P \cup \mathit{facts}(R^+) \cup R$ is inconsistent for all $R \in \mathcal{P}^e_{\langle \mathcal{D} \setminus R^-, \emptyset \rangle}$. By Proposition 3, this is further the case iff for each classical model M' of $P \cup \mathit{facts}(R^+)$ there is a nonempty unfounded set U such that $U \cap M' \neq \emptyset$ and $U \cap (\mathcal{D} \setminus R^-) = \emptyset$. Since M is a model of $P \cup \mathit{facts}(R^+)$ it follows that an unfounded set as required by Condition (ii) exists.

($\Leftarrow$) Let (R^+, R^-) be a pair of sets of atoms such that for all classical models M of P either (i) or (ii) holds. We have to show that it is an IR of P, i.e., $P \cup \mathit{facts}(I)$ is inconsistent for all $I \subseteq \mathcal{D}$ with $R^+ \subseteq I$ and $R^- \cap I = \emptyset$.

Assignments which are no classical models of $P \cup \mathit{facts}(I)$ cannot be answer sets, thus it suffices to show for all classical models of $P \cup \mathit{facts}(I)$ that they are no answer sets. Consider an arbitrary but fixed $I \subseteq \mathcal{D}$ with $R^+ \subseteq I$ and $R^- \cap I = \emptyset$ and let M be an arbitrary classical model of $P \cup \mathit{facts}(I)$. Then M is also a classical model of P and thus, by our precondition, either (i) or (ii) holds.

If (i) holds, then there is an $a \in R^+$ such that $a \notin M$. But since $R^+ \subseteq I$ we have $a \leftarrow \in \mathit{facts}(I)$ and thus M cannot be a classical model and therefore no answer set of $P \cup \mathit{facts}(I)$. If (ii) holds, then there is a nonempty unfounded set U of P wrt. M such that $U \cap M \neq \emptyset$ and $U \cap (\mathcal{D} \setminus R^-) = \emptyset$, i.e., U does not contain elements from $\mathcal{D} \setminus R^-$. Then U is also an unfounded set of $P \cup \mathit{facts}(R^+)$ wrt. M. Then by Proposition 3 we have that $P \cup \mathit{facts}(R^+) \cup R$ is inconsistent for all $R \in \mathcal{P}^e_{\langle \mathcal{D} \setminus R^-, \emptyset \rangle}$. The latter is, by Lemma 1, the case iff (R^+, R^-) be an IR of P wrt. $\mathcal{D}$. □

Intuitively, each classical model M must be excluded from being an answer set of $P \cup \mathit{facts}(I)$ for any $I \subseteq \mathcal{D}$ with $R^+ \subseteq I$ and $R^- \cap I = \emptyset$. This can either be done by ensuring that M does not satisfy R^+ or that some atom $a \in M$ is unfounded if I is added because $a \notin \mathcal{D} \setminus R^-$.

While the previous result characterizes inconsistency of a program precisely, it is computationally expensive to apply because one does not only need to check a condition for all models of the program at hand, but also for all unfounded sets of all models. We therefore present a second, simpler condition, which refers only to the program's models but not the the unfounded sets wrt. these models.

Proposition 5. *Let P be a ground* HEX*-program and $\mathcal{D}$ be a domain such that $H(P) \cap \mathcal{D} = \emptyset$. For a pair of sets of atoms (R^+, R^-) with $R^+ \subseteq \mathcal{D}$ and $R^- \subseteq \mathcal{D}$, if for all classical models M of P we either have (i) $R^+ \not\subseteq M$ or (ii) $R^- \cap M \neq \emptyset$ then (R^+, R^-) is an IR of P.*

Proof. Let (R^+, R^-) be a pair of sets of atoms such that for all classical models M of P we either have $R^+ \not\subseteq M$ or $R^- \cap M \neq \emptyset$. We have to show that it is an IR of P, i.e., $P \cup \mathit{facts}(I)$ is inconsistent for all $I \subseteq \mathcal{D}$ with $R^+ \subseteq I$ and $R^- \cap I = \emptyset$.

Assignments which are no classical models of $P \cup \mathit{facts}(I)$ cannot be answer sets, thus it suffices to show for all classical models that they are no answer sets. Let M be an arbitrary classical model of $P \cup \mathit{facts}(I)$. Then M is also a classical model of P and thus, by our precondition, either (i) or (ii) holds.

If (i) holds, then there is an $a \in R^+$ such that $a \notin M$. But since $R^+ \subseteq I$ we have $a \leftarrow \in \mathit{facts}(I)$ and thus M cannot be a classical model and therefore no answer set of $P \cup \mathit{facts}(I)$. If (ii) holds, then there is an $a \in R^- \cap M$. Since $R^- \cap I = \emptyset$ we have that $a \leftarrow \notin \mathit{facts}(I)$. Moreover, we have that $a \notin H(P)$ by our precondition that $H(P) \cap \mathcal{D} = \emptyset$. But then $\{a\}$ is an unfounded set of $P \cup \mathit{facts}(I)$ wrt. M such that $\{a\} \cap M \neq \emptyset$ and thus, by Proposition 2, M is not an answer set of P. □

However, in contrast to Proposition 4, note that Proposition 5 does *not* hold in reverse direction. That is, the proposition does not characterize inconsistency exactly but provides a practical means for identifying inconsistency in some cases. This is demonstrated by the following example.

Example 3. Let $\mathcal{D} = \{x\}$ and $P = \{\leftarrow \mathrm{not}\, a\}$. Then $R = (\emptyset, \emptyset)$ is an IR of P because P is inconsistent and no addition of any atoms as facts is allowed. However, the classical model $M = \{a\}$ contains R^+ but does not intersect with R^-, hence the precondition of Proposition 5 is not satisfied.

4 Computational Complexity

Next, we discuss the computational complexity of reasoning problems in context of computing inconsistency reasons. For our results we assume that oracle functions can be evaluated in polynomial time wrt. the size of their input.

We start with the problem of checking if a candidate IR of a program is a true IR.

Proposition 6. *Given a* HEX*-program P, a domain $\mathcal{D}$, and a pair $R = (R^+, R^-)$ of sets of atoms with $R^+ \subseteq \mathcal{D}$, $R^- \subseteq \mathcal{D}$ and $R^+ \cap R^- = \emptyset$. Deciding if R is an IR of P wrt. $\mathcal{D}$ is (i) Π_2^P-complete in general and (ii) $coNP$-complete if P is an ordinary disjunction-free program.*

Proof. **Hardness:** Checking if P does not have any answer set is a well-known problem that is Π_2^P-complete for (i) and $coNP$-complete for (ii).

We reduce the problem to checking if a given $R = (R^+, R^-)$ is an IR of P wrt. a domain $\mathcal{D}$, which shows that the latter cannot be easier. Consider $R = (\emptyset, \emptyset)$ and domain $\mathcal{D} = \emptyset$. Then the only $I \subseteq \mathcal{D}$ with $\emptyset \subseteq I$ and $\mathcal{D} \cap I = \emptyset$ is $I = \emptyset$. Then R is an IR iff $P \cup \mathit{facts}(\emptyset) = P$ is inconsistent, which allows for reducing the inconsistency check to the check of R for being an IR of P wrt. $\mathcal{D}$.
Membership: Consider $P' = P \cup \{x \leftarrow \mid x \in R^+\} \cup \{x \leftarrow \text{not}\, nx;\ nx \leftarrow \text{not}\, x \mid x \in \mathcal{D} \setminus R^-\}$, where nx is a new atom for all $x \in \mathcal{D}$. Then P' has an answer set iff for some $I \subseteq \mathcal{D}$ with $R^+ \subseteq I$ and $R^- \cap I = \emptyset$ we have that $P \cup \mathit{facts}(I)$ has an answer set. Conversely, P' does not have an answer set, iff for all $I \subseteq \mathcal{D}$ with $R^+ \subseteq I$ and $R^- \cap I = \emptyset$ we have that $P \cup \mathit{facts}(I)$ has no answer set, which is by Definition 3 exactly the case iff R is an IR of P wrt. $\mathcal{D}$. The observation that checking if P' has no answer set – which is equivalent to deciding if R is an IR – is in Π_2^P resp. $coNP$ for (i) resp. (ii), because the property of being ordinary and disjunction-free carries over from P to P', concludes the proof. □

The complexity of deciding if a program has some IR is then as follows:

Proposition 7. *Given a* HEX*-program P and a domain $\mathcal{D}$. Deciding if there is an IR of P wrt. $\mathcal{D}$ is (i) Σ_3^P-complete in general and (ii) Σ_2^P-complete if P is an ordinary disjunction-free program.*

Proof (Sketch). The hardness proofs are done by reducing satisfiability checking of the QBF formulas $\forall \mathbf{X} \exists \mathbf{Y} \forall \mathbf{Z} \phi(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ for case (i) and $\forall \mathbf{X} \exists \mathbf{Y} \phi(\mathbf{X}, \mathbf{Y})$ for case (ii) to checking (non-)existence of an IR of P; to this end, a saturation encoding is used. The membership proofs are based on the previous results. Due to limited space, we provide the details of this lengthy proof in an extended version at http://www.kr.tuwien.ac.at/research/projects/inthex/inconsistencyanalysis-ext.pdf.

We summarize the complexity results in Table 1.

	Program class	
Reasoning problem	Normal ASP-program	General HEX-program
Checking a candidate IR	$coNP$-c	Π_2^P-c
Checking existence of an IR	Π_2^P-c	Π_3^P-c

Table 1: Summary of Complexity Results

For our envisaged application IRs do not need to be minimal, which is why we leave the formal analysis for future work. However, clearly there is a minimal IR iff there is an IR, while the the complexity of checking a candidate minimal IR is less obvious.

5 Computing Inconsistency Reasons

In this section we discuss various methods for computing IRs. For ordinary normal programs we present a *meta-programming* approach which encodes the computation of IRs as a disjunctive program. For general HEX-programs this is not easily possible due to complexity reasons and we present a procedural method instead.

Inconsistency reasons for normal ASP-programs. If the program P at hand is normal and does not contain external atoms, then one can construct a positive disjunctive meta-program M^P which checks consistency of P. There are multiple encodings for M^P (see e.g. [8]) based on the simulation of an answer set solver for normal programs. However, the details of M^P are not relevant for the further understanding and it suffices to assume that for a given normal ASP-program P, program M^P has the following properties:

- M^P is always consistent;
- if P is inconsistent, then M^P has a single answer set $A_{sat} = A(M^P)$ containing all atoms in M^P including a dedicated atom $incons$ which does not appear in P; and
- if P is consistent, then the answer sets of M^P correspond one-to-one to those of P and none of them contains $incons$.

Then, the atom $incons$ in the answer set(s) of M^P represents inconsistency of the original program P. One can then extend M^P in order to compute the inconsistency reasons of P as follows. We construct

$$\begin{aligned}\tau(\mathcal{D}, P) = M^P & \qquad (1)\\ \cup \{a^+ \vee a^- \vee a^x \mid a \in \mathcal{D}\} \cup & \qquad (2)\\ \cup \{a \leftarrow a^+;\ \leftarrow a, a^-;\ a \vee \bar{a} \leftarrow a^x;\ \mid a \in \mathcal{D}\} & \qquad (3)\\ \cup \{\leftarrow \text{not } incons\}, & \qquad (4)\end{aligned}$$

where a^+, a^-, a^x and $\bar{a}$ are new atoms for all atoms $a \in \mathcal{D}$.

Informally, the idea is that rules (2) guess all possible IRs $R = (R^+, R^-)$, where a^+ represents $a \in R^+$, a^- represents $a \in R^-$ and a^x represents $a \notin R^+ \cup R^-$. Rules (3) guess all possible sets of input facts wrt. the currently guessed IR, where $\bar{a}$ represents that a is not a fact. We know that M^P derives $incons$ iff P together with the facts from rules (3) is inconsistent. This allows the constraint (4) to eliminate all candidate IRs, for which not all possible sets of input facts lead to inconsistency.

One can show that the encoding is sound and complete wrt. the computation of IRs:

Proposition 8. *Let P be an ordinary normal program and $\mathcal{D}$ be a domain. Then (R^+, R^-) is an IR of P wrt. $\mathcal{D}$ iff $\tau(\mathcal{D}, P)$ has an answer set $A \supseteq \{a^\sigma \mid \sigma \in \{+, -\}, a \in R^\sigma\}$.*

Proof. The rules (2) guess all possible explanations (R^+, R^-), where a^+ represents $a \in R^+$, a^- represents $a \in R^-$ and a^x represents $a \notin R^+ \cup R^-$. The rules (3) then guess all possible sets of input facts $I \subseteq \mathcal{D}$ with $R^+ \subseteq I$ and $R^- \cap I = \emptyset$ wrt. the previous guess for (R^+, R^-): if $a \in R^+$ then a must be true, if $a \in R^-$ then a cannot be true, and if $a \notin R^+ \cup R^-$ then a can either be true or not.

Now by the properties of M^P we know that $A(M^P)$ is an answer set of M^P iff P is inconsistent wrt. the current facts computed by rules (3). By minimality of answer sets, $A(\tau(\mathcal{D}, P))$ is an answer set of $\tau(\mathcal{D}, P)$.

Rules (4) ensure that also the atoms a^+, a^- and a^x are true for all $a \in \mathcal{D}$. Since M^P is positive, this does not harm the property of being an answer set wrt. M^P. $\square$

We provide a tool which allows for computing inconsistency reasons of programs, which is available from https://github.com/hexhex/inconsistencyanalysis.

Inconsistency reasons for general HEX**-programs**. Suppose the IRs of a program P wrt. a domain $\mathcal{D}$ can be computed by a meta-program; then this meta-program is consistent iff an IR of P wrt. $\mathcal{D}$ exists. Therefore, consistency checking over the meta-program must necessarily have a complexity not lower than the one of deciding existence of an IR of P. However, we have shown in Proposition 7 that deciding if a general HEX-program has an IR is Σ_3^P-complete. On the other hand, consistency checking over a general HEX-program is only Σ_2^P-complete. Thus, unless $\Sigma_3^P = \Sigma_2^P$, computing the IRs of a general HEX-program cannot be polynomially reduced to a meta-HEX-program (using a non-polynomial reduction is possible but uncommon, which is why we do not follow this possibility). We present two possible remedies.

Giving up completeness. For HEX-programs P without disjunctions we can specify an encoding for computing its IRs, which is sound but not complete. To this end, we make use of a rewriting of P to an ordinary ASP-program $\hat{P}$, which was previously used for evaluating HEX-programs. In a nutshell, each external atom $\&g[\mathbf{p}](\mathbf{c})$ in P is replaced by an ordinary *replacement atom* $e_{\&g[\mathbf{p}]}(\mathbf{c})$ and rules $e_{\&g[\mathbf{p}]}(\mathbf{c}) \leftarrow \text{not}\, ne_{\&g[\mathbf{p}]}(\mathbf{c})$ and $ne_{\&g[\mathbf{p}]}(\mathbf{c}) \leftarrow \text{not}\, e_{\&g[\mathbf{p}]}(\mathbf{c})$ are added to guess the truth value of the former external atom. However, the answer sets of the resulting *guessing program* $\hat{P}$ do not necessarily correspond to answer sets of the original program P. Instead, for each answer set it must be checked if the guesses are correct. An answer set $\hat{A}$ of the guessing program $\hat{P}$ is called a *compatible set* of P, if $f_{\&g}(\hat{A}, \mathbf{p}, \mathbf{c}) = \mathbf{T}$ iff $\mathbf{T}e_{\&g[\mathbf{p}]}(\mathbf{c}) \in \hat{A}$ for all external atoms $\&g[\mathbf{p}](\mathbf{c})$ in P. Each answer set of P is the projection of some compatible set of P. For details about the construction of $\hat{P}$ we refer to [4].

One can exploit the rewriting $\hat{P}$ to compute (some) IRs of P. To this end, one constructs $\tau(\mathcal{D}, \hat{P})$ and computes its answer sets. This yields explanations for the inconsistency of the guessing program $\hat{P}$ rather than the actual HEX-program P. The HEX-program P is inconsistent whenever the guessing program $\hat{P}$ is, and every inconsistency reason of $\hat{P}$ is also one of P. Hence, the approach is sound. However, since $\hat{P}$ might be consistent even if P is inconsistent and $\tau(\mathcal{D}, \hat{P})$ might have no answer set and actual explanations are not found, it is not complete. Formally:

Proposition 9. *Let P be a* HEX*-program and $\mathcal{D}$ be a domain. Then each IR of $\hat{P}$ wrt. $\mathcal{D}$ is also an IR of P wrt. $\mathcal{D}$, i.e., the use of $\hat{P}$ is sound wrt. the computation of IRs.*

Proof. As each answer set of P is the projection of a compatible set of $\hat{P}$ to the atoms in P, inconsistency of $\hat{P} \cup facts(I)$ implies inconsistency of $\hat{P} \cup facts(I)$ for all $I \subseteq \mathcal{D}$. $\square$

The following example shows that using $\hat{P}$ for computing IRs is not complete.

Example 4. Consider the inconsistent HEX-program $P = \{p \leftarrow q, \&neg[p]()\}$ and domain $\mathcal{D} = \{q\}$. An IR is $(\{q\}, \emptyset)$. However, the guessing program $\hat{P} \cup \{q \leftarrow\} = \{e_{\&neg[p]} \vee ne_{\&neg[p]};\ p \leftarrow q, e_{\&neg[p]};\ q \leftarrow\}$ is consistent and has the answer set $\hat{A} = \{e_{\&neg[p]}, p, q\}$; therefore $(\{q\}, \emptyset)$ is not an IR (actually there is none).

Algorithm 1: IRComputation

```
Input: A general HEX-program $P$ and a domain $\mathcal{D}$
Output: All IRs of $P$ wrt. $\mathcal{D}$
for all classical models $M$ of $P$ do
    for all $R^+ \subseteq \mathcal{D}$ such that $R^+ \not\subseteq M$ do
        for all $R^- \subseteq \mathcal{D}$ such that $R^+ \cap R^- \neq \emptyset$ do
            Output $(R^+, R^-)$
    for all $R^- \subseteq \mathcal{D}$ such that there is a UFS $U$ of $P$ wrt. $M$ with $U \cap M \neq \emptyset$ and $U \cap (\mathcal{D} \setminus R^-) = \emptyset$ do
        for all $R^+ \subseteq \mathcal{D}$ such that $R^+ \cap R^- \neq \emptyset$ do
            Output $(R^+, R^-)$
```

In the previous example, the reason why no inconsistency reason is found is that $\hat{A}$ is an answer set of $\hat{P}$ but the value of $e_{\&neg[p]}$ guessed for the external replacement atom differs from the actual value of $\&neg[p]()$ under $\hat{A}$, i.e., $\hat{A}$ is not compatible.

Using a procedural algorithm. A sound and complete alternative is using a procedural algorithm which implements Proposition 4; a naive one is shown in Algorithm 1.

Proposition 10. *For a general* HEX*-program P and a domain $\mathcal{D}$, IRComputation$(P, \mathcal{D})$ outputs all IRs of P wrt. $\mathcal{D}$.*

Proof. The algorithm enumerates all pairs of sets (R^+, R^-) as by Proposition 4. □

However, in general efficient computation of IRs is a challenging problem by itself, which calls for algorithmic optimizations. Since this paper focuses on the notion of IRs and properties, such optimizations are beyond its scope and are left for future work.

6 Discussion and Related Work

The notion of inconsistency reasons is motivated by applications which evaluate programs with fixed proper rules under different sets of facts. Then, inconsistency reasons can be exploited to skip evaluations which are known to yield no answer sets.

A concrete application can be found in optimizations of the evaluation algorithm for HEX-programs [3]. Here, in order to handle programs with expanding domains (value invention), the overall program is partitioned into multiple *program components* which are arranged in an acyclic *evaluation graph*, which is evaluated top-down. To this end, each answer set of a program component is recursively added as facts to the successor component until the final answer sets of the leaf units are computed, which correspond to the overall answer sets. We envisage to exploit inconsistency reasons by propagating them as constraints to predecessor components in order to eliminate interpretations, which would fail to be answer sets of the overall program anyway, already earlier. While we have constructed synthetic examples where the technique leads to an exponential speedup, an empirical analysis using real-world applications is left for future work.

Previous work on inconsistency analysis was mainly in the context of debugging of answer set programs and faulty systems in general, based on symbolic diagnosis [13].

For instance, the approach in [14] computes inconsistency explanations in terms of either minimal sets of constraints which need to be removed in order to regain consistency, or of odd loops (in the latter case the program is called *incoherent*). The realization is based on another (meta-)ASP-program. Also the more general approaches in [1] and [10] rewrite the program to debug into a meta-program using dedicated control atoms. The goal is to support the human user to find reasons for undesired behavior of the program. Possible queries are, for instance, why a certain interpretation is not an answer set or why a certain atom is not true in an answer set. Explanations are then in terms of unsatisfied rules, only cyclically supported atoms, or unsupported atoms. Furthermore, previous work introduced decision criteria on the inconsistency of programs [12] which we exploited in Section 3, but did not introduce a notion of IRs.

Because these approaches are based on meta-programming they are, in this respect, similar to ours. However, an important difference is that we do not compute explanations why a particular interpretation fails to be answer set of a fixed program, but we rather analyze inconsistency of a program wrt. a set of possible extensions by facts. Another difference concerns the goal of the approaches: while the aforementioned ones aim at answer set debugging and therefore at human-readable explanations of the inconsistency, ours is *not* intended to be used as a debugging technique for humans to find errors in a program. In view of our envisaged application, it rather aims at machine-processable explanations wrt. input facts (i.e., it identifies unsatisfiable instances).

In contrast to meta-programming approaches, also procedural algorithms have been proposed which explain why a set of atoms is contained in a given answer set resp. not contained in any answer set [2]. However, they do not introduce a formal definition of an explanation but follow a more practical approach by printing human-readable explanations of varying types. Therefore it is not straightforward to transform their explanations into formal constraints, as this is necessary for the envisaged applications.

Inconsistency management has also been studied for more specialized formalisms such as for multi-context systems (MCSs) [5], which are sets of knowledge-bases (implemented in possibly different formalisms and interconnected by so-called *bridge rules*), and DL-programs [6]; *inconsistency diagnoses* for MCSs use pairs of bridge rules which must be removed resp. added unconditionally to make the system consistent. Their notions are not directly comparable to ours as it refer to specific elements of the respective formalism (such as bridge rules of MCSs and the Abox of ontologies), which do not exist in general logic programs. Hence, our notion defines IRs more generically in terms of input facts. However, the main idea of explaining inconsistency is related.

Our complexity results relate to abductive reasoning using sceptical inference [7].

7 Conclusion and Outlook

We have introduced the novel notion of *inconsistency reasons* for explaining why an ASP- or HEX-program is inconsistent in terms of input facts. In contrast to previous notions from the area of answer set debugging, which usually explain why a certain interpretation is not an answer set of a concrete program, we consider programs with fixed proper rules, which are instantiated with various data parts. Moreover, while debugging approaches aim at explanations which support human users, ours was developed with respect to an envisaged application for optimizations in evaluation algorithms.

The next step is to realize this envisaged application. To this end, an efficient technique for computing IRs in the general case is needed. However, due to complexity this is not always possible, hence there there is a tradeoff between the costs of computing inconsistency reasons and potential benefits. The usage of heuristics for estimating both factors appears to be an interesting starting point.

Acknowledgements. The author thanks Markus Bretterbauer, who developed a prototype implementation for computing inconsistency reasons as studied in this paper.

References

1. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: Debugging ASP programs by means of ASP. In: Baral, C., Brewka, G., Schlipf, J. (eds.) Proc. of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning, (LPNMR'07), Tempe, AZ, USA. Lecture Notes in Artif. Intell., vol. 4483, pp. 31–43. Springer (2007)
2. Brain, M., Vos, M.D.: Debugging logic programs under the answer set semantics. In: Vos, M.D., Provetti, A. (eds.) Answer Set Programming, Advances in Theory and Implementation, Proceedings of the 3rd Intl. ASP'05 Workshop, Bath, UK, September 27-29, 2005. CEUR Workshop Proceedings, vol. 142. CEUR-WS.org (2005)
3. Eiter, T., Fink, M., Ianni, G., Krennwallner, T., Redl, C., Schüller, P.: A model building framework for answer set programming with external computations. Theory and Practice of Logic Programming 16(4), 418–464 (2016)
4. Eiter, T., Fink, M., Krennwallner, T., Redl, C., Schüller, P.: Efficient HEX-program evaluation based on unfounded sets. J. Artif. Intell. Res. 49, 269–321 (February 2014)
5. Eiter, T., Fink, M., Schüller, P., Weinzierl, A.: Finding explanations of inconsistency in multi-context systems. Artif. Intell. 216, 233–274 (2014)
6. Eiter, T., Fink, M., Stepanova, D.: Web Reasoning and Rule Systems: 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings, chap. Inconsistency Management for Description Logic Programs and Beyond, pp. 1–3. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-39666-3_1
7. Eiter, T., Gottlob, G., Leone, N.: Semantics and complexity of abduction from default theories. Artificial Intelligence 90(12), 177 – 223 (1997)
8. Eiter, T., Polleres, A.: Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. TPLP 6(1-2), 23–60 (2006)
9. Faber, W.: Unfounded sets for disjunctive logic programs with arbitrary aggregates. In: Proc. of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005), Diamante, Italy, September 5-8, 2005. vol. 3662, pp. 40–52. Springer (2005)
10. Gebser, M., Puehrer, J., Schaub, T., Tompits, H.: A meta-programming technique for debugging answer-set programs. In: Fox, D., Gomes, C.P. (eds.) AAAI-08/IAAI-08 Proceedings. pp. 448–453 (2008), http://publik.tuwien.ac.at/files/PubDat_167810.pdf
11. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9(3–4), 365–386 (1991)
12. Redl, C.: On equivalence and inconsistency of answer set programs with external sources. In: Proceedings of the Thirty-First AAAI Conference (AAAI 2017), February, 2017, San Francisco, California, USA. AAAI Press (February 2017).
13. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. 32(1), 57–95 (Apr 1987)
14. Syrjänen, T.: Debugging inconsistent answer set programs. In: Proceedings of the 11th International Workshop on Non-Monotonic Reasoning. pp. 77–84. Lake District, UK (May 2006)