Incremental Answer Set Programming: A Preliminary Report

Mona Gharib[1], Robert Mercer[2], and Torsten Schaub[1⋆]

[1] Institut für Informatik, Universität Potsdam, August-Bebel-Str. 89, D-14482 Potsdam, Germany
[2] Cognitive Engineering Laboratory, Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7

Abstract. An alternative approach to *answer set programming* is developed in order to remedy the problems of "relevance" and "modularity" of answer set semantics for logic programs. The fundamental idea is to re-introduce monotonicity. In particular, we introduce the alternative concept of *ι-answer sets*, which are characterized by their applied rules. Furthermore, we develop a sound and complete theorem proving method for our incremental approach to answer set programming, which allows for query-oriented computations. Our proof procedure can manage negation in logic programming. Moreover, it enables us to deal with variables using unification instead of grounding.

1 Introduction

Answer Set Programming (ASP; [1]) was originally conceived as a declarative semantics for logic programming [2]. This is achieved by a bottom-up approach that is free of any procedural flavor. However, now that ASP has become an attractive tool for knowledge representation and reasoning, it is sometimes desirable to have top-down constructions that allow for query-answering (see eg. [3] for an application motivated approach). However, classical ASP and query-orientation appear to be completely diametrical. As detailed in [4], "*STABLE*[ASP] *does not allow for a goal-oriented computation*". Related to this is the fact that answer set semantics is neither "relevant" nor "modular". The principle of relevance states that the truth value of an atom only depends on the subprogram connected to this atom in the underlying dependency graph [5]. Accordingly, modularity stipulates that the semantics of the overall program can be composed of the semantics of subprograms (connected in the dependency graph [5]).

In this paper, we investigate an alternative approach to ASP that allows for query-oriented computation. In particular, we are interested in *incremental* constructions supporting relevance and modularity. This is accomplished via the definition of a *ι-answer set*. The respective concept of ι-answer sets is characterized by applied rules, which can be determined one by one in the construction of a ι-answer set. This incremental character guarantees the existence of ι-answer sets for every logic program and facilitates their computation.

⋆ Affiliated with Simon Fraser University, Canada, and Griffith University, Australia.

Moreover, our incremental approach to ASP allows us to overcome another bottleneck given by ground instantiation. We introduce a sound and complete theorem proving method for our incremental approach to ASP, which allows for query-oriented computations. Furthermore, we consider logic programs with variables and show how our proof procedure enables us to deal with variables using unification instead of grounding. Also, it can manage negation in logic programming. That is, the procedure can return an answer also in cases where SLDNF resolution would flounder [6].

The outline of the paper is as follows. Section 2 provides some basic concepts. In Section 3, we introduce ι-answer sets and compare them to (standard) answer sets. In Section 4, we introduce a proof procedure for ι-answer sets. In Section 5, we consider logic programs with variables and show how our proof procedure enables us to deal with variables using unification instead of grounding. Finally, we conclude with Section 6.

2 Background

A *(normal) logic program* is a finite set of *rules* of the form

$$p_0 \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n \tag{1}$$

where $n \geq m \geq 0$, and each p_i ($0 \leq i \leq n$) is an *atom*. Given a rule r as in (1), we denote the *body* of r by $body(r) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$ and the *head* of r by $head(r) = p_0$. Furthermore, we let $body^+(r) = \{p_1, \ldots, p_m\}$ and $body^-(r) = \{p_{m+1}, \ldots, p_n\}$ be the *positive* and *negative* body of r, respectively. For a logic program Π, we let $body^+(\Pi) = \bigcup_{r \in \Pi} body^+(r)$, $body^-(\Pi) = \bigcup_{r \in \Pi} body^-(r)$, and $head(\Pi) = \{head(r) \mid r \in \Pi\}$. A *literal* is either an atom or a negated atom. We denote the set of all atoms occurring in Π by $Atm(\Pi)$.

A program is called *basic* if $body^-(r) = \emptyset$ for all $r \in \Pi$. A set X of atoms is *closed* under a basic program Π if, for any $r \in \Pi$, $head(r) \in X$ if $body^+(r) \subseteq X$. The smallest set of atoms closed under a basic program Π is denoted by $Cn(\Pi)$. The *reduct* of a logic program Π relative to a set X of atoms is

$$\Pi^X = \{head(r) \leftarrow body^+(r) \mid r \in \Pi, body^-(r) \cap X = \emptyset\}.$$

A set X of atoms is an *answer set* of Π if $X = Cn(\Pi^X)$. For a program Π, we let $Cn^+(\Pi) = Cn(\Pi^\emptyset)$. Note that $\Pi^\emptyset = \{head(r) \leftarrow body^+(r) \mid r \in \Pi\}$. For a program Π and a set X of atoms, the *generating rules* of X for Π are

$$\mathcal{R}_\Pi(X) = \{r \in \Pi \mid body^+(r) \subseteq X, body^-(r) \cap X = \emptyset\}.$$

One can show that X is an answer set of Π iff $Cn^+(\mathcal{R}_\Pi(X)) = Cn(\Pi^X) = X$. A set X of atoms is a *model* of Π if, for all $r \in \Pi$, we have $head(r) \in X$ if $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$.

3 ι-Answer Sets

The construction of answer sets is non-modular, that is, all rules of a program need to be inspected. It is thus impossible to incrementally construct an answer set or to locally validate a construction, like a proof, by only looking at a subset of the given program.

In the area of default logics, Łukaszewicz defined justified extensions [7] to overcome these problems. This leads us to *ι-answer sets* [8], where ι indicates the incremental flavor.

Definition 1. *Let Π be a logic program.*

A set X of atoms is a ι-answer set of Π if $X = Cn^+(\Pi')$ for some $\subseteq$-maximal $\Pi' \subseteq \Pi$ such that

(i) $body^+(\Pi') \subseteq Cn^+(\Pi')$ *and*
(ii) $body^-(\Pi') \cap Cn^+(\Pi') = \emptyset$.

Definition 1 characterizes a ι-answer set X of Π in terms of the rules that are applied wrt X. The set Π' of such rules is maximal among all subsets of Π that satisfy conditions (i) and (ii). Condition (i) guarantees that the positive bodies of rules in Π' are justified, while condition (ii) makes sure that the rules in Π' do not block one another.

For illustration, consider the following program Π_1:

$$\begin{array}{l} r_1 : a \leftarrow not\ d \\ r_2 : b \leftarrow not\ e \\ r_3 : c \leftarrow a, b \\ r_4 : e \leftarrow not\ a \end{array} \tag{2}$$

The ι-answer sets of Π_1 are $\{a, b, c\}$ and $\{e\}$. For $\{a, b, c\}$, the corresponding maximal subset of Π_1 is $\Pi_1' = \{r_1, r_2, r_3\}$. We have

$$\begin{array}{l} body^+(\Pi_1') = \{a, b\} \subseteq \{a, b, c\} = Cn^+(\Pi_1'), \text{and} \\ body^-(\Pi_1') \cap Cn^+(\Pi_1') = \{d, e\} \cap \{a, b, c\} = \emptyset. \end{array}$$

Since $body^-(r_4) \cap Cn^+(\Pi_1') = \{a\} \cap \{a, b, c\} = \{a\} \neq \emptyset$, Π_1' is indeed a maximal subset of Π_1 such that condition (i) and (ii) in Definition 1 hold. For the other ι-answer set $\{e\}$, one can verify that $\{r_4\}$ is maximal satisfying condition (i) and (ii).

For a program Π and a set X of atoms, we let the *applied rules* of X for Π be

$$\mathcal{A}_\Pi(X) = \{r \in \Pi \mid body^+(r) \subseteq X, body^-(r) \cap X = \emptyset, head(r) \in X\}.$$

For a ι-answer set X of Π, the applied rules $\mathcal{A}_\Pi(X)$ correspond to Π' in Definition 1. Note that $\mathcal{A}_\Pi(X) \subseteq \mathcal{R}_\Pi(X)$ for any program Π and any set X of atoms, but not vice versa. For instance, observe that $\mathcal{A}_{\Pi_1}(\{e\}) = \{r_4\} \subset \{r_1, r_4\} = \mathcal{R}_{\Pi_1}(\{e\})$.

The following result shows that ι-answer sets can be constructed incrementally, as the corresponding applied rules underlie monotonicity.

Theorem 1. *Let Π be a logic program and X be a set of atoms such that $X = Cn^+(\mathcal{A}_\Pi(X))$.*

Then, there is a ι-answer set X' of Π such that $X \subseteq X'$.

For constructing a ι-answer set, we can thus start with the empty set, whose applied rules are empty as well, and pick "applicable" rules one by one until no further rule can be added without violating either condition (i) or (ii) in Definition 1.

For illustration, consider the following program Π_2:

$$\begin{array}{l} r_1 : a \leftarrow \\ r_2 : b \leftarrow not\ a \end{array} \tag{3}$$

This program has two ι-answer sets: $\{a\}$ and $\{b\}$. As we see, the monotonic inference of a does not override the nonmonotonic inference of b to eliminate the second ι-answer set $\{b\}$ of Π_2.

As a consequence of Theorem 1, every program yields some ι-answer set.

Corollary 1. *For every logic program Π, there is some ι-answer set X of Π.*

To see this, consider the program $\Pi = \{a \leftarrow not\ a\}$. This program has no answer sets, but it has $\emptyset$ as its unique ι-answer set.

As there are programs that do not have any (standard) answer sets, it is clear that ι-answer sets are not necessarily answer sets.

The converse does however hold, that is, any answer set is as well a ι-answer set.

Theorem 2. *Let Π be a logic program and X be an answer set of Π.*
Then, X is a ι-answer set of Π.

The ι-answer sets of a program Π do thus form a superset of the answer sets of Π.

A ι-answer set is also an answer set if its applied and generating rules are identical.

Theorem 3. *Let Π be a logic program and X be a ι-answer set of Π.*
Then, X is an answer set of Π iff $\mathcal{A}_\Pi(X) = \mathcal{R}_\Pi(X)$.

For ι-answer set $\{a, b, c\}$ of Π_1 in (2), we have $\mathcal{A}_{\Pi_1}(\{a, b, c\}) = \{r_1, r_2, r_3\} = \mathcal{R}_{\Pi_1}(\{a, b, c\})$, that is, $\{a, b, c\}$ is an answer set of Π_1. In contrast, we have $\mathcal{A}_{\Pi_1}(\{e\}) = \{r_4\} \neq \{r_1, r_4\} = \mathcal{R}_{\Pi_1}(\{e\})$, thus, the ι-answer set $\{e\}$ of Π_1 is not an answer set of Π_1. In fact, set $\{e\}$ is not a model of Π_1 because rule r_1 is not satisfied.

Based on the concept of models, we can reformulate Theorem 3 as follows.

Theorem 4. *Let Π be a logic program and X be a ι-answer set of Π.*
Then, X is an answer set of Π iff X is a model of Π.

Any answer set of a program is as well a model of the program, while ι-answer sets are not necessarily models. In fact, condition (ii) in Definition 1 allows for denying the application of a rule if the head would block some applied rule. However, a ι-answer set that is a model of the given program is also an answer set. The property of being a model distinguishes answer sets from ι-answer sets.

In [9], we have extended our framework with *integrity constraints*, providing us with versatile means to filter ι-answer sets. For instance, we can use integrity constraints to deny any ι-answer set that is not a model. But, in this work we concentrate on developing a sound and complete theorem proving method for our incremental approach to answer set programming, which allows for query-oriented computations.

4 Query-Answering

Our primary goal is to furnish a theory for answer set programming that allows for local computations based on top-down query answering. So the first question is how to identify a substructure of a given program that allows for deriving a query. To answer it, we introduce the concept of a *ι-proof*.

Definition 2. *Let Π be a logic program and p be an atom.*
We define Π' as a ι-proof for p from Π iff $\Pi' \subseteq \Pi$ such that

(i) $p \in head(\Pi')$,
(ii) $body^+(\Pi') \subseteq Cn^+(\Pi')$, *and*
(iii) $body^-(\Pi') \cap Cn^+(\Pi') = \emptyset$.

That is, any ι-proof for an atom p consists of rules allowing to derive p.

Reconsider program Π_1 in (2) having two ι-answer sets: $\{a, b, c\}$ and $\{e\}$. Atom a has ι-proof $\Pi_1' = \{r_1\} = \{a \leftarrow not\ d\}$. We have

$$\begin{array}{l} a \in head(\Pi_1'), \\ body^+(\Pi_1') = \emptyset \subseteq \{a\} = Cn^+(\Pi_1'), \text{and} \\ body^-(\Pi_1') \cap Cn^+(\Pi_1') = \{d\} \cap \{a\} = \emptyset. \end{array}$$

Also, we get that $\Pi_2' = \{r_2\}$, $\Pi_3' = \{r_1, r_2, r_3\}$, and $\Pi_4' = \{r_4\}$ are ι-proofs for b, c, and e from Π_1, respectively.

The existence of a ι-proof for an atom p from a program Π furnishes a necessary and sufficient condition for guaranteeing the existence of a ι-answer set of Π containing p.

Theorem 5. *Let Π be a logic program and p be an atom.*
Then, p has a ι-proof from Π iff $p \in X$ for some ι-answer set X of Π.

For example, reconsider program $\Pi = \{a \leftarrow not\ a\}$. According to Definition 2, a has no ι-proof from Π. Thus, Π has no ι-answer set containing a. Indeed, Π has $\emptyset$ as its unique ι-answer set.

We now introduce a calculus along with the concept of a derivation such that, given a logic program Π and a goal g, the derivation succeeds iff g holds with respect to a ι-answer set of Π. A *goal* g is of the form $\leftarrow p_1, \ldots, p_n$, where p_i $(1 \leq i \leq n)$ is an atom. A *context* C is a set of literals.

Definition 3. *Let g be a goal and C be a context.*
We define the ordered pair (g, C) as a contextual goal; g is said to be a goal in context C.

We denote the set of positive atoms in C by C^+ and the set of atoms preceded by not by C^-. A *selection function* is a function from a set of goals to a set of atoms that maps a goal to one of its contained atoms.

In the following, we introduce a proof procedure analogous to SLD resolution [6].

Definition 4. *Let Π be a logic program, f be a selection function, and (g, C) be a contextual goal.*

Then, a contextual goal (g', C') is derivable from (g, C) and a rule $r \in \Pi$ via f, if the following conditions hold:

(i) $p = f(g)$,
(ii) $p = head(r)$,
(iii) $g' = \leftarrow (body(g) \backslash \{p\}) \cup body^+(r)$,
(iv) $C' = (C \cup body^-(r) \cup \{p\})$,
(v) $C'^+ \cap C'^- = \emptyset$.

Definition 5. *Let Π be a logic program, f be a selection function, and (g, C) be a contextual goal.*

A ι-derivation of g from Π in context C via f consists of a finite sequence $\langle (g_0, C_0), \ldots, (g_n, C_n) \rangle$ of contextual goals and a finite sequence of rules $\langle r_1, \ldots, r_n \rangle$ such that each (g_{i+1}, C_{i+1}) is derived from (g_i, C_i) and $r_{i+1} \in \Pi$ via f, where $0 \leq i < n$ and $(g_0, C_0) = (g, C)$.

A *successful* ι-derivation is one that ends in an ordered pair with an empty goal (i.e. $body(g) = \emptyset$), otherwise, it is called a *failed* ι-derivation. We refer to the last ordered pair in a successful ι-derivation of g from Π in context C via f by $(\Box, C_\Box)$, where $C_\Box$ is the projective mapping

$$C_\Box : \langle (g_0, C_0), \ldots, (\Box, C_n) \rangle \mapsto C_n.$$

For illustration, consider the following logic program Π_3:

$$\begin{array}{l} r_1 : p \leftarrow \\ r_2 : q \leftarrow p, not\ r \\ r_3 : q \leftarrow r, not\ p \\ r_4 : r \leftarrow p, not\ s \end{array} \tag{4}$$

and the contextual goal $(\leftarrow q, \emptyset)$. A successful ι-derivation for q from Π_3 in context $\emptyset$ is:

(1)	$(\leftarrow q, \emptyset)$	
(2)	$(\leftarrow p, \{q, not\ r\})$	derived from (1) and r_2
(3)	$(\leftarrow \Box, \{p, q, not\ r\})$	derived from (2) and r_1
		success.

A failed ι-derivation for q from Π_3 in context $\emptyset$ is:

(1)	$(\leftarrow q, \emptyset)$	
(2)	$(\leftarrow r, \{q, not\ p\})$	derived from (1) and r_3
(3)	$(\leftarrow p, \{r, q, not\ s, not\ p\})$	derived from (2) and r_4
	$\ldots$	failure.

The following result shows the soundness and completeness of our derivation procedure.

Theorem 6. *Let Π be a logic program, f be a selection function, and p be an atom.*

Then, there exists a ι-proof of p from Π iff there exists a successful ι-derivation of p from Π in context $\emptyset$ via f.

The following result is obtained from Theorems 5 and 6.

Corollary 2. *Let Π be a logic program, f be a selection function, and p be an atom.*

Then, p has a successful ι-derivation from Π in context $\emptyset$ via f iff $p \in X$ for some ι-answer set X of Π.

The derivations from (g, C) to $(\Box, C_{\Box})$ can be viewed as search trees, which we call *ι-trees*. These trees are obtained as follows.

Definition 6. *Let Π be a logic program, f be a selection function, and (g, C) be a contextual goal.*

A ι-tree of g from Π in context C via f is a tree satisfying the following conditions:

(i) Each node of the tree is a contextual goal.
(ii) The root node is (g, C).
(iii) If (g', C') is a node of the tree and (g'', C'') is a child of (g', C'), then (g'', C'') is derivable from (g', C') and some rule $r \in \Pi$ via f.

Each branch of a ι-tree is a derivation of g from Π in context C via f. Branches corresponding to successful derivations are called *success* branches and branches corresponding to failed derivations are called *failure* branches.

Reconsider program Π_3 in (4) and contextual goal $(\leftarrow q, \emptyset)$. Figure 1 shows the ι-tree of q from Π_3 in context $\emptyset$. Note that in Figure 1, we identify one successful ι-derivation of q from Π_3 in context $\emptyset$.

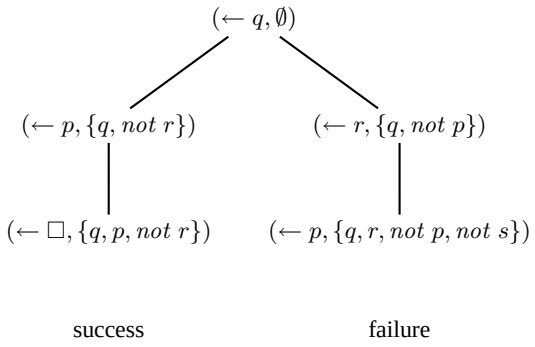

Fig. 1. The ι-tree of $(\leftarrow q, \emptyset)$ from Π_3 in (4).

5 Logic Programs with Variables

In this section, we show how ι-proofs can serve as a basis for query answering on (non-ground) logic programs with variables by using unification instead of ground instantiations.

For a logic program Π, its *Herbrand universe* U^Π is the set of constants in Π and its *Herbrand base* B^Π is the set of ground atoms constructible from U^Π. The *ground instances* of a rule $r \in \Pi$ are the ground rules obtained by replacing all variables in r by elements from U^Π:

$$ground(r) = \{r\theta \mid \theta : var(r) \to U^\Pi\}$$

where $var(r)$ stands for the set of all variables occurring in r; θ is a (ground) substitution. We define the *ground instantiation* of Π as:

$$ground(\Pi) = \{ground(r) \mid r \in \Pi\}.$$

We use the logic programming convention that terms beginning with a capital letter are variables.

For illustration, consider the following (non-ground) logic program Π_4:

$$\begin{array}{c} arc(a,b) \leftarrow \\ nonterminal(X) \leftarrow arc(X,Y) \end{array} \tag{5}$$

We have

$U^{\Pi_4} = \{a, b\}$ and

$B^{\Pi_4} = \{arc(a,a), arc(a,b), arc(b,a), arc(b,b), nonterminal(a), nonterminal(b)\}$.

By instantiating all variables in the second rule, we obtain $ground(\Pi_4)$ as follows:

$$\begin{array}{c} arc(a,b) \leftarrow \\ nonterminal(a) \leftarrow arc(a,a) \\ nonterminal(a) \leftarrow arc(a,b) \\ nonterminal(b) \leftarrow arc(b,a) \\ nonterminal(b) \leftarrow arc(b,b) \end{array}$$

For computing answer sets of (non-ground) logic programs with variables, the size of ground instantiations is critical because the available methods for computing answer sets, e.g., [10–16], require ground rules. Clearly, producing ground instantiations can lead to space problems.

In the following, we generalize the definition of a ι-derivation to non-ground programs as follows.

Definition 7. *Let Π be a (non-ground) logic program, f be a selection function, and (g, C) be a contextual goal.*

Then, a contextual goal (g', C') is derivable from (g, C) and a variant v of a rule $r \in \Pi$ via f using the most general unifier θ (mgu for short) if the following conditions hold:

(i) $p = f(g)$,
(ii) θ *is the mgu of* p *and* $head(v)$,
(iii) $g' = \ \leftarrow ((body(g) \backslash \{p\}) \cup body^+(v))\theta$,
(iv) $C' = (C \cup body^-(v) \cup \{p\})\theta$,
(v) $C'^+ \cap C'^- = \emptyset$.

Definition 8. *Let Π be a (non-ground) logic program, f be a selection function, and (g, C) be a contextual goal.*

A ι-derivation of g from Π in context C via f consists of a finite sequence $\langle (g_0, C_0), \ldots, (g_n, C_n) \rangle$ of contextual goals, a finite sequence of variants $\langle v_1, \ldots, v_n \rangle$ of rules in Π, and a finite sequence $\langle \theta_1, \ldots, \theta_n \rangle$ of mgus such that each (g_{i+1}, C_{i+1}) is derivable from (g_i, C_i) and v_{i+1} via f using θ_{i+1}, where $0 \leq i < n$ and $(g_0, C_0) = (g, C)$.

A ι-derivation is *successful* if the last goal is empty; in this case, we say that the goal g has a successful ι-derivation from Π in context C via f using mgu θ. The composition of the mgus computed during the derivation, restricted to the variables of g, is called an *answer substitution*.

We extend the definition of ι-trees to non-ground logic programs as follows.

Definition 9. *Let Π be a (non-ground) logic program, f be a selection function, and (g, C) be a contextual goal.*

A ι-tree of g from Π in context C via f is a tree satisfying the following conditions:

(i) Each node of the tree is a contextual goal.
(ii) The root node is (g, C).
(iii) If (g', C') is a node of the tree and (g'', C'') is a child of (g', C'), then (g'', C'') is derivable from (g', C') and a variant of some rule in Π via f using an mgu θ.

For illustration, consider the (non-ground) program Π_5:

$$\begin{array}{ll} r_1 : p(s,t) \leftarrow & \\ r_2 : \quad q(t) \leftarrow not\ r(X) & \\ r_3 : \ \ r(Y) \leftarrow p(X,Y), not\ q(X) & \end{array} \tag{6}$$

and ground atom $r(t)$. Figure 2 shows the ι-tree of $r(t)$ from Π_5 in context $\emptyset$. Thus, we conclude that a ι-proof Π_5' of $r(t)$ from Π_5 is

$$\begin{array}{l} p(s,t) \leftarrow \\ \quad r(t) \leftarrow p(s,t), not\ q(s) \end{array}$$

which contains all ground rules used in the successful ι-derivation. By adding the rule $q(t) \leftarrow not\ r(s)$ to Π_5', we obtain the consequences $\{p(s,t), r(t), q(t)\}$, which is the unique ι-answer set of Π_5 containing $r(t)$.

The results for the ground setting can be extended to the general case as follows.

Theorem 7. (Soundness) *Let Π be a (non-ground) logic program, g be a goal, and f be a selection function.*

If g has a successful ι-derivation from Π in context $\emptyset$ via f with answer substitution θ, then, for all ground substitutions σ, we have $g\theta\sigma \in X$ for some ι-answer set X of Π.

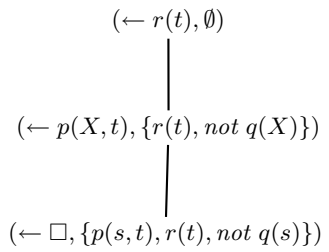

Fig. 2. The ι-tree of $(\leftarrow r(t), \emptyset)$ from Π_5 in (6).

Theorem 8. (Completeness) *Let Π be a (non-ground) logic program, g be a goal, and f be a selection function.*

If for some ground substitution σ, we have $g\sigma \in X$ for some ι-answer set X of Π, then g has a successful ι-derivation from Π in context $\emptyset$ via f and some answer substitution θ more general than σ.

Finally, we want to stress the ability of our ι-derivation procedure to manage negation in logic programming. For instance, consider the following logic program $\Pi = \{a \leftarrow not\ b;\ b \leftarrow not\ a\}$. By applying a solver based on SLDNF resolution [6], the goal $\leftarrow a$ produces an infinite loop because of the circularity of rules (a is defined by b and b is defined by a). This circularity does not lead to an infinite loop in our ι-derivation procedure because it uses a context for storing positive and negative sub-goals, and for identifying contradictions. A meta-interpreter based upon the ι-derivation calculus has been implemented in *ECLiPSe*-Prolog [17].

6 Conclusion

In this work, we have introduced an alternative approach to answer set programming. This is accomplished via the definition of a ι-answer set. Every logic program has at least one ι-answer set, which can be constructed incrementally based on applicable rules. The ι-answer sets of a logic program form a superset of its (standard) answer sets. Based on the concept of models, we have shown that a ι-answer set that is a model of the given program is also an answer set.

We have furnished a theory for answer set programming that provides local computations via top-down query answering. We have developed a sound and complete proof procedure for our incremental approach to answer set programming that allows for query-oriented computations. Important properties and advantages of our procedure are: It can be used to decide whether an atom belongs to some ι-answer set of a program. It is able to manage negation in logic programming. Furthermore, it enables us to deal with logic programs containing variables by using unification instead of ground instantiation.

A resolution calculus for skeptical stable model semantics is given in [18]. Interestingly, this calculus is not derived from credulous inference; also, it does not require the given program to be instantiated before reasoning.

We remark also that our theorem prover can be the basis of a method for computing ι-answer sets of logic programs. The idea is based on ι-proofs. Suppose that we are interested in checking whether a ι-answer set of a program Π contains a given atom p. Then, in the first step, we can determine a ι-proof Π' for p from Π. In the second step, we can add rules to Π' until we obtain the set of rules generating some ι-answer set containing p. This approach has the advantage to focus only on ι-answer sets that contain the desired atom.

References

1. Baral, C.: Knowledge representation, reasoning and declarative problem solving with Answer sets. Cambridge University Press (2003)
2. Gelfond, M., Lifschitz, V.: Logic programs with classical negation. In: Proceedings of the International Conference on Logic Programming. (1990) 579–597
3. Cumbo, C., Faber, W., Greco, G.: Improving query optimization for disjunctive datalog. In Buccafurri, F., ed.: Proceedings of the Joint Conference on Declarative Programming (AGP'03). (2003) 252–262
4. Dix, J., Furbach, U., Niemelä, I.: Nonmonotonic reasoning: Towards efficient calculi and implementations. In Voronkov, A., Robinson, A., eds.: Handbook of Automated Reasoning, Elselvier and MIT Press (2001) 1241–1354
5. Dix, J.: A framework for representing and characterizing semantics of logic programs. In Nebel, B., Rich, C., Swartout, W., eds.: Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning, Morgan Kaufmann Publishers (1992) 591–602
6. Lloyd, J.: Foundations of Logic Programming. Springer-Verlag (1987)
7. Łukaszewicz, W.: Considerations on default logic: An alternative approach. Computational Intelligence **4** (1988) 1–16
8. Delgrande, J., Gharib, M., Mercer, R., Risch, V., Schaub, T.: Łukaszewicz-style answer set programming: A preliminary report. In De Vos, M., Provetti, A., eds.: Proceedings of the Second International Workshop on Answer Set Programming (ASP'03). Volume 78 of CEUR Workshop Proceedings. (2003)
9. Gebser, M., Gharib, M., Schaub, T.: Incremental answer sets and their computation. In: Proceedings of the Fourth International Workshop on Answer Set Programming (ASP'07). (2007) To appear.
10. Cholewiński, P., Marek, V., Truszczyński, M.: Default reasoning system DeReS. In: Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning, Morgan Kaufmann Publishers (1996) 518–528
11. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
12. Leone, N., Faber, W., Pfeifer, G., Eiter, T., Gottlob, G., Koch, C., Mateis, C., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7**(3) (2006) 499–562
13. Lin, F., Zhao, Y.: Assat: computing answer sets of a logic program by sat solvers. Artificial Intelligence **157**(1-2) (2004) 115–137
14. Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. Journal of Automated Reasoning **36**(4) (2006) 345–377

15. Anger, C., Gebser, M., Linke, T., Neumann, A., Schaub, T.: The nomore++ approach to answer set solving. In Sutcliffe, G., Voronkov, A., eds.: Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2005), Springer-Verlag (2005) 95–109
16. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In Veloso, M., ed.: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07), AAAI Press/The MIT Press (2007) 386–392
17. `http://eclipse.crosscoreop.com/`
18. Bonatti, P.: Resolution for skeptical stable model semantics. Journal of Automated Reasoning **27**(4) (2001) 391–421