Incremental Answer Sets and Their Computation

Martin Gebser, Mona Gharib, and Torsten Schaub*

Institut für Informatik, Universität Potsdam, August-Bebel-Str. 89, D-14482 Potsdam, Germany

Abstract. In answer set programming, the existence of an answer set for a logic program is not guaranteed. In order to remedy this problem, we utilize the alternative concept of ι-answer sets, which are characterized by their applied rules. The ι-answer sets of a logic program amount to the justified extensions of the default theory corresponding to the program. On the one hand, every logic program has at least one ι-answer set, which can be constructed incrementally based on applicable rules. On the other hand, a ι-answer set may lack characteristic properties of standard answer sets, such as being a model of the given program. We show how integrity constraints can be used to re-establish such properties, even up to correspondence with standard answer sets. Furthermore, we introduce a translation from logic programs to propositional formulas that modifies Clark's completion to preserve the ι-answer sets of a given program. Based on our notion of program completion, we present a DPLL-like algorithm for computing the ι-answer sets of a logic program that satisfy a given set of integrity constraints.

1 Introduction

Answer Set Programming (ASP; [1]) has emerged as an attractive paradigm for declarative problem solving [2–4]. Originally, it was developed as a declarative branch of logic programming [5], where the semantics of logic programs is given by their answer sets [6]. The answer set semantics is closely related to other nonmonotonic formalisms, such as Reiter's default logic [7] and Clark's completion [8]. Similar to them, the existence of an answer set for a logic program is not guaranteed. In order to remedy this problem, a variant of default logic, called justified, has been proposed in [9]; the approach has been mapped to ASP in [10]. 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. On the one hand, this incremental character of ι-answer sets guarantees their existence for every logic program and facilitates computations. On the other hand, a ι-answer set may lack characteristic properties of standard answer sets, such as being a model of the given program. Such properties can be re-established by using integrity constraints. Indeed, one-to-one correspondence between standard answer sets and ι-answer sets satisfying a set of integrity constraints can be achieved by a simple syntactic transformation. In this view, ι-answer sets are a priori less restrictive than standard answer sets; in contrast to the latter, their existence is guaranteed for every logic program. By additionally incorporating integrity constraints, the expressiveness of standard answer sets can however be obtained if it is desired.

* Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada, and IIIS at Griffith University, Brisbane, Australia.

The basic algorithms of (standard) answer set solvers, such as *dlv* [11] and *smodels* [12], are modifications of the Davis-Putnam-Logemann-Loveland procedure (DPLL; [13, 14]). To our knowledge, no such algorithm has been described yet for computing either ι-answer sets of logic programs or justified extensions of default theories. (An analytic tableau method for the latter has been proposed in [15]; it is based on a conversion into disjunctive normal form.) We present a DPLL-like algorithm for computing the ι-answer sets of a logic program that satisfy a given set of integrity constraints. Our algorithm is based on a modification of Clark's completion that preserves the ι-answer sets of a given program and on the well-founded semantics for logic programs [16]. In the absence of integrity constraints, the algorithm computes a ι-answer set in polynomial time. Our algorithm also handles integrity constraints; however, the decision problem whether there is some ι-answer set that satisfies all integrity constraints is NP-complete. We have implemented our algorithm as a prototype in *ECLiPSe*-Prolog [17]; the implementation is publicly available at [18].

The outline of this 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 extend our framework with integrity constraints as a mechanism to filter ι-answer sets. Section 5 characterizes ι-answer sets in terms of propositional logic via a modification of Clark's completion. In Section 6, we show how unit propagation can be applied within the deterministic part of the ι-answer set computation. Based on this, Section 7 provides a DPLL-like algorithm for computing ι-answer sets. We conclude with Section 8.

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 head of r by $head(r) = p_0$ and the body of r by $body(r) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$. Furthermore, $body^+(r) = \{p_1, \ldots, p_m\}$ and $body^-(r) = \{p_{m+1}, \ldots, p_n\}$ are the positive and negative body of r, respectively. For a program Π, we let $body^+(\Pi) = \bigcup_{r\in\Pi} body^+(r)$ and $body^-(\Pi) = \bigcup_{r\in\Pi} body^-(r)$. We denote the set of all atoms occurring in Π by $At(\Pi)$.

A set X of atoms is a model of a logic program Π if, for every $r \in \Pi$, $head(r) \in X$ if $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$. Program Π is called basic if $body^-(\Pi) = \emptyset$. The $\subseteq$-minimal model of 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 $X = Cn(\Pi^X)$. Note that any answer set of Π is also a model of Π. The *generating rules* of X for Π are

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

We have that X is a model of Π iff $head(r) \in X$ for all $r \in \mathcal{R}_\Pi(X)$. Furthermore, let $Cn^+(\Pi) = Cn(\Pi^\emptyset)$. Note that $\Pi^\emptyset = \{head(r) \leftarrow body^+(r) \mid r \in \Pi\}$. One can show that X is an answer set of Π iff $Cn^+(\mathcal{R}_\Pi(X)) = Cn(\Pi^X) = X$.

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 [9] to remedy these problems; they lead us to *ι-answer sets* [10], where ι indicates the incremental flavor.

Definition 1. *A set X of atoms is a ι-answer set of a logic program 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

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

Since $body^-(r_4) \cap Cn^+(\Pi_1') = \{a\} \cap \{a, b, c\} \neq \emptyset$, 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 concept of applied rules yields an alternative characterization of ι-answer sets.

Proposition 1. *Let Π be a logic program and X be a set of atoms.*

Then, X is a ι-answer set of Π iff $X = Cn^+(\mathcal{A}_\Pi(X))$ and, for every $r \in \Pi \setminus \mathcal{A}_\Pi(X)$, either

(i) $body^+(r) \not\subseteq X$,
(ii) $body^-(r) \cap X \neq \emptyset$, *or*
(iii) $head(r) \in body^-(\mathcal{A}_\Pi(X) \cup \{r\})$.

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

Proposition 2. *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 either condition (i), (ii), or (iii) in Proposition 1 holds for every remaining rule.

As a consequence of Proposition 2, every program has some ι-answer set.

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

Since 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.

Proposition 3. *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.

Proposition 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 $\mathcal{A}_\Pi(X) = \mathcal{R}_\Pi(X)$.

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, $\{e\}$ is not a model of Π_1 because rule r_1 is not satisfied.

Based on the concept of models, we can reformulate Proposition 4 as follows.

Corollary 1. *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 (iii) in Proposition 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 the next section, we extend 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. In this way, we can achieve a one-to-one correspondence between ι-answer sets and answer sets.

4 Integrity Constraints

For a program Π, let $\Pi_{\mathcal{C}} = \{r \in \Pi \mid head(r) \in body^-(r)\}$, that is, $\Pi_{\mathcal{C}}$ contains all "self-blocking" rules of Π. For a (standard) answer set X of $\Pi \setminus \Pi_{\mathcal{C}}$, we have that X is an answer set of Π iff $body^+(r) \not\subseteq X$ or $body^-(r) \cap X \neq \emptyset$ for all $r \in \Pi_{\mathcal{C}}$. Thus, rules in $\Pi_{\mathcal{C}}$ can effectively prune answer sets of $\Pi \setminus \Pi_{\mathcal{C}}$. This does not apply to ι-answer sets because $\Pi' \cap \Pi_{\mathcal{C}} = \emptyset$ for any $\Pi' \subseteq \Pi$ satisfying condition (i) and (ii) in Definition 1.

In fact, the rules in $\Pi_{\mathcal{C}}$ have no effect on the ι-answer sets of Π.

Proposition 5. *Let Π be a logic program and X be a set of atoms.*
Then, X is a ι-answer set of Π iff X is a ι-answer set of $\Pi \setminus \Pi_{\mathcal{C}}$.

This shows that the standard approach to prune undesired answer sets via self-blocking rules does not work with ι-answer sets.

For filtering ι-answer sets, we thus extend our framework: An *integrity constraint* is of the form

$$\leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n \tag{3}$$

where $n \geq m \geq 0$ and each p_i ($1 \leq i \leq n$) is an atom. Given an integrity constraint c as in (3), we let $body(c) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$, $body^+(c) = \{p_1, \ldots, p_m\}$, and $body^-(c) = \{p_{m+1}, \ldots, p_n\}$ denote the body, the positive body, and the negative body of c, respectively. We say that c is *satisfied* wrt a set X of atoms if $body^+(c) \not\subseteq X$ or $body^-(c) \cap X \neq \emptyset$. For a set $\mathcal{C}$ of integrity constraints, we denote the set of all atoms occurring in $\mathcal{C}$ by $At(\mathcal{C})$.

We now extend the definition of ι-answer sets to the case that integrity constraints must be satisfied.

Definition 2. *A set X of atoms is a ι-answer set of a logic program Π wrt a set $\mathcal{C}$ of integrity constraints if X is a ι-answer set of Π such that every $c \in \mathcal{C}$ is satisfied wrt X.*

Similar to pruning standard answer sets via $\Pi_{\mathcal{C}}$, one can use $\mathcal{C}$ to prune ι-answer sets.

We next show how integrity constraints can be used to achieve a one-to-one correspondence between ι-answer sets and answer sets. By Corollary 1, we have to deny ι-answer sets that are no models of the given program. For a program Π, let

$$\mathcal{C}_\Pi = \{\leftarrow body(r), not\ head(r) \mid r \in \Pi\}.$$

The integrity constraints in $\mathcal{C}_\Pi$ prohibit that rules whose bodies are true are not applied.

Any ι-answer set of Π wrt $\mathcal{C}_\Pi$ is a model and thus also an answer set of Π.

Proposition 6. *Let Π be a logic program and X be a set of atoms.*
Then, X is an answer set of Π iff X is a ι-answer set of Π wrt $\mathcal{C}_\Pi$.

For illustration, reconsider program Π_1 in (2). We obtain the following set $\mathcal{C}_{\Pi_1}$ of integrity constraints:

$$\begin{aligned} c_1 &: \leftarrow not\ d, not\ a \\ c_2 &: \leftarrow not\ e, not\ b \\ c_3 &: \leftarrow a, b, not\ c \\ c_4 &: \leftarrow not\ a, not\ e\ . \end{aligned}$$

Observe that all integrity constraints in $\mathcal{C}_{\Pi_1}$ are satisfied wrt ι-answer set $\{a, b, c\}$ of Π_1, thus, $\{a, b, c\}$ is a model and an answer set of Π_1. Wrt the second ι-answer set $\{e\}$, integrity constraint c_1 is not satisfied, indeed, $\{e\}$ is neither a model nor an answer set of Π_1.

As a second example, consider the following program Π_2, which does not have any answer sets:

$$\begin{array}{l} a \leftarrow not\ b \\ b \leftarrow not\ c \\ c \leftarrow not\ a\ . \end{array} \tag{4}$$

The ι-answer sets of Π_2 are $\{a\}$, $\{b\}$, and $\{c\}$. Neither of them satisfies all integrity constraints in $\mathcal{C}_{\Pi_2} = \{\leftarrow not\ b, not\ a; \leftarrow not\ c, not\ b; \leftarrow not\ a, not\ c\}$. Thus, there is no ι-answer set of Π_2 wrt $\mathcal{C}_{\Pi_2}$. This example demonstrates that ι-answer sets, whose existence is guaranteed for every program, are a priori less restrictive than answer sets. By additionally incorporating integrity constraints, the same expressiveness as with answer sets can however be obtained if it is desired.

5 ι-Completion

In this section, we introduce a translation from logic programs to propositional formulas, which we call ι-completion. The ι-completion of a program modifies Clark's completion [8] to preserve the ι-answer sets of the program. The models of Clark's completion form a superset of the program's answer sets. In the same way, the models of a program's ι-completion form a superset of the program's ι-answer sets. For some programs, Clark's completion does not have any models. However, every program has some ι-answer set (as long as no integrity constraints must be satisfied); therefore, the ι-completion must also have a model.

Let us first provide some auxiliary definitions. For a program Π and an atom p, let $rule(p) = \{r \in \Pi \mid head(r) = p, p \notin body^-(r)\}$ be the rules defining p. Note that we exclude any $r \in \Pi$ such that $p \in body^-(r)$ from the rules defining p. This is justified by the fact that self-blocking rules do not have any effect on the ι-answer sets of Π (cf. Proposition 5). Based on $rule(p)$, we define the *support formula* for p as

$$sup(p) = \bigvee_{r \in rule(p)} (\bigwedge_{p \in body^+(r)} p \wedge \bigwedge_{p \in body^-(r)} \neg p).$$

Assuming that $head(r) \notin body^-(r)$ for every $r \in \Pi$, we let *Clark's completion* of a program Π and a set $\mathcal{C}$ of integrity constraints be

$$\begin{array}{rl} Clark(\Pi, \mathcal{C}) = & \{\, p \equiv sup(p) \mid p \in At(\Pi) \cup At(\mathcal{C}) \,\} \cup \\ & \{\, \bigvee_{p \in body^+(c)} \neg p \vee \bigvee_{p \in body^-(c)} p \mid c \in \mathcal{C} \,\}. \end{array}$$

Note that a rule r such that $head(r) \in body^-(r)$ can be expressed as the integrity constraint $\leftarrow body(r)$. Self-blocking rules are therefore unnecessary with our explicit definition of integrity constraints.

For Π_2 in (4), we have $Clark(\Pi_2, \emptyset) = \{a \equiv \neg b, b \equiv \neg c, c \equiv \neg a\}$; one can check that $Clark(\Pi_2, \emptyset)$ does not have any models. As mentioned above, the ι-completion

must have a model for every program, as every program has some ι-answer set. Thus, we cannot rely on $p \equiv sup(p)$ in the ι-completion of a program.

In order to preserve a program's ι-answer sets, we need to include an additional conjunct in the equivalence defining an atom. For an atom p, let $neg(p) = \{r \in \Pi \mid p \in body^-(r), head(r) \notin body^-(r)\}$. The set $neg(p)$ contains all rules that can be blocked by p, again we exclude any $r \in \Pi$ such that $head(r) \in body^-(r)$. Based on $neg(p)$, we define the *block formula* for p as

$$block(p) = \bigvee_{r \in neg(p)} (head(r) \wedge \bigwedge_{p \in body^+(r)} p \wedge \bigwedge_{p \in body^-(r)} \neg p).$$

Combining $sup(p)$ and $block(p)$ for atoms p, we define the *ι-completion* of a program Π and a set $\mathcal{C}$ of integrity constraints as

$$\begin{aligned} Comp(\Pi, \mathcal{C}) = \{\, p \equiv sup(p) \wedge \neg block(p) \mid p \in At(\Pi) \cup At(\mathcal{C}) \,\} \cup \\ \{\, \bigvee_{p \in body^+(c)} \neg p \vee \bigvee_{p \in body^-(c)} p \mid c \in \mathcal{C} \,\}. \end{aligned} \tag{5}$$

Including $\neg block(p)$ in (5) permits p being false if some $r \in \Pi$ such that $p \in body^-(r)$ is applied. In this case, satisfying the body of some rule defining p does not make p true. This is different from Clark's completion; the latter asserts that atoms must be equivalent to (the disjunction of) their defining rules' bodies.

For Π_2 in (4), $Comp(\Pi_2, \emptyset)$ contains the following equivalences:

$$\begin{aligned} a &\equiv \neg b \wedge \neg(c \wedge \neg a) \\ b &\equiv \neg c \wedge \neg(a \wedge \neg b) \\ c &\equiv \neg a \wedge \neg(b \wedge \neg c)\ . \end{aligned}$$

The models of $Comp(\Pi_2, \emptyset)$ are $\{a\}$, $\{b\}$, and $\{c\}$; they correspond to the ι-answer sets of Π_2. For instance wrt model $\{a\}$, we have $\{a\} \models block(b) = a \wedge \neg b$. This makes b false, although we have $\{a\} \models sup(b) = \neg c$. Observe that model $\{a\}$ of $Comp(\Pi_2, \emptyset)$ does not satisfy $Clark(\Pi_2, \emptyset)$; the same applies to $\{b\}$ and $\{c\}$.

The above example shows that models of the ι-completion are not necessarily models of Clark's completion. The converse however holds.

Proposition 7. *Let Π be a logic program and $\mathcal{C}$ be a set of integrity constraints. Then, we have $Clark(\Pi, \mathcal{C}) \models Comp(\Pi, \mathcal{C})$.*

This shows that the notion of ι-completion is indeed a relaxation of Clark's completion.

Similar to Clark's completion and answer sets, we have the following relationship between the models of a program's ι-completion and the program's ι-answer sets.

Proposition 8. *Let Π be a logic program and $\mathcal{C}$ be a set of integrity constraints. For every ι-answer set X of Π wrt $\mathcal{C}$, we have $X \models Comp(\Pi, \mathcal{C})$.*

The following is an immediate consequence of Theorem 1 and Proposition 8.

Corollary 2. *For every logic program Π, there is some model of $Comp(\Pi, \emptyset)$.*

Note that the existence of ι-answer sets or models of the ι-completion, respectively, is only guaranteed in the absence of integrity constraints.

The converse of Proposition 8 holds if a model X of the ι-completion is justified by the given program Π, that is, if $X = Cn^+(\mathcal{A}_\Pi(X))$ holds.

Theorem 2. *Let Π be a logic program, $\mathcal{C}$ be a set of integrity constraints, and X be a set of atoms.*

Then, X is a ι-answer set of Π wrt $\mathcal{C}$ iff $X \models Comp(\Pi, \mathcal{C})$ and $X = Cn^+(\mathcal{A}_\Pi(X))$.

Note that $X = Cn^+(\mathcal{A}_\Pi(X))$ implies that condition (i) in Definition 1 holds for $\mathcal{A}_\Pi(X)$, and $X \models Comp(\Pi, \mathcal{C})$ makes sure that $\mathcal{A}_\Pi(X)$ is indeed a maximal subset of Π satisfying condition (i) and (ii) in Definition 1.

In view of [16], we can characterize the ι-answer sets X of a program Π by unfounded sets. A set $U \subseteq At(\Pi)$ is *unfounded* for Π wrt a set X of atoms if, for any $r \in \Pi$ such that $head(r) \in U$, either $body^+(r) \not\subseteq X$, $body^-(r) \cap X \neq \emptyset$, or $body^+(r) \cap U \neq \emptyset$. For all $r \in \mathcal{A}_\Pi(X)$, we have $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$. If we have $X = Cn^+(\mathcal{A}_\Pi(X))$, then, for any nonempty $U \subseteq X$, there is some $r \in \mathcal{A}_\Pi(X)$ such that $head(r) \in U$ and $body^+(r) \cap U = \emptyset$. Thus, the only unfounded set contained in a ι-answer set X of Π is the empty set, which is unfounded by definition.

Based on unfounded sets, we can reformulate Theorem 2 as follows.

Proposition 9. *Let Π be a logic program, $\mathcal{C}$ be a set of integrity constraints, and $X \subseteq At(\Pi)$.*

Then, X is a ι-answer set of Π wrt $\mathcal{C}$ iff $X \models Comp(\Pi, \mathcal{C})$ and no nonempty subset of X is unfounded for Π wrt X.

For a program Π and a set X of atoms, one can show that the union $U_1 \cup U_2$ of two unfounded sets U_1 and U_2 for Π wrt X is also unfounded for Π wrt X. Thus, the union of all unfounded sets for Π wrt X yields the *greatest unfounded set* for Π wrt X, denoted by $\mathcal{U}_\Pi(X)$. As shown in [16], X is an answer set of Π iff $\mathcal{U}_\Pi(X) = At(\Pi) \setminus X$. This does not apply to ι-answer sets. For Π_1 in (2) and the ι-answer set $\{e\}$ of Π_1, we have $\mathcal{U}_{\Pi_1}(\{e\}) = \{b, c, d\} \subset \{a, b, c, d\} = At(\Pi_1) \setminus \{e\}$.

The following is a reformulation of Proposition 9 in terms of greatest unfounded sets.

Proposition 10. *Let Π be a logic program, $\mathcal{C}$ be a set of integrity constraints, and $X \subseteq At(\Pi)$.*

Then, X is a ι-answer set of Π wrt $\mathcal{C}$ iff $X \models Comp(\Pi, \mathcal{C})$ and $\mathcal{U}_\Pi(X) \subseteq At(\Pi) \setminus X$.

In contrast to answer sets, we may have $\mathcal{U}_\Pi(X) \subset At(\Pi) \setminus X$ for a ι-answer set X.

Finally, Proposition 9 allows us to provide a modification of the Lin-Zhao theorem [19] for ι-answer sets. For a program Π, the (positive) dependency graph of Π is $(At(\Pi), \{(p, p') \mid r \in \Pi, head(r) = p, p' \in body^+(r)\})$. A set $L \subseteq At(\Pi)$ is a *loop* of Π if, for every pair $p_1 \in L, p_2 \in L$, there is a path of non-zero length from p_1 to p_2 in the dependency graph of Π such that all atoms in the path belong to L. For a loop L of Π, we define the *loop formula* of L for Π as

$$LF_\Pi(L) = \bigvee_{p \in L} p \rightarrow \bigvee_{r \in \Pi, head(r) \in L, body^+(r) \cap L = \emptyset} (\bigwedge_{p \in body^+(r)} p \wedge \bigwedge_{p \in body^-(r)} \neg p).$$

Intuitively, $LF_\Pi(L)$ enforces non-circular support if some atom of L is true. We denote by $LF(\Pi)$ the set of loop formulas $LF_\Pi(L)$ for all loops L of Π.

For a program Π, a set $\mathcal{C}$ of integrity constraints, and $X \subseteq At(\Pi)$, the Lin-Zhao theorem states that X is an answer of Π such that every $c \in \mathcal{C}$ is satisfied wrt X iff $X \models Clark(\Pi, \mathcal{C}) \cup LF(\Pi)$.[1] By replacing Clark's completion with ι-completion, we obtain a modification of the Lin-Zhao theorem for ι-answer sets.

Theorem 3. *Let Π be a logic program, $\mathcal{C}$ be a set of integrity constraints, and $X \subseteq At(\Pi)$.*

Then, X is a ι-answer set of Π wrt $\mathcal{C}$ iff $X \models Comp(\Pi, \mathcal{C}) \cup LF(\Pi)$.

Note that the number of loop formulas in $LF(\Pi)$ is exponential in the worst case [20]. Thus, it is impractical to explicitly construct $LF(\Pi)$. Fortunately, constructing $LF(\Pi)$ is unnecessary since violations wrt a set X of atoms can be detected efficiently, for instance, using $Cn^{+}(\mathcal{A}_{\Pi}(X))$.

6 Unit Propagation

We now exploit the concept of ι-completion for identifying deterministic propagation steps to be applied in the computation of ι-answer sets. This paves the way for the DPLL-like algorithm, presented in the next section, that combines deterministic propagation and non-deterministic splitting. To our knowledge, no such algorithm has been described yet for computing either ι-answer sets of logic programs or the closely related justified extensions of default theories [9]. Also, the propagation within (standard) answer set solvers, such as *clasp* [21], *dlv* [11], and *smodels* [12], is based on Clark's completion and can thus not be used for computing ι-answer sets. Though we cannot directly use existent propagation frameworks, the fact that the ι-completion of a program is a propositional theory allows us to identify propagation principles for the ι-answer set computation within the setting of DPLL [13, 14].

The DPLL procedure works on propositional formulas in conjunctive normal form (CNF). A formula in CNF is a set $\{C_1, \ldots, C_m\}$ representing a conjunction of clauses $C_i = \{l_1, \ldots, l_n\}$ $(1 \leq i \leq m)$, where every l_j $(1 \leq j \leq n)$ is a (propositional) literal. That is, either $l_j = p$ or $l_j = \neg p$ for some proposition p, and C_i expresses the disjunction of its literals. We denote the complement of a literal l by $\bar{l}$ where $\bar{l} = \neg p$ if $l = p$ and, respectively, $\bar{l} = p$ if $l = \neg p$. We extend this notation to clauses $C = \{l_1, \ldots, l_n\}$ and let $\overline{C} = \{\bar{l}_1, \ldots, \bar{l}_n\}$.

An assignment A is a consistent set of literals, that is, $p \notin A$ or $\neg p \notin A$ for every proposition p. For a clause C, the unit clause rule implies a literal $l \in C$ wrt A if $\overline{C} \setminus A = \{\bar{l}\}$, that is, all literals of C except for l are false wrt A. If l is implied wrt A, it is the only literal that can satisfy C; in order to construct a model, DPLL thus adds l to A. Such an application of the unit clause rule is called unit propagation.

For a formula Γ in CNF, an assignment A is a fixpoint of unit propagation if either

(i) $\overline{C} \subseteq A$ for some clause $C \in \Gamma$ or
(ii) $l \in A$ for every clause $C \in \Gamma$ such that $\overline{C} \setminus A = \{\bar{l}\}$.

[1] We assume $head(r) \notin body^{-}(r)$ for every $r \in \Pi$.

In case (i), clause C is violated wrt A, and thus A cannot be extended to a model of Γ. In case (ii), all implied literals are already contained in A, that is, A cannot be extended further by unit propagation. For an assignment A', we say that A is a *unit fixpoint* of A' if A is a fixpoint of unit propagation such that $A' \subseteq A$ and all literals in $A \setminus A'$ have successively been added to A' by unit propagation.

In order to use unit propagation in the computation of ι-answer sets, we have to represent the ι-completion of a program in CNF. For avoiding an exponential blow-up, our CNF representation uses structural propositions in addition to the atoms of the given program, as usually introduced in CNF conversions [22]. Given a program Π and a set $\mathcal{C}$ of integrity constraints, we uniquely associate every rule $r \in \Pi$ with a proposition p_r and every body $b \in \{body(r) \mid r \in \Pi \cup \mathcal{C}\}$ with a proposition p_b. We below assert $p_r \equiv head(r) \wedge p_{body(r)}$ as well as $p_b \equiv p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n$ for $b = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$.

For a program Π and a set $\mathcal{C}$ of integrity constraints, we use the following clause sets to represent $Comp(\Pi, \mathcal{C})$ in CNF:

$$\Gamma_r(\Pi) = \big\{\{p_r, \neg head(r), \neg p_{body(r)}\}, \{\neg p_r, head(r)\}, \{\neg p_r, p_{body(r)}\} \mid r \in \Pi, head(r) \notin body^-(r)\big\} \quad (6)$$

$$\Gamma_b(\Pi, \mathcal{C}) = \big\{\{p_b, \neg p_1, \ldots, \neg p_m, p_{m+1}, \ldots, p_n\}, \{\neg p_b, p_1\}, \ldots, \{\neg p_b, p_m\}, \{\neg p_b, \neg p_{m+1}\}, \ldots, \{\neg p_b, \neg p_n\} \mid (b = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}) \in \{body(r) \mid r \in \Pi \cup \mathcal{C}\}\big\} \quad (7)$$

$$\Gamma_p(\Pi, \mathcal{C}) = \big\{\{p, \neg p_{body(r_1)}, p_{n_1}, \ldots, p_{n_m}\}, \ldots, \{p, \neg p_{body(r_k)}, p_{n_1}, \ldots, p_{n_m}\}, \{\neg p, p_{r_1}, \ldots, p_{r_k}\} \mid p \in At(\Pi) \cup At(\mathcal{C}), rule(p) = \{r_1, \ldots, r_k\}, neg(p) = \{n_1, \ldots, n_m\}\big\} \quad (8)$$

$$\Gamma_{Comp}(\Pi, \mathcal{C}) = \Gamma_r(\Pi) \cup \Gamma_b(\Pi, \mathcal{C}) \cup \Gamma_p(\Pi, \mathcal{C}) \cup \big\{\{\neg p_{body(c)}\} \mid c \in \mathcal{C}\big\}\,. \quad (9)$$

For $r \in \Pi$ and $b \in \{body(r) \mid r \in \Pi \cup \mathcal{C}\}$, respectively, the clauses $\Gamma_r(\Pi)$ in (6) and $\Gamma_b(\Pi, \mathcal{C})$ in (7) assert the equivalences mentioned above.[2] For $p \in At(\Pi) \cup At(\mathcal{C})$, the clauses $\Gamma_p(\Pi, \mathcal{C})$ in (8) reflect the equivalence $p \equiv sup(p) \wedge \neg block(p)$ in the definition of $Comp(\Pi, \mathcal{C})$ (cf. (5)). Finally, $\Gamma_{Comp}(\Pi, \mathcal{C})$ in (9) represents $Comp(\Pi, \mathcal{C})$ in CNF. In addition to the clauses described above, it asserts that integrity constraints $c \in \mathcal{C}$ must be satisfied, that is, $p_{body(c)}$ must be false. This fact can potentially be exploited by unit propagation, even if no literal of $body(c)$ can be falsified immediately (using the first clause in (7)), because there might be rules $r \in \Pi$ such that $body(r) = body(c)$, in which case we have $p_{body(r)} = p_{body(c)}$.

The following result shows that, for a program Π and a set $\mathcal{C}$ of integrity constraints, $\Gamma_{Comp}(\Pi, \mathcal{C})$ is indeed a conservative extension of $Comp(\Pi, \mathcal{C})$.

Proposition 11. *Let Π be a logic program, $\mathcal{C}$ be a set of integrity constraints, and $X \subseteq At(\Pi)$.*

Then, $X \models Comp(\Pi, \mathcal{C})$ iff $X = Y \cap At(\Pi)$ for a (unique) $Y \subseteq At(\Pi) \cup \{p_r \mid r \in \Pi, head(r) \notin body^-(r)\} \cup \{p_{body(r)} \mid r \in \Pi\}$ such that $Y \models \Gamma_{Comp}(\Pi, \mathcal{C})$.

[2] By Proposition 5, we can in (7) ignore rules $r \in \Pi$ such that $head(r) \in body^-(r)$. Our prototype implementation, described in the next section, removes such rules in preprocessing.

The clauses in $\Gamma_{Comp}(\Pi, \mathcal{C})$ have the desirable property that, for an assignment A such that $p \in A$ or $\neg p \in A$ for either all $p \in At(\Pi)$, all $p \in \{p_r \mid r \in \Pi, head(r) \notin body^-(r)\}$, or all $p \in \{p_{body(r)} \mid r \in \Pi, head(r) \notin body^-(r)\}$, a unit fixpoint of A either violates some clause in $\Gamma_{Comp}(\Pi, \mathcal{C})$ or decides all atoms, bodies, and applied rules of Π. (In the latter case, $A \cap At(\Pi)$ is a ι-answer set of Π wrt $\mathcal{C}$, provided that $Cn^+(\mathcal{A}_\Pi(A \cap At(\Pi))) = A \cap At(\Pi)$.) Thus, we can restrict the search space for ι-answer sets to either atoms, bodies, or rules of Π. In the next section, we detail an algorithm that exploits this property for efficiently computing ι-answer sets (at least in the absence of integrity constraints).

7 ι-Answer Set Computation

We now provide a DPLL-like algorithm for computing the ι-answer sets of a program Π that satisfy a set $\mathcal{C}$ of integrity constraints. By Proposition 8 and 11, we can use unit propagation on $\Gamma_{Comp}(\Pi, \mathcal{C})$ for deterministically extending an assignment A. In addition, we have to make sure that true atoms are not unfounded, that is, $A \cap At(\Pi) \subseteq Cn^+(\{r \in \Pi \mid head(r) \notin body^-(r), \neg p_r \notin A\})$ must hold. Conversely, all atoms $p \in At(\Pi) \setminus Cn^+(\{r \in \Pi \mid head(r) \notin body^-(r), \neg p_r \notin A\})$ must be false, and we can add $\neg p$ to A. Thus, there are two kinds of deterministic inferences: literals implied by unit propagation on $\Gamma_{Comp}(\Pi, \mathcal{C})$ and unfounded atoms.

If deterministic propagation does neither yield a total assignment nor a conflict, DPLL splits on some undecided proposition. As mentioned above, it for $\Gamma_{Comp}(\Pi, \mathcal{C})$ is sufficient to split either only on atoms, bodies, or rules. In the algorithm below, we only split on rules or their corresponding propositions, respectively. We even go further and stipulate $body^+(r) \subseteq Cn^+(\{r \in \Pi \mid p_r \in A\})$, for a rule $r \in \Pi$ used for splitting. Indeed, such a rule r must exist if there are rules whose application is not yet decided.

Our strategy is motivated by the desire to efficiently compute ι-answer sets in the absence of integrity constraints. By Proposition 2, ι-answer sets can be constructed incrementally based on their applied rules. For a model Y of $\Gamma_{Comp}(\Pi, \mathcal{C})$ and $X = Y \cap At(\Pi)$, we have $\mathcal{A}_\Pi(X) = \{r \in \Pi \mid p_r \in Y\}$. If we choose to apply a rule $r \in \Pi$ such that $body^+(r) \subseteq Cn^+(\{r \in \Pi \mid p_r \in A\})$ for an assignment A, we make sure that $head(r) \in Cn^+(\mathcal{A}_\Pi(X))$, as stipulated in Theorem 2. In the absence of integrity constraints, our splitting strategy thus allows for directly constructing ι-answer sets. In fact, we can compute one ι-answer set of a program without any backtracking.

Our algorithm for computing the ι-answer sets of a program Π satisfying a set $\mathcal{C}$ of integrity constraints is shown in Algorithm 1. In lines 1–7, we deterministically extend assignment A until either a conflict is encountered or no further literals can be inferred. A conflict occurs if either a clause in $\Gamma_{Comp}(\Pi, \mathcal{C})$ is violated (line 3) or a true atom is unfounded (line 6). If no conflict has been encountered, we in line 8 determine all rules $r \in \Pi$ such that $head(r) \notin body^-(r)$, the corresponding proposition p_r is undecided wrt A (that is, $\{p_r, \neg p_r\} \cap A = \emptyset$), and $body^+(r) \subseteq Cn^+(\{r \in \Pi \mid p_r \in A\})$. If no such rule exists, then the true atoms in A form a ι-answer set of Π wrt $\mathcal{C}$ and are printed in line 9. Otherwise, if propagation yields a partial assignment, we in line 11 non-deterministically select one of the rules determined in line 8. We then first analyze the case that r is applied (line 12) and afterwards the case that r is not applied (line 13).

Algorithm 1: ι-ANSWER-SETS

Input : A logic program Π, a set $\mathcal{C}$ of integrity constraints, and an assignment A.

1 **repeat**
2 $\quad A \leftarrow$ some unit fixpoint of A for $\Gamma_{Comp}(\Pi, \mathcal{C})$
3 $\quad$ **if** $\overline{C} \subseteq A$ for some $C \in \Gamma_{Comp}(\Pi, \mathcal{C})$ **then return**
4 $\quad A' \leftarrow A$
5 $\quad A \leftarrow A \cup \{\neg p \mid p \in At(\Pi) \setminus Cn^+(\{r \in \Pi \mid head(r) \notin body^-(r), \neg p_r \notin A\})\}$
6 $\quad$ **if** $\{p \in At(\Pi) \mid p \in A\} \cap \{p \in At(\Pi) \mid \neg p \in A\} \neq \emptyset$ **then return**
7 **until** $A = A'$
$\pi \leftarrow \{r \in \Pi \mid head(r) \notin body^-(r), \{p_r, \neg p_r\} \cap A = \emptyset,$
8 $\qquad body^+(r) \subseteq Cn^+(\{r \in \Pi \mid p_r \in A\})\}$
9 **if** $\pi = \emptyset$ **then print** $A \cap At(\Pi)$
10 **else**
11 $\quad r \leftarrow$ **select**(π)
12 $\quad \iota$-ANSWER-SETS$(\Pi, \mathcal{C}, A \cup \{p_r\})$
13 $\quad \iota$-ANSWER-SETS$(\Pi, \mathcal{C}, A \cup \{\neg p_r\})$

All ι-answer sets of a program Π that satisfy a set $\mathcal{C}$ of integrity constraints are computed by ι-ANSWER-SETS$(\Pi, \mathcal{C}, \emptyset)$. It is straightforward to modify Algorithm 1 to compute a desired number of ι-answer sets, if they exist. If $\mathcal{C} = \emptyset$, our splitting strategy allows us to construct one ι-answer set without any backtracks (without ever executing line 13 in Algorithm 1). If $\mathcal{C} \neq \emptyset$, we cannot guarantee this any longer. Though the decision problem whether Π has a ι-answer set is trivial, deciding whether there is some ι-answer set of Π wrt $\mathcal{C}$ is NP-complete. The latter is a direct consequence of Proposition 1 and 6.

We have implemented Algorithm 1 as a prototype in *ECLiPSe*-Prolog [17]. Our implementation is publicly available at [18]. For illustrating its usage, assume that file `pi.lp` contains the following program Π_3 (in Prolog syntax):

```
a :- not d.
b :- not a.
b :- c, not d.
c :- b.
d :- not c.
```

Via the query

```
?- lsolve('pi.lp', R, X, Y).
```

we can compute all ι-answer sets `X` of Π_3, where `R` is the set $\mathcal{A}_{\Pi_3}(\mathtt{X})$ of applied rules and `Y` is the set $body^-(\mathcal{A}_{\Pi_3}(\mathtt{X}))$. By backtracking, we get the following answers:

```
R=[a:-not d]                          X=[a]    Y=[d];      (10)
R=[b:-not a, c:-b, b:-(c, not d)]     X=[b,c]  Y=[a,d];    (11)
R=[b:-not a, d:-not c]                X=[b,d]  Y=[a,c];    (12)
No.
```

Indeed, the ι-answer sets of Π_3 are $\{\mathtt{a}\}$, $\{\mathtt{b},\mathtt{c}\}$, and $\{\mathtt{b},\mathtt{d}\}$. (Observe that Π_3 does not have any standard answer sets.) All solutions found by backtracking correspond to distinct ι-answer sets, thanks to splitting in Algorithm 1.

Assume next that file `ic.lp` contains Π_3 augmented with the following integrity constraint:

```
:- b, not c.                                    (13)
```

As expected, the query

```
?- lsolve('ic.lp', R, X, Y).
```

yields solutions (10) and (11), but not (12). In fact, (13) is satisfied wrt the ι-answer sets $\{\mathtt{a}\}$ and $\{\mathtt{b},\mathtt{c}\}$, but not wrt $\{\mathtt{b},\mathtt{d}\}$.

8 Conclusion

In this work, we have elaborated upon the concept of ι-answer sets, providing an alternative incremental approach to answer set programming. The ι-answer sets of a logic program amount to the justified extensions of the default theory corresponding to the program. Similar to justified extensions of default theories but different from (standard) answer sets, every logic program has at least one ι-answer set. In comparison to answer sets, the distinguishing feature of ι-answer sets is the possibility to construct them incrementally. In fact, the rules of a logic program can be applied one by one in order to eventually obtain a ι-answer set of the program. A ι-answer set may however lack characteristic properties of answer sets, such as being a model of the given program. We have seen how integrity constraints can be used to re-establish such properties. Indeed, a one-to-one correspondence between ι-answer sets and answer sets can be achieved by a simple syntactic transformation. Thus, ι-answer sets are a priori less restrictive than answer sets; by using integrity constraints, the same expressiveness as with answer sets can however be obtained if it is desired.

The Clark's completion of a logic program preserves the answer sets of the program. We have provided a translation from logic programs into propositional logic that modifies Clark's completion to preserve the ι-answer sets of a given logic program. Paralleling the models of Clark's completion and answer sets, the models of the ι-completion approximate ι-answer sets. That is, every ι-answer set satisfies the ι-completion, but a model of the ι-completion must be non-circularly supported for being a ι-answer set. In fact, the ι-answer sets of a logic program can be captured in terms of propositional logic by replacing Clark's completion with ι-completion in the Lin-Zhao theorem. In contrast to answer sets, the set of atoms outside a ι-answer set does not have to be unfounded. Given that the ι-answer sets of a logic program are not necessarily models, some false atoms might actually be (non-circularly) supported by the program.

Based on the ι-completion, we have provided a DPLL-like algorithm for computing the ι-answer sets of a logic program satisfying a given set of integrity constraints. To our knowledge, no such algorithm has been described yet for computing either ι-answer sets of logic programs or justified extensions of default theories. In the absence of integrity constraints, the algorithm computes one ι-answer set in polynomial time,

without any backtracking. Our algorithm also handles integrity constraints, but the decision problem whether there is some ι-answer set that satisfies all integrity constraints is NP-complete. Thus, the construction of a ι-answer set loses its incremental character if some constraints must eventually be satisfied.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
2. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence **25**(3-4) (1999) 241–273
3. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In Apt, K., Marek, W., Truszczyński, M., Warren, D., eds.: The Logic Programming Paradigm: a 25-Year Perspective. Springer-Verlag (1999) 375–398
4. Lifschitz, V.: Answer set programming and plan generation. Artificial Intelligence **138**(1-2) (2002) 39–54
5. Lloyd, J.: Foundations of Logic Programming. Springer-Verlag (1987)
6. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385
7. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13**(1-2) (1980) 81–132
8. Clark, K.: Negation as failure. In: Logic and Data Bases. Plenum Press (1978) 293–322
9. Łukaszewicz, W.: Considerations on default logic: an alternative approach. Computational Intelligence **4** (1988) 1–16
10. 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). (2003)
11. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic **7**(3) (2006) 499–562
12. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
13. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7** (1960) 201–215
14. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5** (1962) 394–397
15. Risch, V.: Analytic tableaux for default logics. Journal of Applied Non-Classical Logics **6**(1) (1996) 71–88
16. Van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. Journal of the ACM **38**(3) (1991) 620–650
17. `http://eclipse.crosscoreop.com/`
18. `http://www.cs.uni-potsdam.de/~gebser/lsolve.tar.gz`
19. Lin, F., Zhao, Y.: ASSAT: computing answer sets of a logic program by SAT solvers. Artificial Intelligence **157**(1-2) (2004) 115–137
20. Lifschitz, V., Razborov, A.: Why are there so many loop formulas? ACM Transactions on Computational Logic **7**(2) (2006) 261–268
21. 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/MIT Press (2007) 386–392
22. Plaisted, D., Greenbaum, S.: A structure-preserving clause form translation. Journal of Symbolic Computation **2**(3) (1986) 293–304