# Approaching the Core of Unfounded Sets

**Christian Anger** and **Martin Gebser** and **Torsten Schaub***
Institut für Informatik
Universität Potsdam
Postfach 90 03 27, D–14439 Potsdam
{christian, gebser, torsten}@cs.uni-potsdam.de

**Abstract**

We elaborate upon techniques for unfounded set computations by building upon the concept of loops. This is driven by the desire to minimize redundant computations in solvers for Answer Set Programming. We begin by investigating the relationship between unfounded sets and loops in the context of partial assignments. In particular, we show that subset-minimal unfounded sets correspond to active elementary loops. Consequentially, we provide a new loop-oriented approach along with an algorithm for computing unfounded sets. Unlike traditional techniques that compute greatest unfounded sets, we aim at computing small unfounded sets and rather let propagation (and iteration) handle greatest unfounded sets. This approach reflects the computation of unfounded sets employed in the *nomore++* system. Beyond that, we provide an algorithm for identifying active elementary loops within unfounded sets. This can be used by SAT-based solvers, like *assat*, *cmodels*, or *pbmodels*, for optimizing the elimination of invalid candidate models.

## Introduction

Search strategies of solvers for *Answer Set Programming* (ASP) naturally decompose into a deterministic and a non-deterministic part. While the non-deterministic part is realized through heuristically driven choice operations, the deterministic one is based on advanced propagation operations, often amounting to the computation of *well-founded semantics* (van Gelder *et al.* 1991). The latter itself can be broken up into techniques realizing *Fitting's operator* (Fitting 2002) and the computation of *unfounded sets* (van Gelder *et al.* 1991). The notion of an unfounded set captures the intuition that its atoms might circularly support themselves but have no support from "outside." Hence, there is no reason to believe in the truth of an unfounded set, and the contained atoms must be false. The opposites of unfounded sets are *externally supported sets* (Lee 2005), their atoms have a non-circular support.

While genuine ASP-solvers, like *dlv* (Leone *et al.* 2006) and *smodels* (Simons *et al.* 2002), aim at determining *greatest* unfounded sets, SAT-based ASP-solvers, like *assat* (Lin & Zhao 2004), *cmodels* (Lierler & Maratea 2004), and *pbmodels* (Liu & Truszczyński 2005), use *loops* and associated *loop formulas* (Lin & Zhao 2004; Lee 2005) for eliminating models containing unfounded sets. Both approaches comprise certain redundancies: For instance, not all elements of a greatest unfounded set need to be determined by special-purpose unfounded set techniques. Alternatively, one may restrict attention to crucial unfounded sets and handle the remaining ones via simpler forms of propagation and iteration. In fact, we show that a subset of a program's loops grants the same propagation strength as obtained with greatest unfounded sets. Further on, the problem with the standard concept of loops is that it tolerates the generation of ineffective loop formulas within SAT-based solvers. That is, unfounded subsets of a loop might recur, causing the need to generate additional loop formulas. Both redundancy issues are addressed by *(active) elementary loops* (Gebser & Schaub 2005), on which the computational approaches presented in this paper build upon.

We consider two diametrical computational tasks dealing with unfounded sets: first, falsification of greatest unfounded sets and, second, identification of subset-minimal unfounded sets. Greatest unfounded sets are worthwhile when the aim is setting unfounded atoms to false, as done within genuine ASP-solvers. Subset-minimal unfounded sets can serve best when one needs to eliminate an undesired model of a program's *completion* (Clark 1978) by a loop formula, which is important for SAT-based solvers.

First, we turn our attention to greatest unfounded sets computed by genuine ASP-solvers. In *dlv*, operator $\mathcal{R}_{\Pi_C,I}$ is applied to so-called *head-cycle-free components* $C$ of a disjunctive program $\Pi$, where $I$ is a (partial) interpretation (Calimeri *et al.* 2001).[1] The fixpoint, $\mathcal{R}^{\omega}_{\Pi_C,I}(C)$, of this operator is the greatest unfounded set with respect to $I$, restricted to atoms inside $C$.[2] Component-wise unfounded set identification is in *dlv* achieved by computing complements,

* Affiliated with the School of Computing Science at Simon Fraser University, Canada.

[1] Such a component $C$ is a strongly connected component of the atom dependency graph, where positive as well as negative dependencies (through $not$) contribute edges. Head-cycle-freeness additionally assures tractability of unfounded set checks, which otherwise are intractable for disjunctive programs.

[2] Note that a "global" greatest unfounded set is not guaranteed to exist for a disjunctive program (Leone *et al.* 1997). However, a head-cycle-free component always has a "local" greatest unfounded set, which can be computed in linear time.

that is, $C \setminus \mathcal{R}^{\omega}_{\Pi_C,I}(C)$. This set is externally supported, all other atoms of $C$ form the greatest unfounded set.

In *smodels*, unfounded set computation follows a similar idea. The respective function, called *Atmost*, is based on *source pointers* (Simons 2000). Each non-false atom has a source pointer indicating a rule that provides an external support for that atom. When some source pointers are invalidated (in effect of a choice), *Atmost* proceeds as follows:

> Iterate over the strongly connected components of a program's *(positive) atom dependency graph* (see next section). For the current component, do:
>
> 1. Remove source pointers that point to rules whose bodies are false.
> 2. Remove further source pointers that point to rules whose positive bodies contain some atoms currently not having source pointers themselves.
> 3. Determine new source pointers if possible. That is, re-establish source pointers of atoms that are heads of rules with non-false bodies such that all atoms in the positive parts have source pointers themselves.
> 4. All atoms without a new source pointer are unfounded. Set them to false (possibly invalidating source pointers of other components' atoms) and proceed.

Essentially, Step 1 and 2 check for atoms that might be unfounded due to rules whose bodies have recently become false. Afterwards, Step 3 determines the atoms that are still externally supported and, hence, not unfounded. Observe that the atoms to falsify as a result of Step 4 are precisely the ones that are not found externally supported in the step before. Thus, both *smodels* and *dlv* compute greatest unfounded sets as complements of externally supported sets. Notably, computations are modularized to strongly connected components of atom dependency graphs.

Having considered the falsification of greatest unfounded sets, we now turn to the diametrical problem: determining subset-minimal unfounded sets. The ability to compute subset-minimal unfounded sets is attractive for SAT-based solvers, which compute (propositional) models of a program's completion. Whenever a computed candidate model does not correspond to an answer set,[3] a loop formula that eliminates the model is added to the completion. For the loop formula eliminating the model, the respective loop must be unfounded. SAT-based solver *assat* determines so-called *terminating* loops (Lin & Zhao 2004), which are subset-maximal unfounded loops. Terminating loops are easy to compute: They are strongly connected components of the (positive) atom dependency graph induced by the greatest unfounded set. Given that terminating loops are not necessarily subset-minimal unfounded sets, their loop formulas condense the reason why a model is invalid less precisely than the ones of subset-minimal unfounded sets.

In this paper, we present a novel approach to achieving the aforementioned computational tasks. In fact, both tasks are settled on the same theoretical fundament. On the one hand, we can explain strategies of genuine ASP-solvers to handle greatest unfounded sets, also we present the strategy recently implemented in *nomore++* (Anger *et al.* 2005). On the other hand, we point out how our approach can be exploited by SAT-based solvers for determining more effective loop formulas. The overall contributions are:

- We relate the notion of elementary loops to unfounded sets in the context of partial assignments. Thereby, we reveal unfounded sets that must intrinsically be considered by both SAT-based and genuine ASP-solvers. The developed theoretical fundament fortifies new approaches to computational tasks dealing with unfounded sets.
- We describe a novel algorithm for computing unfounded sets in a loop-oriented way. The algorithm determines unfounded sets directly, avoiding the complementation of externally supported sets. This approach allows us to immediately propagate falsity of atoms in a detected unfounded set and to postpone unprocessed unfounded set checks. We thereby achieve a tighter coupling of unfounded set checks with simpler forms of propagation and localize the causes and effects of operations. The algorithm has recently been implemented in *nomore++*, but may be integrated into other solvers, e.g., *dlv*, as well.
- We present an algorithm for extracting active elementary loops from unfounded sets. The algorithm, which is the first of its kind, exploits particular properties of active elementary loops, building the "cores" of unfounded sets. Active elementary loops can replace terminating loops in SAT-based solvers. Note that a terminating loop is not guaranteed to be elementary, hence, a respective loop formula might be redundant (Gebser & Schaub 2005). Our algorithm can be integrated in solvers like *assat*, *cmodels*, and *pbmodels*. Such an integration could form the base for an empirical evaluation of the effectiveness of active elementary loops.

## Background

Given an alphabet $\mathcal{P}$, a (normal) *logic program* is a finite set of rules of the form

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

where $0 \leq m \leq n$ and each $p_i \in \mathcal{P}$ $(0 \leq i \leq n)$ is an *atom*. A *literal* is an atom $p$ or its negation $not\ p$. For a rule $r$ as in (1), let $head(r) = p_0$ be the *head* of $r$ and $body(r) = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\}$ be the *body* of $r$. Given a set $X$ of literals, let $X^+ = \{p \in \mathcal{P} \mid p \in X\}$ and $X^- = \{p \in \mathcal{P} \mid not\ p \in X\}$. For $body(r)$, we then get $body(r)^+ = \{p_1, \ldots, p_m\}$ and $body(r)^- = \{p_{m+1}, \ldots, p_n\}$. The set of atoms occurring in a logic program $\Pi$ is denoted by $atom(\Pi)$. The set of bodies in $\Pi$ is $body(\Pi) = \{body(r) \mid r \in \Pi\}$. For regrouping rule bodies sharing the same head $p$, define $body(p) = \{body(r) \mid r \in \Pi, head(r) = p\}$. A program $\Pi$ is called *positive* if $body(r)^- = \emptyset$ for all $r \in \Pi$. $Cn(\Pi)$ denotes the smallest set of atoms closed under positive program $\Pi$. The *reduct*, $\Pi^X$, of $\Pi$ relative to a set $X$ of atoms is defined by $\Pi^X = \{head(r) \leftarrow body(r)^+ \mid r \in \Pi,$

[3] Any answer set of a program is a model of the program's completion, whereas the converse does generally not hold (Fages 1994).

$body(r)^- \cap X = \emptyset\}$. A set $X$ of atoms is an *answer set* of a logic program $\Pi$ if $Cn(\Pi^X) = X$.

An unfounded set is defined relative to an *assignment*. In *nomore++*, values are assigned to both atoms and bodies, whereas *smodels* and *dlv* explicitly assign values only to atoms (from which the (in)applicability of rules is determined). Note that an assignment to atoms and bodies can reflect any state resulting from an assignment to atoms, whereas the converse does not hold because a body might be false without yet containing a false literal. Also, the restriction of assignments to atoms limits search to branching on atoms, which may lead to exponentially worse proof complexity than obtained when branching on both atoms and bodies (Gebser & Schaub 2006). Given that assignments to both atoms and bodies provide extra value, we define an *assignment* $\boldsymbol{A}$ for a program $\Pi$ as a (total) function:

$$\boldsymbol{A} : atom(\Pi) \cup body(\Pi) \rightarrow \{\ominus, \odot, \otimes, \oplus\}$$

The four values correspond to those used by *dlv* (Faber 2002); that is, $\ominus$ stands for *false*, $\odot$ for *undefined*, $\otimes$ for *must-be-true*, and $\oplus$ for *true*.[4] We also assume that the abstract ASP-solver, invoking the algorithms presented in the following sections, propagates the four values like *dlv* applied to normal programs (which approximates propagation within *nomore++*) and do not provide any details here.[5] We call an assignment $\boldsymbol{A}$ *positive-body-saturated*, abbreviated *pb-saturated*, if for every $B \in body(\Pi)$, $\boldsymbol{A}(B) = \ominus$ if $\boldsymbol{A}(p) = \ominus$ for some $p \in B^+$. An arbitrary assignment is easily turned into a pb-saturated one by propagation.

What is important to note is the difference between $\otimes$ (must-be-true) and $\oplus$ (true). For our unfounded set check to work, the following invariant must hold for any assignment $\boldsymbol{A}$:

$$\{p \in atom(\Pi) \mid \boldsymbol{A}(p) = \oplus\} \cup \left(\bigcup_{B \in body(\Pi), \boldsymbol{A}(B) = \oplus} B^+\right) \subseteq Cn(\{r \in \Pi \mid \boldsymbol{A}(body(r)) = \oplus\}^\emptyset) \quad (2)$$

The invariant stipulates that all atoms and (positive parts of) bodies assigned $\oplus$ are bottom-up derivable within the part of $\Pi$ assigned $\oplus$. This guarantees that no unfounded set ever contains an atom assigned $\oplus$, and we can safely exclude such atoms as well as bodies assigned $\oplus$ from unfounded set checks. Hence, the invariant helps in avoiding useless work. It also allows for "lazy" unfounded set checks, on which we will come back when discussing the relation of our unfounded set algorithm to *smodels*. Invariant (2) can be maintained by assigning $\oplus$ to an atom, only if some of its bodies is already assigned $\oplus$, and to a body, only if all atoms in the positive part are already assigned $\oplus$. Otherwise, $\otimes$ must be assigned instead of $\oplus$.

[4]Note that the concept of an assignment is to be understood in the sense of a constraint satisfaction problem, rather than an interpretation. This is because answer sets are defined as models that are represented by their entailed atoms. By assigning values to bodies, which can be viewed as conjunctions, we do not construct such a model but deal with problem-relevant variables. For this reason, we use symbolic values instead of ascribed truth values.

[5]When referring to propagation, we mean any technique that deterministically extends assignments except for unfounded set checks, to be detailed in the following sections.

We now come to unfounded sets. For a program $\Pi$, we define a set $U \subseteq atom(\Pi)$ as an *unfounded set* with respect to an assignment $\boldsymbol{A}$ if, for every rule $r \in \Pi$, we have either

- $head(r) \notin U$,
- $\boldsymbol{A}(body(r)) = \ominus$, or
- $body(r)^+ \cap U \neq \emptyset$.

Our definition is close to the original one (van Gelder *et al.* 1991), but differs regarding the second condition, which aims at inapplicable rules. With the original definition, such rules are determined from atoms, that is,

- $\{p \in body(r)^+ \mid \boldsymbol{A}(p) = \ominus\} \neq \emptyset$ or
- $\{p \in body(r)^- \mid \boldsymbol{A}(p) = \otimes \text{ or } \boldsymbol{A}(p) = \oplus\} \neq \emptyset$.

The reason for not determining inapplicable rules from atoms is that, with our definition of an assignment, a body assigned $\ominus$ needs not necessarily contain a false literal. Rather, a body might be inapplicable, that is, assigned $\ominus$, due to any reason (such as a choice or an inference by lookahead). Still it holds that normal programs (in contrast to disjunctive ones (Leone *et al.* 1997)) enjoy the property that the union of distinct unfounded sets is itself an unfounded set. Hence, there always is a *greatest unfounded set*, denoted $GUS_\Pi(\boldsymbol{A})$, for any program $\Pi$ and any assignment $\boldsymbol{A}$.

Finally, we come to loops, which are sets of atoms involved in cyclic program structures. Traditionally, program structure is described by means of atom dependency graphs (Apt *et al.* 1987). When we restrict attention to unfounded sets, it is sufficient to consider *positive* atom dependency graphs. For a program $\Pi$, the *(positive) atom dependency graph* is the directed graph $(atom(\Pi), E)$ where $E = \{(p, p') \mid r \in \Pi, p = head(r), p' \in body(r)^+\}$. That is, the head of a rule has an edge to each atom in the positive body. Following (Lee 2005), we define a *loop* $L$ in a program $\Pi$ as a non-empty subset of $atom(\Pi)$ such that, for any two elements $p \in L$ and $p' \in L$, there is a path from $p$ to $p'$ in the atom dependency graph of $\Pi$ all of whose vertices belong to $L$. In other words, the subgraph of the atom dependency graph of $\Pi$ induced by $L$ is strongly connected. Note that each set consisting of a single atom is a loop, as every atom is connected to itself via a path of length zero.

The significance of loops has first been recognized in (Lin & Zhao 2002), where the concept was also originally defined.[6] In fact, program completion and loop formulas capture answer sets in terms of propositional models. The advantage of loops and their formulas, in comparison to other SAT-reductions (e.g. (Janhunen 2003; Lin & Zhao 2003)), is that the reduction can be done incrementally (SAT-based solvers *assat*, *cmodels*, and *pbmodels* pursue this strategy); the increase in problem size is very small in the best case. The downside is that a program may yield exponentially many loops, leading to exponential worst-case space complexity of loop-based SAT-reductions (Lifschitz & Razborov 2006). Genuine ASP-solvers can, however, exploit loops

[6]Note that in (Lin & Zhao 2002) loops' atoms must be connected via paths of *non-zero* length. By dropping this requirement, we can relate loops and unfounded sets more directly.

without explicitly representing loop formulas. In what follows, we relate loops to unfounded sets paving the way to loop-oriented unfounded set computations. The difference to SAT-based approaches is that we consider loops in the context of partial assignments, and not with respect to total (propositional) models.

## Relating Unfounded Sets and Loops

Recall the definition of an unfounded set given in the previous section. It states that any rule whose head belongs to an unfounded set is either inapplicable or contains an unfounded atom in the positive part of the body. Since unfounded sets are finite, the following is a consequence.

**Proposition 1** *Let $\Pi$ be a logic program, $\boldsymbol{A}$ be an assignment, and $U$ be an unfounded set w.r.t. $\boldsymbol{A}$.*

*If $U \neq \emptyset$, we have $L \subseteq U$ for some loop $L$ in $\Pi$ that is unfounded w.r.t. $\boldsymbol{A}$.*

This result establishes that any non-empty unfounded set is a superset of some loop that is itself unfounded. Note that Proposition 1 would not hold, if we had defined loops according to (Lin & Zhao 2004), where the contained atoms must be connected via paths of non-zero length. Omitting this, a singleton unfounded set $\{p\}$ such that all bodies in $body(p)$ are assigned $\ominus$ is a loop. Otherwise, some element from an unfounded set $U$ must be contained in $B^+$, if $B$ is the body of a rule whose head is in $U$ and not assigned $\ominus$. The latter condition gives rise to inherent cyclicity.

When dealing with greatest unfounded sets, one usually concentrates on the part of an assignment not assigned $\ominus$. In fact, for an atom $p$ assigned $\ominus$ and a set $U$ of atoms such that $p \in U$, any body $B$ such that $p \in B^+$ satisfies the condition of containing an element from $U$ as well as the condition of containing a false literal. Since the latter condition is easy to verify, it is reasonable to exclude atoms assigned $\ominus$ when looking for the relevant part of a greatest unfounded set.

However, our definition of an unfounded set does not look "through" bodies for determining inapplicability. As a minimum requirement, we thus need an assignment to be pb-saturated, before a relevant unfounded set is determined. Certainly this requirement is reasonable, while working on "unsynchronized" assignments of atoms and bodies would be rather weird. For a pb-saturated assignment, the non-false part of the greatest unfounded set is an unfounded set.

**Lemma 1** *Let $\Pi$ be a logic program and $\boldsymbol{A}$ be a pb-saturated assignment.*

*Then, $\{p \in GUS_\Pi(\boldsymbol{A}) \mid \boldsymbol{A}(p) \neq \ominus\}$ is an unfounded set w.r.t. $\boldsymbol{A}$.*

Combining Proposition 1 and Lemma 1 yields the following.

**Theorem 1** *Let $\Pi$ be a logic program, $\boldsymbol{A}$ be a pb-saturated assignment, and $U = \{p \in GUS_\Pi(\boldsymbol{A}) \mid \boldsymbol{A}(p) \neq \ominus\}$.*

*If $U \neq \emptyset$, we have $L \subseteq U$ for some loop $L$ in $\Pi$ that is unfounded w.r.t. $\boldsymbol{A}$.*

The above result is the "partial assignment counterpart" of (Lin & Zhao 2004, Theorem 2), where the latter refers to total (propositional) models. Due to Theorem 1, we can concentrate greatest unfounded set computation on loops: By successively falsifying the atoms of unfounded loops and pb-saturating the resulting assignment, we eventually falsify all atoms in a greatest unfounded set. Clearly, more advanced propagation techniques (such as contraposition) can be applied in addition to pb-saturation. Theorem 1 still grants that there always is an unfounded loop whose atoms are not assigned $\ominus$, as long as there are non-false atoms left in the greatest unfounded set. Note that all answer set solvers we know of apply propagation techniques that are at least as strong as Fitting's operator (Fitting 2002). Whenever this operator has reached a fixpoint, all singleton loops $\{p\}$ such that all bodies in $body(p)$ are assigned $\ominus$ are already set to false. More sophisticated unfounded set checks can thus concentrate on loops as defined in (Lin & Zhao 2004).

Up to now, we have considered loops, which are defined by means of atom dependency graphs. Such graphs do not reflect program-specific connection via the bodies of rules. Given that we are interested in intrinsically relevant unfounded sets, loops are not yet fine-grained enough. To see this, consider the following programs:

$$\begin{array}{lllllll} \Pi_1 & = & \{\, a \leftarrow & b \leftarrow a, c & & & c \leftarrow b \,\} \\ \Pi_2 & = & \{\, a \leftarrow & b \leftarrow a & b \leftarrow c & & c \leftarrow b \,\} \end{array}$$

Though sharing the same atom dependency graph, the single answer set of $\Pi_1$ is $\{a\}$, whereas we obtain $\{a, b, c\}$ for $\Pi_2$. The reason for this is that the apparently different rules, $b \leftarrow a, c$ in $\Pi_1$ as well as $b \leftarrow a$ and $b \leftarrow c$ in $\Pi_2$, contribute the same edges to an atom dependency graph. However, rule $b \leftarrow a$ provides an external support for the set $\{b, c\}$, whereas rule $b \leftarrow a, c$ does not.

For distinguishing between putative and virtual external supports, we have to consider elementary loops (Gebser & Schaub 2005). We define a loop $L$ in a program $\Pi$ as *elementary* if, for each non-empty proper subset $K$ of $L$, there is a rule $r \in \Pi$ such that

- $head(r) \in K$,
- $body(r)^+ \cap K = \emptyset$, and
- $body(r)^+ \cap L \neq \emptyset$.

In words, a loop is elementary if each of its non-empty proper subsets has a rule whose head is in the subset and whose body positively relies on the loop, but not on the subset itself.[7] A particular property of elementary loops, rather than general ones, is that they potentially provide an external support to any of their non-empty proper subsets, even when they are unfounded. If such a situation arises, we say that an elementary loop is active. Formally, an elementary loop $L$ in a program $\Pi$ is *active* w.r.t. an assignment $\boldsymbol{A}$ if

- $L$ is an unfounded set w.r.t. $\boldsymbol{A}$ and
- $L$ is elementary in $\{r \in \Pi \mid \boldsymbol{A}(body(r)) \neq \ominus\}$.

Due to the first condition, an active elementary loop always is unfounded. The next result tells us that any non-empty unfounded set contains an active elementary loop.

[7] In analogy to general loops, every singleton is an elementary loop by definition. This is different from (Gebser & Schaub 2005), where loops are defined according to (Lin & Zhao 2004).

**Proposition 2** *Let $\Pi$ be a logic program, $\boldsymbol{A}$ be an assignment, and $U$ be an unfounded set w.r.t. $\boldsymbol{A}$.*

*If $U \neq \emptyset$, we have $L \subseteq U$ for some elementary loop $L$ in $\Pi$ that is active w.r.t. $\boldsymbol{A}$.*

This result strengthens Proposition 1. For a pb-saturated assignment, it together with Lemma 1 grants the existence of an active elementary loop none of whose atoms is assigned $\ominus$, whenever the greatest unfounded set contains non-false atoms. It is thus sufficient to concentrate unfounded set computations on active elementary loops. Going beyond is impossible: Any non-empty proper subset of an active elementary loop is externally supported.

**Proposition 3** *Let $\Pi$ be a logic program, $\boldsymbol{A}$ be an assignment, and $L$ be an active elementary loop in $\Pi$ w.r.t. $\boldsymbol{A}$.*

*Then, any non-empty proper subset of $L$ is not unfounded w.r.t. $\boldsymbol{A}$.*

The following is a "partial assignment counterpart" of results on total (propositional) interpretations in (Gebser *et al.* 2006).[8] It is a consequence of Proposition 2 and 3.

**Theorem 2** *Let $\Pi$ be a logic program, $\boldsymbol{A}$ be an assignment, and $L \subseteq atom(\Pi)$.*

*Then, $L$ is an active elementary loop in $\Pi$ w.r.t. $\boldsymbol{A}$ iff $L$ is a subset-minimal non-empty unfounded set w.r.t. $\boldsymbol{A}$.*

This result shows that active elementary loops form in fact the "cores" of unfounded sets. Any proper superset of an active elementary loop contains atoms that are unnecessary for identifying (parts of) the set as unfounded. In turn, no non-empty proper subset of an active elementary loop can be identified as unfounded. Active elementary loops motivate novel computational approaches in two aspects: First, they can be used to make unfounded set computations less exhaustive by not aiming at greatest unfounded sets; second, they reveal intrinsically relevant unfounded sets and rule out superfluous ones. In the next sections, we provide respective computational approaches.

## Greatest Unfounded Sets

We now exploit Theorem 1 and Proposition 2, granting the existence of an active elementary loop as a subset of the non-false part of a greatest unfounded set, and design an algorithm aiming at such loops. In order to restrict computations to necessary parts, we make the following assumptions:

- Invariant (2) on assignments holds. It guarantees that neither an atom assigned $\oplus$ nor an atom from the positive part of a body assigned $\oplus$ is unfounded.
- If, for an atom, the bodies of all rules with the atom as head are assigned $\ominus$, then the atom is assigned $\ominus$. Vice versa, an atom is assigned $\oplus$ if it has a body assigned $\oplus$.
- A body is assigned $\ominus$ if some of its literals is false, that is, an atom from the positive part is assigned $\ominus$ or one from the negative part is assigned either $\otimes$ or $\oplus$. Also, a body is assigned $\oplus$ if all atoms in its positive part are assigned $\oplus$ and all atoms in the negative part $\ominus$.

Due to the first assumption, the external support of atoms and bodies assigned $\oplus$ is granted. Furthermore, atoms and bodies already assigned $\ominus$ need not be considered anyway. We can thus restrict attention to atoms and bodies assigned either $\odot$ or $\otimes$. The second and third assumption grant that anything decidable by Fitting's operator is already assigned. (Note that this implies assignments to be pb-saturated.) Fixpoints of Fitting's operator are computed by *dlv*, *smodels*, and *nomore++* before an unfounded set check is initiated.

The unfounded sets we are aiming at are loops. Loops are bound from above by the strongly connected components of a program's atom dependency graph. For conveniently arranging both atoms and bodies into strongly connected components, we extend dependency graphs to bodies. For a program $\Pi$, we define the *(positive) atom-body dependency graph* as the directed graph $(atom(\Pi) \cup body(\Pi), E \cup E_0)$ where $E = \{(head(r), body(r)) \mid r \in \Pi\}$ and $E_0 = \{(body(r), p) \mid r \in \Pi, p \in body(r)^+\}$.[9] The strongly connected components of such graphs are understood in the standard graph-theoretical sense, loops are the atoms contained in strongly connected subgraphs.

We are now ready to describe our algorithm for computing an unfounded set. It accesses the following global variables.

$\Pi$**:** The underlying logic program.

$\boldsymbol{A}$**:** The current assignment.

$\mathsf{SCC}$**:** The vertices of a strongly connected component of the atom-body dependency graph of $\Pi$.

$\mathsf{Set}$**:** A set of atoms such that $\mathsf{Set} \subseteq \mathsf{SCC} \cap atom(\Pi)$.

$\mathsf{Ext}$**:** The set $\mathsf{Ext} = \bigcup_{p \in \mathsf{Set}} \{B \in body(p) \mid B^+ \cap \mathsf{Set} = \emptyset, \boldsymbol{A}(B) \neq \ominus\}$ of bodies.

$\mathsf{Source}$**:** A subset of $body(\Pi)$.

$\mathsf{Sink}$**:** A subset of $atom(\Pi)$.

Variable $\mathsf{Set}$ contains the atoms to be extended to an unfounded set. All atoms in $\mathsf{Set}$ belong to the same strongly connected component: $\mathsf{SCC}$. Set $\mathsf{Ext}$ of bodies can be thought of as a todo list. It comprises bodies that provide external supports for the atoms in $\mathsf{Set}$, hence, some atoms from their positive parts must be added to $\mathsf{Set}$. Synonymously to *smodels*' source pointers, set $\mathsf{Source}$ contains bodies for which it is known that external supports for their positive parts exist. Set $\mathsf{Sink}$ contains atoms some of whose non-false bodies are in $\mathsf{Source}$ or in a different strongly connected component; such atoms are not contained in any unfounded set. A source pointer in *smodels* can be thought of as a link from an atom in $\mathsf{Sink}$ to a body in $\mathsf{Source}$ or outside $\mathsf{SCC}$.

Our unfounded set algorithm is shown in Algorithm 1. The designated initial situation is that some atom, assigned either $\odot$ or $\otimes$, has been chosen to start an unfounded set check from. This atom is initially contained in $\mathsf{Set}$, its "external bodies" in $\mathsf{Ext}$. For the computation being reasonable, each external body is supposed to be contained in

[8] Please note that the reformulation of (active) elementary loops provided here is inspired by the notion of an *elementary set* (Gebser *et al.* 2006), for which similar results in the context of total (propositional) interpretations were developed first.

[9] So-called *body-head graphs* are used in (Linke & Sarsakov 2005) for describing isomorphisms between dependency graphs and syntactically restricted program classes.

**Algorithm 1**: UNFOUNDED SET

```
1  while Ext ≠ ∅ do
2      Ext ← Ext \ {B} for some B ∈ Ext
3      if there is some p ∈ B+ ∩ SCC such that p ∉ Sink and A(p) ≠ ⊕ then
           J ← {B ∈ body(p) | B ∉ SCC, A(B) ≠ ⊖} ∪
4              {B ∈ body(p) | B ∈ Source, A(B) ≠ ⊖}
5          if J = ∅ then
6              Set ← Set ∪ {p}
7              Ext ← Ext \ {B ∈ Ext | p ∈ B+}
8              Ext ← Ext ∪ {B ∈ body(p) | B+ ∩ Set = ∅, A(B) ≠ ⊖}
9          else
10             Sink ← Sink ∪ {p}
11             Ext ← Ext ∪ {B}
12     else
13         Source ← Source ∪ {B}
14         R ← {p ∈ Set | B ∈ body(p)}
15         while R ≠ ∅ do
16             Set ← Set \ R
17             Sink ← Sink ∪ R
               J ← {B ∈ body(Π) ∩ SCC | B+ ∩ R ≠ ∅, A(B) ≠ ⊖,
18                 {p ∈ B+ ∩ SCC | p ∉ Sink, A(p) ≠ ⊕} = ∅}
19             Source ← Source ∪ J
20             R ← {p ∈ Set | body(p) ∩ J ≠ ∅}
21         Ext ← ⋃p∈Set {B ∈ body(p) | B+ ∩ Set = ∅, A(B) ≠ ⊖}
```

SCC \ Source. The outer while-loop from line 1 to 21 is iterated as long as there are external bodies. Note that we have Ext $= \emptyset$ whenever Set $= \emptyset$; in this case, the empty Set indicates that no unfounded set contains any atom that has temporarily been in Set.

If Ext $\neq \emptyset$, we select in line 2 an external body B from whose positive part an atom should be added to Set next. Such an atom p must be contained in SCC, but not in Sink, and not be assigned $\oplus$ (line 3). If there is such an atom p, we determine in line 4 all bodies of atom p that are not assigned $\ominus$ and either not contained in SCC or contained in Source. If such bodies exist, that is, J $\neq \emptyset$, p is externally supported, and we add it to Sink (line 10). Otherwise, we can extend Set with atom p (line 6). All bodies that were formerly external but positively rely on p are then removed from Ext (line 7). Finally, we add bodies of rules with head p to Ext if they do not positively rely on Set and are not assigned $\ominus$ (line 8).

From line 12 to 21, we handle the case that no atom from the positive part of body B can be added to Set. Then, we add B to Source as it is externally supported (line 13). In line 14, we determine the atoms from Set that occur as heads of rules with body B. These atoms are as well externally supported and must be removed from Set. Note that we always have R $\neq \emptyset$ because B occurs as body of at least one atom in Set. From line 15 to line 20, we remove atoms from Set and add them to Sink as long as further bodies and associated head atoms are found externally supported. The crucial line is 18: Here we determine bodies $B$ from SCC, not assigned $\ominus$, such that some atoms in the positive part have recently been removed from Set ($B^+ \cap$ R $\neq \emptyset$) and all other atoms are either not contained in SCC, contained in Sink, or assigned $\oplus$. In a bottom-up fashion, we derive such externally supported bodies and add them to Source (line 19), respective head atoms are successively removed from Set and added to Sink (lines 16, 17, and 20). Finally, we update in line 21 the external bodies of the atoms still in Set.

Like unfounded set detection algorithms of *dlv* and *smodels*, Algorithm 1 can be implemented such that it works in linear time. The distinguishing element to other algorithms is that it extends the set of considered atoms on demand, that is, if there are bodies from whose positive parts no atom is included yet. The algorithm stops and does not explore any more atoms when such bodies do not exist. The aim is to keep a computed unfounded set as small as possible. This is motivated as follows: Propagation of single atoms and bodies can be done very efficiently and does, in contrast to unfounded set checks, not risk "wasted" work yielding no inferences. Simpler forms of propagation, like Fitting's operator, are thus in *nomore++* applied as soon as possible, in the hope that pending unfounded set checks can be avoided in effect. For enabling such "early" propagation, it is important that we compute unfounded sets directly, as it is done by Algorithm 1, and do not complement externally supported sets, as done within *dlv* and *smodels*.

Let us now consider ways of integrating Algorithm 1 into solvers. Any solver using Algorithm 1 has to grant that potential external support for bodies in Source and atoms in Sink really exists, since the elements of these sets are not examined by the algorithm. The same applies to atoms and bodies assigned $\oplus$. Systems *dlv* and *nomore++* assure the latter by assigning *must-be-true* or $\otimes$, when later unfoundedness of a true atom or body cannot be excluded. Detecting unfoundedness of program parts that must be true leads to a conflict, which has to be detected for soundness reasons.

The strategy of *smodels* is different, it does not use an analog for $\otimes$. Unfounded program parts, whether they contain true elements or not, are determined from source pointers. Such source pointers correspond to elements of Source and Sink. They are maintained during the solving process, and invalid ones are removed during the "first stage" of function *Atmost*, before it performs the actual unfounded set check. For a true atom, the removal of its source pointer can be seen as turning the value from $\oplus$ to $\otimes$, in order to make the atom accessible to a pending unfounded set check.

In contrast to *smodels*' *Atmost*, *dlv* and *nomore++* do not have a "first stage" for canceling outdated external support information. They simply start their unfounded set computations from head atoms of rules whose bodies have become false since the last unfounded set check. (Such atoms are also the starting points for *Atmost* to remove source pointers.) Unfounded set checks are done locally for strongly connected components of the respective dependency graphs. After processing a component, no information is kept, and no updates are necessary upon a revisit. Another parallel between *dlv* and *nomore++* is that the former propagates a component's greatest unfounded set before initiating further unfounded set checks (Faber 2006). Though not the same, this is quite similar to *nomore++* immediately propagating an unfounded set determined by Algorithm 1.

The discussion above shows that Algorithm 1 can potentially be put into various contexts, using different strategies to maintain acquired information and to combine unfounded set checks with propagation. Concerning the latter, Algorithm 1 is designed to stop as soon as an unfounded set is detected. In this way, a solver can immediately propagate falsity of the contained atoms. This allows unfounded set checks to always work on an up-to-date assignment, possibly reducing the overall efforts of a computation. Finally, let us mention that Algorithm 1, though aiming at loops, only guarantees that the atoms of a computed unfounded set belong to the same strongly connected component. They do not necessarily form a loop because of the inherent sensitivity to the order in which atoms are assumed to belong to an unfounded set (the order in which they are added to Set).

## Subset-Minimal Unfounded Sets

Having considered the falsification of greatest unfounded sets, we now turn to the diametrical problem: determining subset-minimal unfounded sets, which, by Theorem 2, are active elementary loops. The ability to determine active elementary loops is attractive for SAT-based ASP-solvers, computing (propositional) models of a program's completion and adding loop formulas to eliminate invalid candidate models. To this end, the SAT-based solver *assat* determines terminating loops, which are subset-maximal unfounded loops. Clearly, terminating loops are not necessarily active elementary loops. However, the loop formula of an active elementary loop eliminates an invalid candidate model, like the one of a terminating loop. In addition, undesired models that are not eliminated by the loop formula of a terminating loop might be excluded in future invocations of the underlying SAT-solver (cf. Section 5 in (Gebser & Schaub 2005) for an example). In this section, we show how an active elementary loop can be extracted from a given unfounded set, which might be a terminating loop. Within SAT-based solvers, active elementary loops can thus replace terminating loops.

Though elementary loops, as defined before, suggest that all subsets of a loop must be examined, deciding whether a loop is elementary is tractable. Indeed, elementary loops can also be characterized by elementary subgraphs of a program's atom-body dependency graph (Gebser & Schaub 2005). For a program $\Pi$ and a set $L \subseteq atom(\Pi)$, we define $B(L) = \{body(r) \mid r \in \Pi, head(r) \in L, body(r)^+ \cap L \neq \emptyset\}$ and $E(L) = \{(p, B) \mid p \in L, B \in B(L), p \leftarrow B \in \Pi\}$. The *elementary subgraph* of $L$ in $\Pi$ is the directed graph $(L \cup B(L), E(L) \cup EC(L))$ where:

$$\begin{aligned} EC^0(L) &= \emptyset \\ EC^{i+1}(L) &= EC^i(L) \cup \{(B,p) \mid B \in B(L), p \in B^+ \cap L, \\ &\qquad \text{each } p' \in B^+ \cap L \text{ has a path to } p \\ &\qquad \text{in } (L \cup B(L), E(L) \cup EC^i(L))\} \\ EC(L) &= \textstyle\bigcup_{i \geq 0} EC^i(L) \end{aligned}$$

By (Gebser & Schaub 2005, Theorem 10), the elementary subgraph allows for deciding elementariness.

**Theorem 3** *Let $\Pi$ be a logic program and $L \subseteq atom(\Pi)$. If $L \neq \emptyset$, $L$ is an elementary loop in $\Pi$ iff the elementary subgraph of $L$ in $\Pi$ is strongly connected.*

If a loop is elementary, its elementary subgraph has the following property (Gebser & Schaub 2005, Proposition 12).

**Proposition 4** *Let $\Pi$ be a logic program, $L$ be an elementary loop in $\Pi$, and $(L \cup B(L), E(L) \cup EC(L))$ be the elementary subgraph of $L$ in $\Pi$.*

*Then, every subgraph $(L \cup B(L), E(L) \cup EC'(L))$ such that $EC'(L) \subseteq EC(L)$ and $\{B \mid (B,p) \in EC'(L)\} = B(L)$ is strongly connected.*

Due to the above property, considering only a single edge from a body to a contained loop atom is sufficient for deciding elementariness by elementary subgraph construction. This "don't care" character of elementary subgraphs greatly facilities elementary loop computation: Instead of considering all edges in an atom-body dependency graph, we can select one contained atom as a canonical representative to be reached from a body. Considering the definition of elementary subgraphs, this representative should be a body atom that is reached from all other body atoms under consideration. Proceeding in this way, we can compute active elementary loops by implicitly constructing elementary subgraphs, where bodies reach canonical representatives, reflecting the single edges required to obtain a strongly connected graph.

We have now settled the fundament of Algorithm 2 for extracting an active elementary loop from an unfounded set. Algorithm 2 uses the global variable Set, containing the atoms of an unfounded set. Initially, Set might be the result of Algorithm 1 (which is not necessarily a loop) or a terminating loop. In effect of Algorithm 2, Set will contain the atoms of an active elementary loop, obtained through removing superfluous atoms. The variables Act, Q, and N are local to Algorithm 2. Set Act contains the atoms that are temporarily assumed to be elements of the final active elementary loop. Variable Q is a priority queue of atoms that need to be visited. Each atom p has an associated id, accessible via p.id, atoms in Q are then sorted by their ids in increasing order. Via operation Q.rem(), the first element of Q is removed from Q and returned. Operation Q.add(p) inserts an atom p into Q at the appropriate position, the operation has no effect if p is already contained in Q. Variable N is a counter, used to assign an id to an atom when it is visited for the first time. Besides the id, each atom p is associated with two more variables: root and exp. Integer value root stores the id of the first visited atom that positively depends on p in the elementary subgraph of Set. The set exp corresponds to a todo list of atoms that positively depend on p, but have not yet been explored. Similar to p.id, we access root and exp of an atom p via p.root and p.exp.

Before we start describing the algorithm, let us sketch its fundamental idea. The initial value for N will be |Set|, and we decrement N whenever an atom is visited for the first time. That is, an atom with a greater id is visited before the atoms with smaller ids. While exploring atoms, we make sure that an atom with a smaller id reaches all atoms with greater ids in the elementary subgraph of Set. In this way, we can safely select the contained atom with the greatest id to explore a body from. In fact, this atom is a canonical

**Algorithm 2**: ACTIVE ELEMENTARY LOOP

```
1  Act ← ∅
2  Q ← ∅
3  N ← |Set|
4  while N ≠ 0 do
5      p.id ← 0 for some p ∈ Set
6      Q.add(p)
7      while Q ≠ ∅ do
8          p ← Q.rem()
9          if p.id = 0 then
10             p.id ← N
11             p.root ← N
12             p.exp ← ∅
13             Act ← Act ∪ {p}
14             N ← N − 1
15             foreach B ∈ body(Π) such that p ∈ B+, B+ ∩ Set ⊆ Act,
               and A(B) ≠ ⊖ do
                   let p′ ∈ B+ ∩ Act such that
16                     p′.id = max{p.id | p ∈ B+ ∩ Act}
17                 p′.exp ← p′.exp ∪ {p ∈ Set | B ∈ body(p)}
18                 Q.add(p′)
19         if p.exp ≠ ∅ then
20             Q.add(p)
21             p.exp ← p.exp \ {p′} for some p′ ∈ p.exp
22             if p′ ∈ Act then p.root ← max{p.root, p′.root}
23             else if p′ ∈ Set then
24                 p′.id ← 0
25                 Q.add(p′)
26         else
27             if p.id = p.root then
28                 if Q ≠ ∅ or N ≠ 0 then
29                     Set ← Set \ {p ∈ Act | p.id ≤ p.id}
30                     Act ← Act \ {p ∈ Act | p.id ≤ p.id}
31             else
32                 p′ ← Q.rem()
33                 p′.root ← max{p.root, p′.root}
34                 Q.add(p′)
```

representative, as discussed below Proposition 4. Whenever an atom is not reached from any atom with a greater id in the elementary subgraph of Set or there are unvisited atoms in Set, we can safely remove all atoms with smaller ids than that of the current atom from Set. The residual atoms in Set still form an unfounded set. We are done when N reaches zero, indicating that all atoms in Set have been inspected and form an active elementary loop.

We now describe Algorithm 2. Given Set as global variable, Act and Q are initialized to be empty, and N is set to the cardinality of Set (lines 1 to 3). The outer while-loop from line 4 to 34 is iterated until N reaches zero, indicating that all atoms in Set have been inspected. As long as this is not the case, we pick an arbitrary atom p from Set, assign p.id zero, and add p to the front of Q (lines 5 and 6). The atom p with the smallest id is removed from Q in line 8. In line 9, we detect from p.id being zero that p is visited for the first time. We then initialize p.id and p.root with N, and p.exp with the empty set (lines 10 to 12). Adding p to Act in line 13 indicates that p has been visited. In line 14, we decrement N to the number of still unvisited atoms in Set.

Due to visiting an atom p for the first time, a body B such that $\mathsf{p} \in \mathsf{B}^+$ and $\mathsf{B}^+ \cap \mathsf{Set} \subseteq \mathsf{Act}$ becomes accessible, as there is an atom in Act that is reached from all atoms of $\mathsf{B}^+ \cap \mathsf{Act}$ in the elementary subgraph of Set. Of course, $\boldsymbol{A}(\mathsf{B})$ must not be $\ominus$ since we are interested in an active elementary loop w.r.t. $\boldsymbol{A}$. These conditions are checked in line 15. For each body satisfying the conditions, some atom $\mathsf{p}' \in \mathsf{B}^+ \cap \mathsf{Act}$ has the greatest id; this atom $\mathsf{p}'$ is determined in line 16. As discussed above, $\mathsf{p}'$ is a canonical representative to reach B from. Thus, we add the head atoms of body B that are in Set to $\mathsf{p}'$.exp and re-add $\mathsf{p}'$ to Q (lines 17 and 18). Recall that the latter has no effect if $\mathsf{p}'$ is already contained in Q.

After having updated atoms to be explored, we process p.exp for the current atom p from line 19 to 34. If p.exp is non-empty, we re-add p to Q, making sure that p is re-visited later on, and remove some element $\mathsf{p}'$ to be processed next from p.exp (lines 20 and 21). The atom $\mathsf{p}'$ can be already visited, in which case we maximize ids of atoms reaching p among p.root and $\mathsf{p}'$.root (line 22). If $\mathsf{p}'$ is unvisited and has not been removed from Set since it was added to p.exp, we set $\mathsf{p}'$.id to zero and add $\mathsf{p}'$ to the front of Q (lines 24 and 25). On re-entering the outer while-loop from line 7, $\mathsf{p}'$ is the atom visited next. The else-case from line 26 to 34 reflects that no more atom reaches p. If p is not reached from an atom with a greater id ($\mathsf{p.id} = \mathsf{p.root}$ in line 27) and there are atoms not reaching p ($\mathsf{Q} \neq \emptyset$ or $\mathsf{N} \neq 0$ in line 28), we remove all atoms in Act whose ids are not greater than p.id from both Set and Act (lines 29 and 30). The residual atoms of Set still form an unfounded set (otherwise some of them would have reached one of the removed atoms), containing an active elementary loop by Theorem 2. Finally, the else-case from lines 31 to 34 applies when p is reached by some atom with a greater id. In this case, we have $\mathsf{Q} \neq \emptyset$, since at least the atom picked in line 5 is still contained in Q. For not mistakenly considering an atom unreached, we propagate the greatest id of an atom reaching p to the atom $\mathsf{p}'$ that succeeds p in Q (line 33). Atom $\mathsf{p}'$, removed from Q in line 32 and re-added in line 34, is then re-visited in the next iteration of the outer while-loop from line 7.

Regarding complexity of Algorithm 2, note that a body is explored only once, when the last of its atoms contained in Set is visited for the first time. Also, atoms are added to Act only once, upon re-visits only path information is exchanged via root. Visits of bodies and accompanying updates of reached atoms are bound by the number of edges in the part of the atom-body dependency graph that contains atoms in Set and their connecting bodies.

Extracting active elementary loops from unfounded sets might not be important for genuine ASP-solvers, like *dlv*, *smodels*, and *nomore++*, only aiming at falsification of unfounded sets. But active elementary loops can play a role in SAT-based ASP-solvers, such as *assat*, *cmodels*, and *pbmodels*, since their loop formulas eliminate undesired completion models more effectively than those of terminating loops (Gebser & Schaub 2005).

## Discussion

This paper contributes to computational approaches to unfounded set handling, both theoretically and practically. Unlike already done in the literature (cf. (Lin & Zhao 2004; Lee 2005)), where loops are related to total propositional models, we have put loops into the context of partial assignments. The major result is that active elementary loops form the "cores" of unfounded sets. Hence, they must intrinsically be dealt with by any ASP-solver.

Based on active elementary loops, traditional approaches to unfounded set computation can be explained. Beyond that, new algorithms exploiting active elementary loops are fortified. We have presented an algorithm that allows for computing unfounded sets directly, avoiding the complementation of externally supported sets. This approach is currently implemented in the *nomore++* system. However, it can also be incorporated into other ASP-solvers. In fact, using assignments to both atoms and bodies is not an obligation for our theoretical results and algorithms to apply, it merely allows us to state them in a way that accounts for *nomore++* as well. For brevity, we do not provide experimental results and just report that the usage of Algorithm 1 has greatly improved the performance of the *nomore++* system. This improvement is of course of relative nature and does not indicate any superiority of the approach.

Finally, we have provided an algorithm that exploits the properties of elementary subgraphs to extract active elementary loops from unfounded sets. This algorithm, which is the first of its kind, can be used by SAT-based ASP-solvers to replace terminating loops with active elementary loops.

**Acknowledgments.** This work was supported by DFG (SCHA 550/6-4). We are grateful to Martin Brain, Wolfgang Faber, Joohyung Lee, Yuliya Lierler, and the anonymous referees for many helpful suggestions.

## References

Anger, C.; Gebser, M.; Linke, T.; Neumann, A.; and Schaub, T. 2005. The `nomore++` approach to answer set solving. In Sutcliffe, G., and Voronkov, A., eds., *LPAR*, 95–109. Springer-Verlag.

Apt, K.; Blair, H.; and Walker, A. 1987. Towards a theory of declarative knowledge. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. Chapter 2, 89–148.

Calimeri, F.; Faber, W.; Leone, N.; and Pfeifer, G. 2001. Pruning operators for answer set programming systems. Report INFSYS RR-1843-01-07, TU Wien.

Clark, K. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. Plenum. 293–322.

Faber, W. 2002. Enhancing efficiency and expressiveness in answer set programming systems. Dissertation, TU Wien.

Faber, W. 2006. Personal communication.

Fages, F. 1994. Consistency of Clark's completion and the existence of stable models. *J. MLCS* 1:51–60.

Fitting, M. 2002. Fixpoint semantics for logic programming: A survey. *TCS* 278(1-2):25–51.

Gebser, M., and Schaub, T. 2005. Loops: Relevant or redundant? In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *LPNMR*, 53–65. Springer-Verlag.

Gebser, M., and Schaub, T. 2006. Tableau calculi for answer set programming. In Dix, J., and Hunter, A., eds., *NMR*. This volume.

Gebser, M.; Lee, J.; and Lierler, Y. 2006. Elementary sets for logic programs. In Dix, J., and Hunter, A., eds., *NMR*. This volume.

Janhunen, T. 2003. Translatability and intranslatability results for certain classes of logic programs. Report A82, Helsinki UT.

Lee, J. 2005. A model-theoretic counterpart of loop formulas. In Kaelbling, L., and Saffiotti, A., eds., *IJCAI*, 503–508. Professional Book Center.

Leone, N.; Faber, W.; Pfeifer, G.; Eiter, T.; Gottlob, G.; Koch, C.; Mateis, C.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM TOCL*. To appear.

Leone, N.; Rullo, P.; and Scarcello, F. 1997. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Inf. Comput.* 135(2):69–112.

Lierler, Y., and Maratea, M. 2004. Cmodels-2: SAT-based answer sets solver enhanced to non-tight programs. In Lifschitz, V., and Niemelä, I., eds., *LPNMR*, 346–350. Springer-Verlag.

Lifschitz, V., and Razborov, A. 2006. Why are there so many loop formulas? *ACM TOCL*. To appear.

Lin, F., and Zhao, Y. 2002. ASSAT: computing answer sets of a logic program by SAT solvers. In *AAAI*, 112–118. AAAI/MIT Press.

Lin, F., and Zhao, J. 2003. On tight logic programs and yet another translation from normal logic programs to propositional logic. In Gottlob, G., and Walsh, T., eds., *IJCAI*, 853–858. Morgan Kaufmann.

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

Linke, T., and Sarsakov, V. 2005. Suitable graphs for answer set programming. In Baader, F., and Voronkov, A., eds., *LPAR*, 154–168. Springer-Verlag.

Liu, L., and Truszczyński, M. 2005. Pbmodels - software to compute stable models by pseudoboolean solvers. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *LPNMR*, 410–415. Springer-Verlag.

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

Simons, P. 2000. Extending and implementing the stable model semantics. Dissertation, Helsinki UT.

van Gelder, A.; Ross, K.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *J. ACM* 38(3):620–650.