*TLP* **18** (3–4): 319–336, 2018.  
doi:10.1017/S1471068418000145

# *Cautious reasoning in ASP via minimal models and unsatisfiable cores*

MARIO ALVIANO
*DEMACS, University of Calabria, Italy*
(*e-mail:* alviano@mat.unical.it)

CARMINE DODARO
*DIBRIS, University of Genova, Italy*
(*e-mail:* dodaro@dibris.unige.it)

MATTI JÄRVISALO
*HIIT, Department of Computer Science, University of Helsinki, Finland*
(*e-mail:* matti.jarvisalo@helsinki.fi)

MARCO MARATEA
*DIBRIS, University of Genova, Italy*
(*e-mail:* marco@dibris.unige.it)

ALESSANDRO PREVITI
*HIIT, Department of Computer Science, University of Helsinki, Finland*
(*e-mail:* alessandro.previti@helsinki.fi)

*submitted 27 April 2018; accepted 11 May 2018*

**Abstract**

Answer Set Programming (ASP) is a logic-based knowledge representation framework, supporting—among other reasoning modes—the central task of query answering. In the propositional case, query answering amounts to computing cautious consequences of the input program among the atoms in a given set of candidates, where a cautious consequence is an atom belonging to all stable models. Currently, the most efficient algorithms either iteratively verify the existence of a stable model of the input program extended with the complement of one candidate, where the candidate is heuristically selected, or introduce a clause enforcing the falsity of at least one candidate, so that the solver is free to choose which candidate to falsify at any time during the computation of a stable model. This paper introduces new algorithms for the computation of cautious consequences, with the aim of driving the solver to search for stable models discarding more candidates. Specifically, one of such algorithms enforces minimality on the set of true candidates, where different notions of minimality can be used, and another takes advantage of unsatisfiable cores computation. The algorithms are implemented in WASP, and experiments on benchmarks from the latest ASP competitions show that the new algorithms perform better than the state of the art.

*KEYWORDS*: Answer Set Programming, Cautious Reasoning, Query Answering

## 1 Introduction

Answer set programming (ASP; Niemelä 1999; Marek and Truszczyński 1999; Baral 2010; Brewka *et al*. 2011; Janhunen and Niemelä 2016; Lifschitz 2016) is today a popular logic-based knowledge representation and constraint programming framework with constructs specifically designed for industrial applications (Gebser *et al*. 2013; Dodaro *et al*. 2016). In ASP, knowledge is expressed via logic programs, also referred to as ASP programs, or simply programs. A program encodes a particular problem domain of interest. The semantic is given by the so-called stable models (or answer sets; Gelfond and Lifschitz 1988; Gelfond and Lifschitz 1991), which represent the plausible scenarios for the input instance. As a logic-based knowledge representation framework, ASP systems support several computational tasks, many of them characterized by higher complexities than the corresponding problems in classical propositional logic (Eiter *et al*. 1997). One central task is query answering, which in ASP is supported also in presence of uninterpreted function symbols (Alviano *et al*. 2010). Query answering can be addressed with two reasoning modes, namely brave and cautious reasoning, providing answers witnessed by some or all stable models, respectively. In particular, cautious reasoning over ASP programs has a significant number of interesting applications, including consistent query answering (Arenas *et al*. 2003; Manna *et al*. 2013), data integration (Eiter 2005), multi-context systems (Brewka *et al*. 2007), and ontology-based reasoning (Eiter *et al*. 2008).

Focusing on the propositional level on which the stable model search algorithm of ASP systems typically operates on, query answering amounts to computing *cautious consequences* of the input program among the atoms in a given set of candidates, where cautious consequences are atoms contained in all stable models. At a symbolic level, more complex queries can be expressed by means of rules defining intentional predicates, whose ground instances form the set of candidates. For example, a union of conjunctive queries requires a rule for each conjunction in the union, the body being the conjunction, and the head using a predicate not occurring in the program and collecting all selected variables.

Currently, the most efficient algorithms for cautious reasoning in ASP, as the ones implemented by CLASP (Gebser *et al*. 2012) and WASP (Dodaro *et al*. 2011; Alviano *et al*. 2013; Alviano *et al*. 2015), are mainly adaptations of techniques proposed for the computation of backbones in the context of propositional logic (Janota *et al*. 2015). The common idea behind such algorithms is to perform several iterative calls to an ASP oracle for computing a stable model of the input program subject to algorithm-dependent additional constraints. The added constraints inferred from each of the computed stable models are guaranteed to discard some atoms from the set of candidates. Specifically, state-of-the-art algorithms for cautious reasoning in ASP either (i) iteratively verify the existence of a stable model of the input program extended with the complement of one candidate, where the candidate is heuristically selected, or (ii) introduce an integrity constraint that enforces the falsity of at least one candidate, so that the solver is free to choose which candidate to falsify at any time during the computation of a stable model. Efficient computation is achieved thanks to the incremental stable model search procedure implemented by modern solvers, which possibly reuses several learned clauses from previous computations in order to avoid redundantly repeating some of the already performed computations.

This paper introduces new algorithms for determining the cautious consequences of ASP programs. The intuition behind the proposed algorithms is to drive the search for stable models so that a high number of candidate atoms is discarded at a time. In more detail, two different algorithmic approaches are proposed, based on *minimal models* and *unsatisfiable cores*. The first of the algorithmic approaches introduced in the paper enforces minimality on the set of

true candidates. This approach in fact constitutes a meta-algorithm that can be instantiated with several minimality notions and different minimality procedures; two instantiations are presented in this paper, one based on the computation of subset-minimal models via qualitative preferences (Di Rosa *et al*. 2010), and the other based on the computation of cardinality-minimal models via algorithm ONE (Alviano *et al*. 2015) recently proposed in the context of maximum satisfiability (MaxSAT) solving.

The second algorithm for cautious reasoning introduced in this paper takes advantage of unsatisfiable cores computation, which has already proven very effective for solving ASP optimization problems (Alviano and Dodaro 2016). For some intuition on this approach, modern ASP solvers take as input a program and a set of assumption literals, and search for a stable model of the given program extending the assumption literals. If such a stable model does not exist, a set of assumption literals that is sufficient to cause the failure of the search is returned. Such a sufficient set of assumption literals constitutes an unsatisfiable core in the context of ASP optimization (Alviano and Dodaro 2017). Accordingly, an atom is a cautious consequence of a given program if and only if the negation of the atom is an unsatisfiable core. Hence, the new algorithm searches for stable models falsifying all candidates, with the aim of eliminating all remaining candidates at once. As soon as no stable model under the current set of assumptions exists, the returned unsatisfiable core is either minimized to a singleton or used to discard candidates.

The new algorithms are implemented in WASP, which provides a fair basis for comparing the algorithms proposed in this paper with the two state-of-the-art cautious reasoning approaches previously implemented in WASP (Alviano *et al*. 2014). Indeed, in this way all ASP oracle calls are realized by the incremental stable model search procedure provided by WASP, enabling a clean comparison of the behavior and effectiveness of the algorithms. The implemented algorithms are empirically tested on benchmarks from the latest ASP competitions (Calimeri *et al*. 2016; Gebser *et al*. 2017). The experiments show that the algorithms proposed in this paper perform better than the state of the art on the available test cases. Moreover, the efficiency of the new algorithms is confirmed via a comparison with the state-of-art ASP solver CLASP (Gebser *et al*. 2012), which also supports cautious reasoning. While CLASP is known to be typically faster than WASP in deciding the existence of stable models (i.e., satisfiability checking), the new cautious reasoning algorithms proposed in this paper, as implemented in WASP, outperform CLASP on the task of cautious reasoning.

This paper is structured as follows. Section 2 provides necessary background on ASP, including syntax and semantics, and Section 3 includes an overview of existing state-of-the-art algorithms for cautious reasoning. The new algorithms for cautious reasoning are detailed in Section 4, and Section 5 provides results from an empirical comparison of the new algorithms and earlier proposed approaches. Before conclusions, related work is discussed in Section 6.

## 2 Preliminaries

This section overviews preliminaries on ASP and the main steps of stable model search as necessary for the rest of the paper.

*Syntax.* Let $\mathscr{A}$ be a fixed, countable set of (propositional) *atoms* including $\bot$. A *literal* $\ell$ is either an atom $p$, or its negation $\sim p$, where $\sim$ denotes *default negation*. Let $\overline{\ell}$ denote the complement of $\ell$, i.e., $\overline{p} := \sim p$, and $\overline{\sim p} := p$, for all $p \in \mathscr{A}$. For a set $L$ of literals, $\overline{L} := \{\overline{\ell} \mid \ell \in L\}$, $L^+ := L \cap \mathscr{A}$, and $L^- := \overline{L} \cap \mathscr{A}$. A *program* $\Pi$ consists of a set of *disjunctive* and *choice rules*, which have

respectively the following forms:

$$p_1 \vee \cdots \vee p_m \leftarrow \ell_1, \ldots, \ell_n \tag{1}$$

$$\{\ell_1, \ldots, \ell_n\} \geq k \tag{2}$$

where $p_1, \ldots, p_m$ are distinct atoms, $\ell_1, \ldots, \ell_n$ are distinct literals, and $k, m, n \geq 0$. For a rule $r$ of the form (1), the disjunction $p_1 \vee \cdots \vee p_m$ is called the *head* of $r$, denoted $H(r)$; the conjunction $\ell_1, \ldots, \ell_n$ is the *body* of $r$, denoted $B(r)$. With a slight abuse of notation, $H(r)$ and $B(r)$ also denote the sets of their elements. A *constraint* is a rule of the form (1) such that $H(r) \subseteq \{\bot\}$. For a rule $r$ of the form (2), let $lits(r)$ be $\{\ell_1, \ldots, \ell_n\}$, and $bound(r)$ be $k$.

*Semantics.* An *interpretation* $I$ is a subset of $\mathscr{A} \setminus \{\bot\}$; atoms in $I$ are assigned true, and those in $\mathscr{A} \setminus I$ are assigned false. $I$ is a model of a rule $r$ of the form (1), denoted $I \models r$, if $H(r) \cap I \neq \emptyset$ whenever $B(r)^+ \subseteq I$ and $B(r)^- \cap I = \emptyset$. $I$ is a model of a rule $r$ of the form (2), denoted $I \models r$, if $|lits(r)^+ \cap I| + |lits(r)^- \setminus I| \geq bound(r)$ holds. $I$ is a model of a program $\Pi$, denoted $I \models \Pi$, if it is a model of all rules in $\Pi$. The definition of stable model is based on the following notion of reduct. Let $\Pi$ be a program, and $I$ be an interpretation. The *reduct* of $\Pi$ with respect to $I$, denoted $\Pi^I$, is obtained from $\Pi$ by deleting each rule $r$ of the form (1) such that $B(r)^- \cap I \neq \emptyset$, by replacing each rule $r$ of the form (2) with rules of the form $p \leftarrow$ for all $p \in lits(r) \cap I$, and removing negated atoms in the remaining rules. An interpretation $I$ is a *stable model* of $\Pi$ if $I \models \Pi^I$ and there is no $J \subset I$ such that $J \models \Pi^I$. Let $SM(\Pi)$ denote the set of stable models of $\Pi$. If $SM(\Pi) \neq \emptyset$ then $\Pi$ is *coherent*, otherwise it is *incoherent*. An atom $p \in \mathscr{A}$ is a *cautious consequence* of a program $\Pi$ if $p$ belongs to all stable models of $\Pi$. The set of cautious consequences of $\Pi$ is denoted $CC(\Pi)$.

*Example 2.1* (*Running example*)
Let $\Pi_{run}$ be the following program:

$$\begin{array}{llllllll} r_1: & a \vee b \leftarrow & r_2: & c \vee d \leftarrow & r_3: & q_1 \leftarrow a & r_4: & q_1 \leftarrow b \\ r_5: & q_2 \leftarrow c & r_6: & q_3 \leftarrow {\sim}c & r_7: & q_3 \leftarrow {\sim}d & r_8: & q_4 \leftarrow d. \end{array}$$

$SM(\Pi_{run})$ comprises the following stable models: $I_1 := \{a, d, q_1, q_3, q_4\}$, $I_2 := \{a, c, q_1, q_2, q_3\}$, $I_3 := \{b, d, q_1, q_3, q_4\}$, and $I_4 := \{b, c, q_1, q_2, q_3\}$. Therefore, the set $CC(\Pi_{run})$ of cautious consequences of $\Pi_{run}$ is $\{q_1, q_3\}$. ◁

*Stable model search.* State-of-the-art algorithms for computing stable models of a given ASP program $\Pi$ extend the conflict-driven clause learning (CDCL) algorithm (Silva and Sakallah 1999) with ASP-specific search techniques (Kaufmann *et al.* 2016). The input of the algorithm comprises a propositional program $\Pi$, and a set $A$ of literals, called *assumption literals* (or simply assumptions). Its output is a pair $(I, \emptyset)$, where $I$ is a stable model of $\Pi$ such that $A^+ \subseteq I$ and $A^- \cap I = \emptyset$, if such an $I$ does exist. Otherwise, if there is no $I \in SM(\Pi)$ such that $A^+ \subseteq I$ and $A^- \cap I = \emptyset$, the algorithm returns as output a set $(\bot, C)$, where $C \subseteq A$ is such that $SM(\Pi \cup \{\bot \leftarrow \overline{\ell} \mid \ell \in C\}) = \emptyset$; such a set $C$ is called an *unsatisfiable core* of $\Pi$ with respect to $A$. In what follows, ComputeStableModel$(\Pi, A)$ denotes a call to the stable model search algorithm.

*Example 2.2* (*Continuing Example 2.1*)
ComputeStableModel$(\Pi_{run}, \{b, {\sim}q_2\})$ necessarily returns $(I_3, \emptyset)$, where $I_3$ is the stable model $\{b, d, q_1, q_3, q_4\}$, while ComputeStableModel$(\Pi_{run}, \{{\sim}q_1, {\sim}q_2\})$ must return $(\bot, C)$, where $C$ is either the unsatisfiable core $\{{\sim}q_1\}$, or the unsatisfiable core $\{{\sim}q_1, {\sim}q_2\}$. ◁

**Algorithm 1:** CautiousReasoning(REFINE, $\Pi$,$Q$)

```
1 U := ∅;   O := Q;
2 (I,C) := ComputeStableModel(Π, ∅);
3 if I = ⊥ then return ⊥;
4 O := O ∩ I;
5 while U ≠ O do REFINE;              // ICT, OR, MIN, CM, or other procedure
6 return U;
```

**Procedure** OR

```
1 (I,C) := ComputeStableModel(Π ∪ {⊥ ← O}, ∅);
2 if I = ⊥ then U := O;
3 else O := O ∩ I;
```

## 3 Existing algorithms for cautious reasoning in ASP

Algorithm 1 provides a common skeleton shared by several algorithms for computing cautious consequences of ASP programs (Alviano *et al*. 2014). These algorithms take as input a program $\Pi$ and a set of atoms $Q$, and produce as output either $Q \cap CC(\Pi)$ (i.e., the largest subset of $Q$ that only contains cautious consequences of $\Pi$) in case $\Pi$ is coherent, or $\bot$ when $\Pi$ is incoherent. To this aim, an underestimation $U$ and an overestimation $O$ are initially set to $\emptyset$ and $Q$, respectively, and updated during computation. A first coherence test of $\Pi$ is performed by means of function ComputeStableModel (line 2), so that $\bot$ is returned if $\Pi$ is incoherent (line 3). Otherwise, if a stable model $I$ is found, it is used to possibly reduce the overestimation (line 4), as atoms not in $I$ cannot be cautious consequences of $\Pi$.

*Example 3.1* (*Execution of Algorithm 1*)
Consider program $\Pi_{run}$ from Example 2.1, and the execution of CautiousReasoning (REFINE, $\Pi_{run}, \{q_1, q_2, q_3, q_4\}$). The call to ComputeStableModel($\Pi_{run}, \emptyset$) at line 2 returns a stable model of $\Pi_{run}$, say $I_1$. Hence, $O$ is updated to $\{q_1, q_3, q_4\}$, i.e., the candidate $q_2$ is discarded. ◁

After the initial steps, the underestimation and the overestimation are refined until they are equal, both representing exactly the cautious consequences of $\Pi$ (lines 5–6). The way the estimations are iteratively refined depends on the cautious reasoning algorithm (or strategy) at hand. Two previously proposed and implemented strategies, referred to as *overestimate reduction* (OR) and *iterative coherence testing* (ICT), are detailed next.

Strategy OR iteratively searches for new stable models to improve the overestimation $O$. To this aim, a constraint of the form $\bot \leftarrow O$ is added to $\Pi$, so that at least one of the atoms in $O$ must be false in the computed stable model. If such a stable model does not exist, then the underestimation is set equal to the overestimation to terminate the computation.

*Example 3.2* (*Execution of* OR*; continuing Example 3.1*)
A stable model of $\Pi_{run} \cup \{\bot \leftarrow q_1, q_3, q_4\}$ is searched for. Say that $I_4$ is found. Then $O$ is set to $\{q_1, q_3\}$, and a stable model of $\Pi_{run} \cup \{\bot \leftarrow q_1, q_3\}$ is searched for. However, the program is incoherent, and therefore $U$ is set to $\{q_1, q_3\}$, and the algorithm terminates. ◁

Strategy ICT can improve both estimations during its computation. In fact, one cautious consequence candidate, say $a$, is selected by a heuristic function OneOf. The complement of the

**Procedure** ICT

1 $a := \text{OneOf}(O \setminus U)$;
2 $(I, C) := \text{ComputeStableModel}(\Pi, \{\sim a\})$;
3 **if** $I = \bot$ **then** $U := U \cup \{a\}$ ;
4 **else** $O := O \cap I$ ;

**Procedure** MIN

1 $I := \text{ComputeMinStableModel}(\Pi, O)$;
2 **if** $I \supseteq O$ **then** $U := O$ ;
3 **else** $O := O \cap I$ ;

selected candidate is put in the assumption literals in order to search for a stable model of $\Pi$ not containing $a$. If such a stable model does not exist, then the underestimation can be improved by adding $a$. Otherwise, the stable model found is used to improve the overestimation.

*Example 3.3* (*Execution of* ICT*; continuing Example 3.1*)
Say that $\text{OneOf}(\{q_1, q_3, q_4\})$ selects $q_1$. $\text{ComputeStableModel}(\Pi_{run}, \{\sim q_1\})$ returns $(\bot, \{\sim q_1\})$ because there are no stable models of $\Pi_{run}$ where $q_1$ is false. Hence $q_1$ is added to $U$. After that, say that $\text{OneOf}(\{q_3, q_4\})$ selects $q_4$. $\text{ComputeStableModel}(\Pi_{run}, \{\sim q_4\})$ returns a stable model, say $I_4 = \{b, c, q_1, q_2, q_3\}$. Hence, $O$ is set to $\{q_1, q_3\}$, and $\text{OneOf}(\{q_3\})$ has to return $q_3$. Finally, $\text{ComputeStableModel}(\Pi_{run}, \{\sim q_3\})$ returns $(\bot, \{\sim q_3\})$, and therefore $q_3$ is added to $U$, and the algorithm terminates returning $\{q_1, q_3\}$. ◁

## 4 New algorithms for cautious reasoning in ASP

The skeleton provided by Algorithm 1 can be used by strategies other than ICT and OR, as long as the estimations are iteratively updated and are guaranteed to converge to the set of cautious consequences of the input program $\Pi$ among the candidates in $Q$. In this section, new strategies are presented, with the aim of discarding several candidates at once and thereby speeding up the convergence of the estimations. Specifically, one of the new strategies is based on the search of minimal stable models, and actually results into a meta-algorithm that can be instantiated with different search procedures (Section 4.1). The other strategy introduced here is based on an alternative characterization of cautious consequences in terms of the notion of unsatisfiable cores used in this paper (Section 4.2).

### *4.1 Cautious reasoning via minimal models*

A stable model $I$ of a program $\Pi$ is said to be *minimal* with respect to a set $O$ of objective atoms if there is no $I' \in SM(\Pi)$ such that $I' \cap O \subset I \cap O$. Strategy MIN takes advantage of this notion, specifically of function ComputeMinStableModel, whose input are a coherent program $\Pi$ and a set $O$ of objective atoms, and whose output is a minimal stable model of $\Pi$. Armed with such a function, strategy MIN searches for stable models of $\Pi$ that are minimal with respect to the current overestimation $O$. The stable models returned by ComputeMinStableModel either discard some candidate from $O$, which would lead to a new iteration of the strategy, or are such

**Function** OPT($\Pi$, $O$)

```
1 A := [];                                   // stack of assumptions, also used as a set
2 loop
3     while O \ (A ∪ Ā) ≠ ∅ do                          // unassigned candidates?
4         A.push(~OneOf(O \ (A ∪ Ā)));
5         propagate(Π, A);
6     (I, C) := ComputeStableModel(Π, A);
7     if I ≠ ⊥ then return I;                          // I is a minimal model
8     while C ⊈ A do A.pop();                          // restore consistency
```

that all atoms in $O$ are true. In the latter case, the set $O$ only contains cautious consequences of $\Pi$, and therefore the underestimation is updated to terminate the algorithm.

*Example 4.1* (*Execution of* MIN*; continuing Example 3.1*)
Say that ComputeMinStableModel($\Pi_{run}, \{q_1, q_3, q_4\}$) returns $I_2 = \{a, c, q_1, q_2, q_3\}$. Hence $O$ is set to $\{q_1, q_3\}$. Say that the next call to ComputeMinStableModel($\Pi_{run}, \{q_1, q_3\}$) returns $I_3 = \{b, d, q_1, q_3, q_4\}$. Since $I_3 \supseteq O$, $U$ is set to $\{q_1, q_3\}$, which is returned by the algorithm. ◁

*Theorem 4.1*
Given a program $\Pi$, and a set $Q$ of atoms, CautiousReasoning(MIN, $\Pi, Q$) terminates and returns either $\bot$ (if $SM(\Pi) = \emptyset$) or $CC(\Pi) \cap Q$ (otherwise).

*Proof*
Let $\Pi$ be coherent (otherwise correctness is immediate). Termination is guaranteed by the fact that at each iteration a stable model $I$ is found, and either $O$ is shrunk (if $I \not\supseteq O$), or $U$ is set equal to $O$. In fact, if $O \neq CC(\Pi) \cap Q$, then there is a stable model $I$ of $\Pi$ such that $I \not\supseteq O$, and therefore any stable model $I'$ of $\Pi$ such that $I' \supseteq O$ is not minimal with respect to $O$; hence, in this case $O$ is shrunk. Otherwise, if $O = CC(\Pi) \cap Q$, then every minimal stable model $I$ of $\Pi$ with respect to $O$ must be such that $I \supseteq O$, and therefore $O$ is the answer to be provided in output. □

MIN is actually a meta-algorithm that can be instantiated in several ways by using different versions of function ComputeMinStableModel. Two versions of this function are presented, one based on an algorithm developed in the context of qualitative preferences (OPT; Di Rosa *et al*. 2010) and another based on a MaxSAT algorithm (ONE; Alviano *et al*. 2015).

OPT. Subset-minimality is among the qualitative preferences analyzed by Di Rosa *et al*. (2010), and addressed by modifying the search procedure of a SAT solver; the resulting algorithm is referred to as OPT in the literature, and was also adapted to ASP (Alviano *et al*. 2015; Gebser *et al*. 2013). Intuitively, the branching heuristic of the solver is forced to select $\overline{p}$ for $p \in O$, where $O$ is the set of objective atoms, before any other unassigned literal. In this way, the search is driven to falsify as many atoms in $O$ as possible. When all atoms in $O$ are assigned, standard stable model search procedure is applied without subjecting the branching heuristic to any further guidance. Hence, if a stable model is found, it is guaranteed to be minimal with respect to the set of objective atoms (Di Rosa *et al*. 2010). If instead the current assignment to atoms in $O$ cannot be extended to a stable model, then a conflict involving some objective atom has to be detected. In this case, the assignment of some atom in $O$ is flipped because of the constraint learned by

**Function** ONE($\Pi$, $O$)

1 $S := \overline{O}$;
2 **loop**
3 $(I, C) := \mathit{ComputeStableModel}(\Pi, S)$;
4 **if** $I \neq \bot$ **then return** $I$;
5 Let $C$ be $\{\ell_0, \ldots, \ell_n\}$ (for some $n \geq 0$), and $p_1, \ldots, p_n$ be $|C| - 1$ fresh atoms;
6 $S := (S \setminus C) \cup \{\sim p_1, \ldots, \sim p_n\}$;
7 $\Pi := \Pi \cup \{\{\ell_0, \ldots, \ell_n, p_1, \ldots, p_n\} \geq n\} \cup \{\bot \leftarrow p_i, \sim p_{i-1} \mid i \in [2..n]\}$;

analyzing the conflict, and hence the procedure is repeated with a different assignment for the objective atoms. Function OPT reports such a strategy. A stack $A$ of assumption literals, initially empty (line 1), is populated with complements of candidates in $O$ (line 4). After each insertion, $\mathit{propagate}(\Pi, A)$ is used to extend $A$ with (unit) implied literals, if possible; otherwise, in case of conflict, $\Pi$ is extended with a learned clause, some literal in $A$ is flipped, and propagation is repeated. When all atoms in $O$ occur in $A$, a stable model extending $A$ is searched (line 6). If it is found, it is returned (line 7). Otherwise, some literals in $A$ are removed so to not incur again in the returned unsatisfiable core (line 8).

*Example 4.2* (*Execution of* OPT*; continuing Example 3.1*)
Say that $\text{OPT}(\Pi_{run}, \{q_1, q_3, q_4\})$ starts by selecting $\sim q_1$. Hence $\sim a$ and $\sim b$ are inferred via $r_3$ and $r_4$, respectively. However, this is not possible due to $r_1$, that is, there is a conflict. The program is therefore extended with the learned constraint $\bot \leftarrow \sim q_1$, so that $q_1$ cannot be selected anymore by the branching heuristic. Now, say that $\sim q_4$ is selected. Literal $\sim d$ is inferred via $r_8$, and therefore $c$ and $q_3$ are inferred via $r_2$ and $r_7$, respectively. (In addition, $q_2$ is then inferred via $r_5$.) After that, the branching heuristic is not subject to any guidance, and a stable model of $\Pi_{run}$, say $I_2 = \{a, c, q_1, q_2, q_3\}$, is returned. ◁

ONE. MaxSAT algorithms, being the state-of-the-art for the computation of cardinality-minimal models, can be also used to compute subset-minimal models. Specifically, the algorithm ONE has been used to enumerate models of circumscribed theory (Alviano 2017a). Simplified to the setting of this paper, function ONE takes as input a coherent program $\Pi$ and a set $O$ of objective atoms, and returns a minimal stable model of $\Pi$ with respect to $O$. Hence, it is a valid instantiation of ComputeMinStableModel. In more detail, ONE keeps a set $S$ of *soft literals* to be maximized, initially set to the complements of the atoms in $O$ (line 1). At each step of computation, a stable model of $\Pi$ subject to the assumptions $S$ is searched (line 3), and eventually returned (line 4). When the search fails, instead, soft literals in the computed unsatisfiable core are replaced by fresh literals (line 6), and a choice rule enforcing the satisfaction of at least $n$ literals among those in the unsatisfiable core and those fresh is added to the program (line 7). Since the next stable model search is forced to assign false to the fresh atoms, the choice rule actually drives the search to a stable model containing at least $n$ literals among those in the unsatisfiable core. Additionally, constraints of the form $\bot \leftarrow p_i, \sim p_{i-1}$ are added to the program to eliminate symmetric solutions.

*Example 4.3* (*Execution of* ONE*; continuing Example 3.1*)
$\text{ONE}(\Pi_{run}, \{q_1, q_3, q_4\})$ initially searches for a stable model of $\Pi_{run}$ subject to the assumption literals $\{\sim q_1, \sim q_3, \sim q_4\}$. Say that $(\bot, \{\sim q_1, \sim q_4\})$ is returned. $\Pi_{run}$ is extended with $\{\sim q_1, \sim q_4, p_1\} \geq$

**Procedure** CM

1 $C := \overline{O \setminus U}$; $C' := \emptyset$;
2 **while** $C \neq \emptyset$ **do**
3 $(I, C) :=$ ComputeStableModel$(\Pi, C)$;
4 **if** $I \neq \bot$ **then**
5 $O := O \cap I$; $C := C'$; $C' := \emptyset$;
6 **else**
7 $C' :=$ {OneOf$(C)$}; $C := C \setminus C'$;
8 **if** $C' \neq \emptyset$ **then** $U := U \cup \overline{C'}$;

1, and $S$ is modified to $\{\sim q_3, \sim p_1\}$. Say that the next stable model search returns $(\bot, \{\sim q_3\})$, so that $S$ is now $\{\sim p_1\}$. The next stable model search returns a model, say $I_2 = \{a, c, q_1, q_2, q_3\}$. ◁

### *4.2 Cautious reasoning via unsatisfiable core minimization*

Unsatisfiable cores, as considered in this paper, provide an alternative characterization of the notion of cautious consequences, formalized as the following proposition.

*Proposition 4.1*
Assume that $\Pi$ is a program and $A$ is a set of atoms. Then $p \in CC(\Pi) \cap A$ if and only if $\{\sim p\}$ is an unsatisfiable core of $\Pi$ with respect to $A$.

*Proof*
Follows from $p \in CC(\Pi) \cap A$ if and only if $p \in I$ for all $I \in SM(\Pi)$ if and only if $SM(\Pi \cup \{\bot \leftarrow \sim p\}) = \emptyset$ if and only if $\{\sim p\}$ is an unsatisfiable core of $\Pi$ with respect to $A$. Note that the claim also holds for incoherent programs: $SM(\Pi) = \emptyset$ implies $CC(\Pi) = \mathscr{A}$, and $SM(\Pi \cup \{\bot \leftarrow \sim p\}) = \emptyset$ for all $p \in \mathscr{A}$. □

An immediate consequence of Proposition 4.1 is that a subset-minimal unsatisfiable core containing more than one atom can be used to discard candidates, since none of the atoms in the unsatisfiable core are in fact cautious consequences of the input program in this case. Algorithm CM takes advantage of this fact, and searches for subset-minimal unsatisfiable cores. If a singleton is found, a cautious consequence of the input program is identified. Otherwise, all atoms in the unsatisfiable core are discarded. Actually, in order to compute a subset-minimal unsatisfiable core, an initial unsatisfiable core is minimized by searching for stable models containing all atoms in the unsatisfiable core but one. It turns out that such a stable model can be already used to discard several candidates, even before the minimization process is completed.

In more detail, CM stores in $C$ the unsatisfiable core candidate, that is, the assumption literals for stable model searches performed by the algorithm. Initially, $C$ contains the complements of all unknown candidates (line 1), and a stable model is searched (line 3). If the search fails, then the unsatisfiable core identified by ComputeStableModel is partitioned among $C$ and $C'$, where $C'$ only contains one heuristically selected literal (line 7). This process is repeated until either $C$ is empty, or a stable model is found. A stable model would necessarily assign all literals in $C$ as true, so that the associated candidates can be discarded at once (line 5); in this case the only candidate possibly involved in a singleton unsatisfiable core is the one stored in $C'$, if any, and therefore it is moved into $C$ (line 5; if it is involved in an unsatisfiable core, it will be moved

back to C' at line 7). When the while-loop terminates, if $C'$ still contains a single candidate, it is guaranteed to be a singleton unsatisfiable core, and therefore a cautious consequence of the input program (line 8).

*Example 4.4* (*Execution of* CM*; continuing Example 3.1*)
Initially $C = \{\sim q_1, \sim q_3, \sim q_4\}$ and $C' = \emptyset$. Say that ComputeStableModel($\Pi_{run}, C$) returns ($\bot$, $\{\sim q_1, \sim q_4\}$), and that OneOf($\{\sim q_1, \sim q_4\}$) returns $\sim q_1$, so that the next iteration has $C' = \{\sim q_1\}$ and $C = \{\sim q_4\}$. A stable model is found, say $I_2 = \{a, c, q_1, q_2, q_3\}$, and $O$ is shrunk to $\{q_1, q_3\}$. Moreover, the next iteration has $C = \{\sim q_1\}$ and $C' = \emptyset$. Hence, $(\bot, \{\sim q_1\})$ is returned by ComputeStableModel, and $q_1$ is added to $U$ (after moving its complement into $C'$). ◁

*Theorem 4.2*
Given a program $\Pi$, and a set $Q$ of atoms, CautiousReasoning(CM, $\Pi, Q$) terminates, and returns either $\bot$ (if $SM(\Pi) = \emptyset$) or $CC(\Pi) \cap Q$ (otherwise).

*Proof*
Let $\Pi$ be coherent (otherwise correctness is immediate). First of all, any execution of procedure CM terminates. In fact, if $\overline{O \setminus U}$ is not an unsatisfiable core of $\Pi$, then the stable model returned by ComputeStableModel is used to discard all unknown candidates, and $C$ is assigned the empty set. Otherwise, if $\overline{O \setminus U}$ is an unsatisfiable core of $\Pi$, ComputeStableModel returns a possibly smaller unsatisfiable core, which is partitioned among $C$ and the singleton $C'$; the process is repeated ($|O| - |U|$ times in the worst case) until $C$ is not an unsatisfiable core, and therefore possibly used to discard candidates from $O$. At this point, if $C'$ is an unsatisfiable core, the complement of the literal in $C'$ is a cautious consequence of $\Pi$ because of Proposition 4.1. Second, the algorithm itself terminates because each execution of procedure CM shrinks $O$ or extends $U$ (or both); hence, CM is executed at most $|O| - |U|$ times. Finally, note that no cautious consequence of $\Pi$ can be eliminated at line 5 because $CC(\Pi) \subseteq I$ holds for all $I \in SM(\Pi)$, and therefore all of them are added to $U$ at line 8 during some execution of procedure CM. □

## 5 Experiments

The new algorithms are implemented in WASP (Alviano *et al*. 2014), and their performance was measured on all instances from the latest ASP Competitions (Calimeri *et al*. 2014; Calimeri *et al*. 2016; Gebser *et al*. 2017) involving non-ground queries. (Ground queries are not included in the experiment because they are usually handled by adding an integrity constraint to the input program, and by searching for a single answer set.) Instances are grounded by GRINGO v. 4.4.0 (Gebser *et al*. 2011), which therefore also produces the set of candidates. As a reference to the state of the art, CLASP v. 3.3.3 (Gebser *et al*. 2012) was also tested; CLASP implements algorithm OR. The DLV solver is not considered here as its performance on cautious reasoning has been shown in earlier work to be dominated by the other approaches considered here (Alviano *et al*. 2014). The experiments were run on computing nodes with Intel Xeon 2.4-GHz processors and 16 GB of memory. Per-instance time and memory limits were set to 600 seconds and 15 GB, respectively. In the following, WASP-$X$ refers to WASP executing the strategy $X \in \{$OR, ICT, OPT, ONE, CM$\}$.

Results are reported in the cactus plot of Figure 1, showing for each algorithm the number of instances solved (x-axis) under different per-instance time limits (y-axis). As a first observation, WASP cannot reach the performance of CLASP on the execution of algorithm OR, and indeed

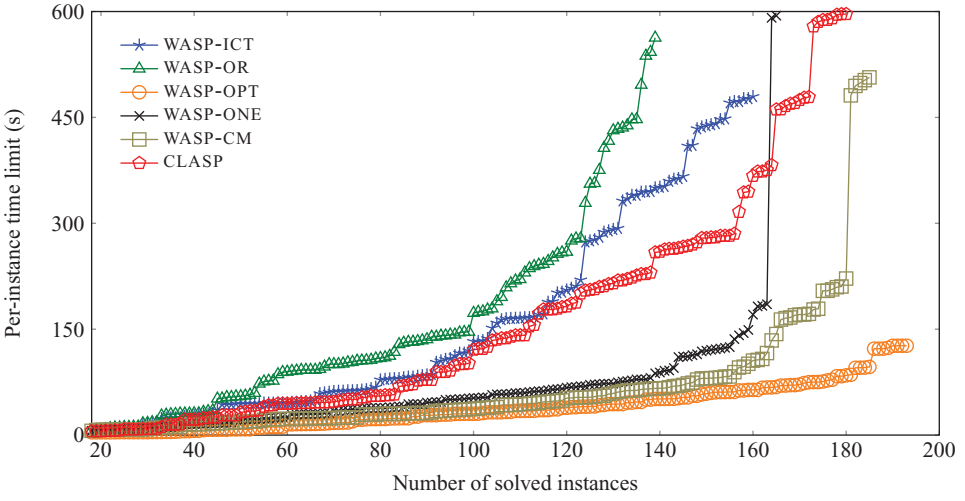


Fig. 1. Performance comparison on non-ground queries in ASP Competitions: number of solved instances within a given per-instance time limit.

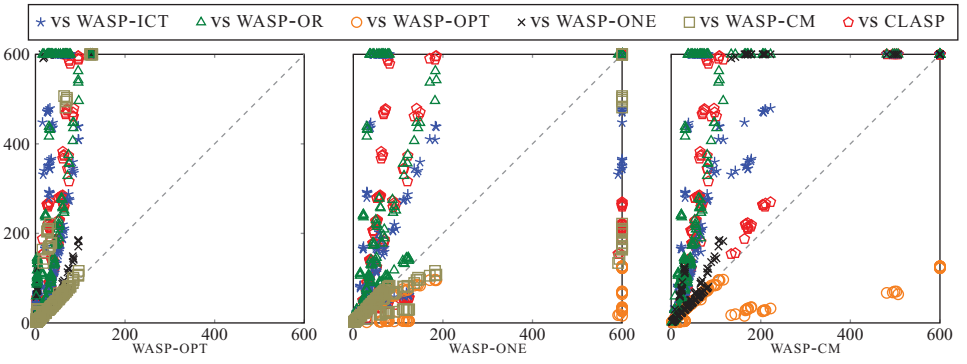


Fig. 2. Instance-by-instance comparison on non-ground queries in ASP Competitions.

CLASP solved 41 instances more than WASP-OR. However, such a huge gap is completely filled by WASP-OPT and WASP-CM, which actually solve 13 and 5 instances more than CLASP, respectively. The best performance overall is obtained by WASP-OPT, which solves all the instances with an average running time of 35.8 seconds. Interestingly, WASP-OPT is able to solve 95% of the tested instances within two minutes, and anyhow all test cases within 150 seconds of computation.

An instance-by-instance comparison is reported on Figure 2. The scatter plots clearly show that the advantage of WASP-OPT and WASP-CM is uniform on all test cases, while the result is mixed for WASP-ONE. For the tested instances, WASP-OPT is faster than WASP-ONE because computing cardinality-minimal models is usually harder than computing subset-minimal models; WASP-OPT is faster than WASP-CM because WASP-CM performs some hard stable model searches due to the unsatisfiable core minimization.

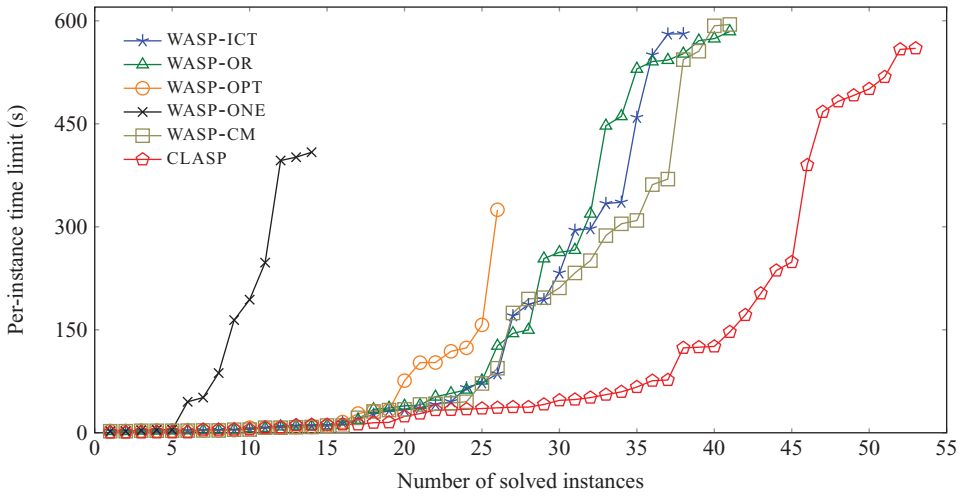


Fig. 3. Performance comparison on computation of cautious consequences for *easy* instances of ASP Competitions: number of solved instances within a given per-instance time limit.

Details on solved instances and the cumulative running times for benchmarks including non-ground queries from ASP Competitions are given in Table 1. As already observed before, the best performance is obtained by WASP-OPT and WASP-CM.

### *5.1 Additional experiments*

Additional experiments are obtained from *easy* instances of ASP Competitions, and from abstract argumentation frameworks by adapting an encoding used by PYGLAF (Alviano 2017b). The performance gain of the new algorithms is less marked on these instances, either because the previous algorithms are already very efficient (abstract argumentation frameworks), or because only few atoms are actually cautious consequences of combinatorial problems in ASP Competitions.

Concerning the instances from 7th ASP Competition, the benchmark set comprises instances classified as easy, that is, those for which a stable model is found within 20 seconds of computation by mainstream ASP systems; these instances were selected for the computationally more demanding task of enumerating all cautious consequences of the programs. However, here one

Table 1. *Numbers of solved instances and cumulative running time (in seconds; each timeout adds 600 seconds) on the benchmarks with non-ground queries from ASP Competitions. Best performance emphasized in bold (within this respect, differences up to 10 seconds are ignored).*

| | | WASP-OR | | WASP-ICT | | WASP-OPT | | WASP-ONE | | WASP-CM | | CLASP | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Benchmark** | **#** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** |
| CQA-Q3 | 40 | 40 | 4346.7 | 40 | 3384.9 | 40 | 1276.3 | 40 | **1272.5** | 40 | 1312.8 | 40 | 4353.9 |
| CQA-Q6 | 40 | 40 | 8410.5 | 40 | 6799.5 | 40 | **1956.1** | 40 | 2948.5 | 40 | 2149.1 | 40 | 8505.0 |
| CQA-Q7 | 40 | 16 | 17636.9 | 20 | 15999.3 | 40 | **1680.9** | 40 | 1701.1 | 40 | 1740.8 | 40 | 8929.5 |
| MCSQ | 73 | 43 | 20646.6 | 60 | 15875.5 | **73** | **1995.3** | 45 | 20050.1 | 65 | 11006.5 | 60 | 12701.0 |
| **Total** | 193 | 139 | 51040.7 | 160 | 42059.2 | **193** | **6908.5** | 165 | 25972.2 | 185 | 16209.3 | 180 | 34489.4 |

Table 2. *Solved instances and cumulative running time (in seconds; each timeout adds 600 seconds) on computation of cautious consequences for* easy *instances of ASP Competitions. Best performance emphasized in bold (within this respect, differences up to 10 seconds are ignored).*

| | | WASP-OR | | WASP-ICT | | WASP-OPT | | WASP-ONE | | WASP-CM | | CLASP | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Benchmark** | **#** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** |
| GracefulGraphs | 1 | 1 | 149.9 | 1 | 85.8 | 1 | **32.0** | 1 | 45.2 | 1 | 45.2 | 1 | 51.2 |
| GraphCol | 1 | 0 | 600.0 | 0 | 600.0 | 0 | 600.0 | 0 | 600.0 | 0 | 600.0 | 0 | 600.0 |
| IncrSched | 6 | 4 | 1904.0 | 1 | 3003.4 | 1 | 3015.7 | 1 | 3164.2 | 2 | 2691.6 | **5** | **857.0** |
| KnightTour | 2 | 0 | 1200.0 | 0 | 1200.0 | 0 | 1200.0 | 0 | 1200.0 | 0 | 1200.0 | **2** | **626.1** |
| Labyrinth | 32 | 1 | 19184.5 | 0 | 19200.0 | 0 | 19200.0 | 0 | 19200.0 | 0 | 19200.0 | **6** | **18377.2** |
| NoMystery | 2 | **1** | **657.4** | 1 | 672.7 | 0 | 1200.0 | 1 | 1001.3 | 1 | 694.0 | 1 | 1091.3 |
| PPM | 15 | 15 | 88.7 | 15 | 94.4 | 15 | **75.6** | 10 | 4350.7 | 15 | **80.8** | 15 | 264.0 |
| QualSpatReas | 18 | 13 | 6569.2 | 15 | 4893.3 | 7 | 7406.4 | 0 | 10800.0 | 17 | 4537.4 | **18** | **1018.6** |
| Sokoban | 36 | **4** | **20353.6** | 3 | 20859.8 | 1 | 21102.1 | 1 | 21051.3 | 3 | 20665.2 | 3 | 20529.0 |
| VisitAll | 2 | 2 | 529.0 | 2 | 364.2 | 1 | 757.1 | 0 | 1200.0 | 2 | 407.8 | **2** | **80.2** |
| **Total** | 115 | 41 | 51236.2 | 38 | 50973.5 | 26 | 54588.9 | 14 | 62612.7 | 41 | 50122.0 | **53** | **43494.7** |

should note that these programs are no *per se* meant to be queried, and therefore the benchmark set is to be considered a crafted one, in contrast to the benchmarks already reported on, which have a clear semantical interpretation in terms of specific applications.

Results are given in the cactus plot of Figure 3 and in Table 2. On the crafted benchmark set, WASP-OR and WASP-CM performed better than WASP-ICT, WASP-ONE, and WASP-OPT. Indeed, the best performance overall was observed for WASP-CM, which solved 41 out of 115 instances with an average running time of 139.6 seconds, while WASP-OR solved the same number of instances but with an higher average running time of 166.7 seconds. A slightly worse performance was reached by WASP-ICT, which solved only 3 instances less than WASP-OR and WASP-CM, while the algorithms based on MIN, i.e., WASP-ONE and WASP-OPT, were less effective, and solved respectively 27 and 15 less instances than WASP-CM and WASP-OR. Moreover, the gap between WASP-OR and CLASP is observed also on these instances, with CLASP solving 12 instances more than WASP-OR. All in all, the benchmark provides a positive result for CM.

The other benchmark is obtained by considering instances of skeptical acceptance under complete extensions in the context of abstract argumentation frameworks. The 2nd International Competition on Computational Models of Argumentation (ICCMA'17) provides a broad set of 350 test cases originating from different contexts; instances can be downloaded from `http://argumentationcompetition.org/2017`. Complete extensions can be computed by adapting the propositional encoding implemented by PYGLAF to the language of ASP as follows:

```
{in(X)} :- arg(X).                              % guess an extension
:- att(X,Y), in(X), in(Y).                      % conflict-free
attacked(X) :- att(Y,X), in(Y).                 % attacked relation
:- att(Y,X), in(X), not attacked(Y).            % admissible
:- arg(X); attacked(Y) : att(Y,X); not in(X). % complete
#show.
#show in/1.
```

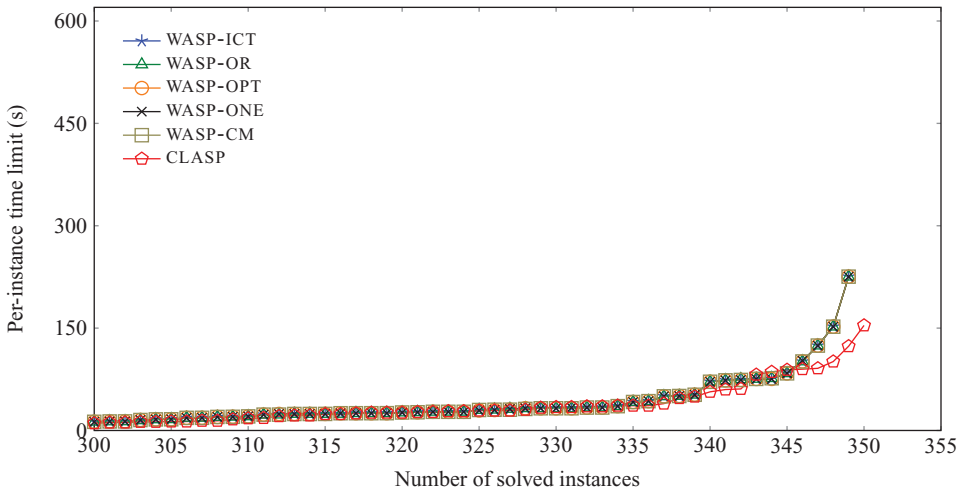


Fig. 4. Performance comparison on skeptical acceptance under complete extensions: number of solved instances within a given per-instance time limit.

Results are given in the cactus plot of Figure 4 and in Table 3. All algorithms behave similarly on these instances, which are easily solved by WASP and CLASP.

## 6 Related work

Cautious reasoning over answer set programs is the underlying computational task of several practical applications, among them Multi Context Systems Querying (MCSQ; Brewka and Eiter 2007) and Consistent Query Answering (CQA; Arenas *et al*. 2003). MCSQ allows to query heterogeneous knowledge bases, called contexts, linked together by bridge rules modeling the flow of information among contexts. CQA, instead, allows to query inconsistent databases by taking advantage of the notion of repair, that is, a maximal and consistent revision of the database (Arenas *et al*. 2003; Manna *et al*. 2013; Manna *et al*. 2015). Both benchmarks were used in Section 5.

In ASP, cautious reasoning has been previously implemented in DLV (Leone *et al*. 2006; Alviano *et al*. 2010; Alviano *et al*. 2017), CLASP (Gebser *et al*. 2012), and WASP (Alviano *et al*.

Table 3. *Solved instances and cumulative running time (in seconds; each timeout adds 600 seconds) on skeptical acceptance under complete extensions. Best performance emphasized in bold (within this respect, differences up to 10 seconds are ignored).*

| | | WASP-OR | | WASP-ICT | | WASP-OPT | | WASP-ONE | | WASP-CM | | CLASP | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Benchmark** | **#** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** | **sol.** | **sum t** |
| C – Dataset 1 | 50 | 50 | 62.5 | 50 | 62.4 | 50 | 62.6 | 50 | 62.4 | 50 | 62.4 | 50 | **18.7** |
| C – Dataset 2 | 50 | 50 | 553.1 | 50 | 552.8 | 50 | 553.3 | 50 | 553.9 | 50 | 553.4 | 50 | **310.3** |
| C – Dataset 3 | 100 | 100 | **628.6** | 100 | **629.1** | 100 | **628.5** | 100 | **630.4** | 100 | **627.6** | 100 | 650.7 |
| C – Dataset 4 | 150 | 149 | 1788.1 | 149 | 1785.8 | 149 | 1785.7 | 149 | 1786.6 | 149 | 1787.1 | **150** | **1184.1** |
| **Total** | 350 | 349 | 3032.3 | 349 | 3030.1 | 349 | 3030.1 | 349 | 3033.3 | 349 | 3030.5 | **350** | **2163.8** |

2015). These systems implement dedicated techniques built on top of their search procedures for computing stable models. The algorithm implemented in DLV is based on the enumeration of all stable models for computing their intersection. CLASP instead implements OR, and WASP implements both OR and ICT (Alviano *et al*. 2014). An abstract framework of cautious reasoning, capturing the above-mentioned algorithms, has been presented by Brochenin and Maratea (2015).

The task of computing the set of cautious consequences of a program $\Pi$ is closely related to computing the backbone of a propositional theory (Janota *et al*. 2015; Zhu *et al*. 2011). Some of the algorithms presented are inspired by previous work in the context of propositional logic. Algorithm OR is based on a strategy for computing the backbone of a conjunctive normal formal propositional formula (Zhu *et al*. 2011), and extended with some optimization techniques by Janota *et al*. (2015). Algorithm ICT is a reimplementation of Algorithm 3 from Janota *et al*. (2015).

Algorithm MIN captures a family of strategies based on the computation of stable models that are subset-minimal with respect to a set of atoms. In particular, the approach to computing subset-minimal stable models adapts the OPTSAT algorithm (Di Rosa *et al*. 2010; Giunchiglia *et al*. 2008) which has been previously employed for the computation of optimal stable models (Alviano *et al*. 2015) in the context of ASP, as well as for computing the ideal semantics in the context of abstract argumentation (Previti and Järvisalo 2018); and another approach is based on algorithm ONE, which was introduced for the first time as a MaxSAT algorithm (Alviano *et al*. 2015). Stable models that are subset-minimal with respect to a set of atoms have also been previously employed in the computation of paracoherent answer sets (Amendola *et al*. 2017), and for circumscription (Alviano 2017a). Other strategies for the computation of minimal models have been previously proposed in early work (Niemelä 1996; Hasegawa *et al*. 2000; Bry and Yahya 2000) preceding the invention of conflict-driven solvers; this limitation was overcome by Koshimura *et al*. (2009). Going beyond the above, general approaches such as the algorithms for computing minimal sets over monotone predicates (MSMP; Janota and Marques-Silva 2016) could also be adapted for computing minimal models.

Although CM is a novel algorithm, it shares some ideas with Algorithm 6 by Janota *et al*. (2015). In particular, both algorithms try to flip all candidates at once. In case a core of size one is returned, both algorithms add the single atom (literal) to the set of cautious consequences (backbone). However, the two algorithms differ in their behavior when a core with size greater than one is returned. The algorithm presented by Janota *et al*. simply stores the literals in the core in a new set, and before termination processes them with an alternative algorithm. Instead, CM starts a minimization procedure of the returned core. If the size of the reduced core is one, then a cautious consequence has been found. Otherwise, during the minimization, the computed stable model is used to improve the overestimation. Moreover, algorithm CM is also *anytime* by itself, since it is able to improve the underestimation during its computation, while algorithm MIN can be made anytime using the technique suggested by Alviano *et al*. (2014).

As a final remark, all considered algorithms work on ground programs. It turns out that they can be used as a back end of query optimization techniques that work at the symbolic level, as for example magic sets (Alviano and Faber 2011; Alviano *et al*. 2012).

## 7 Conclusion

Cautious reasoning over answer set programs is a central query answering task supported by ASP systems and is motivated by various applications. The main contributions of this paper are two new algorithms for cautious reasoning in ASP, their implementation, and empirical evaluation

against earlier proposed approaches to cautious reasoning in ASP. Actually, one of the new algorithms make use of minimal models, and can be instantiated with several procedures, two of them considered in this paper. The other algorithm harnesses the ability of ASP solvers to extract unsatisfiable cores for expediting the elimination of candidate atoms that are not cautious consequences. The empirical results show that the proposed algorithms improve the current state of the art in cautious reasoning on standard benchmarks with non-ground queries from the latest ASP competitions.

## Acknowledgments

Mario Alviano was partially supported by the POR CALABRIA FESR 2014-2020 project "DLV Large Scale" (CUP J28C17000220006), by the EU H2020 PON I&C 2014-2020 project "S2BDW" (CUP B28I17000250008), and by Gruppo Nazionale per il Calcolo Scientifico (GNCS-INdAM). Carmine Dodaro and Marco Maratea were partially supported by project GESTEC - Service Oriented Technologies for Integrated ICT platforms (Italian Ministry for Education Research and University, DM 64565). Matti Järvisalo and Alessandro Previti were supported by Academy of Finland (grants 276412 and 312662).

## References

ALVIANO, M. 2017a. Model enumeration in propositional circumscription via unsatisfiable core analysis. *Theory and Practice of Logic Programming 17,* 5-6, 708–725.

ALVIANO, M. 2017b. The pyglaf argumentation reasoner. In *Technical Communications of the International Conference on Logic Programming*. OASICS, vol. 58. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2:1–2:3.

ALVIANO, M., CALIMERI, F., DODARO, C., FUSCÀ, D., LEONE, N., PERRI, S., RICCA, F., VELTRI, P., AND ZANGARI, J. 2017. The ASP system DLV2. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 10377. Springer, 215–221.

ALVIANO, M. AND DODARO, C. 2016. Anytime answer set optimization via unsatisfiable core shrinking. *Theory and Practice of Logic Programming 16,* 5-6, 533–551.

ALVIANO, M. AND DODARO, C. 2017. Unsatisfiable core shrinking for anytime answer set optimization. In *International Joint Conference on Artificial Intelligence*. ijcai.org, 4781–4785.

ALVIANO, M., DODARO, C., FABER, W., LEONE, N., AND RICCA, F. 2013. WASP: A native ASP solver based on constraint learning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 8148. Springer, 54–66.

ALVIANO, M., DODARO, C., LEONE, N., AND RICCA, F. 2015. Advances in WASP. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 9345. Springer, 40–54.

ALVIANO, M., DODARO, C., MARQUES-SILVA, J., AND RICCA, F. 2015. Optimum stable model search: algorithms and implementation. *Journal of Logic and Computation*. To appear.

ALVIANO, M., DODARO, C., AND RICCA, F. 2014. Anytime computation of cautious consequences in answer set programming. *Theory and Practice of Logic Programming 14,* 4-5, 755–770.

ALVIANO, M., DODARO, C., AND RICCA, F. 2015. A MaxSAT algorithm using cardinality constraints of bounded size. In *International Joint Conference on Artificial Intelligence*. AAAI Press, 2677–2683.

ALVIANO, M. AND FABER, W. 2011. Dynamic magic sets and super-coherent answer set programs. *AI Communications 24,* 2, 125–145.

ALVIANO, M., FABER, W., GRECO, G., AND LEONE, N. 2012. Magic sets for disjunctive datalog programs. *Artificial Intelligence 187*, 156–192.

ALVIANO, M., FABER, W., AND LEONE, N. 2010. Disjunctive ASP with functions: Decidable queries and effective computation. *Theory and Practice of Logic Programming 10,* 4-6, 497–512.

ALVIANO, M., FABER, W., LEONE, N., PERRI, S., PFEIFER, G., AND TERRACINA, G. 2010. The Disjunctive Datalog System DLV. In *Datalog Reloaded*. Lecture Notes in Computer Science, vol. 6702. Springer, 282–301.

AMENDOLA, G., DODARO, C., FABER, W., LEONE, N., AND RICCA, F. 2017. On the computation of paracoherent answer sets. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 1034–1040.

ARENAS, M., BERTOSSI, L. E., AND CHOMICKI, J. 2003. Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Logic Programming 3,* 4-5, 393–424.

BARAL, C. 2010. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

BREWKA, G. AND EITER, T. 2007. Equilibria in heterogeneous nonmonotonic multi-context systems. In *AAAI Conference on Artificial Intelligence*. AAAI Press, 385–390.

BREWKA, G., EITER, T., AND TRUSZCZYNSKI, M. 2011. Answer set programming at a glance. *Communications of the ACM 54,* 12, 92–103.

BREWKA, G., ROELOFSEN, F., AND SERAFINI, L. 2007. Contextual default reasoning. In *International Joint Conference on Artificial Intelligence*. 268–273.

BROCHENIN, R. AND MARATEA, M. 2015. Abstract answer set solvers for cautious reasoning. In *Technical Communications of the International Conference on Logic Programming*. CEUR Workshop Proceedings, vol. 1433. CEUR-WS.org.

BRY, F. AND YAHYA, A. H. 2000. Positive unit hyperresolution tableaux and their application to minimal model generation. *Journal of Automated Reasoning 25,* 1, 35–82.

CALIMERI, F., GEBSER, M., MARATEA, M., AND RICCA, F. 2016. Design and results of the Fifth Answer Set Programming Competition. *Artificial Intelligence 231*, 151–181.

CALIMERI, F., IANNI, G., AND RICCA, F. 2014. The third open answer set programming competition. *Theory and Practice of Logic Programming 14,* 1, 117–135.

DI ROSA, E., GIUNCHIGLIA, E., AND MARATEA, M. 2010. Solving satisfiability problems with preferences. *Constraints 15,* 4, 485–515.

DODARO, C., ALVIANO, M., FABER, W., LEONE, N., RICCA, F., AND SIRIANNI, M. 2011. The birth of a WASP: preliminary report on a new ASP solver. In *Italian Conference on Computational Logic*, F. Fioravanti, Ed. CEUR Workshop Proceedings, vol. 810. CEUR-WS.org, 99–113.

DODARO, C., GASTEIGER, P., LEONE, N., MUSITSCH, B., RICCA, F., AND SCHEKOTIHIN, K. 2016. Combining answer set programming and domain heuristics for solving hard industrial problems (application paper). *Theory and Practice of Logic Programming 16,* 5-6, 653–669.

EITER, T. 2005. Data integration and answer set programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 3662. Springer, 13–25.

EITER, T., GOTTLOB, G., AND MANNILA, H. 1997. Disjunctive datalog. *ACM Transactions on Database Systems 22,* 3, 364–418.

EITER, T., IANNI, G., LUKASIEWICZ, T., SCHINDLAUER, R., AND TOMPITS, H. 2008. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence 172,* 12-13, 1495–1539.

GEBSER, M., KAMINSKI, R., KÖNIG, A., AND SCHAUB, T. 2011. Advances in *gringo* series 3. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Lecture Notes in Computer Science, vol. 6645. Springer, 345–351.

GEBSER, M., KAUFMANN, B., ROMERO, J., OTERO, R., SCHAUB, T., AND WANKO, P. 2013. Domain-specific heuristics in answer set programming. In *AAAI Conference on Artificial Intelligence*, M. desJardins and M. L. Littman, Eds. AAAI Press.

GEBSER, M., KAUFMANN, B., AND SCHAUB, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence 187*, 52–89.

GEBSER, M., MARATEA, M., AND RICCA, F. 2017. The Sixth Answer Set Programming Competition. *Journal of Artificial Intelligence Research 60*, 41–95.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *International Conference and Symposium on Logic Programming*. MIT Press, 1070–1080.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9,* 3/4, 365–386.

GIUNCHIGLIA, E., LEONE, N., AND MARATEA, M. 2008. On the relation among answer set solvers. *Annals of Mathematics and Artificial Intelligence 53,* 1-4, 169–204.

HASEGAWA, R., FUJITA, H., AND KOSHIMURA, M. 2000. Efficient minimal model generation using branching lemmas. In *International Conference on Automated Deduction*. Lecture Notes in Computer Science, vol. 1831. Springer, 184–199.

JANHUNEN, T. AND NIEMELÄ, I. 2016. The answer set programming paradigm. *AI Magazine 37,* 3, 13–24.

JANOTA, M., LYNCE, I., AND MARQUES-SILVA, J. 2015. Algorithms for computing backbones of propositional formulae. *AI Communications 28,* 2, 161–177.

JANOTA, M. AND MARQUES-SILVA, J. 2016. On the query complexity of selecting minimal sets for monotone predicates. *Artificial Intelligence 233*, 73–83.

KAUFMANN, B., LEONE, N., PERRI, S., AND SCHAUB, T. 2016. Grounding and solving in answer set programming. *AI Magazine 37,* 3, 25–32.

KOSHIMURA, M., NABESHIMA, H., FUJITA, H., AND HASEGAWA, R. 2009. Minimal model generation with respect to an atom set. In *International Workshop on First-Order Theorem Proving*. CEUR Workshop Proceedings, vol. 556. CEUR-WS.org.

LEONE, N., PFEIFER, G., FABER, W., EITER, T., GOTTLOB, G., PERRI, S., AND SCARCELLO, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

LIFSCHITZ, V. 2016. Answer sets and the language of answer set programming. *AI Magazine 37,* 3, 7–12.

MANNA, M., RICCA, F., AND TERRACINA, G. 2013. Consistent query answering via ASP from different perspectives: Theory and practice. *Theory and Practice of Logic Programming 13,* 2, 227–252.

MANNA, M., RICCA, F., AND TERRACINA, G. 2015. Taming primary key violations to query large inconsistent data via ASP. *Theory and Practice of Logic Programming 15,* 4-5, 696–710.

MAREK, V. AND TRUSZCZYŃSKI, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*. Springer, 375–398.

NIEMELÄ, I. 1996. A tableau calculus for minimal model reasoning. In *International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*. Lecture Notes in Computer Science, vol. 1071. Springer, 278–294.

NIEMELÄ, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3-4, 241–273.

PREVITI, A. AND JÄRVISALO, M. 2018. A preference-based approach to backbone computation with application to argumentation. In *ACM/SIGAPP Symposium on Applied Computing*. ACM. To appear.

SILVA, J. P. M. AND SAKALLAH, K. A. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers 48,* 5, 506–521.

ZHU, C. S., WEISSENBACHER, G., SETHI, D., AND MALIK, S. 2011. SAT-based techniques for determining backbones for post-silicon fault localisation. In *IEEE International High Level Design Validation and Test Workshop*. IEEE Computer Society, 84–91.