Compute Paracoherent Answer Sets via Saturation

Francesco Ricca

Department of Mathematics and Computer Science, University of Calabria, Italy
ricca@mat.unical.it

Abstract. Answer Set Programming (ASP) is a well-established formalism for nonmonotonic reasoning. Paracoherent semantics for ASP have been suggested as a remedy, to handle cases in which no answer set exists due to the usage of cyclic default negation. In this paper we present a reduction of the task of computing a paracoherent answer set in the one of computing a stable model of a plain ASP program. The strategy is based on a direct encoding of a paracoherent semantics in plain ASP, which works on normal (i.e., non-disjunctive) ASP programs. This encoding is based on the traditional epistemic transformation and on the so-called saturation technique that is basically used to simulate a coNP check with the standard stable model check.

Keywords: Logic Programming · Answer Set Programming · Paracoherent Reasoning · Semi-Equilibrium Models.

1 Introduction

Answer Set Programming (ASP) [17, 20, 16] is a well-established formalism for nonmonotonic reasoning, with a robust solving technology [18, 21–24, 29, 9, 13–15, 32, 31]. ASP can model in a natural declarative way, usually NP-hard, combinatorial problems by encoding them as a logic program and computing its answer sets, which encode the problem solutions. The availability of efficient solvers made possible the development of a large variety of applications to Artificial Intelligence [27, 26, 8, 5, 4], Databases [30], Game Theory [12], Information Extraction [1], including industrial ones [28]. It is known that there are some circumstances, arising in presence of cyclic definitions that involve negation as failure, some logic programs have no answer sets. While this is sometimes desired for encoding problems that admit no solutions, it is sometimes perceived as detrimental, especially when dealing with query answering. Addressing this issue, paracoherent semantics based on answer sets have been proposed to draw meaningful conclusions also from incoherent programs [3, 11, 10]. The term paracoherent has been chosen to highlight both similarities and differences to paraconsistent semantics: their goal is similar, but the latter addresses classical logical contradictions, while the former addresses contradictions due to unstratified ("cyclic") negation.

Practical applications of these paracoherent semantics hinge on the availability of efficient algorithms and implementations [6, 7]. There is a vast potential of applications, the most immediate ones being debugging of ASP and incoherence-tolerant query answering. But also applications in diagnosis, planning, and reasoning about actions are conceivable [10].

In this paper we present a reduction of the task of computing a paracoherent answer set in the one of computing a stable model of a plain ASP program. The strategy is based on a direct encoding of a paracoherent semantics in plain ASP, which works on normal (i.e., non-disjunctive) ASP programs. This encoding is based on the traditional epistemic transformation and on the so-called saturation technique that is basically used to simulate a coNP check with the standard stable model check. The proposed encoding provides both an alternative proof of the complexity bounds of the task of computing paracoherent answer sets, and might be used to implement a paracoherent ASP solver just running an ASP solver.

2 Preliminaries

We start with recalling answer set semantics, and then present the paracoherent semantics of semi-stable and semi-equilibrium models. Finally, we conclude this section recalling computational complexity

2.1 Answer Set Programming

We concentrate on logic programs over a propositional signature Σ. A *disjunctive rule* r is of the form

$$a_1 \vee \cdots \vee a_l \leftarrow b_1,...,b_m, not\ c_1,...,not\ c_n, \tag{1}$$

where all a_i, b_j, and c_k are atoms (from Σ); $l,m,n \geq 0$, and $l+m+n > 0$; *not* represents *negation-as-failure*. The set $H(r) = \{a_1,...,a_l\}$ is the *head* of r, while $B^+(r) = \{b_1,...,b_m\}$ and $B^-(r) = \{c_1,\dots,c_n\}$ are the *positive body* and the *negative body* of r, respectively; the *body* of r is $B(r) = B^+(r) \cup B^-(r)$. We denote by $At(r) = H(r) \cup B(r)$ the set of all atoms occurring in r. A rule r is a *fact*, if $B(r) = \emptyset$ (we then omit $\leftarrow$); a *constraint*, if $H(r) = \emptyset$; *normal*, if $|H(r)| \leq 1$ and *positive*, if $B^-(r) = \emptyset$. A *(disjunctive logic) program* P is a finite set of disjunctive rules. P is called *normal* [resp. *positive*] if each $r \in P$ is normal [resp. positive]. We let $At(P) = \bigcup_{r\in P} At(r)$, that is the set of all atoms occurring in the program P.

Any set $I \subseteq \Sigma$ is an *interpretation*; it is a *model* of a program P (denoted $I \models P$) if and only if for each rule $r \in P$, $I \cap H(r) \neq \emptyset$ if $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$ (denoted $I \models r$). A model M of P is *minimal*, if and only if no model $M' \subset M$ of P exists. We denote by $MM(P)$ the set of all minimal models of P and by $AS(P)$ the set of all *answer sets (or stable models)* of P, i.e., the set of all interpretations I such that $I \in MM(P^I)$, where P^I is the well-known *Gelfond-Lifschitz reduct* [25] of P with respect to I, i.e., the set of rules $a_1 \vee ... \vee a_l \leftarrow b_1,\dots,b_m$, obtained from rules $r \in P$ of form (1), such that $B^-(r) \cap I = \emptyset$. Finally, we say that a program P is *consistent*, if it admits some model, otherwise it is *inconsistent*; whereas it is *coherent*, if it admits some answer set (i.e., $AS(P) \neq \emptyset$), otherwise, it is *incoherent*.

Example 1. Consider the following logic program

$$P = \{a \leftarrow not\ b;\ b \leftarrow c,\ not\ d;\ c \leftarrow a\}.$$

For instance, a model of P is $\{b,d\}$. Then, P is consistent. Moreover, the set of all minimal models of P is given by $MM(P) = \{\{b\},\{a,c,d\}\}$. However, P is incoherent. Indeed, $P^{\{b\}} = \{b \leftarrow c;\ c \leftarrow a\}$, but $\{b\}$ is not a minimal model of $P^{\{b\}}$; and $P^{\{a,c,d\}} = \{a;\ c \leftarrow a\}$, but $\{a,c,d\}$ is not a minimal model of $P^{\{a,c,d\}}$.

2.2 Paracoherent ASP

Here, we introduce two paracoherent semantics that allow for keeping a system responsive when a logic program has no answer set due to cyclic default negation. These semantics satisfy three desiderata properties identified by [10] and used also in other contexts [2].

Semi-Stable Models. Inoue and Sakama [33] introduced *semi-stable model semantics*. We consider an extended signature $\Sigma^\kappa = \Sigma \cup \{Ka \mid a \in \Sigma\}$. Intuitively, Ka can be read as a is believed to hold. Semantically, we resort to subsets of Σ^κ as interpretations I^κ and the truth values false $\bot$, believed true **bt**, and true **t**. where $\bot \preceq \mathbf{bt} \preceq \mathbf{t}$. The truth value assigned by I^κ to a propositional variable a is defined by

$$I^\kappa(a) = \begin{cases} \mathbf{t} & \text{if } a \in I^\kappa, \\ \mathbf{bt} & \text{if } Ka \in I^\kappa \text{ and } a \notin I^\kappa, \\ \bot & \text{otherwise.} \end{cases}$$

The semi-stable models of a program P are obtained from its *epistemic κ-transformation* P^κ.

Definition 1 (Epistemic κ-transformation P^κ). *Let P be a program. Then its epistemic κ-transformation is defined as the program P^κ obtained from P by replacing each rule r of the form (1) in P, such that $B^-(r) \neq \emptyset$, with:*

$$\lambda_{r,1} \vee \ldots \vee \lambda_{r,l} \vee Kc_1 \vee \ldots \vee Kc_n \leftarrow b_1, \ldots, b_m, \quad (2)$$

$$a_i \leftarrow \lambda_{r,i}, \quad (3)$$

$$\leftarrow \lambda_{r,i}, c_j, \quad (4)$$

$$\lambda_{r,i} \leftarrow a_i, \lambda_{r,k}, \quad (5)$$

for $1 \leq i,k \leq l$ and $1 \leq j \leq n$, where the $\lambda_{r,i}$, $\lambda_{r,k}$ are fresh atoms.

Note that for any program P, its epistemic κ-transformation P^κ is positive. For every interpretation I^κ over $\Sigma' \supseteq \Sigma^\kappa$, let $\mathscr{G}(I^\kappa) = \{Ka \in I^\kappa \mid a \notin I^\kappa\}$ denote the atoms believed true but not assigned true, also referred to as the gap of I^κ. Given a set $\mathscr{F}$ of interpretations over Σ', an interpretation $I^\kappa \in \mathscr{F}$ is *maximal canonical in $\mathscr{F}$*, if no $J^\kappa \in \mathscr{F}$ exists such that $\mathscr{G}(I^\kappa) \supset \mathscr{G}(J^\kappa)$. By $mc(\mathscr{F})$ we denote the set of maximal canonical interpretations in $\mathscr{F}$. Semi-stable models are then defined as *maximal canonical* interpretations among the answer sets of P^κ. Then we can equivalently paraphrase the definition of semi-stable models in [33] as follows.

Definition 2 (Semi-stable models). *Let P be a program over Σ. An interpretation I^κ over Σ^κ is a* semi-stable model *of P, if $I^\kappa = S \cap \Sigma^\kappa$ for some maximal canonical answer set S of P^κ. The set of all semi-stable models of P is denoted by $SST(P)$, i.e., $SST(P) = \{S \cap \Sigma^\kappa \mid S \in mc(AS(P^\kappa))\}$.*

Example 2. Consider the program

$$P = \{b \leftarrow not\, a;\ c \leftarrow not\, b;\ a \leftarrow c;\ d \leftarrow not\, d\}.$$

Its epistemic κ-transformation is

$$P^\kappa = \left\{ \begin{array}{l} \lambda_1 \vee Ka;\ b \leftarrow \lambda_1;\ \leftarrow a, \lambda_1;\ \lambda_1 \leftarrow b, \lambda_1; \\ \lambda_2 \vee Kb;\ c \leftarrow \lambda_2;\ \leftarrow b, \lambda_2;\ \lambda_2 \leftarrow c, \lambda_2; \\ \quad a \leftarrow c; \\ \lambda_3 \vee Kd;\ d \leftarrow \lambda_3;\ \leftarrow d, \lambda_3;\ \lambda_3 \leftarrow d, \lambda_3 \end{array} \right\},$$

which has the answer sets $M_1 = \{Ka, Kb, Kd\}$, $M_2 = \{\lambda_1, b, Kb, Kd\}$, and $M_3 = \{Ka, \lambda_2, a, c, Kd\}$. Since $\mathscr{G}(M_1) = \{Ka, Kb, Kd\}$, $\mathscr{G}(M_2) = \{Kd\}$, and $\mathscr{G}(M_3) = \{Kd\}$, among them M_2 and M_3 are maximal canonicals. Hence, $M_2 \cap \Sigma^\kappa = \{b, Kb, Kd\}$ and $M_3 \cap \Sigma^\kappa = \{a, c, Ka, Kd\}$ are semi-stable models of P.

Semi-Equilibrium Models. Semi-equilibrium models were introduced by [10] to avoid some anomalies in semi-stable model semantics concerning properties of modal logic. Like semi-stable models, semi-equilibrium models may be computed as maximal canonical answer sets, of an extension of the epistemic κ-transformation.

Definition 3 (Epistemic HT-transformation P^{HT}). *Let P be a program over Σ. Then its epistemic HT-transformation P^{HT} is defined as the union of P^κ with the set of rules:*

$$Ka \leftarrow a, \tag{6}$$
$$Ka_1 \vee \ldots \vee Ka_l \vee Kc_1 \vee \ldots \vee Kc_n \leftarrow Kb_1, \ldots, Kb_m, \tag{7}$$

for $a \in \Sigma$, respectively for every rule $r \in P$ of the form (1).

Definition 4 (Semi-equilibrium models). *Let P be a program over Σ, and let I^κ be an interpretation over Σ^κ. Then, $I^\kappa \in SEQ(P)$ if, and only if, $I^\kappa \in \{M \cap \Sigma^\kappa \mid M \in mc(AS(P^{HT}))\}$, where $SEQ(P)$ is the set of semi-equilibrium models of P.*

Example 3. Consider the program P of Example 2. Its epistemic HT-transformation is

$$P^{HT} = P^\kappa \cup \left\{ \begin{array}{l} Ka \leftarrow a;\ Kb \leftarrow b;\quad Kc \leftarrow c;\quad Kd \leftarrow d; \\ Kb \vee Ka;\ Kc \vee Kb;\ Ka \leftarrow Kc;\ Kd \leftarrow Kd \end{array} \right\},$$

which has the answer sets $\{Ka, Kb, Kd\}$, $\{\lambda_1, b, Kb, Kd\}$, and $\{Ka, \lambda_2, a, c, Kc, Kd\}$. Therefore, the semi-equilibrium models of P are $\{b, Kb, Kd\}$ and $\{a, c, Ka, Kc, Kd\}$.

In the following, we refer to semi-stable models or semi-equilibrium models as *paracoherent answer sets*. We conclude this preliminary section with some useful observations concerning the computational complexity of the computation of a paracoherent answer set.

Reasoning tasks	Normal ASP programs		Disjunctive ASP programs	
	ASP	*SST*/*SEQ*	*ASP*	*SST*/*SEQ*
Checking	P	coNP-c	coNP-c	Π_2^P-c
Brave inference	NP-c	Σ_2^P-c	Σ_2^P-c	Σ_3^P-c
Cautious inference	coNP-c	Π_2^P-c	Π_2^P-c	Π_3^P-c
Existence	NP-c	NP-c	Σ_2^P-c	NP-c

Table 1. Computational complexity of some reasoning tasks for answer set semantics, semi-stable model semantics and semi-equilibrium model semantics (completeness results).

2.3 Complexity Considerations

The complexity of various reasoning tasks with paracoherent answer sets has been analyzed in [10]. Determining the existence of paracoherent answer sets is NP-complete (it is sufficient to test for existence of classical models). Paracoherent answer set checking is Π_2^P-complete, leading to Σ_3^P-completeness for brave, and Π_3^P-completeness for cautious reasoning in case of disjunctive ASP programs; while paracoherent answer set checking is coNP-complete, leading to Σ_2^P-completeness for brave, and Π_2^P-completeness for cautious reasoning in case of normal ASP programs. These results are summarized in Table 2.3, where results for answer set semantics are also reported.

In this paper, we consider the computation of one paracoherent answer set, which is a functional problem. From previous work it is clear that this task is in FΣ_3^P for disjunctive ASP programs, and actually in FΘ_3^P (functional polynomial time with a logarithmic number of calls to a Σ_2^P-complete oracle), because for computing one paracoherent answer set it is sufficient to solve a cardinality-optimization problem. Hence, this task is in FΣ_2^P for normal ASP programs, and actually in FΘ_2^P (functional polynomial time with a logarithmic number of calls to an NP-complete oracle).

3 Saturation technique

In this section, we introduce an encoding for the computation of paracoherent answer sets based on saturation. Clearly, by computational complexity results, this encoding into a standard answer set program works only for normal ASP programs.

Let P be a normal ASP program. We consider the epistemic transformation program extended with the gap atoms $\Pi = P^{\chi} \cup P_g$. Then, we build an answer set program Π' as follows. First, starting from Π, we build a positive program, replacing each occurrence of an atom a with a fresh atom xa, and each occurrence of a negated atom $not\, a$ with a fresh atom na. More formally, given a fresh atom w, for each rule $r \in \Pi$ of the form (1), we build a rule, named $W(r)$, of the form

$$w \leftarrow na_1,\ \ldots,\ na_l,\ xb_1,\ \ldots,\ xb_m,\ nc_1,\ \ldots,\ nc_n. \tag{8}$$

Let $W(\Pi) = \bigcup\{W(r) | r \in \Pi\}$ be the set of all rules of the form (8). Then, given a rule $r \in \Pi$ such that $|H(r)| > 1$, for each atom a_i appearing in the head of r, we consider two fresh atoms, namely ca_i^r and nca_i^r, and we build the following set of rules:

$$ca_i^r \leftarrow na_1,\ \ldots,\ na_{i-1},\ na_{i+1},\ \ldots,\ na_l,\ xb_1,\ \ldots,\ xb_m,\ nc_1,\ \ldots,\ nc_n; \tag{9}$$

$$nca_i^r \leftarrow xa_j; \qquad \forall j = 1,\ldots,i-1,i+1,\ldots,m; \tag{10}$$

$$nca_i^r \leftarrow nb_j; \qquad \forall j = 1,\ldots,m; \tag{11}$$

$$nca_i^r \leftarrow xc_j; \qquad \forall j = 1,\ldots,n. \tag{12}$$

Moreover, whenever $B(r) \neq \emptyset$, we also consider the following set of rules:

$$xb_j \leftarrow ca_i^r; \qquad \forall j = 1,\ldots,m; \tag{13}$$

$$nc_j \leftarrow ca_i^r; \qquad \forall j = 1,\ldots,n. \tag{14}$$

We denote by $C(r,a_i)$ the set of rules of the form (9), (10), (11), (12), (13), and (14). Note that in case of $B(r) = \emptyset$, then $C(r,a_i)$ contains only rules of the form (9), (10), (11), and (12). We denote by $crules(a)$ the set of all rules r of Π such that a occurs in the head of r and $|H(r)| > 1$. Then, for each atom $a \in \Pi$, we construct the following rule (denoted by $C(a)$):

$$w \leftarrow xa,\ \{nca^r : r \in crules(a)\}. \tag{15}$$

Note that, whenever $crules(a) = \emptyset$, the rule of the form (15) will be $w \leftarrow xa$. Now, let $C(\Pi)$ be the set of all rules of the form $C(r,a_i)$ and $C(a)$ that can be constructed from Π. That is, $C(\Pi) = \bigcup\{C(r,a_i) : a_i \in At(\Pi) \text{ and } r \in crules(a_i)\} \cup \bigcup\{C(a) : a \in At(\Pi)\}$. Then, for each atom $a \in At(\Pi)$, we construct a disjunctive rule of the form $xa \vee na.$ (denoted by $D(xa)$), and for each atom of the form $ca_i^r \in C(\Pi)$, we construct a disjunctive rule of the form $ca_i^r \vee nca_i^r.$ (denoted by $D(ca_i^r)$). Hence, we set $D(\Pi) = \bigcup\{D(xa) | a \in At(\Pi)\} \cup \bigcup\{D(ca_i^r) | ca_i^r \in C(\Pi)\}$. Finally, we consider the following program:

$$\Pi' = W(\Pi) \cup C(\Pi) \cup D(\Pi).$$

Note that the program Π' so constructed is a sort of completion of the original program Π [19]. Therefore, the set of all answer sets of the positive program Π' extended by the constraint $\leftarrow w.$, corresponds to the set of all answer sets of the original program Π. More formally, by setting $M_x = \{a \in At(\Pi) : xa \in M\}$ and $AS_x(\Pi') = \{M_x : M \in AS(\Pi')\}$, it holds that

Theorem 1. *Let Π be a normal ASP program. Then, $AS_x(\Pi' \cup \{\leftarrow w.\}) = AS(\Pi)$.*

Now, to minimize the gap atoms, we consider the following set of rules:

$$\Pi_{check} = \left\{ \begin{array}{rll} un(gap(Ka)) \leftarrow & xgap(Ka),\ not\, gap(Ka); & \forall a \in At(P); \\ eq(gap(Ka)) \leftarrow & xgap(Ka),\ gap(Ka); & \forall a \in At(P); \\ eq(gap(Ka)) \leftarrow & ngap(Ka),\ not\, gap(Ka); & \forall a \in At(P); \\ w \leftarrow & un(gap(Ka)); & \forall a \in At(P); \\ w \leftarrow & \{eq(gap(Ka)) : a \in At(P)\} & \end{array} \right\}$$

Intuitively, the program Π_{check} minimizes the gap atoms, by comparing any two answer sets of Π. Indeed, the atom w is derived whenever the comparison shows that the set of the gap atoms of an answer set is not smaller (that is *uncomparable* or *equal*) than the set of the gap atoms of another answer set. Indeed, the first rule means that whenever a gap atom belongs to an answer set M' of the completion Π' (that mimics an answer set of the original program Π, see Theorem 1), but that atom does not belong to another answer set M of the original program Π, then the gap of M' is *uncomparable* to the gap of M. In this case, we derive w from the fourth rule. Whereas, the fifth rule means that we derive w, if we have derived each atom of the form $eq(gap(Ka))$. These atoms can be derived by the third and the fourth rules, whenever the set of gap atoms of an answer set of Π' is *equal* to the set of gap atoms of an answer set of Π.

Finally, we consider the following set of rules (so-called saturation rules) which derive each new atom created in the construction of Π' and Π_{check}, whenever w is derived.

$$\Pi_{saturation} = \left\{ \begin{array}{rl} a \leftarrow w; & \forall a \in At(\Pi'); \\ un(gap(Ka)) \leftarrow w; & \forall a \in At(P); \\ eq(gap(Ka)) \leftarrow w; & \forall a \in At(P); \\ \leftarrow not\, w & \end{array} \right\}$$

Note that, the last rule force to derive w to obtain an answer set, otherwise the program should be incoherent. Now, by considering Π_{rew} as the union of Π, Π', Π_{check}, and $\Pi_{saturation}$, there is a one-to-one correspondance between answer sets of Π_{rew} and paracoherent answer sets of P. More formally,

Theorem 2. *Let P be a normal program. Then, $M \in AS(\Pi_{rew})$ implies $M \cap At(P) \in PAS(P)$, and $M' \in PAS(P)$ implies $M' \cup (At(\Pi_{rew}) \setminus At(P)) \in AS(\Pi_{rew})$.*

Intuitively, Π' is a "copy" of Π, and the program Π_{check} compares any two answer sets of Π (by comparing answer sets of Π and Π'). Now, if M is an answer set of Π_{rew}, then $w \in M$, and so each new atom occurring in Π' and Π_{check} belongs to M. Moreover, there is an answer set J of Π contained into M. As M is an answer set, it is a minimal model of the reduct of Π_{rew}, that is $\Pi^J \cup \Pi' \cup \Pi^J_{check} \cup \Pi_{saturation} \setminus \{\leftarrow not\, w.\}$. This means that the set of gap atoms of each other answer set of Π and the set of gap atoms of J are uncomparable or equal. Otherwise, M should not be a minimal model of the reduct. Hence, the gap minimality property is satisfied by M, thus $M \cap At(P)$ is a paracoherent answer set of P. The same intuition also explains why $M' \cup (At(\Pi_{rew}) \setminus At(P))$ is an answer set of Π_{rew}, whenever M' is a paracoherent answer set of P.

4 Conclusion

Paracoherent semantics have been suggested as a remedy, which extend the classical notion of answer sets to draw meaningful conclusions also from incoherent programs. In this paper we presented an encoding of paracoherent answer set in plain ASP. The proposed encoding confirms (being usable in an alternative proof) the complexity bounds of the task of computing paracoherent answer sets, and might be used to implement a paracoherent ASP solver just running an ASP solver. As far as future work is concerned, we plan to test in an experimental analysis whether the encoding can be fruitfully employed for implementing an efficient paracoherent ASP solver.

References

1. Adrian, W.T., Manna, M., Leone, N., Amendola, G., Adrian, M.: Entity set expansion from the web via ASP. In: ICLP (Technical Communications). OASICS, vol. 58, pp. 1:1–1:5. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
2. Alviano, M., Amendola, G., Peñaloza, R.: Minimal undefinedness for fuzzy answer sets. In: AAAI 2017. pp. 3694–3700 (2017)
3. Amendola, G.: Dealing with incoherence in ASP: split semi-equilibrium semantics. In: DWAI@AI*IA. CEUR Workshop Proceedings, vol. 1334, pp. 23–32. CEUR-WS.org (2014)
4. Amendola, G.: Preliminary results on modeling interdependent scheduling games via answer set programming. In: RCRA@AI*IA. p. to appear. CEUR Workshop Proceedings, CEUR-WS.org (2018)
5. Amendola, G.: Solving the stable roommates problem using incoherent answer set programs. In: RCRA@AI*IA. p. to appear. CEUR Workshop Proceedings, CEUR-WS.org (2018)
6. Amendola, G., Dodaro, C., Faber, W., Leone, N., Ricca, F.: On the computation of paracoherent answer sets. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA. pp. 1034–1040 (2017)
7. Amendola, G., Dodaro, C., Faber, W., Ricca, F.: Externally supported models for efficient computation of paracoherent answer sets. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, February 2-7, 2018, New Orleans, Louisiana, USA. pp. 1034–1040 (2018)
8. Amendola, G., Dodaro, C., Leone, N., Ricca, F.: On the application of answer set programming to the conference paper assignment problem. In: AI*IA. Lecture Notes in Computer Science, vol. 10037, pp. 164–178. Springer (2016)
9. Amendola, G., Dodaro, C., Ricca, F.: ASPQ: an asp-based 2qbf solver. In: QBF@SAT. CEUR Workshop Proceedings, vol. 1719, pp. 49–54. CEUR-WS.org (2016)
10. Amendola, G., Eiter, T., Fink, M., Leone, N., Moura, J.: Semi-equilibrium models for paracoherent answer set programs. Artif. Intell. **234**, 219–271 (2016)
11. Amendola, G., Eiter, T., Leone, N.: Modular paracoherent answer sets. In: Logics in Artificial Intelligence - 14th European Conference, JELIA2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings. pp. 457–471 (2014)
12. Amendola, G., Greco, G., Leone, N., Veltri, P.: Modeling and reasoning about NTU games via answer set programming. In: IJCAI 2016. pp. 38–45 (2016)
13. Amendola, G., Ricca, F., Truszczynski, M.: Generating hard random boolean formulas and disjunctive logic programs. In: IJCAI. pp. 532–538. ijcai.org (2017)
14. Amendola, G., Ricca, F., Truszczynski, M.: A generator of hard 2qbf formulas and asp programs. In: KR. AAAI Press (2018)
15. Amendola, G., Ricca, F., Truszczynski, M.: Random models of very hard 2qbf and disjunctive programs: An overview. In: ICTCS. CEUR Workshop Proceedings, CEUR-WS.org (2018)
16. Bonatti, P.A., Calimeri, F., Leone, N., Ricca, F.: Answer set programming. In: 25 Years GULP. Lecture Notes in Computer Science, vol. 6125, pp. 159–182. Springer (2010)
17. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Com. ACM **54**(12), 92–103 (2011)
18. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the Fifth Answer Set Programming Competition. Artif. Intell. **231**, 151–181 (2016)
19. Clark, K.L.: Negation as failure. In: Logic and Data Bases. pp. 293–322 (1977)
20. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Morgan & Claypool Publishers (2012)

21. Gebser, M., Leone, N., Maratea, M., Perri, S., Ricca, F., Schaub, T.: Evaluation techniques and systems for answer set programming: a survey. In: IJCAI 2018. pp. 5450–5456 (2018)
22. Gebser, M., Maratea, M., Ricca, F.: The Design of the Sixth Answer Set Programming Competition. In: LPNMR. LNCS, vol. 9345, pp. 531–544. Springer (2015)
23. Gebser, M., Maratea, M., Ricca, F.: What's hot in the answer set programming competition. In: AAAI. pp. 4327–4329. AAAI Press (2016)
24. Gebser, M., Maratea, M., Ricca, F.: The sixth answer set programming competition. Journal of Artificial Intelligence Research **60**, 41–95 (2017)
25. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. **9**(3/4), 365–386 (1991). https://doi.org/10.1007/BF03037169, `http://dx.doi.org/10.1007/BF03037169`
26. Grasso, G., Iiritano, S., Leone, N., Lio, V., Ricca, F., Scalise, F.: An asp-based system for team-building in the gioia-tauro seaport. In: PADL. Lecture Notes in Computer Science, vol. 5937, pp. 40–42. Springer (2010)
27. Grasso, G., Iiritano, S., Leone, N., Ricca, F.: Some DLV applications for knowledge management. In: LPNMR. Lecture Notes in Computer Science, vol. 5753, pp. 591–597. Springer (2009)
28. Grasso, G., Leone, N., Manna, M., Ricca, F.: ASP at work: Spin-off and applications of the DLV system. In: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning. pp. 432–451. LNCS 6565 (2011)
29. Lierler, Y., Maratea, M., Ricca, F.: Systems, engineering environments, and competitions. AI Magazine **37**(3), 45–52 (2016)
30. Manna, M., Ricca, F., Terracina, G.: Taming primary key violations to query large inconsistent data via ASP. TPLP **15**(4-5), 696–710 (2015)
31. Maratea, M., Pulina, L., Ricca, F.: A multi-engine approach to answer-set programming. TPLP **14**(6), 841–868 (2014)
32. Maratea, M., Ricca, F., Faber, W., Leone, N.: Look-back techniques and heuristics in DLV: implementation, evaluation, and comparison to QBF solvers. J. Algorithms **63**(1-3), 70–89 (2008)
33. Sakama, C., Inoue, K.: Paraconsistent stable semantics for extended disjunctive programs. J. Log. Comput. **5**(3), 265–285 (1995)