Modular Paracoherent Answer Sets[⋆]

Giovanni Amendola[1], Thomas Eiter[2], and Nicola Leone[1]

[1] Department of Mathematics and Computer Science, University of Calabria
Via P. Bucci, Cubo 30b, 87036 Rende (CS), Italy amendola@mat.unical.it,
leone@mat.unical.it
[2] Institute of Information Systems, Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
eiter@kr.tuwien.ac.at

Abstract. The answer set semantics may assign a logic program no model due to classic contradiction or cyclic negation. The latter can be remedied by resorting to a paracoherent semantics given by semi-equilibrium (SEQ) models, which are 3-valued interpretations that generalize the logical reconstruction of answer sets given by equilibrium models. While SEQ-models have interesting properties, they miss modularity in the rules, such that a natural modular (bottom up) evaluation of programs is hindered. We thus refine SEQ-models using splitting sets, the major tool for modularity in modeling and evaluating answer set programs. We consider canonical models that are independent of any particular splitting sequence from a class of splitting sequences, and present two such classes whose members are efficiently recognizable. Splitting SEQ-models does not make reasoning harder, except for deciding model existence in presence of constraints (without constraints, split SEQ-models always exist).

1 Introduction

As well-known, the answer set semantics [11] does not assign to every logic program a model. This can be either due to a logical contradiction, as emerging e.g. in the program $\{open \leftarrow not\ closed, \neg open \leftarrow\ \}$, or due to cyclic negation, as e.g. present in the program $\{shaves(joe, joe) \leftarrow not\ shaves(joe, joe)\}$, which is a paraphrase of Russell's paradox (where joe is the barber).

In order to avoid trivialization of reasoning from such programs, Inoue and Sakama [28] have introduced paraconsistent semantics for answer set programs. While dealing with explicit contradictions can be achieved with similar methods as for (non-)classical logic (cf. also [5, 1, 17]), dealing with cyclic negation turned out to be tricky. With the idea that atoms may also be possibly true (i.e., by belief), Inoue and Sakama defined a semi-stable semantics which for Russell's paradox above yields the model that $shaves(joe, joe)$ is possibly true, which seems reasonable. In fact, semi-stable semantics *approximates* answer set semantics and coincides with it whenever a program has some answer set; otherwise, it yields under Occam's razor models with a least set of

⋆ This work was partially supported by Regione Calabria under the EU Social Fund and project PIA KnowRex POR FESR 2007- 2013, and by the Italian Ministry of University and Research under PON project "Ba2Know (Business Analytics to Know) S.I.-LAB n. PON03PE_0001.

atoms believed to be true. That is, the intrinsic *closed world assumption (CWA)* of logic programs is slightly relaxed for achieving stability of models.

In a similar vein, we can regard many semantics for non-monotonic logic programs that relax answer sets as *paracoherent semantics*, e.g. [3, 9, 18, 22, 23, 25, 27, 29, 31, 32].[3] Ideally, such a relaxation meets for a program P the following desiderata [8]:

(D1) Every (consistent) answer set of P corresponds to a model (*answer set coverage*).
(D2) If P has some (consistent) answer set, then its models correspond to answer sets (*congruence*).
(D3) If P has a classical model, then P has a model (*classical coherence*).

In particular, (D3) intuitively says that in the extremal case, a relaxation should renounce to the selection principles imposed by the semantics on classical models (in particular, if a single classical model exists).

However, only few paracoherent semantics satisfy all three desiderata (cf. [8]). A recent one are semi-equilibrium (SEQ) models [8], which improve semi-stable models by avoiding some anomalies. SEQ-models are a relaxation of Pearce's well-known equilibrium models [19], which provide a logical reconstruction of answer sets alias stable models in terms of a non-monotonic version of Heyting's [12] logic of here and there. Roughly speaking, SEQ-models are 3-valued interpretations in which atoms can be true, false or believed to be true; the gap between believed and derivably true atoms is globally minimized by SEQ-models. Note that the distinction between believed and derivably true atoms in models is important; other approaches, e.g. CR-Prolog [3], make a distinction at the rule level.

While the SEQ-semantics has nice properties, it may select models that do not respect modular structure in the rules. To illustrate this, consider the following example.

Example 1 *Suppose we have a program that captures knowledge about friends of a person regarding visits to a party, where $go(X)$ informally means that X will go:*

$$P = \left\{ \begin{array}{c} go(John) \leftarrow not\ go(Mark); \\ go(Peter) \leftarrow go(John), not\ go(Bill); \\ go(Bill) \leftarrow go(Peter) \end{array} \right\}$$

Then P has no answer set; its semi-equilibrium models are $M_1 = (\emptyset, \{go(Mark)\})$, and $M_2 = (\{go(John)\}, \{go(John), go(Bill)\})$. Informally, a key difference between M_1 and M_2 concerns the beliefs on Mark and John. In M_2 Mark does not go, and, consequently, John will go (moreover, Bill is believed to go, and Peter will not go). In M_1, instead, we believe Mark will go, thus John will not go (likewise Peter and Bill).

None of the two models provides a fully coherent view (on the other hand, the program is incoherent, having no answer set). Nevertheless, M_2 appears preferable over M_1, since, according with a layering (stratification) principle, which is widely agreed in LP, one should prefer $go(John)$ rather than $go(Mark)$, as there is no way to derive $go(Mark)$ (which does not appear in the head of any rule of the program).

Modularity via rule dependency as in the example above is widely used in problem modeling and logic programs evaluation; in fact, program decomposition is crucial for

[3] Notably, Seipel's Evidential Stable Models for disjunctive LPs [29] coincide with SEQ-models.

efficient answer set computation. For the program P above, advanced answer set solvers like DLV and clasp immediately set $go(Mark)$ to false, as $go(Mark)$ does not occur in any rule head. In a customary bottom up computation along program components, answer sets are gradually extended until the whole program is covered, or incoherence is detected at some component (in our example for the last two rules). But rather than to abort the computation, we would like to switch to a paracoherent mode and continue with building semi-equilibrium models, as an approximation of answer sets.

In this general setting, we refine SEQ-models with the following contributions.

– Resorting to splitting sets [15], the major tool for modularity in modeling and evaluating answer set programs, we define *split SEQ-models* (Section 3), for which the program is evaluated in progressive layers according to a *splitting sequence* of the atoms. In the example above, the natural sequence $S = (\{go(Mark)\}, \{go(Mark), go(John)\}, \{go(Mark), go(John), go(Bill), go(Peter)\})$ will yield the expected result.

– In general, the resulting split SEQ-models depend on the particular splitting sequence S. We thus introduce *canonical splitting sequences*, with the property that the models are *independent* of any particular from a class of splitting sequences, and thus yield canonical models (Section 4). This is analogous to the *perfect models* of a (disjunctive) stratified program, which are independent of a concrete stratification [2, 26]. For constraint-free programs P, the class derived from the strongly connected components (SCCs) of P warrants this property, as well as modularity property. For arbitrary programs, independence is held by a similar class derived from the maximal joined components (MJCs), merging SCCs involved in constraints.

– We characterize the computational complexity of split SEQ-model semantics, for canonical models and various classes of logic programs (Section 5). It appears that the refined semantics has the same complexity as SEQ-semantics, except for the model existence problem, which gets harder in general. This provides useful insight for defining canonical models that satisfy all desiderata (D1)-(D3) for arbitrary programs, which we briefly discuss here (Section 7).

The refined semantics, and in particular the SCC-models semantics, lends for a modular use and bottom up evaluation of programs. Cautious merging of components, as done for MJC-models, aims at preserving independence of components and thus possible parallel evaluation. This makes the refined semantics attractive for incorporation into answer set evaluation frameworks, in order to add paracoherent features.

2 Preliminaries

We start with recalling answer set semantics and fixing notation, and then present the paracoherent semantics of semi-equilibrium models.

Answer Set Programs. Following the traditional grounding view [11], we concentrate on programs over a propositional signature Λ. A *disjunctive rule* r is of the form

$$a_1 \vee \cdots \vee a_l \leftarrow b_1, ..., b_m, not\ b_{m+1}, ..., not\ b_n, \qquad (1)$$

where all a_i and b_j are atoms (from Λ) and $l \geq 0$, $n \geq m \geq 0$ and $l + n > 0$; 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 $B^-(r) = \{b_{m+1}, \dots, b_n\}$ 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. For any set of atoms S, we let $not\ S = \{not\ a \mid a \in S\}$; rules of form (1) will also be written (in abuse of notation) $H(r) \leftarrow B^+(r), not\ B^-(r)$. 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 $At(P) = \bigcup_{r \in P} At(r)$.

Any set $I \subseteq \mathit{\Lambda}$ is an *interpretation*; it is a *model* of a program P (denoted $I \models P$) iff 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*, iff no model $M' \subset M$ of P exists. We denote the by $MM(P)$ 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* [11] of P w.r.t. I.

Semi-equilibrium paracoherent semantics. We call logic programs that lack answer sets due to cyclic dependency of atoms among each other by rules through negation *incoherent* (cf. Russel's paradox). The semi-equilibrium semantics [8] avoids incoherence by resorting to the view of answer sets in the *logic of here and there* (*HT-logic*) [19, 20]. We focus here on formulas ϕ of the form

$$b_1 \wedge ... \wedge b_m \wedge \neg b_{m+1} \wedge ... \wedge \neg b_n \rightarrow a_1 \vee ... \vee a_l, \qquad (2)$$

which correspond in an obvious way to rules of form (1). In *HT-logic*, *interpretations* are pairs (X, Y), $X \subseteq Y \subseteq \mathit{\Lambda}$, where X is the *here* world and Y the *there* world. Intuitively, the atoms in X are true (value $\mathbf{t}$), atoms not in Y are false ($\mathbf{f}$), and the atoms in $gap(X, Y) = Y \setminus X$ are believed to be true ($\mathbf{bt}$). For any set A of HT-interpretations, we denote by $mc(A)$ the set of maximal canonical interpretations $(X, Y) \in A$, i.e., no $(X', Y') \in A$ exists such that $gap(X', Y') \subset gap(X, Y)$. We define (X, Y) to be an *HT-model* of the formula ϕ, denoted $(X, Y) \models \phi$, in a recursive way:

1. $(X, Y) \models a$ iff $a \in X$;
2. $(X, Y) \not\models \bot$; ($\bot$ is falsity)
3. $(X, Y) \models \phi \wedge \psi$ iff $(X, Y) \models \phi$ and $(X, Y) \models \psi$;
4. $(X, Y) \models \phi \vee \psi$ iff $(X, Y) \models \phi$ or $(X, Y) \models \psi$;
5. $(X, Y) \models \phi \rightarrow \psi$ iff (*i*) $(X, Y) \not\models \phi$ or $(X, Y) \models \psi$, and (*ii*) $Y \models \phi \rightarrow \psi$;[4]
6. $(X, Y) \models \neg\phi$ iff $(X, Y) \models \phi \rightarrow \bot$.

In particular, $(X, Y) \models \neg a$ iff $a \notin Y$, and $(X, Y) \models r$ for a rule r of form (2) iff either $\{a_1 \dots, a_k\} \cap X \neq \emptyset$, $\{b_1, \dots, b_m\} \not\subseteq Y$, or $\{b_{m+1}, \dots, b_n\} \cap Y \neq \emptyset$. A HT-interpretation (X, Y) is an HT-model of a theory (i.e., a set of formulas) $\mathit{\Theta}$, denoted $(X, Y) \models \mathit{\Theta}$ iff $(X, Y) \models \phi$ for each $\phi \in \mathit{\Theta}$. It is an *equilibrium (EQ) model* of $\mathit{\Theta}$ iff $X = Y$ and for every $X' \subset Y$ it holds that $(X', Y) \not\models \mathit{\Theta}$.

Example 2 *Consider the program $P = \{a \leftarrow b;\ b \leftarrow not\ c;\ c \leftarrow not\ a\}$, and the corresponding theory $\mathit{\Theta}_P = \{b \rightarrow a;\ \neg c \rightarrow b;\ \neg a \rightarrow c\}$. As easily checked, $(\emptyset, ac)$, (a, ab), (a, abc), and (c, c) are HT-models of $\mathit{\Theta}_P$; the only equilibrium model is (c, c).*

[4] Note that in condition 5.(*ii*) '$\models$' is the standard operator of classical propositional logic.

As shown by Pearce [19], $M \subseteq At(P)$ fulfills $M \in AS(P)$ iff (M, M) is an EQ-model of Θ_P. Paracoherent answer sets emerge with minimal sets of believed atoms.

Definition 1 ([8]) *A semi-equilibrium (SEQ) model (or* paracoherent answer set*) of a program P is any HT-model (X, Y) of P s.t. (i) $(X', Y) \not\models P$, for all $X' \subset X$ (h-minimality) and (ii) no HT-model (X', Y') of P satisfies h-minimality and $gap(X', Y') \subset gap(X, Y)$ (gap-minimality).*

The set of all semi-equilibrium models of P is denoted by $SEQ(P)$.

Example 3 *Consider the program $P = \{a \leftarrow b;\ b \leftarrow not\ a\}$. Its HT-models are $(\emptyset, a)$, $(\emptyset, ab)$, (a, a), (a, ab), (b, ab) and (ab, ab). Hence, there is no equilibrium model for P, while $SEQ(P) = \{(\emptyset, a)\}$.*

3 Split Semi-Equilibrium Semantics

In this section, we introduce a refinement to the semi-equilibrium semantics. In fact we observe that sometimes gap minimization is too weak. Consider the following example.

Example 4 *Let $P = \{c \leftarrow b, not\ c;\ b \leftarrow not\ a\}$; then $SEQ(P) = \{\,(b, bc), (\emptyset, a)\}$. Here (b, bc) is more appealing than $(\emptyset, a)$ because a is not derivable, as no rule has a in the head. Moreover, intuitively, $P_1 = \{b \leftarrow not\ a\}$ is a lower (coherent) part feeding into the upper part $P_2 = \{c \leftarrow b, not\ c\}$.*

To overcome this limitation, we introduce a refined paracoherent semantics, called *split semi-equilibrium semantics*. It coincides with the answer sets semantics in case of coherent programs, and selects a subset of the SEQ-models otherwise. The main results of this section are two model-theoretic characterizations which identify necessary and sufficient conditions for deciding whether a SEQ-model is selected.

Splitting sets and sequences. Splitting sets [15] allow us to divide a program P into a lower and a higher part which can be evaluated bottom up. More formally, a set $S \subseteq At(P)$ is a *splitting set* of P, if for every rule r in P such that $H(r) \cap S \neq \emptyset$ we have that $At(r) \subseteq S$. We denote by $b_S(P) = \{r \in\mid At(r) \subseteq S\}$ the *bottom* part of P, $t_S(P) = P \setminus b_S(P)$ the *top* part of P relative to S.

Splitting sets naturally lead to splitting sequences. A *splitting sequence* $S = (S_1, \ldots, S_n)$ of P is a sequence of splitting sets of P such that $S_i \subseteq S_j$ for each $i < j$.

Split semi-equilibrium models. We now introduce the notion of *SEQ-models related to a splitting set*. First given a splitting set S for a program P and an HT-interpretation (I, J) for $b_S(P)$, we let

$$P^S(I, J) = P \setminus b_S(P) \cup \{a \mid a \in I\} \cup \{\leftarrow not\ a \mid a \in J\} \cup \{\leftarrow a \mid a \in S \setminus J\}.$$

Informally, the bottom part of P w.r.t. S is replaced with rules and constraints which fix in any EQ-model of the remainder (= $t_S(P)$) the values of the atoms in S to (I, J).

Definition 2 (Semi-equilibrium models related to a splitting set) *Let S be a splitting set of a program P. Then the semi-equilibrium models of P related to S are defined as*

$$SEQ^S(P) = mc\Big(\bigcup_{(I,J)\in SEQ(b_S(P))} SEQ(P^S(I, J))\Big). \qquad (3)$$

Example 5 (cont'd) *For the splitting set $S = \{a, b\}$ of P in Example 4, $b_S(P) = \{b \leftarrow not\ a\}$ and $SEQ(b_S(P)) = \{(b, b)\}$. Hence, $P^S(b, b) = \{c \leftarrow b, not\ c;\ b;\ \leftarrow a\}$ and $SEQ^S(P) = SEQ(P^S(b, b)) = \{(b, bc)\}$.*

For any HT-model (X, Y) and splitting set S of a program P, we define the *restriction of* (X, Y) *to* S as $(X, Y)|_S = (X \cap S, Y \cap S)$.

Proposition 1 *Let S be a splitting set of a program P. If $(X, Y) \in SEQ^S(P)$, then $(X, Y)|_S \in SEQ(b_S(P))$.*

The following result shows that each semi-equilibrium model related to a given splitting set is always a semi-equilibrium model of the program.

Theorem 1 (Soundness) *Let S be a splitting set of a program P. If $(X, Y) \in SEQ^S(P)$, then $(X, Y) \in SEQ(P)$.*

The converse does not hold in general; in fact if we consider the program of Example 4 and the splitting set $S = \{a, b\}$ we have $SEQ^S(P) = \{(b, bc)\}$, while $SEQ(P) = \{(b, bc), (\emptyset, a)\}$. It is also clear that $SEQ^S(P)$ depends on the choice of S; in fact if $S = \emptyset$ then $SEQ^\emptyset(P) = SEQ(P)$.

Moreover for the validity of Theorem 1, the selection of maximal canonical HT-models is necessary. Indeed, for $P = \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ c \leftarrow b, not\ c\}$ and the splitting set $S = \{a, b\}$, we have $SEQ(b_S(P)) = \{(a, a), (b, b)\}$; hence $SEQ(P^S(a, a)) \cup SEQ(P^S(b, b)) = \{(a, a), (b, bc)\}$, while $SEQ(P) = \{(a, a)\}$.

We have seen so far two necessary conditions for an HT-model to qualify as a semi-equilibrium model related to a given splitting set. These conditions are also sufficient.

Theorem 2 (Completeness) *Let S be a splitting set of a program P. If $(X, Y) \in SEQ(P)$ and $(X, Y)|_S \in SEQ(b_S(P))$, then $(X, Y) \in SEQ^S(P)$.*

Putting together the various results obtained so far we have proved the following semantic characterization for semi-equilibrium models related to a splitting set:

Theorem 3 *Let S be a splitting set of a program P. Then $(X, Y) \in SEQ^S(P)$ iff $(X, Y) \in SEQ(P)$ and $(X, Y)|_S \in SEQ(b_S(P))$.*

Now we generalize the use of splitting sets to compute the SEQ-models of a program via splitting sequences.

Definition 3 (Semi-equilibrium models related to a splitting sequence) *Given a splitting sequence $S = (S_1, \ldots, S_n)$ for a program P, we let $S' = (S_2, ..., S_n)$ and define the semi-equilibrium models of P related to the splitting sequence $S = (S_1, ..., S_n)$ as*

$$SEQ^S(P) = mc\Big(\bigcup_{(I,J) \in SEQ(b_{S_1}(P))} SEQ^{S'}(P^{S_1}(I, J)) \Big). \tag{4}$$

The *SEQ-models related to a splitting sequence* can be characterized similarly as those related to a splitting set. To ease presentation, for a program P and splitting sequence $S = (S_1, ..., S_n)$, we let $P_0 = P$ and $P_k = (P_{k-1})^{S_k}(I_k, J_k)$, where $(I_k, J_k) \in SEQ(b_{S_k}(P_{k-1}))$, $k = 1, ..., n$. We now state the main result of this section.

Theorem 4 *Let $S = (S_1, ..., S_n)$ be a splitting sequence of a program P. Then $(X, Y) \in SEQ^S(P)$ iff $(X, Y) \in SEQ(P)$ and $(X, Y)|_{S_k} \in SEQ(b_{S_k}(P_{k-1}))$, for $k = 1, ..., n$.*

Finally we observe that a classically consistent program does not necessarily have *split semi-equilibrium models* (but always *semi-equilibrium models*). In fact, if we consider $P = \{\leftarrow b;\ b \leftarrow not\ a\}$ and the splitting set $S = \{a\}$, we obtain $SEQ(b_S(P)) = \{(\emptyset, \emptyset)\}$ and so $SEQ^S(P) = \emptyset$. However (a, a) and $(\emptyset, a)$ are HT-models of P.

4 Canonical Semi-Equilibrium Models

The split semi-equilibrium semantics depends on the choice of the particular splitting sequence, which is not much desirable. We thus consider a way to obtain a refined split SEQ-semantics that is independent of a particular splitting sequence, but imposes conditions on sequences that come naturally with the program and can be easily tested.

Attractive for this purpose are the *strongly connected components* (SCCs) of a given program, which are at the heart of bottom up evaluation algorithms in ASP systems. In absence of constraints, we get the desired independence of a particular splitting sequence, such that we can then talk about the *SCC-models of a program*. Allowing for constraints will need a slight extension.

4.1 SCC-split sequences and models

Recall that the *dependency graph* of a program P is the directed graph $DG(P) = \langle V_{DG}, E_{DG}\rangle$, where $V_{DG} = At(P)$ and $E_{DG} = \{(a, b) \mid a \in H(r), b \in B(r) \cup (H(r) \setminus \{a\}), r \in P\}$. The SCCs of P, denoted $SCC(P)$, are the SCCs of $DG(P)$, and the supergraph of P is the graph $SG(P) = \langle V_{SG}, E_{SG}\rangle$, where $V_{SG} = SCC(P)$ and $E_{SG} = \{(C, C') \mid C \neq C' \in SCC(P), \exists a \in C, \exists b \in C', (a, b) \in E_{DG}\}$. Note that $SG(P)$ is a directed acyclic graph (dag); recall that a *topological ordering* of a dag $G = \langle V, E\rangle$ is an ordering $v_1, v_2, ..., v_n$ of its vertices, denoted $\leq$, such that for every $(v_i, v_j) \in E$ we have $i > j$. Such an ordering always exists, and the set $O(G)$ of all topological orderings of G is nonempty. Any such ordering of $SG(P)$ naturally induces a splitting sequence as follows.

Definition 4 *Let P be a program and let $\leq = (C_1, ..., C_n)$ be a topological ordering of $SG(P)$. Then the splitting sequence induced by $\leq$ is $S_\leq = (S_1, ..., S_n)$, where $S_1 = C_1$ and $S_j = S_{j-1} \cup C_j$, for $j = 2, ..., n$.*

We call any such $S_\leq$ a *SCC-splitting sequence*; note that $S_\leq$ is indeed a splitting sequence of P. We now have the following result.

Theorem 5 *Let P be a constraint-free program. For every $\leq, \leq' \in O(SG(P))$, we have $SEQ^{S_\leq}(P) = SEQ^{S_{\leq'}}(P)$.*

This result allows to define the *SCC-models of P* as $M^{SCC}(P) = SEQ^{S_\leq}(P)$ for an arbitrary topological ordering of $SG(P)$. We then obtain:

Proposition 2 *The SCC-models semantics, given by $M^{SCC}(P)$ for constraint-free P, satisfies (D1)-(D3).*

Example 6 *Consider $P = \{a \leftarrow c, not\ a;\ a \leftarrow not\ b;\ c \leftarrow not\ d;\ b \leftarrow not\ e\}$; its SCCs are $C_1 = \{a\}$, $C_2 = \{b\}$, $C_3 = \{c\}$, $C_4 = \{d\}$ and $C_5 = \{e\}$. For $\leq = (C_4, C_5, C_3, C_2, C_1)$, we obtain that $SEQ^{S_\leq}(P) = SEQ^{(S_2,S_3,S_4,S_5)}(P^{S_1}(\emptyset, \emptyset)) = SEQ^{(S_3,S_4,S_5)}(P_1^{S_2}(\emptyset, \emptyset)) = SEQ^{(S_4,S_5)}(P_2^{S_3}(c, c)) = SEQ^{(S_5)}(P_3^{S_4}(bc, bc)) = \{(bc, abc)\}$; hence $M^{SCC}(P) = \{(bc, abc)\}$. For $\leq' = (C_5, C_2, C_4, C_3, C_1)$, we obtain $SEQ^{S_{\leq'}}(P) = \{(bc, abc)\}$, in line with Theorem 5. Note that $SEQ(P) = \{(bc, abc), (b, bd), (ac, ace)\}$.*

Finally, if we replace in Equation (4) SEQ, SEQ^S, and $SEQ^{S'}$ all by M^{SCC}, then the resulting equation holds; i.e., we can compute SCC-models modularly bottom up along an arbitrary splitting sequence (using always M^{SCC}).

4.2 MJC-split sequences and models

Theorem 5 fails if we allow constraints in P. E.g., the program $P = \{b;\ \leftarrow\ b, not\ a\}$ has the SCCs $\{a\}$ and $\{b\}$; hence $O(SG(P)) = \{(\{a\}, \{b\}), (\{b\}, \{a\})\}$. But the respective semi-equilibrium models are different: $SEQ^{(\{a\},\{a,b\})}(P) = \emptyset$ and $SEQ^{(\{b\},\{a,b\})}(P) = \{(b, ba)\}$. Note here that semi-equilibrium semantics is able to distinguish constraints $\leftarrow Body$ from rules $f \leftarrow Body, not\ f$; the latter can always be satisfied by believing f (and thus be viewed as soft constraints). On the other hand, Theorem 5 extends to the case without cross-component constraints, i.e., each constraint r is embedded in some SCC C_i ($B(r) \subseteq C_i \in SCC(P)$); otherwise, the order in which unrelated components appear in a splitting sequence may matter.

We thus consider merging SCCs of a program in such a way that independence of concrete topological orderings is preserved and merging is done only if deemed necessary. This is embodied by the *maximal joinable components* of a program, which lead to so called MJC-split sequences and models. Informally, relevant SCCs that are unordered (thus unproblematic in evaluation) are merged if they intersect with a constraint.

We start with introducing *related pairs* and *joinable pairs* of SCCs. We call (K_1, K_2) from $SCC(P)^2$ a *related pair*, if either $K_1 = K_2$ or some constraint $r \in P$ fulfills $At(r) \cap K_1 \neq \emptyset$ and $At(r) \cap K_2 \neq \emptyset$; by $C_{(K_1,K_2)}(P)$ we denote the set of all such constraints.

Definition 5 *A related pair (K_1, K_2) is a joinable pair iff $K_1 = K_2$ or some $(C_1, \dots, C_n)$ in $O(SG(P))$ exists such that (i) $K_1 = C_s$ and $K_2 = C_{s+1}$ for some $1 \leq s < n$, (ii) $(K_2, K_1) \notin E_{SG}$ and (iii) there exists $r \in C_{(K_1,K_2)}(P)$ s.t. $At(r) \subseteq C_1 \cup \ldots \cup C_{s+1}$.*

We denote by $JP(P)$ the set of all *joinable pairs*. Intuitively item (i) states that in some topological ordering K_1 immediately precedes K_2; item (ii) states that no atom in K_2 directly depends on an atom from K_1. If this does not hold, joining K_1 and K_2 to achieve independence is not necessary as their ordering is fixed. Finally item (iii) requires that some constraint must access the two SCCs and appear in the evaluation in the bottom of the program computed so far.

Example 7 *For $P = \{\leftarrow b, not\ a;\ \leftarrow b, not\ c;\ d \leftarrow not\ a;\ c \leftarrow not\ e;\ b \leftarrow c\}$, we have $SCC(P) = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}\}$. We observe that $(\{c\}, \{b\})$ is a related, but not a joinable pair, because $(\{c\}, \{b\})$ satisfies conditions (i) and (iii), but not (ii). On the other hand, $(\{a\}, \{b\})$ is a joinable pair.*

We now extend joinability from pairs to any number of SCCs.

Definition 6 *Let P be a program. Then $K_1, ..., K_m \in SCC(P)$ are joinable iff $m = 2$ and some $K \in SCC(P)$ exists such that $(K_1, K), (K, K_2) \in JP(P)$, or otherwise K_i, K_j are joinable for each $i, j = 1, ..., m$. We let $JC(P) = \{\bigcup_{i=1}^{m} K_i \mid K_1, ..., K_m \in SCC(P)$ are joinable$\}$ and call $MJC(P) = \{J \in JC(P) \mid \forall J' \in JC(P) : J \not\subset J'\}$ the set of all* maximal joined components *(MJCs) of P.*

Note that $(K_1, K_2) \in JP(P)$ implies that K_1 and K_2 are joinable.

Example 8 *In Ex. 7, $(\{a\}, \{b\})$ is the only nontrivial joinable pair; hence $MJC(P) = \{\{a, b\}, \{c\}, \{d\}, \{e\}\}$.*

As easily seen, $MJC(P)$ is a partitioning of $At(P)$ that results from merging SCCs. We define the *MJC graph* of P as $JG(P) = \langle V_{JG}, E_{JG} \rangle$, where $V_{JG} = MJC(P)$ and $E_{JG} = \{(J, J') \mid J \neq J' \in MJC(P), \exists a \in J, \exists b \in J', (a, b) \in E_{DG}\}$. Note that $JG(P)$ is like $SG(P)$ a dag, and hence admits a topological ordering; we denote by $O(JG(P))$ the set of all such orderings. We thus define

Definition 7 *Let P be a program and $\leq\ = (J_1, ..., J_m)$ be a topological ordering of $JG(P)$. Then the splitting sequence induced by $\leq$ is $S_{\leq} = (S_1, ..., S_m)$, where $S_1 = J_1$ and $S_k = S_{k-1} \cup J_k$, for $k = 2, \ldots, m$.*

The sequence $S_{\leq}$ is again indeed a splitting sequence, which we call a *MJC-splitting sequence*. We obtain a result analogous to Theorem 5, but in presence of constraints.

Theorem 6 *Let P be a program. For every $\leq, \leq' \in O(JG(P))$, we have $SEQ^{S_{\leq}}(P) = SEQ^{S_{\leq'}}(P)$.*

Similarly as SCC-models, we thus can define the *MJC-models of P* as $M^{MJC}(P) = SEQ^{S_{\leq}}(P)$ for an arbitrary topological ordering $\leq$ of $JG(P)$.

Example 9 (cont'd) *Reconsider P in Example 7. Then for the ordering $\leq= (\{a\}, \{d\}, \{e\}, \{c\}, \{b\})$ we obtain $SEQ^{S_{\leq}}(P) = \emptyset$, while for $\leq'= (\{e\}, \{c\}, \{b\}, \{a\}, \{d\})$ we obtain $SEQ^{S_{\leq'}}(P) = \{(bc, abc)\}$. On the other hand, $JG(P)$ has the single topological ordering $\leq= (\{e\}, \{c\}, \{a, b\}, \{d\})$, and $SEQ^{S_{\leq}}(P) = \{(bc, abc)\}$; hence $M^{MJC}(P) = \{(bc, abc)\}$. Note that $SEQ(P) = \{(bc, abc), (d, de)\}$.*

The problem in Section 4.2 disappears when we use the MJCs. The program $P = \{\leftarrow b, not\ a;\ b\}$ there has the single MJC $J = \{a, b\}$, since the two SCCs $\{a\}$ and $\{b\}$ are related through the constraint $\leftarrow b, not\ a$ and thus joinable. As desired, we get (b, ab) as the (single) MJC-model of P.

Note that trivially, the MJC- and the SCC-semantics coincide for constraint-free programs (in fact, also in absence of cross-constraints). As for the desiderata, we note:

Proposition 3 *The MJC-models semantics, given by $M^{MJC}(P)$ for any program P, satisfies (D1)-(D2).*

Classical coherence (D3), however, is not ensured by MJC-models, due to lean component merging that fully preserves dependencies. To obtain a model, blurring strict dependencies can be necessary, where two aspects need to taken into account.

(A1) Inconsistency may still emerge from cross-component constraints.

Example 10 *The program $P = \{\leftarrow b, not\ a;\ b;\ b \leftarrow a\}$ has $MJC(P) = \{\{b\}, \{a\}\}$ as $\{b\}, \{a\}$ are not joinable. As the single MJC-splitting sequence, $(\{a\}, \{a, b\})$, admits no split SEQ-model, $M^{MJC}(P) = \emptyset$.*

This can be remedied by suitably merging components that intersect the same constraint.

(A2) A second, orthogonal aspect is dependence.

Example 11 *The program $P = \{\leftarrow b;\ b \leftarrow not\ a\}$ has no MJC-model, as the MJC-splitting sequence $S = (\{a\}, \{a, b\})$ admits no split SEQ-model; the culprit is a, which does not occur in the constraint.*

Clearly, the problem extends to dependence via an (arbitrarily long) chain of rules (e.g., change in Example 11 the rule to $b \leftarrow c_1$, $c_1 \leftarrow c_{i+1}$, $1 \leq i < n$, $c_n \leftarrow not\ a$). Again, this can be remedied by merging components. Many merging policies to ensure (D3) are conceivable; however, such a policy should ideally not dismiss structure unless needed, and it should be efficiently computable; we defer further discussion to Section 7, as the next section will provide useful insight for it.

5 Complexity and Computation

In this section, we consider the computational complexity of the following major reasoning tasks for programs under split SEQ-semantics.

(MCH) Given a program P, a splitting sequence S and an HT-interpretation (X, Y), decide whether (X, Y) is a split semi-equilibrium model of P.

(INF) Given a program P, a splitting sequence S, an atom a and $v \in \{\mathbf{t}, \mathbf{f}, \mathbf{bt}\}$, decide if a is a brave [resp. cautious] *SEQ^S-consequence* of P with value v, denoted $P \models_S^{b,v} a$ [resp. $P \models_S^{c,v} a$], i.e., a has in some (all) $(X, Y) \in SEQ^S(P)$ value v.

(CON) Given a program P and a splitting sequence S, decide whether $SEQ^S(P) \neq \emptyset$.

We consider also SCC- and MJC-splitting sequences and several classes of programs, viz. normal, disjunctive, stratified, and headcycle-free programs.[5] Recall that a program P is *stratified*, if for each $r \in P$ and $C \in SCC(P)$ either $H(r) \cap C = \emptyset$ or $B^-(r) \cap C = \emptyset$; P is *headcycle-free (hcf)*, if $|H(r) \cap C| \leq 1$ for each $r \in P$ and $C \in SCC(P')$, where $P' = \{a \leftarrow B^+(r) \mid r \in P,\ a \in H(r)\}$.

Positive programs are here of less interest, as $SEQ^S(P) = \{(M, M) \mid M \in MM(P)\}$ for each splitting sequence S. Furthermore, hcf-programs are under SEQ-semantics sensitive to body shifts; e.g., $P = \{a \vee b;\ \ a \leftarrow not\ a;\ \ b \leftarrow not\ b\}$ has the SEQ-models (a, ab) and (b, ab), while its shift $P_{\rightarrow} = \{a \leftarrow not\ b;\ \ b \leftarrow not\ a;\ \ a \leftarrow not\ a;\ \ b \leftarrow not\ b\}$ has the single SEQ-model $(\emptyset, ab)$.

[5] Note that [8] did not consider stratified and hcf programs.

Table 1. Complexity of split SEQ-models (completeness results). The same results hold for canonical models (SCC-, MJC-split seq. S); diverging results for SEQ-models are in brackets.

Problem / Program P:		normal, strat. normal, headcycle-free	disj. stratified, disjunctive
(MCH) Model checking:	$(X,Y) \in SEQ^S(P)$?	coNP	Π_2^p
(INF) Brave reasoning:	$P \models_S^{b,v} a$?	Σ_2^p	Σ_3^p
Cautious reasoning:	$P \models_S^{c,v} a$?	Π_2^p	Π_3^p
(CON) Existence:	$SEQ^S(P) \neq \emptyset$?	Σ_2^p [NP]	Σ_3^p [NP]

Overview of complexity results. Our complexity results are summarized in Table 1. Briefly, they show that split SEQ-models have the same complexity as SEQ-models (i.e., structural information does not affect complexity) except on Problem CON, which is harder. The reason is that coherence (D3) no longer holds. In particular, this means that imposing a structural condition on building SEQ-models along SCCs may eliminate such models. Furthermore, it implies that no polynomial-time method μ exists that associates with P a splitting sequence $S = \mu(P)$, using a polynomial-time checkable criterion on P, such that (i) μ respects structure, i.e., $\mu(P) \neq (At(P))$ if $SEQ^S(P) \neq \emptyset$ for some $S \neq (At(P))$, and (ii) μ preserves consistency, i.e., $SEQ(P) \neq \emptyset$ implies $SEQ^S(P) \neq \emptyset$; this holds even if μ may be nondeterministic, i.e., can "guess" a suitable S for P. In other words, the price for ensuring coherence with tractable (or NP) effort is to merge sometimes more components than necessary.

Problems MCH and INF do not become harder, as MCH reduces to polynomially many MCH instances without splitting. The hardness results for arbitrary splitting sequences are inherited from respective results without splitting; we also provide results for stratified and hcf programs.

For SCC and MJC splitting sequences, we obtain analogous results; informally, the problems do not get easier as splitting can be blocked by irrelevant rules.

Details on the derivation of the results in Table 1 are omitted for space reasons and included in extended version of this paper.

Constructing and recognizing canonical splitting sequences. It is well-known that $SCC(P)$ and $SG(P)$ are efficiently computable from P (using Tarjan's [30] algorithm even in linear time); hence, it is not hard to see that one can recognize a SCC-splitting sequence S in polynomial time, and that every such S can be (nondeterministically) generated in polynomial time (in fact, in linear time). We obtain similar tractability results for $MJC(P)$ and MJC-splitting sequences. To this end, we first note the following useful proposition.

Proposition 4 *Let P be a program and let $K_1, K_2 \in SCC(P)$. Then K_1 and K_2 satisfy (i) and (ii) of Definition 5 iff they are disconnected in $SG(P)$, i.e., no path from K_1 to K_2 and vice versa exists.*

Theorem 7 *Given a program P, $MJC(P)$ and $JG(P)$ are computable in polynomial time (in time $O(cs \cdot \|P\|)$, where $cs = |\{r \in P \mid H(r) = \emptyset\}|$ and $\|P\|$ is the size of P).*

Proof (Sketch). For every constraint r, determine all $C_1, \ldots, C_k$ in $SCC(P)$ such that $C_i \cap B(r) \neq \emptyset$; suppose $C_1, \ldots, C_l$, $l \leq k$ are the maximal among them in $SG(P)$. Using Proposition 4, it can be shown that the pairs (C_i, C_j), $1 \leq i \neq j \leq l$ are the joinable pairs witnessed by r (i.e., satisfying (iii)). One can compute $C_1, \ldots, C_l$ efficiently, e.g. using a stratified program P_r with the following rules:

1. $r_j \leftarrow$, for each $C_j \in V_{SG}$ such that $C_j \cap B(r) \neq \emptyset$;
2. $r_j \leftarrow r_i$ and $n\_max\_r_j \leftarrow r_i$, for each $(C_i, C_j) \in E_{SG}$;
3. $max\_r_j \leftarrow r_j,\ not\ n\_max\_r_j$, for each $C_i \in V_{SG}$.

The answer set of P_r yields $C_1, \ldots, C_l$, whose union $C_r = \bigcup_{i=1}^{l} C_i$ is contained in a (unique) MJC C (i.e., $C_r \subseteq C$). The set $MJC(P)$ is built by merging C_r and $C_{r'}$ s.t. $C_r \cap C_{r'} \neq \emptyset$ repeatedly. From $MJC(P)$ and $SG(P)$, computing $JG(P)$ is easy.

Each step: building $SCC(P)$, $SG(P)$ and P; evaluating P_r; computing C_r; merging the C_r's; and building $JG(P)$ from $MJC(P)$ and $SG(P)$ is feasible in linear time, except evaluating P_r, which takes $O(cs{\cdot}\|P\|)$ time; in total, this is $O(cs{\cdot}\|P\|)$ time. □

6 Application: Inconsistency-Tolerant Query Answering

The standard answer set semantics may be regarded as appropriate when a knowledge base, i.e., logic program, is properly specified adopting the CWA principle to deal with incomplete information. Query answering over a knowledge base then resorts usually to brave or cautious inference from the answer sets of a knowledge base; let us focus on the latter here. However, if (unexpected) incoherence arises, then we lose all information and query answers are trivial. This, however, may not be satisfactory, especially if it is not possible to modify the knowledge base, which may be due to various reasons. Paracoherent semantics can be exploited to overcome this problem and to render query answering operational, without trivialization. In particular, SEQ-semantics is attractive as it builds on simple grounds and (1) brings in "unsupported" assumptions, (2) stays in model building close to answer sets, but distinguishes atoms that require such assumptions from atoms derivable without them, (3) keeps the CWA/LP spirit of minimal assumptions, and (4) easily lifts to extensions (nested programs, arbitrary formulas, aggregates, etc).

For instance, consider a variant of the Russell paraphrase from the Introduction [28]:

$$P = \{shaves(joe, X) \leftarrow not\ shaves(X, X);\ man(paul)\}.$$

While this program has no answer set, SEQ-semantics gives us the model

$$(\{shave(joe, paul), man(paul)\}, \{shave(joe, paul), man(paul), shave(joe, joe)\});$$

here the incoherent rule $shaves(joe, joe) \leftarrow not\ shaves(joe, joe)$ obtained by grounding is isolated from rest of the program, avoiding the absence of solutions (a similar intuition is underlying the definition of CWA inhibition rule in [21], used for contradiction removal in a logic program), and allows us to derive, for instance, that $shave(joe, paul)$ and $man(paul)$ are true; furthermore, we can infer that $shave(joe, joe)$ can not be false. Such a capability seems very attractive in query answering.

Now reconsider the program in Example 1, and let us ask for query $go(John)$. Again answer set semantics yields only a trivial answer to the query. However the local incoherence is due to the second and the third rule, and the CWA implies that

$go(Mark)$ is false; hence there is no reason to avoid the answer. Moreover split-SEQ semantics yields the unique model $(\{go(John)\}, \{go(John), go(Bill)\})$ and removes the SEQ-model ambiguity, as it makes stronger gap minimization through the bottom-up evaluation. In this way, the relaxation of CWA is minimized.

Notice that also the well-founded semantics (WFS) [31] avoids cyclic incoherence, but resorts to undefinedness that is cautiously propagated, such that reasoning by cases may be abandoned. For example, consider the program

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

and ask the query c. The program is incoherent due to the last rule; under WFS, c is undefined (as a and b are, due to the first two rules), while split-SEQ semantics yields the models (ac, acd) and (bc, bcd), from which c is a cautious consequence as expected.

7 Discussion and Conclusion

Related work. CR-Prolog [3] adds, roughly speaking, a subset-minimal set R of rules from a pool R' to program P such that $P \cup R$ is coherent, and accepts all answer sets of $P \cup R$. This is a (syntactic) inconsistency management strategy (possibly missing cases), not a logic-level semantic treatment of incoherence. Even for R' consisting of all atoms, it may disagree with SEQ-semantics, as adding facts is stronger than blocking negated atoms (admitting more answer sets).

To our knowledge, modularity aspects of paracoherent semantics have not been studied extensively. A noticeable exception is [9], which studied the applicability of splitting sets for several partial models semantics, among them the L-stable semantics. The latter is in spirit close to semi-equilibrium semantics but uses a different 3-valued logic. Unsurprisingly, it does not satisfy the splitting property. Huang et al. [13] showed that hybrid knowledge bases, which generalize logic programs, have modular paraconsistent semantics for stratified knowledge bases; however, the semantics aims at dealing with classical contradictions and not with incoherence in terms of instability through cyclic negation. Pereira and Pinto [24], using the layering notion, that is similar to SCC-split sequences, introduce Layered Models (LM) semantics which is an alternative semantics that extends the stable models semantics for normal logic programs. But LM are just a superset of stable models, and do not coincide with them on coherent programs, so the CWA is too relaxed. Finally, Faber et al. [10] introduced a notion of modularity for answer set semantics, based on syntactic relevance, which has paracoherent features. However, this notion was geared towards query answering rather than model building, and did not incorporate gap minimization at a semantic level.

Further issues. By the results of Section 5, tractable merging policies that ensure classical coherence (D3) will sometimes merge more components than necessary. To deal with the issues (1) and (2) in Section 4.2, a parametric approach that gradually merges SCCs seems attractive. Let $D_k(C)$ denote the set of all descendants of C in $SG(P)$ within distance $k \geq 0$; then we may proceed as follows.

Create a graph G_k with a node v_r for each constraint r in P, which is labeled with the set of SCCs $\lambda(v_r) = cl_p(\bigcup\{D_k(C_i) \mid C_i \in SCC(P), C_i \cap B(r) \neq \emptyset\})$; here $cl_p(D)$ is a 'closure operation that for a set D of SCCs yields D plus all SCCs that are in $SG(P)$ on some path between two SCCs from D. Merge then nodes v_r and $v_{r'}$ (and their labels, using cl_p) such that $\lambda(v_r) \cap \lambda(v'_r) \neq \emptyset$ as long as possible. After that, add an edge from v to v', if $v \neq v'$ and $SG(P)$ has some edge (C_i, C_j) where $C_i \in \lambda(v)$ and $C_j \in \lambda(v')$. The resulting graph G_k is acyclic and distinct nodes have disjoint labels. Similar as for $JG(P)$, any topological ordering $\leq$ of G_k induces a splitting sequence $S_\leq$ (via the node labels); thanks to an analog of Theorem 6, one can define the M_k-models of P as $M_k(P) = SEQ^{S_\leq}(P)$ for an arbitrary $\leq$.

Clearly, $M_k(P) \subseteq M_{k+1}(P)$ holds for every $k \geq 0$, and $M_k(P) = SEQ(P)$ for large enough k; as $M^{MJC}(P) \subseteq M_0(P)$ holds, we have a hierarchy of models between $M^{MJC}(P)$ and $SEQ(P)$ which eventually establishes (D3); however, predicting the least k such that $M_k(P) \neq \emptyset$ is intractable.

Other relaxed notions of models (using different parameters for cross-constraints and direct dependency) are conceivable; we leave this for future study.

Summary and outlook. We have studied a refinement of SEQ-semantics that respects modular structure, and we gave a semantics via splitting sets that is amenable to bottom up evaluation of programs.

The generic framework of Equilibrium Logic makes it easy to define SEQ-semantics via gap minimization for many extensions of the programs considered here, such as nested programs, programs with aggregates and external atoms, hybrid knowledge bases etc; programs with classical negation require to use more truth values [17]. It remains to consider modularity in these extensions and to define suitable refinements of SEQ-models. Particularly interesting are modular logic programs [14, 6] where explicit (by module encapsulation) and implicit modularity (by splitting sets) occur at the same time.

Besides language extensions, another issue is generalizing the model selection. To this end, preference of gap minimization at higher over lower levels must be supported; however, this intuitively requires more guessing and hinders bottom up evaluation. Finally, efficient algorithms and an implementation are to be done, as well integration into an answer set building framework.

References

1. Alcântara, J., Damásio, C.V., Pereira, L.M.: A declarative characterization of disjunctive paraconsistent answer sets. In: Proc. ECAI-04. pp. 951–952. IOS Press (2004)
2. Apt, K., Blair, H., Walker, A.: Towards a theory of declarative knowledge. In: Minker [16], pp. 89–148
3. Balduccini, M., Gelfond, M.: Logic programs with consistency-restoring rules. In: McCarthy, J., Williams, M.A. (eds.) Int'l Symp. Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symp. Series. pp. 9–18 (2003)
4. Ben-Eliyahu, R., Dechter, R.: Propositional semantics for disjunctive logic programs. Ann. Math. & Artif. Intell. 12, 53–87 (1994)
5. Blair, H.A., Subrahmanian, V.S.: Paraconsistent logic programming. Theor. Comput. Sci. 68(2), 135–154 (1989)
6. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Modular nonmonotonic logic programming revisited. In: Proc. ICLP-09. pp. 145–159. LNCS 5649, Springer (2009)
7. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Ann. Math. & Artif. Intell. 15(3/4), 289–323 (1995)

8. Eiter, T., Fink, M., Moura, J.: Paracoherent answer set programming. In: F. Lin, U. Sattler and M. Truszczyński (eds.), Proc. KR-10, May 9-13, 2010, Toronto, Canada. pp. 486–496. AAAI Press (2010)
9. Eiter, T., Leone, N., Saccà, D.: On the partial semantics for disjunctive deductive databases. Ann. Math. & Artif. Intell. 19(1/2), 59–96 (1997)
10. Faber, W., Greco, G., Leone, N.: Magic sets and their application to data integration. J. Comput. Syst. Sci. 73(4), 584–609 (2007)
11. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365–385 (1991)
12. Heyting, A.: Die formalen Regeln der intuitionistischen Logik. Sitzungsberichte der Preussischen Akademie der Wissenschaften 16(1), 42–56 (1930)
13. Huang, S., Li, Q., Hitzler, P.: Reasoning with inconsistencies in hybrid MKNF knowledge bases. Logic Journal of the IGPL 21(2), 263–290 (2013)
14. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity aspects of disjunctive stable models. J. Artif. Intell. Res. (JAIR) 35, 813–857 (2009)
15. Lifschitz, V., Turner, H.: Splitting a logic program. In: Proc. ICLP-94. pp. 23–38. MIT-Press (1994)
16. Minker, J. (ed.): Foundations of Deductive Databases and Logic Programming. Morgan Kaufman, Washington DC (1988)
17. Odintsov, S.P., Pearce, D.: Routley semantics for answer sets. In: Proc. LPNMR-05. LNCS 3662, pp. 343–355. Springer (2005)
18. Osorio, M., Ramírez, J.R.A., Carballido, J.L.: Logical weak completions of paraconsistent logics. J. Log. Comput. 18(6), 913–940 (2008)
19. Pearce, D.: Equilibrium logic. Ann. Math. & Artif. Intell. 47(1-2), 3–41 (2006)
20. Pearce, D., Valverde, A.: Quantified equilibrium logic and foundations for answer set programs. In: Proc. ICLP-08. LNCS 5366, pp. 546–560. Springer (2008)
21. Pereira, L.M., Alferes, J.J., Aparício, J.N.: Contradiction removal semantics with explicit negation. In: Logic at Work. LNCS 808, pp. 91–105. Springer (1992)
22. Pereira, L.M., Pinto, A.M.: Revised stable models - a semantics for logic programs. In: Proc. EPIA-05. LNCS 3808, pp. 29–42. Springer (2005)
23. Pereira, L.M., Pinto, A.M.: Approved models for normal logic programs. In: Proc. LPAR-07. LNCS 4790, pp. 454–468. Springer (2007)
24. Pereira, L.M., Pinto, A.M.: Layered models top-down querying of normal logic programs. In: Proc. PADL-09. LNCS 5418, pp. 254–268. Springer (2009)
25. Przymusinski, T.: Stable semantics for disjunctive programs. New Generation Computing 9, 401–424 (1991)
26. Przymusinski, T.C.: On the declarative semantics of deductive databases and logic programs. In: Minker [16], pp. 193–216
27. Saccà, D., Zaniolo, C.: Partial models and three-valued stable models in logic programs with negation. In: Subrahmanian, V. et al. (ed.) Proc. LPNMR-91. pp. 87–101. MIT Press (1991)
28. Sakama, C., Inoue, K.: Paraconsistent stable semantics for extended disjunctive programs. J. Log. Comput. 5(3), 265–285 (1995)
29. Seipel, D.: Partial evidential stable models for disjunctive deductive databases. In: Proc. LPKR-97. LNCS 1471, pp. 66–84. Springer (1997)
30. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. 1(2), 146–160 (1972)
31. van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. J. ACM 38(3), 620–650 (1991)
32. You, J.H., Yuan, L.: A three-valued semantics for deductive databases and logic programs. J. Comput. Syst. Sci. 49, 334–361 (1994)