Künstliche Intelligenz manuscript No.
(will be inserted by the editor)

Aggregates in Answer Set Programming

Mario Alviano · Wolfgang Faber

Received: date / Accepted: date

Abstract Aggregates are among the most important linguistic extensions of Answer Set Programming (ASP), allowing for compact representations of properties and inductive definitions involving sets of propositions. Common use cases of aggregates in ASP are reported in this paper, which mainly focus on the semantics implemented by mainstream solvers, namely the F-stable model semantics. Other well-established semantics are also briefly discussed, providing a historical perspective on the foundation of logic programs with aggregates.

Keywords answer set programming · recursive aggregates · nonmonotonic reasoning

1 Introduction

ASP is a declarative language for knowledge representation and reasoning [16], where highly complex computational problems are modeled by means of logic programs comprising disjunctive logic rules, possibly using default negation under stable model semantics in their bodies [24,25]. One of the origins of ASP is Datalog, a database query language based on logic programming. Since constructs for aggregate functions like SUM or COUNT are well-established in database query languages, it was natural to introduce them into Datalog [41], and consequently ASP as well [12,19,21,26, 31,34,39]. Indeed, aggregation functions allow for expressing properties on sets of atoms declaratively, and have found applications in several contexts. Common use cases are enforcing *functional dependencies*, constraining *nondeterministic guesses*, and *inductive definitions* involving recursive aggregations. Some examples are provided later on in Section 3.

Mainstream ASP solvers [2,3,17,18,20,23,27,33,35] mostly agree on the semantics of aggregates [19,21], here referred to as the F-stable model semantics. Some alternatives will be briefly reviewed in Section 4, but it is important to stress here that several semantics agree on a large class of commonly used aggregates, referred to as *convex* in the literature [32]; the consensus is lost for non-convex aggregations, which also introduce a new complexity source for several reasoning tasks.

According to the F-stable model semantics, a model I of a propositional program Π is *stable* if there is no $J \subset I$ satisfying all rules of the *program reduct* Π^I, obtained from Π by removing all rules whose body is false with respect to I, and by fixing the interpretation of negated literals in the remaining rules. The semantics of programs with symbolic variables can be defined in terms of their *grounding*, obtained by substituting variables occurring in rules with ground terms in all possible ways. A few additional insights into the syntax and semantics of aggregates is given in Section 2.

M. Alviano
DEMACS, University of Calabria, Italy
E-mail: alviano@mat.unical.it

W. Faber
AINF, Alpen-Adria-Universität Klagenfurt, Austria
E-mail: wf@wfaber.com

2 Syntax and Semantics

The notion of F-stable models, as given in the introduction, is not significantly different from the original definition of stable models for programs without aggregates [25]. Actually, the aspects that require clarification are *how aggregates are written*, *how aggregates are grounded*, and *how ground aggregates are interpreted.* This section provides sufficient notions to understand these concepts; familiarity with standard syntax and semantics of ASP is assumed.

An *aggregate element* has the following form:

$$t_1, \ldots, t_m : \varphi$$

where $t_1, \ldots, t_m$ ($m \geq 1$) are terms, and φ is a conjunction of literals; all variables in $t_1, \ldots, t_m$ occur in φ. The intuitive meaning is the set of tuples $t_1, \ldots, t_m$ for which φ holds. An *aggregate* has the following form:

$$f\{e_1; \cdots; e_n\} \odot t$$

where f is an aggregation function, $e_1, \ldots, e_n$ ($n \geq 1$) are aggregate elements, $\odot$ is an arithmetic comparator, and t is a ground term. The intuition is to apply f on the multiset of the first terms of all tuples represented by the entries and compare the result to t using $\odot$. Common aggregation functions are #count, #sum, #min, and #max.

The grounding of programs with aggregates is based on the notions of *global* and *local variables.* Global variables of a rule are variables occurring in at least one literal not involved in any aggregation; other variables are local to the aggregate element they occur in. A ground instance of a rule is obtained by replacing global variables with ground terms (of a fixed set), and then by expanding local variables of each aggregate element. Specifically, the expansion of an aggregate element comprises all ground aggregate elements obtained by substituting local variables with ground terms in all possible ways.

Example 1 Consider the following rule:

```
:- p(X), #sum{Y : q(X,Y,Z)} >= 2,
   #sum{Y,Zp : q(X,Y,Zp)} <= 3.
```

Here, X is a global variable, while Y, Z and Zp are local variables. Note that the same variable Y is used as a local variable by the two aggregates above, but this does not create any link between the two aggregates; in fact, Y could be replaced by Yp in either of the two aggregates, and the grounding of the above rule would not change for any set of ground terms.

The grounding of the rule above, for the set $\{1, 2\}$ of ground terms, yields the following ground rules:

```
:- p(1), #sum{1 : q(1,1,1); 1 : q(1,1,2);
        2 : q(1,2,1); 2 : q(1,2,2)} >= 2,
   #sum{1,1 : q(1,1,1); 1,2 : q(1,1,2);
        2,1 : q(1,2,1); 2,2 : q(1,2,2)} <= 3.
:- p(2), #sum{1 : q(2,1,1); 1 : q(2,1,2);
        2 : q(2,2,1); 2 : q(2,2,2)} >= 2,
   #sum{1,1 : q(2,1,1); 1,2 : q(2,1,2);
        2,1 : q(2,2,1); 2,2 : q(2,2,2)} <= 3.
```

These rules form the basis for Example 2. ■

An interpretation I is a set of atoms; atoms in I are true, those not in I are false. In order to apply I to ground aggregates, first consider a set S of ground aggregate elements, and let $I(S)$ be the following set of tuples:

$$\{\langle t_1, \ldots, t_m \rangle \mid (t_1, \ldots, t_m : \varphi) \in S \wedge I \models \varphi\}.$$

essentially comprising the tuples of ground terms associated with true conjunctions. The set $I(S)$ is used to obtain a multiset by collecting the first elements of all tuples, on which the aggregation function is applied:

$$I(f(S)) := f([t_1 \mid \langle t_1, \ldots, t_m \rangle \in I(S)]).$$

The natural interpretation of the comparator defines satisfaction:

$I \models f(S) \odot t$ if and only if $I(f(S)) \odot t$ holds.

Example 2 Let S_1, S_2 be the following sets of ground aggregate elements from Example 1:

$$S_1 := \{\mathtt{1 : q(1,1,1); 1 : q(1,1,2);}$$
$$\mathtt{2 : q(1,2,1); 2 : q(1,2,2)}\}$$
$$S_2 := \{\mathtt{1,1 : q(1,1,1); 1,2 : q(1,1,2);}$$
$$\mathtt{2,1 : q(1,2,1); 2,2 : q(1,2,2)}\}$$

and let I be $\{\mathtt{q(1,1,1), q(1,1,2), q(1,2,1)}\}$. Then $I(S_1)$ is $\{\langle 1\rangle, \langle 2\rangle\}$ and $I(S_2)$ is $\{\langle 1,1\rangle, \langle 1,2\rangle, \langle 2,1\rangle\}$. Hence, $I(\texttt{\#sum}(S_1)) = \texttt{\#sum}([1,2]) = 3$, and therefore I satisfies $\texttt{\#sum}(S_1)$ >= 2. On the other hand, $I(\texttt{\#sum}(S_2)) = \texttt{\#sum}([1,1,2]) = 4$, and $\texttt{\#sum}(S_2)$ <= 3 is not satisfied by I. ■

3 Typical Use Cases

In many ASP programs, *functional dependencies* are enforced by means of COUNT aggregates, as in the following rule:

```
:- time(T), #count{A : do(A,T)} > 1.
```

The above rule in fact constrains relation `do` to satisfy the functional dependency $\texttt{do}[2] \rightarrow \texttt{do}[1]$, that is, there must be a unique value of the first argument of relation `do` for any value in the second argument of `do`. In this case, the functional dependency stems from a planning scenario where at most one action can be done at each time step.

Similarly, aggregate functions are also commonly used in ASP to constrain nondeterministic guesses, as for example in the *knapsack problem*, where the total weight of the selected items must not exceed a given limit, which can be modeled by means of a rule of the following form:

```
:- lim(L), #sum{W,O: obj(O,W,V), in(O)} > L.
```

where `O` is an object with weight `W` and value `V`. In this case using `W,O` rather than just `W` guarantees that multiple objects with the same weight are accounted for correctly.

The examples above use aggregation functions in a rather simple form, that is, only in *integrity constraints*, which are rules with empty heads. However, ASP does not restrict the use of aggregation functions in rules with nonempty heads, which allows for representing problems that require inductive definitions involving aggregates in rule bodies.

As an example, consider the *company controls problem* in the area of stock markets, in which the goal is to determine pairs of companies x, y such that x controls y, given information on stock possessions of companies. In order to control a company y, a company x has to exert control over more than 50% of the stock of y. Company x can exert control over stock of y in two different ways: x itself may possess a certain percentage of stock of y, in this case we say that x *directly exerts control* over the respective percentage of stock of y. Alternatively, if x controls a company z, which exerts control over stock of y, then we say that x *indirectly exerts control* over the respective stock of y. Company x then exerts control over the *sum* of stock of y over which it directly and indirectly exerts control.

Alternatively, we can define the control relation inductively, by saying that company x controls y if the sum of the shares in y possessed by x and by companies controlled by x is more than 50%. The following rule encodes this inductive definition:

```
controls(X,Y) :- company(X), company(Y),
  #sum{S : owns(X,Y,S);
     S,Z : controls(X,Z), owns(Z,Y,S)} > 50.
```

In this case, for each pair x, y of ground terms, the aggregation is on the multiset obtained from the set

$$\{\langle s\rangle \mid \texttt{owns}(x,y,s)\} \cup$$
$$\{\langle s,z\rangle \mid \texttt{controls}(x,z) \wedge \texttt{owns}(z,y,s)\}$$

of tuples by projecting all elements but the first. Interestingly, the company controls problem, as stated above, can be solved in polynomial time, but represented a challenge for ASP systems due to the presence of recursive aggregates.

Inductive definitions involving aggregates can be combined with other constructs of ASP, as for example with *choice rules*, often used to define the search space for nondeterministic guesses. Continuing with the company control problem, one may be interested in checking whether the acquisition of some shares, available on the stock exchange, may guarantee to exert control on some target companies. In this case, if available packages of shares are represented by predicate `onSale`, the following choice rule and inductive definition can be used in combination with an integrity constraint to assert the target controls:

```
{buy(Pkg)} :- onSale(Pkg,Comp,Share,Cost).
controls(X,Y) :- company(X), company(Y),
  #sum{S : owns(X,Y,S);
     S,Z : controls(X,Z), owns(Z,Y,S);
     S,P : buy(P), onSale(P,Y,S,C)} > 50.
:- target(X,Y), not control(X,Y).
```

Note that in this case, for each pair x, y of ground terms, the aggregation is on the multiset resulting from the set

$$\{\langle s\rangle \mid \texttt{owns}(x,y,s)\} \cup$$
$$\{\langle s,z\rangle \mid \texttt{controls}(x,z) \wedge \texttt{owns}(z,y,s)\} \cup$$
$$\{\langle s,p\rangle \mid \texttt{buy}(p) \wedge \exists c\ \texttt{onSale}(p,y,s,c)\},$$

where packages and companies are assumed to have disjoint identifiers. This problem belongs to the first level of the polynomial hierarchy (it is NP-complete), and its source of complexity is not the aggregation, but the combination of choice rules and integrity constraints.

As a final use case, consider the *generalized subset sum* problem [1,6,7,13], a prototypical problem for the second level of the polynomial hierarchy, that is, a Σ_2^P-complete problem. In the generalized subset sum problem, two vectors u, v of integers, and an integer b are given, and the task is to decide whether the formula

$$\exists x \forall y (ux + vy \neq b)$$

is true, where x and y are vectors of binary variables of the same length as u and v, respectively. For example, for $u = [1, 2]$, $v = [2, 3]$, and $b = 5$, the task is to decide whether the following formula is true:

$$\exists x_1 x_2 \forall y_1 y_2 (1 \cdot x_1 + 2 \cdot x_2 + 2 \cdot y_1 + 3 \cdot y_2 \neq 5).$$

By combining several constructs and patterns provided by ASP, such a complex computational problem can be naturally encoded by the following rules:

```
{true(exists,X,C)} :- var(exists,X,C).
true(forall,Y,C) :- var(forall,Y,C),unequal.
:- not unequal.
unequal :- #sum{C,Q,X : true(Q,X,C)} != b.
```

First, a choice rule is used to encode the existential guess, while the second and third rules are used to encode the universal quantification on variables Ys. Intuitively, any model I of the program is forced to contain true(forall,y,c) for every var(forall,y,c) in the input because of the integrity constraint; however, the integrity constraint does not belong to the program reduct, and therefore I is stable only if there is no assignment for variables Ys such that the aggregate in the last rule becomes false. Indeed, for the following facts:

```
var(exists,1,1). var(exists,2,2).
var(forall,1,2). var(forall,2,3).
#const b = 5.
```

the program admits a unique stable model representing the only solution for $u = [1, 2]$, $v = [2, 3]$, and $b = 5$, that is, x_1 true and x_2 false. In this case, the aggregation is one of the source of complexities of the above encoding, as in fact it does not belong to the class of convex aggregates, the complexity boundary of ASP programs under F-stable semantics [4].

4 History and Alternative Semantics

Logic programs had been endowed with aggregates for quite some time in Datalog, a database query language. In the late 1990ies these were also introduced to ASP, first to enhance the languages of ASP systems. In SMODELS [39], weight and cardinality constraints were supported [36], while DLV [29,8] had a slightly more general framework similar to the one presented in this paper [20]. While SMODELS allowed for aggregates in recursive definitions, DLV did not for some time.

It was soon realized that recursive definitions pose some peculiar semantic issues. In order to overcome these, initially two kinds of semantics were suggested: one sometimes termed PSP (for Pelov [37], and Son and Pontelli [40]) and another known as FLP (for Faber, Leone, Pfeifer [19], and Ferraris [21]). The latter forms the basis for the semantics mentioned here (F-stable, for Ferraris), which deals with negated aggregates in an arguably better way. These two groups of semantics coincide on a significant number of programs, namely all programs that do not contain aggregates occurring in recursion, and programs that contain only aggregates that are convex. Many aggregates are indeed convex, a notable exception being #sum when the summation is extended for both positive and negative numbers.

While the discussion on the appropriate semantics for aggregates is still continuing (see for example [5,26,38]), it appears as if there were consensus on programs that contain only aggregates that are convex, and in particular on programs that do not involve aggregates in inductive definitions. The latter have a clear semantics, and even most of the earliest semantics agree on them. Moreover, most programs seen in practice at the moment fall into this class. And for many semantics, going beyond convex aggregates increases the computational complexity of many reasoning tasks in the general case (if P $\neq$ NP; see for example [10,11]). In this respect, while convex is the complexity boundary of F-stable models [4], this is not the case for PSP, for which there are several non-convex aggregations that do not increase the complexity of common reasoning tasks [9].

Complexity arguments also motivated the introduction of several techniques to normalize ASP programs with aggregates since the beginning [39]. Specifically, normalized ASP programs only contain monotone aggregates, and therefore some rewritings are correct only in the stratified case [30]. In fact, any *polynomial, faithful, and modular* translation [28] to compile logic programs with non-convex aggregates into equivalent logic programs that only comprise monotone aggregates must introduce a different source of complexity. In a recent polynomial, faithful, and modular translation implemented in GRINGO [22], this source of complexity is recursive disjunction in rule heads [6, 7]. Other relevant rewriting techniques in the literature aim to completely eliminate aggregates [15, 14], and proved to be quite efficient in practice; these rewritings produce aggregate-free programs preserving F-stable models only in the stratified case, or if recursion is limited to convex aggregates, which is guaranteed to be the case for the output of the latest versions of GRINGO.

More extensive discussions of the history of aggregates in ASP can be found in the works of Pelov [37] and Ferraris [21].

Acknowledgements Mario Alviano has been 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). Wolfgang Faber has been partially supported by the EU H2020 Marie Skłodowska-Curie project "MIREL" (#690974) while at the University of Huddersfield, UK.

References

1. Alviano, M.: Evaluating answer set programming with non-convex recursive aggregates. Fundam. Inform. **149**(1-2), 1–34 (2016). DOI 10.3233/FI-2016-1441. URL `https://doi.org/10.3233/FI-2016-1441`
2. Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P., Zangari, J.: The ASP system DLV2. In: M. Balduccini, T. Janhunen (eds.) Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings, *Lecture Notes in Computer Science*, vol. 10377, pp. 215–221. Springer (2017). DOI 10.1007/978-3-319-61660-5_19. URL `https://doi.org/10.1007/978-3-319-61660-5_19`
3. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: F. Calimeri, G. Ianni, M. Truszczynski (eds.) Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings, *Lecture Notes in Computer Science*, vol. 9345, pp. 40–54. Springer (2015). DOI 10.1007/978-3-319-23264-5_5
4. Alviano, M., Faber, W.: The complexity boundary of answer set programming with generalized atoms under the FLP semantics. In: P. Cabalar, T.C. Son (eds.) Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 8148, pp. 67–72. Springer (2013). DOI 10.1007/978-3-642-40564-8_7
5. Alviano, M., Faber, W.: Supportedly stable answer sets for logic programs with generalized atoms. In: B. ten Cate, A. Mileo (eds.) Web Reasoning and Rule Systems - 9th International Conference, RR 2015, Berlin, Germany, August 4-5, 2015, Proceedings, *Lecture Notes in Computer Science*, vol. 9209, pp. 30–44. Springer (2015). DOI 10.1007/978-3-319-22002-4_4
6. Alviano, M., Faber, W., Gebser, M.: Rewriting recursive aggregates in answer set programming: back to monotonicity. TPLP **15**(4-5), 559–573 (2015). DOI 10.1017/S1471068415000228
7. Alviano, M., Faber, W., Gebser, M.: From non-convex aggregates to monotone aggregates in ASP. In: S. Kambhampati (ed.) Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, pp. 4100–4194. IJCAI/AAAI Press (2016). URL `http://www.ijcai.org/Abstract/16/610`
8. Alviano, M., Faber, W., Leone, N., Perri, S., Pfeifer, G., Terracina, G.: The disjunctive datalog system DLV. In: O. de Moor, G. Gottlob, T. Furche, A.J. Sellers (eds.) Datalog Reloaded - First International Workshop, Datalog 2010, Oxford, UK, March 16-19, 2010. Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 6702, pp. 282–301. Springer (2010). DOI 10.1007/978-3-642-24206-9_17. URL `https://doi.org/10.1007/978-3-642-24206-9_17`
9. Alviano, M., Faber, W., Strass, H.: Boolean functions with ordered domains in answer set programming. In: D. Schuurmans, M.P. Wellman (eds.) Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA., pp. 879–885. AAAI Press (2016). URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12078`
10. Alviano, M., Leone, N.: Complexity and compilation of gz-aggregates in answer set programming. TPLP **15**(4-5), 574–587 (2015). DOI 10.1017/S147106841500023X
11. Alviano, M., Leone, N.: On the properties of gz-aggregates in answer set programming. In: S. Kambhampati (ed.) Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, pp. 4105–4109. IJCAI/AAAI Press (2016). URL `http://www.ijcai.org/Abstract/16/611`
12. Bartholomew, M., Lee, J., Meng, Y.: First-order semantics of aggregates in answer set programming via modified circumscription. In: Logical Formalizations of Commonsense Reasoning, Papers from the 2011 AAAI Spring Symposium, Technical Report SS-11-06, Stanford, California, USA, March 21-23, 2011. AAAI (2011)
13. Berman, P., Karpinski, M., Larmore, L.L., Plandowski, W., Rytter, W.: On the complexity of pattern matching for highly compressed two-dimensional texts. J. Comput. Syst. Sci. **65**(2), 332–350 (2002). DOI 10.1006/jcss.2002.1852
14. Bomanson, J., Gebser, M., Janhunen, T.: Improving the normalization of weight rules in answer set programs. In: E. Fermé, J. Leite (eds.) JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8761, pp. 166–180. Springer (2014). DOI 10.1007/978-3-319-11558-0_12
15. Bomanson, J., Janhunen, T.: Normalizing cardinality rules using merging and sorting construc-

tions. pp. 187–199. Springer (2013). DOI 10.1007/978-3-642-40564-8_19. URL `http://dx.doi.org/10.1007/978-3-642-40564-8_19`

16. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011). DOI 10.1145/2043174.2043195
17. Bruynooghe, M., Blockeel, H., Bogaerts, B., de Cat, B., Pooter, S.D., Jansen, J., Labarre, A., Ramon, J., Denecker, M., Verwer, S.: Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with *IDP3*. TPLP **15**(6), 783–817 (2015). DOI 10.1017/S147106841400009X. URL `https://doi.org/10.1017/S147106841400009X`
18. Devriendt, J., Bogaerts, B., Bruynooghe, M., Denecker, M.: On local domain symmetry for model expansion. TPLP **16**(5-6), 636–652 (2016). DOI 10.1017/S1471068416000508. URL `https://doi.org/10.1017/S1471068416000508`
19. Faber, W., Pfeifer, G., Leone, N.: Semantics and complexity of recursive aggregates in answer set programming. Artif. Intell. **175**(1), 278–298 (2011). DOI 10.1016/j.artint.2010.04.002
20. Faber, W., Pfeifer, G., Leone, N., Dell'Armi, T., Ielpa, G.: Design and implementation of aggregate functions in the DLV system. TPLP **8**(5-6), 545–580 (2008). DOI 10.1017/S1471068408003323
21. Ferraris, P.: Logic programs with propositional connectives and aggregates. ACM Trans. Comput. Log. **12**(4), 25 (2011). DOI 10.1145/1970398.1970401
22. Gebser, M., Kaminski, R., König, A., Schaub, T.: Advances in *gringo* series 3. In: J.P. Delgrande, W. Faber (eds.) LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings, *Lecture Notes in Computer Science*, vol. 6645, pp. 345–351. Springer (2011). DOI 10.1007/978-3-642-20895-9_39. URL `http://dx.doi.org/10.1007/978-3-642-20895-9_39`
23. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. **187**, 52–89 (2012). DOI 10.1016/j.artint.2012.04.001
24. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: R.A. Kowalski, K.A. Bowen (eds.) Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19, 1988 (2 Volumes), pp. 1070–1080. MIT Press (1988)
25. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. **9**(3/4), 365–386 (1991). DOI 10.1007/BF03037169
26. Gelfond, M., Zhang, Y.: Vicious circle principle and logic programs with aggregates. TPLP **14**(4-5), 587–601 (2014). DOI 10.1017/S1471068414000222
27. Giunchiglia, E., Lierler, Y., Maratea, M.: Answer set programming based on propositional satisfiability. J. Autom. Reasoning **36**(4), 345–377 (2006). DOI 10.1007/s10817-006-9033-2
28. Janhunen, T.: Some (in)translatability results for normal logic programs and propositional theories. Journal of Applied Non-Classical Logics **16**(1-2), 35–86 (2006). DOI 10.3166/jancl.16.35-86. URL `http://dx.doi.org/10.3166/jancl.16.35-86`
29. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3), 499–562 (2006). DOI 10.1145/1149114.1149117. URL `http://doi.acm.org/10.1145/1149114.1149117`
30. Liu, G., You, J.: Relating weight constraint and aggregate programs: Semantics and representation. TPLP **13**(1), 1–31 (2013). DOI 10.1017/S147106841100038X. URL `http://dx.doi.org/10.1017/S147106841100038X`
31. Liu, L., Pontelli, E., Son, T.C., Truszczynski, M.: Logic programs with abstract constraint atoms: The role of computations. Artif. Intell. **174**(3-4), 295–315 (2010). DOI 10.1016/j.artint.2009.11.016
32. Liu, L., Truszczynski, M.: Properties and applications of programs with monotone and convex constraints. J. Artif. Intell. Res. (JAIR) **27**, 299–334 (2006). DOI 10.1613/jair.2009. URL `http://dx.doi.org/10.1613/jair.2009`
33. Maratea, M., Pulina, L., Ricca, F.: A multi-engine approach to answer-set programming. TPLP **14**(6), 841–868 (2014). DOI 10.1017/S1471068413000094
34. Marek, V.W., Niemelä, I., Truszczynski, M.: Logic programs with monotone abstract constraint atoms. TPLP **8**(2), 167–199 (2008). DOI 10.1017/S147106840700302X
35. Mariën, M., Wittocx, J., Denecker, M., Bruynooghe, M.: SAT(ID): satisfiability of propositional logic extended with inductive definitions. In: H.K. Büning, X. Zhao (eds.) Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings, *Lecture Notes in Computer Science*, vol. 4996, pp. 211–224. Springer (2008). DOI 10.1007/978-3-540-79719-7_20
36. Niemelä, I., Simons, P., Soininen, T.: Stable model semantics of weight constraint rules. In: M. Gelfond, N. Leone, G. Pfeifer (eds.) Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99, El Paso, Texas, USA, December 2-4, 1999, Proceedings, *Lecture Notes in Computer Science*, vol. 1730, pp. 317–331. Springer (1999). DOI 10.1007/3-540-46767-X_23
37. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and stable semantics of logic programs with aggregates. TPLP **7**(3), 301–353 (2007). DOI 10.1017/S1471068406002973. URL `http://dx.doi.org/10.1017/S1471068406002973`
38. Shen, Y., Wang, K., Eiter, T., Fink, M., Redl, C., Krennwallner, T., Deng, J.: FLP answer set semantics without circular justifications for general logic programs. Artif. Intell. **213**, 1–41 (2014). DOI 10.1016/j.artint.2014.05.001
39. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artif. Intell. **138**(1-2), 181–234 (2002). DOI 10.1016/S0004-3702(02)00187-X
40. Son, T.C., Pontelli, E.: A constructive semantic characterization of aggregates in answer set programming. TPLP **7**(3), 355–375 (2007). DOI 10.1017/S1471068406002936
41. Zaniolo, C., Yang, M., Das, A., Shkapsky, A., Condie, T., Interlandi, M.: Fixpoint semantics and optimization of recursive datalog programs with aggregates. TPLP **17**(5-6), 1048–1065 (2017). DOI 10.1017/S1471068417000436. URL `https://doi.org/10.1017/S1471068417000436`