Answer Set Programming without Unstratified Negation

Ilkka Niemelä

Helsinki University of Technology (TKK)
Department of Information and Computer Science
Ilkka.Niemela@tkk.fi

Abstract. The paper argues that for answer set programming purposes it is not necessary to use unstratified negation but it is more appropriate to employ a basic language based on simple choice constructs, integrity constraints, and stratified negation. This offers a framework that enables natural problem encodings and smooth extensions, for instance, with constraints and aggregates.

1 Introduction

The stable model semantics [1] was introduced originally to provide a declarative account of negation as failure in normal logic programs used in logic programming systems such as Prolog. In particular, the challenge was to capture the problematic case of recursion through negation, i.e., *unstratified negation.*

In the mid 1990s systems capable of computing stable models for tens of thousands of (ground) rules were emerging [2,3]. Then it was realized that logic programs with stable models could be used in a novel way to solve challenging search problems [4,5,6]. The name *answer set programming* (ASP) was coined to this new paradigm where the idea is to see rules as constraints characterizing a set of (stable) models. Now a given search problem can be solved by encoding the problem as a set of rules such that the stable models of the rules correspond to the solutions of the original problem. Hence, a solution to a given problem can be found by giving the logic program encoding as input to an ASP solver which computes a stable model of the encoding and then a solution of the original problem can be extracted from the computed stable model.

ASP has its origins in the stable model semantics of normal programs. Naturally this class of programs has provided the basic logic program language for ASP and it has been the starting point for extensions including disjunctions and aggregates which increase expressivity and often also complexity. In particular, unstratified negation is an essential part of ASP based on normal programs because without recursion through negation a normal program has at most one stable model and, hence, is not possible to capture potential solutions to a given problem as alternative stable models. For example, to encode a choice whether to include an atom a in a stable model or not we need to introduce a new atom, say $\overline{a}$, and employ unstratified negation as in the two rules:

$$a \leftarrow \text{not } \overline{a} \qquad \overline{a} \leftarrow \text{not } a.$$

Such encodings using negation through recursion are very challenging to understand and develop. To overcome the problem, normal programs were extended, for example, by disjunctions [7] and various choice constructs [8,9].

The paper reconsiders the role of normal programs allowing unstratified negation as the basic language for ASP. It argues that for ASP purposes it is not necessary to use unstratified negation but it is more suitable to employ a basic language based on simple choice constructs, integrity constraints, and stratified negation. This offers a framework where recursive definitions are allowed, problem encoding can be done in a very direct way using a generate and test approach, and unstratified negation is easy to capture if needed. Moreover, the approach provides a basic language which is straightforward to extend with different kinds of constraints and aggregates without changing the underlying ideas in the semantics and where primitives such as disjunction [7] that increase expressivity (and complexity) can be added.

2 SCI Programs

We put forward a class of logic programs which we call **SCI** programs (for Stratified negation, Choice constructs, and Integrity constraints) where programs consist of normal rules of the form (1), choice rules of the form (2) and integrity constraints of the form (3)

$$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \tag{1}$$
$$\{a_1, \ldots, a_l\} \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \tag{2}$$
$$\leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n \tag{3}$$

where a_i, b_j, c_k are all atoms. As usual the positive literals $b_1, \ldots, b_m$ and negative literals not $c_1, \ldots,$ not c_n are called the body of the rule and a ($\{a_1, \ldots, a_l\}$) is the head for a normal (choice) rule and for integrity constraints the head is empty. For a **SCI** program P we denote by NR(P), CR(P), and IC(P) the sets of normal rules, choice rules and IC(P) integrity constraints, respectively.

In **SCI** programs unstratified negation is not allowed and programs are required to be stratified in the usual sense [10]: for each program there should exist a mapping S from the predicate symbols in the program to natural numbers such that for each rule and each predicate symbol p appearing in the head (i) $S(p) \geq S(q)$ holds for every predicate symbol q appearing in the positive body literals and (ii) $S(p) > S(q)$ holds for every predicate symbol q appearing in the negative body literals of the rule.

3 Semantics of SCI Programs

Choice and integrity rules can be seen as special cases of the cardinality constraints in [11] and, hence, the stable model semantics for **SCI** programs can be defined using a Gelfond-Lifschitz type of a reduct generalized to choice rules as done in [11] (or more generally for abstract constraint atoms in [12]) .

However, here we propose a slightly different method of defining the semantics which coincides with the approach explained above for **SCI** programs but provides a more direct path to extending the language as will be discussed below. We outline the method in the ground case, i.e., for programs without variables. Generalizing it to programs with variables can be done in the usual way by Herbrand instantiation.

Models of a program are sets of atoms and a positive (negative) literal a (not a) is satisfied in a model S if $a \in S$ ($a \notin S$). A normal rule of the form (1) is satisfied by a model S if the head is satisfied whenever all the body literals are satisfied. A choice rule is satisfied in any model and an integrity constraint is satisfied if at least one of the body literals is not. A model S satisfies a set of rules P if it satisfies all the rules (denoted by $S \models P$).

Given a set S of atoms (a candidate model) the **SCI**-reduct P^S of P w.r.t. S is the set of rules including all normal rules $\mathrm{NR}(P)$ in P and a rule

$$a_l \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$$

for each choice rule of the form (2) in $\mathrm{CR}(P)$ with $a_l \in S$. Notice that P^S is a normal stratified program without integrity constraints and it has a unique canonical model which can be defined iteratively bottom up layer by layer in the stratification, for details see [10]. For a normal stratified program P we denote the unique stratified model by $\mathrm{StM}(P)$[1].

Definition 1. *For a* ***SCI*** *program P, a set of atoms S is a stable model of P iff $S \models \mathrm{IC}(P)$ and $S = \mathrm{StM}(P^S)$.*

Example 1. Consider the program P

$$c \leftarrow a \qquad b \leftarrow \text{not } a \qquad \{a\} \leftarrow \text{not } d \qquad \leftarrow \text{not } a$$

Now $S_1 = \{a, c\}$ is a stable model of P because it satisfies the only integrity constraint (last rule) in P and it is the stratified model of the reduct P^{S_1} consisting of the first two rules and the rule $a \leftarrow \text{not } d$. However, $S_2 = \{b\}$ is not a stable model because it does not satisfy the integrity constraint and $S_3 = \{a, b, c\}$ is not a stable model because it is not the stratified model of the reduct P^{S_3}. In fact, S_1 is the only stable model of P.

Example 2. The choice rules enable very natural encodings without using unstratified negation. We illustrate this with an encoding of the Hamiltonian circuit problem for directed graphs, i.e., the problem of finding a path in a graph visiting each node exactly once and returning to the starting node. We assume that the graph is given using a set of facts of the form $edge(v, u), vtx(v)$ specifying the edges and vertices and a fact $start(w)$ for some arbitrary starting vertex for the circuit. In the encoding below the circuit is represented by the predicate $hc(\cdot, \cdot)$. The first rule introduces for each edge a choice whether to include the edge in

[1] Note that for a stratified normal program P the unique model coincides with the stable model and the well-founded model of P.

the circuit or not and the two other rules on the left require that a vertex can have at most one immediate successor and predecessor in the circuit. The rules on the right state that each vertex needs to be reachable through the circuit from the starting node of the circuit.

$$\begin{array}{ll} \{hc(V,U)\} \leftarrow edge(V,U) & r(V) \leftarrow hc(S,V), start(S) \\ \leftarrow hc(V,U), hc(V,W), U \neq W & r(V) \leftarrow r(U), hc(U,V) \\ \leftarrow hc(U,V), hc(W,V), U \neq W & \leftarrow \text{not } r(V) \end{array}$$

4 Capturing Unstratified Negation

In **SCI** programs unstratified negation is not allowed and an interesting question is whether it can be captured with a suitable translation in terms choices, integrity constraints and stratified negation. In fact, this is possible because a literal not p with (unstratified) negation can be seen as a new atom $\overline{p}$ for which a choice needs to be made whether to include $\overline{p}$ in the model such that a model cannot contain both p and $\overline{p}$ but one of them needs to be included.

Hence, a normal program P can be translated to a **SCI** program tr(P) where all unstratified negations (or in fact all negative literals) in P can be eliminated as follows. For each atom p in P, we introduce a new atom $\overline{p}$ and add the following three rules in the translation:

$$\{\overline{p}\} \leftarrow \qquad \leftarrow p, \overline{p} \qquad \leftarrow \text{not } p, \text{not } \overline{p}$$

Then each negative literal not p in the rules can be replaced $\overline{p}$.

It can be shown that given a normal program P (i) if S is a stable model of P, then $S \cup \{\overline{p} \mid p \in \text{At(P)} - S, \}$ is a stable model of the **SCI** program tr(P) and (ii) if S is a stable model of tr(P), then $S \cap \text{At(P)}$ is a stable model of P, where At(P) is the set of atoms in P.

5 Extending SCI Programs

One of the advantages of the approach is that it is very straightforward and unproblematic to extend the framework with new kinds of constraints and aggregates. The idea is to require that these extensions can appear only in bodies of rules and only in a stratified way, i.e., predicates used in a constraint or aggregate need to be defined on an earlier stratum. Then it is straightforward to the generalize the stratified model to handle novel kinds of constraints, see for instance [13].

Notice that a rule with a constraint in the head can be directly represented with a choice rule and an integrity constraint. For example, a rule such as

$$odd(a_1, ..., a_l) \leftarrow b, \text{not } c$$

stating that an odd number of atoms $a_1, ..., a_l$ should be selected if $b, \text{not } c$ hold, can be encoded using two rules:

$$\{a_1, ..., a_l\} \leftarrow b, \text{not } c \qquad \leftarrow \text{not } odd(a_1, ..., a_l), b, \text{not } c.$$

There are some cases where positive recursion through a constraint has clear semantics and is usable in practical applications, for example, in the case of monotone constraints. For such cases it is straightforward to relax the notion of stratification to allow positive recursion through such constraints and to extend the semantics to handle this case [12].

Acknowledgments. The financial support of the Academy of Finland (project 122399) is gratefully acknowledged.

References

1. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of the 5th International Conference on Logic Programming, pp. 1070–1080. The MIT Press, Cambridge (1988)
2. Niemelä, I., Simons, P.: Efficient implementation of the well-founded and stable model semantics. In: Proceedings of the Joint International Conference and Symposium on Logic Programming, pp. 289–303. The MIT Press, Cambridge (1996)
3. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: The KR system dlv: Progress report, comparisons and benchmarks. In: Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning, pp. 406–417. Morgan Kaufmann Publishers, San Francisco (1998)
4. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. In: Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning (1998); Extended version appeared in Annals of Mathematics and Artificial Intelligence 25(3,4), 241–273 (1999)
5. Marek, W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm: a 25-Year Perspective, pp. 375–398. Springer, Heidelberg (1999)
6. Lifschitz, V.: Answer set planning. In: Proceedings of the 16th International Conference on Logic Programming, pp. 25–37. The MIT Press, Cambridge (1999)
7. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365–385 (1991)
8. Soininen, T., Niemelä, I.: Developing a declarative rule language for applications in product configuration. In: Gupta, G. (ed.) PADL 1999. LNCS, vol. 1551, pp. 305–319. Springer, Heidelberg (1999)
9. Niemelä, I., Simons, P., Soininen, T.: Stable model semantics of weight constraint rules. In: Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning, pp. 317–331. Springer, Heidelberg (1999)
10. Apt, K., Blair, H., Walker, A.: Towards a theory of declarative knowledge. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 89–148. Morgan Kaufmann Publishers, San Francisco (1988)
11. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence 138(1-2), 181–234 (2002)
12. Marek, V., Niemelä, I., Truszczyński, M.: Programs with monotone abstract constraint atoms. Theory and Practice of Logic Programming 8(2), 167–199 (2008)
13. Kemp, D.B., Stuckey, P.J.: Semantics of logic programs with aggregates. In: Proceedings of the 1991 International Symposium on Logic Programming, pp. 387–401. MIT Press, Cambridge (1991)