Nested Weight Constraints in ASP[*]

Stefania Costantini[1] and Andrea Formisano[2]

[1] Università di L'Aquila, Via Vetoio, Loc. Coppito, I-67010 L'Aquila, Italy
stefania.costantini@univaq.it
[2] Università di Perugia, via Vanvitelli, 1, I-06123 Perugia, Italy
formis@dmi.unipg.it

Abstract. Weight constraints are a powerful programming construct that has proved very useful within the Answer Set Programming paradigm. In this paper, we argue that practical Answer Set Programming might take profit from introducing some forms of nested weight constraints. We define such empowered constraints (that we call "Nested Weight Constraints") and discuss their semantics and their complexity.

1 Introduction

Answer Set Programming (ASP for short) [8], has evolved over more than two decades as a paradigm that allows for very elegant solutions to many combinatorial problems: in fact, ASP has been successfully applied to many forms of knowledge representation and commonsense reasoning (cf. among others, [1, 7] and the references therein). The paradigm is based upon describing a problem by a logic program in such a way that its answer sets correspond to the solutions of the considered problem.

The ASP paradigm has become even more powerful by extending ASP programs by means of weight constraints [11, 13]. Intuitively, weight constraints allow one to associate weights to the literals occurring in specific subsets of a (candidate) model. Then, bounds can be imposed on the overall weight of each subset. A model is accepted if all these bounds are satisfied. Cardinality constraints are a special case where all weights are equal to one. Weight constraints have proved to be a very useful programming tool in many applications such as planning and configuration. For instance, in the product configuration domain, we need to express cardinality, cost, and resource constraints, which are very difficult to capture using logic programs without weights.

Weight constraints are nowadays adopted (in some form) by most of the ASP inference engines (usually called "ASP solvers").

All common algorithmic tasks related to programs with weight constraints, such as checking the consistency of a program (i.e., whether a program admits stable models), are intractable [5]. Though, as shown in [12], tractability can be achieved by imposing some restrictions on program structure.

We propose an improved form of constraints that admits nesting of weight constraints. Syntactically, nesting allows one to specify a set of weight constraints within a

[*] Research partially funded by GNCS-2011 and MIUR-PRIN-2008 projects, and grants 2009.010.0336 and 2010.011.0403. We are grateful to the anonymous referees for their helpful advice and suggestions.

"container" weight constraint. In turn, such "contained" constraints may include other constraints, and so on. Semantics is given by requiring that the satisfaction of the internal constraints has to be evaluated with respect to the *context* defined by the containing constraints. Hence, the new construct introduces a form of locality in program rules: two identical weight constraints might be differently evaluated depending on the context in which they occur. We will see that nesting can be introduced without affecting complexity.

We argue that practical ASP programming might take profit from the introduction of nested weight constraints. In particular, our proposal is aimed at improving *elaboration tolerance* where, [10]:

> "A formalism is elaboration tolerant to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances. Representations of information in natural language have good elaboration tolerance when used with human background knowledge. Human-level AI will require representations with much more elaboration tolerance than those used by present AI programs, because human-level AI needs to be able to take new phenomena into account. The simplest kind of elaboration is the addition of new formulas. We'll call these additive elaborations. Next comes changing the values of parameters. Adding new arguments to functions and predicates represents more of a change. However, elaborations not expressible as additions to the object language representation may be treatable as additions at a meta-level expression of the facts ..."

One can say that elaboration tolerance implies the ability to cope with minor changes to input problems without major revisions. The introduction of constructs involving forms of locality, as well as modularity, goes in this direction. In what follows, we will take a sample problem (which is however a representative of a wide class) and we will show that the formalization in ASP benefits from the use of nested weight constraints.

The paper is structured as follows. In Section 2 we recall the notions of weight (and cardinality) constraints. Section 3 introduces the enhancements we intend to propose, for the case of ground programs. In Section 4 we further extend the formalism by introducing conditional literals [11] and the use of variables to denote collections of literals. An example is exploited in Section 5 to illustrate nested weight constraints. The complexity issue is addressed in Section 6. Finally, in Section 7 we conclude.

2 Weight and Cardinality Constraints in ASP

Weight and cardinality constraints were introduced in [11, 13], where their semantics is also presented, as well as their implementation in the context of the ASP solver smodels. Deciding whether a program involving ground weight constraints has an answer set is still NP-complete, and computing an answer set is still FNP-complete. Though the computational complexity remains the same, the modeling power of the extended language is higher, as proved by the wide application of this construct.

In what follows we recall the syntax and semantics of (ground) programs with weight constraints by abstracting away from any particular concrete syntax. We assume known the usual notions of constant, predicate, term, atom, literal, etc. Let us consider as fixed an underlying language and consequently let $\mathcal{B}$ denote the corresponding Herbrand base, namely the set of all ground atoms of the given language.

Atoms have the form $p(t_1, \ldots, t_k)$ where p is a predicate symbol and each t_i is a term. For a literal ℓ, let $\pi(\ell)$ denote the predicate symbol of ℓ (e.g., $\pi(p(t_1, \ldots, t_k) = p)$. For a set of literals S, let $\pi(S) = \{\pi(\ell) | \ell \in S\}$.

A *weight literal* over is a pair (a, j) or $(\neg a, j)$ for $a \in \mathcal{B}$ and $j \in \mathbb{N}$, the weight of the literal and $\neg$ denotes *default negation*.[3] A *weight constraint* is a triple (S, l, u) where S is a set of weight literals and $l \leq u$ are non-negative integers, the lower and upper bound. We will often use the symbol ∞ to denote an arbitrarily large upper bound. (This will be useful in situations in which the upper bound is not specified.) Moreover, as a shorthand notation, we denote by a the weight constraint $(\{(a, 1)\}, 1, 1)$.

For a given constraint $c = (S, l, u)$, we indicate S with $Cl(c)$, l with $l(c)$ and u with $u(c)$. A weight constraint where for every weight literal (a, j) and $(\neg a, j)$ we have $j = 1$ is called a *cardinality constraint*.

A rule r is a pair (h, b) where h (the head) is a weight constraint and b (the body) is a set of weight constraints. We indicate h with $\mathtt{H}(r)$ and b with $\mathtt{B}(r)$.

A (ground) program with weight constraints (for short, PWC) is a set of rules.

Given a weight constraint c and a set of atoms I, we define the *weight* of c in I as $W(c, I) = \sum_{(a,j) \in Cl(c) \wedge a \in I} j + \sum_{(\neg a,j) \in Cl(c) \wedge a \notin I} j$.

A set of atoms I is a model of c (denoted by $I \models c$) iff $l(c) \leq W(c, I) \leq u(c)$. (Notice that the second inequality always holds if $u(c) = \infty$.)

For a set of weight constraints C, $I \models C$ iff $I \models c$ for all $c \in C$. Moreover, I is a model of a rule r (denoted by $I \models r$) iff $I \models \mathtt{H}(r)$ whenever it $I \models c$ holds for each $c \in \mathtt{B}(r)$. For a set of rules R, $I \models R$ iff $I \models r$ for all $r \in R$.

Stable models of a PWC are obtained by means of an extension to the GL-reduct [1] that, instead of removing rules where some negative literals in the body are not modeled in a given set of atoms (candidate stable model) I, it removes rules where the upper bound of some weight constraints in the body are not satisfied. The upper bounds of constraints are removed and the lower bounds are rearranged in order to eliminate negative literals. Each rule r is then replaced by a set of rules each of them having as head one of the positive literals in $\mathtt{H}(r)$ which belongs to I. In this manner, a positive PWC is obtained where the heads of rules are atoms. Finally, I is a stable model if it is the unique minimal model of this resulting program. Following [13], we have:

Definition 1 (PWC Semantics). *Let P be a PWC and let $I \subseteq \mathcal{B}$. The* reduct c^I *of a constraint c w.r.t. I is obtained from c by removing all negative literals, by setting the upper bound to be ∞, and by replacing the lower bound with the value $l' = max\big(0, l(c) - \sum_{(\neg a,j) \in Cl(c) \wedge a \notin I} j\big)$.*

The reduct P^I of the program P w.r.t. I is obtained by first removing each rule whose body contains a constraint c with $W(c, I) > u(c)$. Afterwards, each remaining rule r is replaced by the set of all rules of the form (h, b), for $(h, j) \in Cl(\mathtt{H}(r))$ such that $h \in I$ and $b = \{c^I \ : \ c \in \mathtt{B}(r)\}$.

[3] For the sake of simplicity, in this paper we will deal with non-negative integer weights only. Generalizations involving negative values, as well as real numbers, are possible [13].

The set I is a stable model of P iff it is a model of P^I and there exists no $J \subsetneq I$ such that J is a model of P^I.

3 Nested Weight Constraints

In this section we introduce an extension of ASP where weight constraints can be arbitrarily nested. As we will see, in this extension one can specify within an "external" weight constraint a collection of "internal" weight constraints. These represent conditions on the satisfiability of the outer constraint. Conversely, the external constraint affects the interpretation of internal weight literals and defines the *local context* where these weights literals have to be evaluated.

Definition 2. *A* nested weight constraint *(NWC) is a tuple (S, N, l, u) where*
- *S is a finite set of weight literals,*
- *$l \leq u$ are two non-negative integers (as before u can be ∞),*
- *N is a (possibly empty) finite collection of nested weight constraints*

The definitions of rule and program are given as one expects. We also extend to NWCs the notation introduced earlier and, moreover, for any given NWC $c = (S, N, l, u)$ we denote N with $N(c)$. The *depth* of a given NWC c, denoted by $depth(c)$, is defined as:

$$depth(c) = \begin{cases} 1 & \text{if } N(c) = \emptyset \\ 1 + \max_{c' \in N(c)} depth(c') & \text{otherwise} \end{cases}$$

The depth of a given program P is the maximum value among the depths of its NWCs.

For the purposes of this paper, it is not restrictive to assume the finiteness of the Herbrand universe of the underlying language. We will also consider only programs with finite depth.

The notion of satisfaction for NWCs requires some preliminary definitions. In particular, let $X, Y \subseteq \mathcal{B}$ be two disjoint sets of atoms and $c = (S, N, l, u)$ an NWC. Then, we define the *weight* of the constraint c (w.r.t. X, Y) as follows:

$$W(c, X, Y) = \sum_{(a,j) \in S \wedge a \in X} j + \sum_{(\neg a,j) \in S \wedge a \in Y} j \tag{1}$$

We say that a pair of sets of atoms X, Y satisfies the NWC $c = (S, N, l, u)$, and write $(X, Y) \models c$, if the following two conditions hold:

1. $l \leq W(c, X, Y) \leq u$;
2. for all $c' \in N$ it holds that $(U, V) \models c'$, where

$$\begin{array}{l} U = \{a \mid a \in X \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in X \wedge a \in S\} \\ V = \{a \mid a \in Y \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in Y \wedge \neg a \in S\} \end{array} \tag{2}$$

Given a set of atoms I we say that I models an NWC c and write $I \models c$, iff $(I, \mathcal{B} \setminus I) \models c$. For a set Q of NWCs we write $I \models Q$ iff $I \models c$ for each $c \in Q$.

Notice that, in absence of nesting, namely, for an NWC $c = (S, N, l, u)$ with $N = \emptyset$, we obtain the notion introduced earlier for weight constraints. If $N \neq \emptyset$, the satisfaction of c also depends on the satisfaction of the NWCs in N. In turn, the satisfaction of each $c' \in N$ has to be evaluated *within the context* determined by S. In particular, consider the above definition of satisfaction for an NWC c. The weight of a nested constraint $c' \in N$ is evaluated by considering only those atoms belonging to the subsets $U \subseteq X$ and $V \subseteq Y$ (recall that for the overall constraint c we have $X = I$ and $Y = \mathcal{B} \setminus X$ for a given set of atoms I). In this way, all weight literals in c' having an atom in $\mathcal{B} \setminus (U \cup V)$ are assumed to have null weight. More precisely, in evaluating the weight of c' we ignore the weights of all literals ℓ with $\pi(\ell) \in \pi(S)$ not occurring in S. The same procedure is recursively applied in evaluating the weights of the constraints $c'' \in N(c')$, and so on.

Note that, the above definition of satisfaction implicitly exploits, in (2), a partition of a Herbrand base $\mathcal{B}$. Each block of this partition corresponds to a single predicate symbol and consists of all the atoms having such leading symbol. Observe that the approach can be generalized since any partition of $\mathcal{B}$ can be used.

As before, a set of atoms I is a model of a rule r (denoted by $I \models r$) iff $I \models \mathrm{H}(r)$ whenever $I \models \mathrm{B}(r)$ holds. Given a program P, $I \models P$ iff $I \models r$ for all $r \in P$.

Now, we adapt the notion of reduct to deal with the nesting of constraints. Given an NWC $c = (S, N, l, u)$ and a pair of disjoint sets of atoms X, Y, the *reduct* of c w.r.t. X, Y is so defined (a denotes an atom):

$$c^{(X,Y)} = \big(\{(a,j) | (a,j) \in S\},\ \{d^{(U,V)} | d \in N\},\ \max(0, l - \textstyle\sum_{(\neg a,j) \in S \wedge a \in Y} j),\ \infty\big)$$

where U and V are obtained from X, Y and S as explained earlier (cf., (2) of page 4).

For a set Q of NWCs we denote by $Q^{(X,Y)}$ the set $\{c^{(X,Y)} \mid c \in Q\}$.

Given a program with NWCs, the reduct P^I of P w.r.t. a set of atoms I is so defined:

$$P^I = \Big\{ \big(a, (\mathrm{B}(r))^{(I, \mathcal{B} \setminus I)}\big) \mid r \in P,\ (a,j) \in Cl(\mathrm{H}(r)),\ a \in I, \\ W(c, I, \mathcal{B} \setminus I)) \leq u(c) \text{ for all } c \in \mathrm{B}(r) \Big\} \tag{3}$$

Notice that each rule r in P^I has an head of the form $(\{(a,1)\}, 1, 1)$, for some atom a. Moreover, no negative literal occurs in the body of r. Similarly to the case of ordinary weight constraints, we introduce an operator T_{P^I} defined as follows:

$$T_{P^I}(J) = \{a \mid \exists\, r \in P^I,\ a = \mathrm{H}(r),\ J \models \mathrm{B}(r)\} \tag{4}$$

Proposition 1. *Given a program P^I and two sets of atoms J_1 and J_2, if $J_1 \subseteq J_2$ then $T_{P^I}(J_1) \subseteq T_{P^I}(J_2)$.*

Proof. (Sketch). Let $a \in T_{P^I}(J_1)$. There exists a rule $r \in P^I$ of the form $(a, \mathrm{B}(r)$ such that $J_1 \models \mathrm{B}(r)$. Hence, for each NWC $c = (S, N, l, \infty) \in \mathrm{B}(r)$, we have that $l \leq W(c, J_1, \mathcal{B} \setminus J_1)$ and for each $c' \in N$, $(U_1, V_1) \models c'$, with $U_1 = \{a \mid a \in J_1 \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in J_1 \wedge a \in S\}$ and $V_1 = \{a \mid a \in (\mathcal{B} \setminus J_1) \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in (\mathcal{B} \setminus J_1) \wedge \neg a \in S\}$. If $J_1 \subseteq J_2$, then $l \leq W(c, J_2, \mathcal{B} \setminus J_2)$ plainly follows because there are no negative literals in S. Observe now that $U_2 = \{a \mid a \in J_2 \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in J_2 \wedge a \in S\} \supseteq U_1$ and $V_2 = \{a \mid a \in (\mathcal{B} \setminus J_2) \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in (\mathcal{B} \setminus J_2) \wedge \neg a \in S\} \subseteq V_1$. The fact that $(U_2, V_2) \models c'$ holds for each $c' \in N$ can be shown by induction on the maximum depth of nesting in N. In particular, in absence of nesting (namely, if $N = \emptyset$) the result is immediate. The proof of inductive step relies on the fact that no negative literal occurs in N. This allows us to conclude that $J_2 \models \mathrm{B}(r)$, hence $a \in T_{P^I}(J_2)$. □

Given a program P and a set of atoms I, by the previous result, the operator T_{P^I} is monotone and has an unique least fix-point which is obtainable by iterated applications of T_{P^I} starting from the empty set. Let us denote such a fix-point by $T_{P^I}\!\uparrow$.

We have the following notion of stable model for programs with NWCs.

Definition 3. *Given a program with NWCs P, a set I of atoms is a stable model for P iff $I \models P$ and $I = T_{P^I}\!\uparrow$*

4 Conditional literals and the use of variables

Similarly to the approach of [11], in this section we adapt the treatment described in Section 3 to deal with *conditional literals*.

A conditional literal has the form $\ell{:}s$ where ℓ is a weight literal and s is a (possibly empty) set of atoms. The intended meaning is that the conjunction of the atoms in s constitutes a precondition for the satisfiability of ℓ. (Empty conditions, i.e., $s = \emptyset$, are trivially satisfied. Conditional literals of the form $\ell{:}\emptyset$ correspond to weight literals as introduced in Section 3. We will often write ℓ in place of $\ell{:}\emptyset$.)

All the notions introduced in Section 3 can be easily adapted to deal with conditional literals. In what follows we outline the main steps of such an adaptation. For the sake of readability, in doing this we will maintain the same notational conventions. The next definition is the counterpart of Def. 2:

Definition 4. *A* nested weight constraint *(NWC) is a tuple (S, N, l, u) where*

- *S is a finite set of conditional literals,*
- *$l \leq u$ are two non-negative integers (u can be ∞),*
- *N is a (possibly empty) finite collection of nested weight constraints*

Rules and programs are defined as one expects.

The notion of satisfaction for NWCs is slightly complicated w.r.t. the one in Section 3. This is so because the initial set of atoms (i.e., the candidate model, Z in the following) has to be considered in evaluating the preconditions of all conditional literals. Let $Z, X, Y \subseteq \mathcal{B}$ be sets of atoms such that $X \subseteq Z$ and $Y \subseteq (Z \setminus \mathcal{B})$. We define the *weight* of the NWC $c = (S, N, l, u)$, w.r.t. Z, X, Y, as follows:

$$W(c, Z, X, Y) = \sum_{(a,j):s \in S \wedge a \in X \wedge s \subseteq Z} j + \sum_{(\neg a,j):s \in S \wedge a \in Y \wedge s \subseteq Z} j$$

We say that a the sets of atoms Z, X, Y satisfy the NWC $c = (S, N, l, u)$, and write $(Z, X, Y) \models c$, if the following two conditions hold:

1. $l \leq W(c, Z, X, Y) \leq u$;

2. for all $c' \in N$ it holds that $(Z, U, V) \models c'$, where

$$\begin{array}{l} U = \{a \mid a \in X \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in X \wedge (a, j){:}s \in S \wedge s \subseteq Z\} \\ V = \{a \mid a \in Y \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in Y \wedge (\neg a, j){:}s \in S \wedge s \subseteq Z\} \end{array} \tag{5}$$

where, with abuse of notation, we denote by $\pi(S)$ the set $\{\pi(\ell) | \ell{:}s \in S\}$.

Given a set I of atoms, we say that I models an NWC c and write $I \models c$, iff $(I, I, \mathcal{B} \setminus I) \models c$. For a set Q of NWCs we write $I \models Q$ iff $I \models c$ for each $c \in Q$.

Analogously to what seen in Section 3, the satisfaction of each $c' \in N$ has to be evaluated within the context determined by S. The same recursive scheme outlined in Section 3 applies here in evaluating the satisfiability of NWCs.

As before, a set I of atoms is a model of a rule r (denoted by $I \models r$) iff $I \models \mathrm{H}(r)$ whenever $I \models \mathrm{B}(r)$ holds. Given a program P, $I \models P$ iff $I \models r$ for all $r \in P$.

In defining the notion of reduct we have to take into account all preconditions of conditional literals. Let Z, X, Y be sets of atoms. The *reduct* of an NWC $c = (S, N, l, u)$, w.r.t. Z, X, Y, is so defined:

$$\begin{aligned} c^{(Z,X,Y)} = \big(\{(a, j){:}s \mid (a, j){:}s \in S\},\ \{d^{(Z,U,V)} \mid d \in N\}, \\ \max(0, l - \textstyle\sum_{(\neg a, j):s \in S \wedge a \in Y \wedge s \subseteq Z} j),\ \infty\big) \end{aligned}$$

where U and V are obtained from X, Y, Z, and S as explained earlier (cf., (5)).

In analogy with the cases of plain [11, Def. 2] and nested (Section 3) weight constraint, given a program P and a set of atoms I, the reduct P^I is so defined:

$$\begin{aligned} P^I = \Big\{\big(a, \mathrm{B}(I, r, s)\big) \mid r \in P,\ (a, j){:}s \in Cl(\mathrm{H}(r)),\ \{a\} \cup s \subseteq I \\ W(c, I, I, \mathcal{B} \setminus I)) \leq u(c) \text{ for all } c \in \mathrm{B}(r)\Big\} \end{aligned}$$

where $\mathrm{B}(I, r, s)$ denotes the set

$$\mathrm{B}(I, r, s) = \Big\{c^{(I,I,\mathcal{B}\setminus I)} \,|\, c \in \mathrm{B}(r)\Big\} \cup \Big\{(\{(b, 1)\}, \emptyset, 1, \infty) \,|\, b \in s\Big\} \cup \tag{6}$$

$$\bigcup_{c \in \mathrm{B}(r)} \Big\{(\{(a, 1)\}, \emptyset, 1, \infty) \,|\, (\neg b, j){:}r \in Cl(c),\ a \in r \text{ s.t. } b \notin I,\ r \subseteq I\Big\} \tag{7}$$

The definition of the reduct P^I of a program is slightly more involute than the homologous definition given in Section 3. This is so because each negative literal $(\neg b, j){:}r$, with $b \notin I$, occurring in an NWC of the body of a rule r, will give its contribution to the weight of the NWC only if the precondition r holds in I. This requirement must be reflected in the program P^I by adding the set shown in (7). In this manner the body of the resulting rule will be falsified whenever any of such preconditions is false.

Now, we can define an operator T_{P^I} exactly as done in (4). Such an operator is monotone (an analogous to Proposition 1 can be stated) and has an unique least fixpoint. Def. 3 can be properly generalized to the case of NWCs with conditional literals:

Definition 5. *Given a program P with NWCs involving conditional literals, a set of atoms I is a stable model for P iff $I \models P$ and $I = T_{P^I}\!\uparrow$*

Variables can be exploited to denote collections of weight literals. This is done by admitting non-ground conditional weight literals $\ell{:}\varphi$ where ℓ has, in general,[4] the form $(p(X_1,\ldots,X_n),j)$ (or the form $(\neg p(X_1,\ldots,X_n),j)$) and each X_i is a variable (for $n \geq 0$). Similarly, φ is a set of not necessarily ground atoms. Let $var(\varphi) = \{X_1,\ldots,X_n,Y_1,\ldots,Y_m\}$ be the set of all the variables occurring in φ (for $n,m \geq 0$). The variables $X_1,\ldots,X_n$ are said to be *local* to the literal. The variables $Y_1,\ldots,Y_m$ are said to be *global*.

Non-ground NWCs, rules, and programs, are then defined as one expects.

Given a program, it is not restrictive to impose that each local variable occurs in a single conditional literal. We will make this assumption in what follows.

Considering a rule with non-ground NWC, all its global variables should be intended as being universally quantified. The instantiation of a rule is defined as the set of the ground rules each of them obtainable, first, by grounding all global variables (i.e., by uniformly substituting them by ground terms from the Herbrand universe of the underlying language) and then by replacing each non-ground conditional weight literal with the collection of all its ground instances that are obtainable by grounding the local variables. Notice that, in a literal such as (a,j) (or $(\neg a,j)$), we admit j to be a (global) variable. In this manner, each instantiation of this literal may have a different weight, determined through the grounding process. Analogously, the lower and upper bounds of an NWC can be expressed using variables, provided that the grounding process suitably instantiates them to non-negative integers. (In what follows we will adopt this option.)

The instantiation of a program P is defined as the set of all instantiations of rules in P. Stable models of programs involving variables are easily defined as follows:

Definition 6. *Given a (non-ground) program P, a set of ground atoms I is a stable model for P iff it is a stable model for the instantiation of P.*

Concrete syntax. In the following section, we describe a concrete encoding of a running example. In doing this we resort to the smodels-like notation, for programs, rules, and literals. In particular, we indicate by p an NWC of the form $(\{p\},\emptyset,1,1)$. Also, we denote a weight constraint

$$(\{(a_1,w_{a_1}),\ldots,(a_1,w_{a_1}),\ (\neg b_1,w_{b_1}),\ldots,\ (\neg b_m,w_{b_m})\},\ l,\ u)$$

as $l[a_1 = w_{a_1},\ldots,a_n = w_{a_n}, \textit{not}\ b_1 = w_{b_1},\ldots,\textit{not}\ b_m = w_{b_m}]u$ and, similarly, an NWC $(\{(a_1,w_{a_1}),\ldots,(a_1,w_{a_1}),\ (\neg b_1,w_{b_1}),\ldots,\ (\neg b_m,w_{b_m})\},\ \{W_1,\ldots,W_k\},\ l,\ u)$ as $l[a_1 = w_{a_1},\ldots,a_n = w_{a_n}, \textit{not}\ b_1 = w_{b_1},\ldots,\textit{not}\ b_m = w_{b_m} \mid W_1,\ldots,W_k]u$. In both cases, we omit u whenever $u = \infty$.
For the special case of cardinality constraints, i.e., when $w_i = 1$ for all i, we adopt the shorthand notation $l\,\{a_1,\ldots,a_n,\textit{not}\ b_1,\ldots,\textit{not}\ b_m\}\,u$. Conditional literals of the form $(a,j){:}\{b_1,\ldots,b_k\}$ will be denoted as $a : b_1,\ldots,b_k = j$. Moreover, in expressing weights and bounds of constraints we might use variables (intended to be suitably instantiated by the grounding phase). Finally, we denote a program rule as W_0 :- $W_1,\ldots,W_n$, where the W_is are (nested) weight constraints (for $n \geq 0$).

[4] Note that, in concrete encodings, constants are admissible in place of (some of) the X_is. For simplicity, and without loss of generality, we assume that each X_i is a variable.

5 A Case-study

To motivate the introduction of NWC into ASP, we resort to a case-study. Our running example is freely inspired by the Italian Computer Science undergraduate Program, that we shortly describe here in its basic features. Then, we provide an encoding using NWCs.

In order to get a bachelor degree in Computer Science, an Italian student is required to obtain 180 credits. Most of them must be obtained by attending courses and passing the corresponding exams. The remaining ones can be obtained by means of internships and a short thesis. There is a certain flexibility, so usually the number of credits that should be obtained from courses is allowed to vary within a range (say between 153 and 171, in the following encoding; actual ranges vary among different Universities and tracks). There are different possible choices for the courses to attend, so students are required to present what is called a "plan of studies", that must be approved by a Committee. Some courses must be taken at a certain year, for others there is some flexibility. For simplicity, we assume that the latter can be taken at any year and we neglect constraints related to the order in which certain courses should be taken.

Basically, the above (as described up to now) might be summarized by the following rule that characterizes possible plans of studies. (The atom `in_ps(c,j)` means that the course `c` is inserted into the plan of studies, at year `j`.)

```
 Min [in_ps(X,Y):course(X,W),course_year(X,Y)} = W ] Max  :-
                              credits_bounds(Min,Max).
```

This knowledge base describes a possible problem instance:

```
year(1..3).   credits_bounds(153,171).
course_desc(programming, comp_science, 12).
course_desc(computer_architectures, comp_science, 6).
course_desc(databases, comp_science, 12).
course_desc(algorithms, comp_science, 12).
course_desc(theoretical_cs, comp_science, 6).
  ...
course_desc(calculus, mathematics, 6).
course_desc(optimization, mathematics, 6).

course(Course,Creds) :- course_desc(Course,Area,Creds).
```

where each fact `course_desc(c,a,n)` specifies that the course `c` belongs to the area `a` (see below) and corresponds to an amount of `n` credits. Moreover, we might assume the presence of facts of the form `course_year(c,y)` specifying, for each year `y`, the admissible courses `c` for that year.

Clearly, this simple encoding does not model all aspects of the problem at hand. For instance, an aspect which is not represented is that a plan of study cannot include the same course several times. This can be imposed by adding this NWC (actually a cardinality constraint):

```
  0 {in_ps(X,Y):in_ps(X,Y1),neq(Y,Y1)} 0.
```

In our case-study, it is always the case that some mandatory courses must be situated at a certain course year. To model this requirement we add this NWC to the initial rule:

```
  0 {in_ps(X,Y):mandatory(X,Y1),neq(Y,Y1)} 0.
```

For courses that must be included in the solution, but can be situated at any year, we add this extra rule to the encoding:

```
  1{in_ps(C,Y):year(Y)}1 :- mandatory_course(C).
```

The specific instance might specify, for example, this piece of knowledge:

```
mandatory(programming, 1).  mandatory(computer_architectures, 1).
mandatory(algorithms, 2).   mandatory(theoretical_cs, 3).
mandatory_course(databases).
```

Finally, to avoid a student giving too many exams, there is a statement that enforces at least a minimum number of courses of the first two years to weigh 12 credits each. Also, courses are allowed to belong to certain scientific areas, namely Computer Science, Mathematics, Physics, and other different though related topics (within a list). However, there are directions stating that every subject should contribute to the plan of studies for a quota ranging between a minimum and a maximum number of credits.

The next NWCs specify both the number of the 12 credit courses in the first years, and range of credits that can be allowed to the different areas. Here, the constants `comp_science`, `mathematics`, ..., identify (through the facts `course_desc` listed earlier), the area of each course.

```
Min12 {in_ps(X,Y):course(X,W),leq(Y,2),eq(W,12)}
L1 [in_ps(X,Y):course_desc(X,comp_science,W),course_year(X,Y)=W] U1
 ...
Ln [in_ps(X,Y):course_desc(X,mathematics,W),course_year(X,Y)=W] Un
```

where the variables `Min12`, `L1`, `U1`, ..., `Ln`, and `Un`, (to be instantiated through atoms in the rule body) express the bounds on the minimum number of courses worth 12 credits in the first two years, and the minimum/maximum amounts of credits in the different areas.

Summing up, the encoding of our sample problem is as follows (to be joined with a specific instance specifying, together with the pieces of knowledge seen earlier, the predicate `area_bounds`):

```
Min [ in_ps(X,Y):course(X,W),course_year(X,Y)}=W  |
  0 {in_ps(X,Y):in_ps(X,Y1),neq(Y,Y1)} 0,
  0 {in_ps(X,Y):mandatory(X,Y1),neq(Y,Y1)} 0,
  Min12 {in_ps(X,Y):course(X,W),leq(Y,2),eq(W,12)},
  L1 [in_ps(X,Y):course_desc(X,comp_science,W),course_year(X,Y)=W] U1,
      ...
  Ln [in_ps(X,Y):course_desc(X,mathematics,W),course_year(X,Y)=W] Un
] Max  :-  credits_bounds(Min,Max), min_12(Min12),
           area_bounds(comp_science,L1,U1),
                   ...
           area_bounds(mathematics,Ln,Un).

1{in_ps(C,Y):year(Y)}1 :- mandatory_course(C).
```

We hope at this point to have convinced the reader that NWC can easily cope with aspects that can be relevant in a number of applications. Our case-study, in fact, is a simple example of a scheduling problem, where this kind of problems are an important realm of application of ASP. We believe therefore that many kinds of ASP applications might profit from programming constructs that allow for some degree of nesting. In other words, we deem it appropriate to introduce some kind of *contextual* constructs.

6 On the Complexity of Nested Weight Constraints

In [13] it is proved that introducing weight constraints does not affect the complexity of ASP. That is, for instance, the complexity of the problem of checking whether a program has a stable model does not depend on the presence of weight constraints. Here, we are more generally concerned with checking whether a program with NWCs admits stable models.

In what follows we address the complexity issue for NWC programs by focusing on the particular case of ground programs containing NWCs of bounded depth, as defined in Section 3.

Let k-NWC be the class of programs with depth not greater then k, for $k \geq 0$. For $k = 2$ we have the following proposition.

Proposition 2. *Deciding whether a ground* 2*-NWC program admits stable models is NP-complete.*

Proof. (Sketch) The problem of deciding whether a ground 2-NWC program admits a stable model is NP-hard. This follows from the NP-completeness of ASP with weight constraints in absence of nesting [13]. As regards inclusion in NP, this can be verified by showing that, given a set M of atoms, it can be checked in polynomial time whether M is a stable model of P. To do this we have to show that: (a) given a rule $r \in P$, checking if $M \models r$ takes polynomial time; (b) the reduct P_M of the program P has polynomial size w.r.t. the size of P; (c) $T_{P^M}\!\uparrow$ can be computed in polynomial time.

As regards (a), observe that checking whether $M \models c$ for an NWC c involves the evaluation of the weight of c (cf., (1)) and the computation of the sets U, V (cf., (2)). Both the computations can be completed in polynomial time (recall that $depth(c) \leq 2$). Concerning (b), for each rule r in P a linear number of rules is introduced in P_M (cf., (3)). Moreover, for each NWC c occurring in r the computation of the reduct of c takes polynomial time. Finally, (c) can be shown by observing that the computation of the set $I_2 = T_{P^M}(I_1)$, for a given I_1, can proceed by processing, one-by-one, the unsatisfied rules whose head is not in I_1, and checking the satisfaction of their bodies. By (b) we conclude that $T_{P^M}\!\uparrow$ can be computed in polynomial time. □

The previous result generalizes to the case of k-NWC programs, for any fixed k.

From this result, it follows that NWCs might be rephrased in plain ASP. As shown in [13, 6], for weight constraints this can be done at the expense of introducing a (polynomial, but not insignificant) number of new atoms and rules. Moreover, except for cardinality constraints, the translation is quite involved. Therefore, it turns out that weight constraints are a quite substantial programming construct, rather than simple syntactic sugar. This is of course true also for NWCs. Notice that, how to represent NWCs in plain ASP is far from easy to understand. Outlining a translation into plain ASP and evaluating the necessary number of additional atoms and rules is a subject of future work.

7 Concluding Remarks

In this paper, we have introduced an extension to the weight constraint construct, widely used in ASP practical programming. We have illustrated by means of a significant example the potential usefulness of the extension. We have formally defined the extension involving arbitrary nesting of weight constraints and provided a semantics for the enhanced framework. In the case when the depth of nesting is bounded, we proved that the new construct does not affect the complexity of ASP.

Much remains to be done. First of all, the complexity issue for NWC programs has not been completely investigated in the general case (when no bound on the nesting depth is assumed). Moreover, the proposed construct has not been implemented yet and no translation in plain ASP has been designed (this, by Proposition 2, at least for the bounded-depth case, should be achievable). When an implementation will be available, practical use will help us explore the feasibility of further extensions and generalizations. Also, we intend to explore the enrichment of weight constraints by means of complex preferences. In particular, the present work can be easily integrated with the approach to preference handling devised in [2, 3] and extended to weight constraints in [4]. In the resulting setting, referring to the above example one might enrich the formulation with student's preferences, stating for instance with kind of courses are preferred and in which conditions.

We will have to explore both the usefulness in practical applications of nested-constraints, as well as their feasibility in cases where both negative weights and circular definitions are admitted.

From the formal point of view, we intend to extend the method of [6] so as to be able to extend the concept of *strong equivalence* to ASP programs with NWC. Strong equivalence [9] in fact, as widely recognized, provides an important conceptual and practical tool for program simplification, transformation and optimization. In the case of NWC programs, the form of locality implicitly present in NWCs might have interesting consequences. A further issue for future research regards the relation between NWC and (nested) aggregates.

References

[1] C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[2] S. Costantini and A. Formisano. Conditional preferences in P-RASP. In *Proceedings of LANMR'08*, 2008.

[3] S. Costantini and A. Formisano. Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of of Algorithms in Cognition, Informatics and Logic*, 64(1), 2009.

[4] S. Costantini and A. Formisano. Weight constraints with preferences in ASP. In *Proc. of LPNMR'11*, LNCS. Springer, 2011.

[5] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[6] P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. *TPLP*, 5:45–74, 2005.

[7] M. Gelfond. Answer sets. In *Handbook of Knowledge Representation*. Elsevier, 2007.

[8] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proc. of ICLP/SLP'88*. The MIT Press, 1988.

[9] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM TOCL*, 2:526–541, 2001.

[10] J. McCarthy. Elaboration tolerance. In *Proc. of Commonsense'98*, 1998.

[11] I. Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. In *Proc. of LPNMR'99*, number 1730 in LNCS, pages 317–331. Springer, 1999.

[12] R. Pichler, S. Rümmele, S. Szeider, and S. Woltran. Tractable answer-set programming with weight constraints: Bounded treewidth is not enough. In *Proc. of KR'10*. AAAI Press, 2010.

[13] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.