Modeling preferences on resource consumption and production in ASP

Stefania Costantini[*] and Andrea Formisano[**]

Abstract. Recently we have proposed RASP, an extension of Answer Set Programming that permits declarative specification and reasoning on consumption and production of resources. In this paper, we extend this framework to allow the declarative specification of preferences among alternative use of different resources. We provide syntax and semantics for the resulting formalism, where preferences expressed on resource usage induce a preference order on answer sets.
Key words: Answer set programming, quantitative reasoning, preferences, non-monotonic logic programming, language extensions.

Introduction

As it is well-known, Answer Set Programming (ASP) is a form of logic programming based on the answer set semantics [13], where solutions to a given problem are represented in terms of selected models (answer sets) of the corresponding logic program [19]. ASP is nowadays applied in many areas, including problem solving, configuration, information integration, security analysis, agent systems, semantic web, and planning (see, among others, [5, 26, 2]).

However, the possibility was lacking of performing some kind of quantitative reasoning which is instead possible in non-classical logics such as, for instance, Linear Logics [15] and Description Logics [3]. In recent work [9], an extension of ASP, called RASP (standing for Resourced ASP), has been proposed so to support declarative reasoning on consumption and production of resources.

In this paper, we go further in this extension, by adding declarative *preferences* to the specification of production/consumption processes. In particular, in realizing the same process (modeled through the firing of rules), one may prefer to produce and/or consume certain resources rather than another ones. This extension can be particularly useful in configuration applications where one can, for instance, prefer to save money while spending more time or vice versa or may prefer to employ a certain amount of cheap components rather than a little amount of expensive parts.

Let us briefly recall syntax and intended semantics of RASP programs through a simple example. We will then modify such example to informally introduce preferences. A RASP program is composed of r-facts and r-rules, where numbers associated with the heads of r-facts and rules indicate which amount

[*] Università di L'Aquila. Email: `stefcost@di.univaq.it`
[**] Università di Perugia. Email: `formis@dipmat.unipg.it`

of a certain resource is respectively: available, in case of r-facts; produced, in case of r-rules, where production can take place if the body holds (this implies that required resources are either available or produced). Available or produced resources can in turn be consumed: Quantities are associated to atoms occurring in the bodies of r-rules as well. The example concerns the preparation of desserts:

$$\begin{array}{lll} \mathit{cake}{:}1 \leftarrow \mathit{egg}{:}3,\ \mathit{flour}{:}4,\ \mathit{sugar}{:}3. & \mathit{flour}{:}8. & \mathit{egg}{:}3. \\ \mathit{ice\_cream}{:}1 \leftarrow \mathit{egg}{:}2,\ \mathit{sugar}{:}2,\ \mathit{milk}{:}2. & \mathit{sugar}{:}6. & \mathit{milk}{:}3. \end{array}$$

Atoms of the form $q{:}a$ are called *amount-atoms*, where the *amount-symbol* a denotes the quantity of that resource which is either produced (if the amount-atom is in the head of a rule), or consumed (if it is in the body), or available (if it is a fact), respectively. We may notice that different solutions stem, in this case, from the fact that, with the available ingredients, one may prepare either a cake or an ice-cream, but not both.

In general, usual ASP literals (possibly involving negation-as-failure) may occur in rules. Semantics of a RASP program is determined by interpreting usual literals as in ASP (i.e., by exploiting stable model semantics) and amount-atoms in an auxiliary algebraic structure (that supports operations and comparisons). For instance, we could modify the above rule by requiring that ice-cream can be made only if there is a fridge and there is someone who is a good cook:

$$\begin{array}{l} \mathit{ice\_cream}{:}1 \leftarrow \mathit{egg}{:}2,\ \mathit{sugar}{:}2,\ \mathit{milk}{:}2,\ \mathit{fridge},\ \mathit{a\_cook\_is\_here}. \\ \mathit{a\_cook\_is\_here} \leftarrow \mathit{is\_here}(\mathit{remy}),\ \mathit{is\_here}(\mathit{linguini}). \\ \mathit{is\_here}(\mathit{remy}). \qquad \mathit{is\_here}(\mathit{linguini}). \end{array}$$

Intuitively, the first rule of this program is applicable only in correspondence of models that satisfy the literals *fridge* and *a_cook_is_here*.

RASP already offers some constructs to express limited forms of preferences on resource consumption/production. For instance, a number of budget policies are exploitable to control rule firings and, consequently, to influence what resources to produce and in which quantity, and whether the firing of r-rules is optional or mandatory. The various policies can be combined in a mixed strategy by choosing one of them for each single rule of the program. These features, among others, are fully dealt with in [9]. In what follows we develop a general and more expressive form of preference on resource usage.

Recall the initial example and suppose you might prepare a cake either with corn flour or with potato flour. The following rules express the two possibilities, but do not say which one you would prefer, assuming both of them to be feasible:

$$\begin{array}{l} \mathit{cake}{:}1 \leftarrow \mathit{egg}{:}3, \mathit{flour}{:}4, \mathit{sugar}{:}3. \\ \mathit{cake}{:}1 \leftarrow \mathit{egg}{:}3, \mathit{potato\_flour}{:}3, \mathit{sugar}{:}3. \end{array}$$

We propose in this paper P-RASP (RASP with preferences), to allow one to explicitly state which resource (s)he would prefer to use, e.g., the formulation

$$\mathit{cake}{:}1 \leftarrow \mathit{potato\_flour}{:}3 {>} \mathit{flour}{:}4,\ \mathit{egg}{:}3,\ \mathit{sugar}{:}3.$$

indicates that consuming potato flour is preferred onto consuming corn flour. Or also, if the recipe includes milk, one might prefer to use skim milk if available:

$$\mathit{cake}{:}1 \leftarrow \mathit{potato\_flour}{:}3 {>} \mathit{flour}{:}4,\ \mathit{skim\_milk}{:}2 {>} \mathit{whole\_milk}{:}2,\ \mathit{egg}{:}3,\ \mathit{sugar}{:}3.$$

In this reformulation, we have two *preference lists* (or for short *p-lists*). Actually, p-lists may involve any number of amount-atoms. The intuitive reading is that leftmost elements of a p-list have higher priority. P-lists may occur in the head of r-rules, as shown in the example below, where one prefers to employ available ingredients to make an ice-cream instead of two cups of zabaglione:

$$\mathit{ice\_cream}{:}1 > \mathit{zabaglione}{:}2 \leftarrow \mathit{skim\_milk}{:}2 > \mathit{whole\_milk}{:}2,\ \mathit{egg}{:}2,\ \mathit{sugar}{:}3.$$

The introduction of p-lists requires a concept of *preferred answer set.* In case several p-lists occur either in one rule or in different r-rules, it is necessary to establish which answer set better satisfies the preferences. Intuitively, if we choose to consider as "better" the answer sets which satisfy the higher number of leftmost elements, in the last example we would have: Producing an ice-cream with skim milk is the best solution. Producing: (a) an ice-cream with whole milk or (b) two zabagliones with skim milk would be equally good (but worse than the previous solution) as each of them employs the leftmost element of one p-list. Producing two zabagliones with whole milk is the less preferred solution. Clearly, one has to choose the best *possible* solution, given the available resources. One might choose other strategies, e.g. one might give higher priorities to p-lists in rule heads, where consequently solution (a) above would become better than (b). One may also imagine to introduce a choice among different strategies to be employed in different contexts. Finally, preferences may be conditional. One may for instance prefer skim milk when on a diet, i.e. the last rule may become:

$$\begin{aligned}\mathit{ice\_cream}{:}1 > \mathit{zabaglione}{:}2 \leftarrow {} & (\mathit{skim\_milk}{:}2 > \mathit{whole\_milk}{:}2\ \mathit{IF}\ \mathit{diet}),\\ & \mathit{egg}{:}2,\ \mathit{sugar}{:}3.\end{aligned}$$

If *diet* does not hold, then the preference list reduces to a disjunction, i.e. either skim milk or whole milk can be indistinctly used. This generalizes to several p-lists, like in the example below:

$$\begin{aligned}&(\mathit{ice\_cream}{:}1 > \mathit{zabaglione}{:}2\ \mathit{IF}\ \mathit{summer}) \leftarrow\\ &\qquad (\mathit{skim\_milk}{:}2 > \mathit{whole\_milk}{:}2\ \mathit{IF}\ \mathit{diet}),\ \mathit{egg}{:}2,\ \mathit{sugar}{:}3.\end{aligned}$$

In this paper, we propose a definition of P-RASP and its semantics, we briefly addresses the complexity and the implementation issues, and outline a comparison with related work. We notice that P-RASP has been fully implemented but for the sake of space, a description of the implementation is out of the scope of this paper. The interested reader can refer to [9, 10].

1 From RASP to P-RASP: Syntax

In order to formally introduce syntax and semantics of P-RASP, we need to briefly summarize the basic notions about RASP syntax as presented in [9].

To accommodate the new language expressions that involve resources and their quantities, the underlying language of RASP is partitioned into *P*rogram symbols and *R*esource symbols. Precisely, let $\langle \Pi, \mathcal{C}, \mathcal{V} \rangle$ be an alphabet where $\Pi = \Pi_P \cup \Pi_R$ is a set of predicate symbols such that $\Pi_P \cap \Pi_R = \emptyset$, $\mathcal{C} = \mathcal{C}_P \cup \mathcal{C}_R$

is a set of symbols of constant such that $\mathcal{C}_P \cap \mathcal{C}_R = \emptyset$, and $\mathcal{V}$ is a set of variable symbols. The elements of $\mathcal{C}_R$ are said *amount-symbols*, while the elements of Π_R are said *resource-predicates*. A *program-term* is either a variable or a constant symbol. An *amount-term* is either a variable or an amount-symbol.

Amount-atoms are introduced in addition to plain ASP atoms, here called program atoms. Let $\mathcal{A}(X, Y)$ denote the collection of all expressions of the form $p(t_1, \ldots, t_n)$, with $p \in X$ and $\{t_1, \ldots, t_n\} \subseteq Y$. Then, a *program atom* is an element of $\mathcal{A}(\Pi_P, \mathcal{C} \cup \mathcal{V})$. An *amount-atom* is an expression of the form $q{:}a$ where $q \in \Pi_R \cup \mathcal{A}(\Pi_R, \mathcal{C} \cup \mathcal{V})$ and a is an amount-term. Let $\tau_R = \Pi_R \cup \mathcal{A}(\Pi_R, \mathcal{C})$. We call elements of τ_R *resource-symbols*. E.g., in the two expressions $p{:}3$ and $q(2){:}b$, p and $q(2)$ are resource-symbols (with $p, q \in \Pi_R$ and $2 \in \mathcal{C}$) aimed at defining two resources which are available in quantity 3 and b, resp., (with $3, b \in \mathcal{C}_R$ amount-symbols). Expressions such as $p(X){:}V$ where V, X are variable symbols are also allowed, as resources can be either directly specified as constants or derived. Notice that the set of variables is not partitioned, as the same variable may occur both as a program term and as an amount-term. *Ground* amount- or program-atoms contain no variables. As usual, a *program-literal* L is a program-atom A or the negation *not* A of a program-atom (intended as negation-as-failure).[1] If $L = A$ (resp., $L = \textit{not}\, A$) then $\overline{L}$ denotes *not* A (resp., A).

Definition 1. *A* resource-literal *(r-literal) is either a program-literal or an amount-atom.*

Therefore, we do not allow negation of amount-atoms. (See [9] for a discussion about this point.) Finally, we distinguish between plain rules and rules that involve amount-atoms. In particular, a *program-rule* is defined as a regular ASP rule, including the case of ASP *constraints*, i.e., rules with empty head. Beside program-rules we introduce resource-rules which differ from program rules in that they may contain amount-atoms.[2]

Definition 2. *A* resource-proper-rule *has the form*

$$H \leftarrow B_1, \ldots, B_k \tag{1}$$

where $B_1, \ldots, B_k$, $k > 0$ are r-literals and H is either a program-atom or a (non-empty) list of amount-atoms.

Resource-facts are intended to model the fixed amount of resources that are available "from the beginning". They are defined as follows:

Definition 3. *A* resource-fact *(*r-fact*, for short) has the form $H \leftarrow .$, where H is an amount-atom $q{:}a$ and a is an amount-symbol.*

[1] We will only deal with negation-as-failure. Though, classical negation of program literals could be used in (P-)RASP programs and treated as usually done in ASP.

[2] A more general definition of r-rule is given in [9] that offers the possibility of expressing bounds on the (finite) number of times each r-rule is fired. Here, for simplicity, we restrict ourselves to the simpler case in which each r-rule may be fired at most once. The treatment of the general case does not offer significant differences.

According to the definition, the amount of an initially available resource has to be explicitly stated. Thus, in an r-fact the amount-term a cannot be a variable.

Definition 4. *A* resource-rule *(*r-rule*, for short) can be either a resource-proper-rule or a resource-fact. A* RASP-rule *(*rule*, for short) γ is either a program-rule or a resource-rule. An* r-program *is a finite set of RASP-rules.*

Remark 1. Notice that we admit several amount-atoms in the head of a resource-proper-rule, while the case in which a rule γ has an empty head is admitted only if γ is a program-rule (i.e., γ is an ASP *constraint*).

The list of amount-atoms composing the head of an r-rule has to be intended conjunctively, i.e., as a collection of those resources that are all contemporaneously produced by firing the rule.

P-RASP programs are obtained from RASP programs by introducing alternatives in using resources expressed by preference lists:

Definition 5. *A* preference-list *of amount-atoms (*p-list*, for short) is a writing of the form $q_1{:}a_1{>}\cdots{>}q_h{:}a_h$, where $h \geqslant 2$ and $q_1, \ldots, q_h$ are pairwise distinct resource-symbols. We say that the amount-atom $q_i{:}a_i$ has* grade of preference i *in the p-list.*

We have now to extend the definition of an r-rule accordingly. This is done by including p-lists in r-literals:

Definition 6. *A P-RASP* resource-literal *(r-literal) is either a program-literal or an amount-atom or a p-list.*

In practice, P-RASP rules differ from RASP rules in that p-lists are admitted in place of amount-atoms. More precisely, the syntax of an r-rule in P-RASP is defined as in Def. 2 where in (1) some of the $B_1, \ldots, B_k, H$ may be p-lists.

Intuitively speaking, a p-list plays a role similar to an exclusive disjunction of amount-atoms. If a p-list occurs in the body (resp. head) of a rule, it encodes the requirement that one (and only one) resource among $q_1, \ldots, q_h$ has to be consumed (resp. produced), in the indicated amount, if the rule is fired. Moreover, q_i is preferred to q_j, for $i < j$.

Remark 2. The kind of preference among alternative uses of resources expressed by p-lists have a *local scope*: each p-list is seen in the context of a particular rule (which models a specific process in manipulating some amounts of resources). Clearly, such a local aspect is strictly correlated with the constraints on global resource balance and resource availability. Consequently, preferences locally stated for different rules might/should be expected to interact "over distance" with those expressed in other rules. Nevertheless, different preference orders on the same amount-atoms can be expressed in different p-lists.

Example 1. Assembling different PCs requires different sets of components (motherboard, processor(s), ram modules, etc.) and preference might be imposed depending on the kind of PC. For instance, in case of servers one might prefer SCSI disks rather than EIDE disks and vice versa for normal PCs:

cpu:5. $scsihd$:5. $eidehd$:9. $motherboard$:7. ram_module:20.
$pc(server)$:1 $\leftarrow$ cpu:2, ($scsihd$:2>$eidehd$:2), $motherboard$:1, ram_module:4.
$pc(desk)$:1 $\leftarrow$ cpu:1, ($eidehd$:2>$scsihd$:2), $motherboard$:1, ram_module:2.

Notice that completely antithetic orders are expressed in the two r-rules.

2 Semantics of P-RASP

In this section we first define the semantics of ground P-RASP programs. The general case is then easily dealt with by considering the grounding of a program P to be the set of all ground instances of rules of P that are obtainable through ground substitutions using constants occurring in P.

Semantics of a (ground) P-RASP program is determined by interpreting program-literals as in ASP and amount-atoms in an auxiliary algebraic structure that supports operations and comparisons. The rationale behind the proposed semantic definition is the following. On the one hand, we translate each r-rule into a fragment of a plain ASP program, so that we do not have to modify the definition of stability which remains the same: this is of some importance in order to make the several theoretical and practical advances in ASP still available for RASP and P-RASP. However, an answer set of a P-RASP program will support the firing of an r-rule only if: the rule is satisfied (in the usual way) as concerns its program-literals; and the requested amounts are allocated for all the resource-atoms. Hence, an interpretation (and consequently an answer set) for an r-program has two components: a set of program atoms and an allocation of actual quantities to amount-atoms.

In describing the semantics of an r-program P we will proceed as follows. First we fix an algebraic structure to represent quantities and supports operations on them. Then, we develop a representation for collections of amounts with positive balance. Each of these collections will be a potential allocation of quantities to all the amount-atoms relative to a single resource symbol in P. Then, we introduce the notion of r-interpretation of P by selecting an allocation of amounts for each resource symbol in P.

Modeling Amounts. Amounts are modeled by choosing a collection Q of *quantities*, the operations to combine and compare quantities, and a mapping $\kappa : \mathcal{C}_R \to Q$ that associates quantities to amount-symbols. A natural choice is $Q = \mathbb{Z}$. In this case, positive (resp. negative) integers model produced (resp. consumed) amounts of resources. Alternative options for Q are obviously viable. (For instance, one could choose Q to be the set of rational numbers.) For the sake of simplicity, in the rest of the presentation, we will adopt a simplification by identifying $\mathcal{C}_R$ with $\mathbb{Z}$ (and κ being the identity). This will not cause loss in the generality of the treatment.

Notation. Before going on, we introduce some useful notation. Given two sets X, Y, let $\mathcal{FM}(X)$ denote the collection of all finite multisets of elements of X, and let Y^X denote the collection of all (total) functions having X and Y as domain and codomain, respectively. For any (multi)set Z of integers, $\sum(Z)$ denotes their sum (e.g., $\sum(\{\!\{ 2,5,3,3,5 \}\!\}) = 18$).

Given a collection S of (non-empty) sets, a *choice function* $c(\cdot)$ for S is a function having S as domain and such that for each s in S, $c(s)$ is an element of s. In other words, $c(\cdot)$ chooses exactly one element from each set in S.

In order to deal with the preference order syntactically expressed by a p-list, we consider each amount-atom in a p-list as marked with an integer index. Such indexes are intended to represent the grade of preference of the amount-atoms (cf., Def. 5). Operationally, for each p-list, its composing amount-atoms will be associated, from left to right, with successive indexes starting from 1; for simple amount-atoms, the index will always be 0.

As mentioned, the elements of $Q = \mathbb{Z}$ provide the interpretations for amount symbols. To deal with the preference orders expressed by p-lists, we need a structure slightly richer that Q. In fact, to take into account of the preference grades, we will interpret amount-atoms in $\mathbb{N} \times Q$. We call *amount couples* the elements of $\mathbb{N} \times Q$. For instance: an interpretation for a p-list such as *skim_milk*:2>*whole_milk*:2, occurring in the head of an r-rule, will involve one of the couples $\langle 1, 2\rangle$ and $\langle 2, 2\rangle$, where the first components of the couples reflect the grades of preference and the second elements are the quantities.[3] For single amount-atoms (in a head of an r-rule), such as *egg*:2, no preference is involved and a potential interpretation is the amount couple $\langle 0, 2\rangle$.

Given an amount couple $r = \langle n, x\rangle$, let $grade(r) = n$ and $amount(r) = x$. We extend such a notation to sets and multisets, as one expects: namely, if X is a multiset then $grade(X)$ is defined as the multiset $\{\!\{ n \mid \langle n, x\rangle \text{ is in } X \}\!\}$, and similarly for $amount(X)$. E.g., if $X = \{\!\{ \langle 1, 2\rangle, \langle 2, 4\rangle, \langle 3, 1\rangle, \langle 1, 2\rangle \}\!\}$ then $grade(X)$ is $\{\!\{ 1, 2, 3, 1 \}\!\}$ and $amount(X)$ is $\{\!\{ 2, 4, 1, 2 \}\!\}$.

Interpretation of P-RASP Programs. An interpretation for P must determine an allocation of amounts for all occurrences of such amount symbols in P. We represent produced quantities (corresponding to amount-atoms in heads) by positive values, while negative values model consumed amounts (corresponding to amount-atoms in bodies). Since amounts and resource-symbols are used to model production and consumption of "real world" objects, we must take into account the obvious constraint that we cannot consume more than what is produced. In other words, for each resource symbol q, the overall sum of quantities allocated to amount-atoms of the form q:a must not be negative. The collection $\mathbb{S}_P$ of all potential allocations (i.e., those having a non-negative global balance)—for any single resource-symbol occurring in P (considered as a set of rules)—is the following collection of mappings:

$$\mathbb{S}_P = \Big\{ F \in (\mathcal{FM}(\mathbb{N} \times Q))^P \mid 0 \leqslant \sum \Big(\bigcup_{\gamma \in P} amount\big(F(\gamma)\big) \Big) \Big\} \tag{2}$$

[3] Recall that we are identifying the set of amount-symbols $\mathcal{C}_R$ with the domain of quantities $Q = \mathbb{Z}$. Consequently, the symbol 2 in the amount couple $\langle 1, 2\rangle$ is an element of Q, whereas the symbol 2 in *skim_milk*:2 is an element of $\mathcal{C}_R$.

The rationale behind the definition of $\mathbb{S}_P$ is as follows: Let q be a fixed resource-symbol. Each element $F \in \mathbb{S}_P$ is a function that associates to every rule $\gamma \in P$ a (possibly empty) multiset $F(\gamma)$ of amount couples, assigning certain quantities to each occurrence of amount-atoms of the form $q{:}a$ in γ. All such Fs satisfy, by definition of $\mathbb{S}_P$, the requirement that, considering the entire P, the global sum of all the quantities F assigns must be non-negative. As we will see later, only some of these allocations will actually be acceptable as a basis for a model.

An r-interpretation of the amount symbols in a ground r-program P is defined by providing a mapping $\mu : \tau_R \to \mathbb{S}_P$. Such a function determines, for each resource-symbol $q \in \tau_R$, a mapping $\mu(q) \in \mathbb{S}_P$. In turn, this mapping $\mu(q)$ assigns to each rule $\gamma \in P$ a multiset $\mu(q)(\gamma)$ of quantities, as explained above. The use of multisets allows us to handle multiple copies of the same amount-atom. Each of them corresponds to a different amount of resource to be taken into account.

Let $\mathcal{B}(X, Y)$ denote the collection of all ground atoms built up from predicate symbols in X and terms in Y. We have the following

Definition 7. *An* r-interpretation *for a (ground) r-program P is a pair $\mathcal{I} = \langle I, \mu \rangle$, with $I \subseteq \mathcal{B}(\Pi_P, \mathcal{C})$ and $\mu : \tau_R \to \mathbb{S}_P$.*

Intuitively: I plays the role of a usual answer set assigning truth values to program-literals; μ describes an allocation of resources.

Example 2. Let $\Pi_P = \{\mathit{have\_black\_powder}, \mathit{have\_gun\_powder}\}$ and $\Pi_R = \{\mathit{saltpeter}, \mathit{charcoal}, \mathit{sulfur}\}$, and consider the following program P_1:

(γ_1) $\mathit{have\_black\_powder} \leftarrow \mathit{saltpeter}{:}15, \mathit{charcoal}{:}3, \mathit{sulfur}{:}2.$
(γ_2) $\mathit{have\_gun\_powder} \leftarrow \mathit{saltpeter}{:}7, \mathit{charcoal}{:}3.$
(γ_3) $\mathit{sulfur}{:}4.$ (γ_4) $\mathit{saltpeter}{:}18.$ (γ_5) $\mathit{charcoal}{:}5.$

An r-interpretation for P_1 is $\langle I, \mu \rangle$ with $I = \{\mathit{have\_gun\_powder}\}$ and μ such that $\mu(\mathit{saltpeter})(\gamma_2) = \{\!\!\{ \langle 0, -7 \rangle \}\!\!\}$, $\mu(\mathit{saltpeter})(\gamma_4) = \{\!\!\{ \langle 0, 18 \rangle \}\!\!\}$, $\mu(\mathit{charcoal})(\gamma_2) = \{\!\!\{ \langle 0, -3 \rangle \}\!\!\}$, $\mu(\mathit{charcoal})(\gamma_5) = \{\!\!\{ \langle 0, 5 \rangle \}\!\!\}$, $\mu(\mathit{sulfur})(\gamma_3) = \{\!\!\{ \langle 0, 4 \rangle \}\!\!\}$, and $\mu(q)(\gamma_i) = \{\!\!\{ \;\}\!\!\}$ otherwise.

The firing of an r-rule (which involves consumption/production of resources) can happen only if the truth values of the program-literals satisfy the rule. We reflect the fact that the satisfaction of an r-rule γ depends on the truth of its program-literals by introducing a suitable fragment of ASP program $\widehat{\gamma}$. Let the r-rule γ have $L_1, \ldots, L_k$ as program-literals and $R_1, \ldots, R_h$ as amount-atoms (or p-lists). The ASP-program $\widehat{\gamma}$ is so defined:

$$\widehat{\gamma} = \begin{cases} \{ \leftarrow \overline{L_1}, \ldots, \leftarrow \overline{L_k} \} & \text{if the head of } \gamma \text{ consists of amount-atoms or p-lists} \\ \{ \leftarrow \overline{L_1}, \ldots, \leftarrow \overline{L_k}, & \text{if } \gamma \text{ has the program-atom } H \text{ as head} \\ \quad H \leftarrow L_1, \ldots, L_k \} & \quad \text{and } h > 0 \\ \{ \gamma \} & \text{otherwise (e.g., } \gamma \text{ is a program-rule).} \end{cases}$$

Def. 8, to be seen, states that in order to be a model, an r-interpretation $\mathcal{I}$ that allocates non-void amounts to some amount-atoms of γ (i.e., γ is fired), has to model the ASP-rules in $\widehat{\gamma}$. (Notice that if γ is a program rule then $\widehat{\gamma} = \{\gamma\}$.)

So far we have developed a semantic structure in which r-rules are interpretable by singling-out suitable collections of amount couples.

Different ways of allocating amount of resources to an r-program are possible. In order to be acceptable, an allocation has to reflect, for each p-list r in P, one of the admissible choices that r implicitly represents. In order to extract from P the information about such admissible choices we need some further notation.

Let ℓ be either an amount-atom or a p-list in a resource-rule γ. Let

$$\mathit{setify}(\ell) = \begin{cases} \{\langle 0, q, a\rangle\} & \text{if } \ell \text{ is } q{:}a \\ \{\langle 1, q_1, a_1\rangle, \dots, \langle h, q_h, a_h\rangle\} & \text{if } \ell \text{ is } \; q_1{:}a_1{>}\cdots{>}q_h{:}a_h \end{cases}$$

We will use *setify* to represent the amount-atoms of rules as triples denoting: the position in each preference list where they occur; the resource-symbol they contain; the amount that is required for this resource-symbol in that preference list. We generalize the notion to any multiset X of amount-atoms and p-lists: $\mathit{setify}(X) = \{\!\!\{\, \mathit{setify}(\ell) \mid \ell \text{ in } X \,\}\!\!\}$.

Let $r\text{-}\mathit{head}(\gamma)$ and $r\text{-}\mathit{body}(\gamma)$ denote the multiset of amount-atoms or p-lists occurring in the head and in the body of γ, respectively. In order to distinguish, in the representation, between amount-atoms occurring in heads and in bodies, we define $\mathit{setify}_b(\gamma)$ and $\mathit{setify}_h(\gamma)$ as the multisets $\{\!\!\{\, \mathit{setify}(x) \mid x \in r\text{-}\mathit{body}(\gamma) \,\}\!\!\}$ and $\{\!\!\{\, \mathit{setify}(x) \mid x \in r\text{-}\mathit{head}(\gamma) \,\}\!\!\}$, respectively.

At this point we can associate to each r-rule γ, a set $\mathcal{R}(\gamma)$ of multisets, intended to represent the collection of admissible choices we mentioned above:

$$\mathcal{R}(\gamma) = \Big\{ \{\!\!\{\, \langle i, q, a\rangle \mid \langle i, q, a\rangle = c_1(S_1) \text{ and } S_1 \text{ in } \mathit{setify}_h(\gamma) \,\}\!\!\} \cup \{\!\!\{\, \langle i, q, -a\rangle \mid \langle i, q, a\rangle = c_2(S_2) \text{ and } S_2 \text{ in } \mathit{setify}_b(\gamma) \,\}\!\!\} \;\Big|\; \text{for } c_1 \text{ and } c_2 \text{ choice functions for } \mathit{setify}_h(\gamma) \text{ and } \mathit{setify}_b(\gamma), \text{ resp.} \Big\}$$

where c_1 (resp. c_2) ranges on all possible choice functions for $\mathit{setify}_h(\gamma)$ (resp. for $\mathit{setify}_b(\gamma)$). Each element of $\mathcal{R}(\gamma)$ represents a possible admissible selection of one amount-atom from each of the p-lists in γ and an actual allocation of an amount to it. Negative quantities are associated to amount-atoms of the body of γ, as these resources are *consumed.*[4] Vice versa, the quantities associated to amount-atoms occurring in the head are positive, as these resources are *produced.*

Definition 8. *Let $\mathcal{I} = \langle I, \mu\rangle$ be an r-interpretation for a (ground) r-program P. $\mathcal{I}$ is an* answer set *for P if the following conditions hold:*

- *for all rules $\gamma \in P$*
$$\Big(\forall\, q \in \tau_R \,\big(\mu(q)(\gamma) = \emptyset\big)\Big) \vee \Big(\bigcup_{q \in \tau_R} \Big\{ \{\!\!\{\, \langle i, q, v\rangle \mid \langle i, v\rangle \textit{ is in } \mu(q)(\gamma) \,\}\!\!\} \Big\} \in \mathcal{R}(\gamma)\Big)$$
- *I is a stable model for the ASP-program $\widehat{P}$, so defined*
$$\widehat{P} = \bigcup \left\{ \widehat{\gamma} \;\middle|\; \begin{array}{l} \gamma \textit{ is a program-rule in } P, \;\; \textit{or} \\ \gamma \textit{ is a resource-rule in } P \textit{ and } \exists\, q \in \tau_R \,\big(\mu(q)(\gamma) \neq \emptyset\big) \end{array} \right\}$$

[4] To be precise, the admissible quantity corresponds to the negation of the amount occurring in an amount-atom of the body. One may also specify negative *byproducts* directly in the body, as in the amount-atom $q{:}{-2}$, for instance. In this case, amounts of q are produced and not consumed (cf., [9]).

The two disjuncts in the formula in Def. 8 correspond to the two cases: a) the rule γ is not fired, so null amounts are allocated to all its amount-atoms; b) the rule γ is actually fired and all needed amounts are allocated (by definition this happens if and only if $\exists\, q \in \tau_R \left(\mu(q)(\gamma) \neq \emptyset\right)$ holds). (Again, notice that case b) imposes that the amount couples assigned by μ to a resource q in a rule γ reflect one of the possible choices in $\mathcal{R}(\gamma)$.)

We now formally introduce the notion *resource balance*:

Definition 9. *Let $\mathcal{I} = \langle I, \mu \rangle$ be an answer set for a (ground) r-program P. The* resource balance *for P, w.r.t. $\langle I, \mu \rangle$, is the mapping $\varphi : \tau_R \to Q$ defined as:*

$$\varphi(q) = \sum \left(\left\{\!\!\left[\sum \left(\mathit{amount}\big(\mu(q)(\gamma)\big) \right) \mid \gamma \in P \right]\!\!\right\} \right)$$

which summarizes consumptions and productions of all resources.

Finally, we say that an r-interpretation $\mathcal{I}$ is an answer set of an r-program P if it is an answer set for the grounding of P.

Note that the above definition however does not in general fulfill the preferences expressed through p-lists. In order to impose a preference order on the answer sets of an r-program, we need to provide a *preference criterion* $\mathcal{PC}$ to compare answer sets. Such a criterion should impose an order on the collection of answer sets by reflecting the (preference grades in the) p-lists. Any criterion has to take into account that each rule determines a (partial) preference ordering on answer sets. In a sense, $\mathcal{PC}$ should aggregate/combine all "local" partial order to obtain a global one.

Fundamental techniques for combining preferences (seen as generic binary relations) can be found for instance in [1]. Regarding combination of preferences in Logic Programming, criteria are also given, for instance, in [4, 7, 6, 24].

Here we will just consider for P-RASP two of the simpler criteria among the variety of alternative possible choices. As a first example, we directly exploit the ordering of amount-atoms in the p-lists (i.e., their relative position). For any multiset m in $\mathcal{FM}(\mathbb{N} \times Q)$ and $i \in \mathbb{N}$, let be $\beta_i(m) = |\{\!\!\{ \langle i, v \rangle \mid \langle i, v \rangle \text{ is in } m \}\!\!\}|$. A partial order on answer sets can be defined as follows. Given two answer sets $\mathcal{I}_1 = \langle I_1, \mu_1 \rangle$ and $\mathcal{I}_2 = \langle I_2, \mu_2 \rangle$ for an r-program P, with $\mu_1 \neq \mu_2$, let m_i be the multiset

$$m_i = \bigcup_{\gamma \in P,\, q \in \tau_R} \mu_i(q)(\gamma),$$

for $i \in \{1, 2\}$, and let j be the minimum natural number such that $\beta_j(m_1) \neq \beta_j(m_2)$. We put $\mathcal{I}_1 \prec_1 \mathcal{I}_2$ if and only if $\beta_j(m_1) > \beta_j(m_2)$.

Our first preference criterion $\mathcal{PC}_1$ states that $\mathcal{I}_1$ is preferred to $\mathcal{I}_2$ if it holds that $\mathcal{I}_1 \prec_1 \mathcal{I}_2$. The *preferred answer sets* with respect to $\mathcal{PC}_1$ are those answer sets that are $\prec_1$-minimal. In a sense, the criterion $\mathcal{PC}_1$ has a "positional flavor": the answer sets that selects the highest possible number of leftmost elements (in the p-lists) are preferred.

Our second criterion brings into play the magnitude of the preference grades. This can be done by considering the grades as weights and by optimizing with respect to the global weight expressed by the entire answer set. (Clearly, more complex assignments of weights are viable.) For any answer set $\mathcal{I} = \langle I, \mu \rangle$ let

$$\omega(\mathcal{I}) = \sum_{\gamma\in P,\, q\in\tau_R} grade\big(\mu_i(q)(\gamma)\big).$$

Given $\mathcal{I}_1$ and $\mathcal{I}_2$ as before, we put $\mathcal{I}_1 \prec_2 \mathcal{I}_2$ if and only if $\omega_j(m_1) < \omega_j(m_2)$. Consequently, our second preference criterion $\mathcal{PC}_2$ states that $\mathcal{I}_1$ is preferred to $\mathcal{I}_2$ if it holds that $\mathcal{I}_1 \prec_2 \mathcal{I}_2$. As before, the preferred answer sets, with respect to $\mathcal{PC}_2$, are those that are $\prec_2$-minimal.

3 Conditional preferences on resources.

Let us extend the syntax of r-rules by admitting p-lists (or amount-atoms) whose activation is subject to the truth of a conjunctive condition. A *conditional p-list* (cp-list, for short) is a writing of the form

$$(r \;\; \textbf{\textit{if}} \;\; L_1, \ldots, L_m)$$

where r is a p-list $q_1{:}a_1 > \cdots > q_h{:}a_h$ (or simply an amount-atom), and $L_1, \ldots, L_m$ are program-literals. The intended meaning of a cp-list occurring in the body of a r-rule γ (the case of the head is analogous) is that whenever γ is fired the rule has to consume one of the resources occurring in r. If the firing occurs in correspondence of an answer set that satisfies the literals $L_1, \ldots, L_m$, then the choice of which resource to consume is determined by the preference expressed by the p-list. Otherwise, if any of the L_i is not satisfied, a non-deterministic choice is performed. (Hence the conjunction $L_1, \ldots, L_m$ need not to be satisfied in order to fire γ.) More precisely, the r-rule containing the cp-list becomes, if $L_1, \ldots, L_m$ does not hold, equivalent to h r-rules, each containing exactly one of the amount-atoms $q_j{:}a_j$, in place of the cp-list.

Such an extension of P-RASP can be treated by translating the rules involving cp-lists into regular r-rules. For instance, the rule

$$H \leftarrow B_1, \ldots, B_k, (r \;\; \textbf{\textit{if}} \;\; L_1, \ldots, L_m)$$

is translated into this fragment of r-program:

$$\begin{array}{lll}
p \leftarrow not\ np. & np \leftarrow not\ p. & \\
\leftarrow np, L_1, \ldots, L_m. & \leftarrow p, \overline{L_i}. & \text{for } i \in \{1, \ldots, m\} \\
H \leftarrow B_1, \ldots, B_k, r, p. & & \\
H \leftarrow B_1, \ldots, B_k, q_j{:}a_j, pq_j, np. & & \text{for } j \in \{1, \ldots, h\} \\
npq_i \leftarrow pq_j. & pq_j \leftarrow not\ npq_j. & \text{for } i, j \in \{1, \ldots, h\}, i \neq j
\end{array}$$

where p, np, npq_j and pq_j (for each $j \in \{1, \ldots, h\}$) are fresh program atoms. Consequently, the semantics of cp-lists is given in terms of that of p-lists.

Similarly, one can introduce cp-lists with different semantics. For example, one might imagine a cp-list that, differently from the previous case, when some L_i does not hold the firing does not require any consumption of resources in r.

4 On Complexity and Implementation of P-RASP

We limit ourselves to shortly address the main aspects of complexity and implementation, a detailed presentation goes beyond the scope of this paper.

As regards the implementation, a solver for (P-)RASP built on top of an existing ASP-solver is described in [9, 10]. Basically, a preliminary translation converts an r-program in ASP, by rendering the semantics presented in Section 2. This ASP program is then joined to an ASP specification of an inference engine which performs the real reasoning on resources allocation and that remains independent from the particular r-program at hand. Preference criteria (as well as some cost-based features and budget policies) are encoded in the inference engine by exploiting optimization statement commonly supported by ASP-solvers such as `smodels` or `clasp`.

The analysis of the complexity of PRASP can be made by establishing a relationship with LPOD [7]. In this approach, one can define rules of the form:

$$A_1 \, X \, A_2 \, \ldots \, A_n \leftarrow \mathit{Body}$$

meaning that one or more of the A_i's can be derived provided that *Body* holds, where A_1 is the best preferred option, A_2 the second best, and so on. These preferences can be expressed only in the head of rules and have a global flavor, i.e., their scope is the entire program. Apart form the notation, there is a clear similarity with PRASP p-lists when occurring in the head of r-rules. Then, by considering resource atoms as plain atoms and adapting the syntax, a PRASP r-rule with a p-list in the head and no p-lists in the body can be considered to be an LPOD rule.

As concerns an r-rule, say γ_i, with p-lists in the body, i.e., of the form:

$$H \leftarrow \ldots, \, B_1 {>} B_k \, ldots$$

we can rephrase it in LPOD terms as the set of rules (where q_i is a fresh symbol)

$$\begin{array}{l} H \leftarrow \ldots, \, q_i \, ldots \\ q_i \leftarrow B_1 \\ \ldots q_i \leftarrow B_k \\ B_1 \, X \, \ldots \, X \, B_k \end{array}$$

We may notice that the k atoms in the p-list have been replaced by the single atom q_i, but then appear in the LPOD fact. Then, we have added k rules with two atoms each: therefore, we have substituted k atoms with $k + 2k + 1$ atoms, which ensures that the program resulting from this transformation is just linearly larger than the original one. For the transformed program, as discussed in [7], credulous reasoning is either Σ_P^2 complete or stays in Δ_P^2 according to the chosen preference criterium for selecting preferred answer sets (all this considering that credulous reasoning for plain RASP it has been proved in [9] to be NP-complete).

The approach of [21] has the same complexity, which means that each formalism can be translated into each other. However, as concerns the programming style in the mentioned approaches preferences are global, i.e., imposed all over the program, while in our case preferences are local to rules (i.e., the same amount-atoms might be ordered differently in different p-lists, cf., Example 1). Reflecting such a “locality” character by means of global preferences would originate as seen above an unnatural representation, also making it harder to design an efficient implementation and to prove its correctness. To ease these difficulties, we have provided an "autonomous" semantics which better reflects, in our opinion, the intuitive meaning that a programmer assigns to resources and quantities.

5 Related Work

The seminal work in linear logic [15] introduced the possibility of reading "A implies B" as "if you give me an A, I will give you a B" or, more precisely, "give me as many As as I might need and I will give you one B". RASP takes a very similar position, assuming "give me as many As as I might need and I will give you a certain number of Bs". Logic programming languages based on fragments of linear logic are fairly general, as they allow the programmer to exercise a significant degree of control over the pattern of use of certain program clauses (or resources) during proof search. In addition, linear logic has been used for giving to the notion of computation an overall logical setting, also for concurrent prolog systems.

In RASP, we go back to the original intuition of linear logic though in the context of the ASP semantics, where different rules which "compete" for acquiring resources give rise to different answer sets, reflecting the different allocations. Despite of the limitations (e.g., finite domain) we stay within a decidable setting.

Concerning preferences, we are not aware of approaches to preferences in linear logic. In order to understand whether P-RASP might be rephrased as a fragment of linear logic, a direct comparison would be needed, that can be a subject of future work.

Among the various approaches proposed to equip LP with some notion of resource, [20] exploits (a variant of) linear logic to define a *resource programming language (RPL)*. An operational semantics of RPL is given in terms of deduction rules: *Storage operators* and *resource transformation rules* model availability and transformation of resources, respectively. The deduction proceeds by applying these rules in a Prolog-like goal-directed fashion. A notion of step-by-step evolution of the state of the world is implicit in the rule application mechanism.

To deal with resources, [18] proposes a concurrent Prolog inference engine for clauses enriched with pre/post-conditions on resource availability. Resources are represented by multisets of atoms and terms (non-unit amounts of a resource are rendered through multiple copies of the same atom/term).

Both in [18] and [20] the operational semantics of the proposed frameworks can be given in terms of (refinements of) the SLD-procedure and (default) negation is not handled. Moreover, both the programming languages of [20] and [18] offer little separation between the resource/amounts representation symbols and program symbols: resources and amounts are represented by program terms. The distinction is left to programmer's discipline.

A form of resource treatment is described in [23, 22] to model product configuration problems. The proposed framework is based on stable model semantics and encompasses default negation and disjunctive choices.

Recently, [8] proposed the action description language $\mathcal{CARD}$ where resources are rendered through multi-valued fluents and the use of resources is implicitly modeled by the changes in fluents' values caused by actions. The approach emphasizes the use of resources in planning problems and the semantics is given in terms of transition systems (in the spirit of [14]). $\mathcal{CARD}$ also supports some form of preferences on actions. With respect to $\mathcal{CARD}$, in RASP there is a neater distinction between what is a resource and what is not. Moreover, the arithmetic of amounts (as well as the constraints on balances) is implicitly handled by RASP's inference engine. It seems that in $\mathcal{CARD}$ these aspects have to be encoded in the problem specification. On the other hand, since $\mathcal{CARD}$ is tailored to model action theories, time and state evolution are easily dealt with.

Preferences enable natural forms of commonsense reasoning. Here we briefly mention some of the proposed approaches. (See [12] for a comprehensive treatment of preferences in non-monotonic reasoning.) As regards Prolog-based frameworks, we mention [17, 16, 11] as interesting attempts to introduce preferences in (constraint) logic programming. Various forms of preferences have also been introduced in ASP (see [12]). Most of the proposed approaches on preference reasoning in ASP are based on establishing priorities/preferences among rules. In [4], A-Prolog is enriched with ordered disjunction and preferences among rules are handled by means of a rule-naming mechanism. In the case of *ordered logic programs* [25], preferences are expressed through a partial order imposed on the set of rules. The order is used to implement defeating of less-preferred rules.

Other approaches express priorities among answer sets. Intuitively, this is done by declaring those atoms whose truth is "preferred" (typically, in these cases some forms of disjunction in the heads of rules is introduced). In *prioritized logic programs* [21], a set of *priorities* determines preferences on literals: From priorities, a preference relation on answer sets is drawn.

In [7] preferences on atoms are modeled by *ordered disjunction* in the head of rules. Considering a given answer set of a program, for each rule a *degree of satisfaction* is determined depending on which atom of the head is satisfied. Satisfaction degrees of all rules are then combined, according to some criterion, to rank the answer sets. Through similar ideas, a *Preference Description Language* is defined in [6] to formalize penalty-based preference handling in ASO. A comparison of these approaches can be found in [25].

Notice that in almost all the above mentioned cases, preferences are expressed globally, e.g., by providing an order relation that applies on all the rules (or atoms) of the program. In P-RASP, as shown, preferences are imposed, by using p-lists, on some of the atoms of a rule. In this sense preference in P-RASP has a local character, cf., Remark 2 and Example 1.

Conclusions

In this paper, we have presented a refinement of the RASP approach (that allows for production/consumption of resources in ASP) to include preferences on which resources to exploit/produce. Preferences are expressed by means of p-lists of amount-atoms, where leftmost ones are assumed to have higher priority in consumption/production. P-lists, that can be conditional, can in fact occur both in the body and in the head of r-rules. We have extended both syntax and semantics of RASP to account for this kind of preferences and we have introduced a concept of preferred answer set as a partial ordering among possible solution according to a certain strategy.

In future work, we intend to further generalize P-RASP by introducing preferences among sets of amount-atoms (i.e., one might e.g. prefer to use resources a and b instead of resources x, y and z), as well as (explicit) preferences on rules. We intend to apply P-RASP to practical problems, e.g., of configuration, so as to have the ground for defining and experimenting different strategies for choosing preferred answer sets.

References

[1] H. Andréka, M. Ryan, and P.-Y. Schobbens. Operators and laws for combining preference relations. *J. Log. Comput.*, 12(1):13–53, 2002.

[2] C. Anger, T. Schaub, and M. Truszczyński. ASPARAGUS – the Dagstuhl Initiative. *ALP Newsletter*, 17(3), 2004. See `http://asparagus.cs.uni-potsdam.de`.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.

[4] M. Balduccini and V. S. Mellarkod. CR-Prolog$_2$ with ordered disjunction. In *Proc. of ASP'03*, 2003.

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

[6] G. Brewka. Complex preferences for answer set optimization. In *Proc. of KR'04*, 2004.

[7] G. Brewka, I. Niemelä, and T. Syrjänen. Logic programs with ordered disjunction. *Comput. Intell.*, 20(2):335–357, 2004.

[8] S. Chintabathina, M. Gelfond, and R. Watson. Defeasible laws, parallel actions, and reasoning about resources. In *Proc. of CommonSense'07*, 2007.

[9] S. Costantini and A. Formisano. Modeling resource production and consumption in answer set programming. In *Proc. of ASP07*, 2007. Extended version in `www.dipmat.unipg.it/~formis/papers/report2008_04.ps.gz`.

[10] S. Costantini and A. Formisano. Modeling preferences on resource consumption and production in ASP. Rep. 9/08, Dip. di Matematica e Informatica, Univ. di Perugia, 2008. In `www.dipmat.unipg.it/~formis/papers/report2008_09.ps.gz`.

[11] B. Cui and T. Swift. Preference logic grammars: Fixed point semantics and application to data standardization. *Artif. Intell.*, 138(1-2):117–147, 2002.

[12] J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Comput. Intell.*, 20(12):308–334, 2004.

[13] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of ICLP'88*, pp. 1070–1080. The MIT Press, 1988.

[14] M. Gelfond and V. Lifschitz. Action languages. *ETAI*, 2:193–210, 1998.

[15] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[16] K. Govindarajan, B. Jayaraman, and S. Mantha. Preference queries in deductive databases. *New Generation Comput.*, 19(1):57–86, 2000.

[17] H.-F. Guo and B. Jayaraman. Mode-directed preferences for logic programs. In *Proc. of ACM-SAC'05*, pp. 1414–1418, 2005.

[18] J.-M. Jacquet and L. Monteiro. Towards resource handling in logic programming: The PPL framework and its semantics. *Comput. Lang.*, 22(2/3):51–77, 1996.

[19] V. W. Marek and M. Truszczyński. *Stable logic programming - an alternative logic programming paradigm*, pp. 375–398. Springer, 1999.

[20] Y. U. Ryu. A logic-based modeling of resource consumption and production. *Decision Support Systems*, 22(3):243–257, 1998.

[21] C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artif. Intell.*, 123(1-2):185–222, 2000.

[22] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proc. of PADL'99*, 1999.

[23] T. Soininen, I. Niemelä, J. Tiihonen, and R. Sulonen. Representing configuration knowledge with weight constraint rules. In *Proc. of ASP'01*, 2001.

[24] T. C. Son and E. Pontelli. Planning with preferences using logic programming. *TPLP*, 6(5):559–607, 2006.

[25] D. Van Nieuwenborgh and D. Vermeir. Preferred answer sets for ordered logic programs. *TPLP*, 6(1-2):107–167, 2006.

[26] WASP–WP5 Report: Model applications and proofs-of-concept, 2005. Working Group on ASP: See `http://www.kr.tuwien.ac.at/projects/WASP/report.html`.