Conditional preferences in P-RASP

Stefania Costantini[1] and Andrea Formisano[2]

[1] Università di L'Aquila. stefcost@di.univaq.it
[2] Università di Perugia. formis@dipmat.unipg.it

Abstract. In this paper, we extend our previous work where we have introduced the possibility of defining and using resources in ASP. In fact, in P-RASP (Resourced ASP with Preferences) one can define resources with their amounts. Available resources can be used for producing other resources, thus consuming either all or just a part of the available amount. The remaining amount, if any, can be used in a different way. It is possible to express preferences about which resources should be either consumed or produced. Moreover, *conditional* preferences, of three different forms, allow one to express preferences according to certain conditions, that are to be evaluated "dynamically.", namely, with respect to the specific answer set at hand. The semantic rendering of P-RASP with conditional preferences is in terms of plain P-RASP, though the translation is not straightforward and thus the new features are not *syntactic sugar*.
Key words: Answer set programming, quantitative reasoning, preferences, non-monotonic logic programming, language extensions.

1 Introduction

Recently (cf. [6]), we have extended Answer Set Programming (ASP) (which is a form of logic programming based on the answer set semantics [9]) in order to support declarative reasoning on consumption and production of resources, drawing inspiration from what is possible e.g. in non-classical logics such as Linear Logics [11] and Description Logics [2]. Going further, [7] introduces the possibility of expressing declarative *preferences* in the specification of production/consumption processes. In particular, in realizing the same process (modeled as we will see below through the "firing" of certain rules), one may prefer to produce and/or consume certain resources rather than others. The extension has been called P-RASP, standing for Resourced ASP with Preferences.

We are convinced that many of the applications of ASP (that nowadays include declarative problem solving, configuration, information integration, security analysis, agent systems, semantic web, and planning for which the reader can see, e.g., [3, 17, 1, 4, 10, 8, 12, 13, 15, 16]) might benefit from these features.

In the present paper, we discuss the possibility of expressing contextual preferences i.e., preferences that apply only if certain conditions hold. This extension can be particularly useful in applications when preferences may depend on contextual elements that are not foreseeable in advance.

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

– available, in case of r-facts;
– produced, in case of r-rules, where production can take place if required resources are either available or have been produced.

As resources that are either available or produced can be in turn consumed, then numbers (quantities) can be associated to atoms occurring in the bodies of r-rules as well, as shown below.

The example concerns the preparation of desserts. 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. Atoms of the form q:a are called *amount atoms*, where the *amount symbol a* states which is 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. In P-RASP one can specify that any given rule can be repeatedly fired, where the writing [N-M]: prefixed to a rule indicates the minimum N and the maximum M of times a rule can be used (here, to produce from 1 to 3 cakes). Clearly, a suitable quantity of required resources must be available *for each* firing of a rule.

$$
\begin{array}{llll}
[1\text{-}3]: & \mathit{cake}{:}1 \leftarrow \mathit{egg}{:}3,\ \mathit{flour}{:}4,\ \mathit{sugar}{:}3. & & \\
 & \mathit{ice\_cream}{:}1 \leftarrow \mathit{egg}{:}2,\ \mathit{sugar}{:}2,\ \mathit{milk}{:}2. & & \\
 & \mathit{egg}{:}3. \quad \mathit{flour}{:}8. & \mathit{sugar}{:}6. & \mathit{milk}{:}3.
\end{array}
$$

In general, usual ASP literals (possibly involving negation-as-failure) may occur in rules. Semantics of a P-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).

RASP already offers some constructs that can be used to express basic forms of preferences on rule firing and resource consumption/production by specifying whether the firing of a rule, whenever sufficient resources are available, is either forced or optional. These and other features are fully dealt with in [6]. However, P-RASP also provides explicit preferences, illustrated in the following re-elaboration of the previous example. Suppose you might prepare a cake either with corn flour or with potato flour. In P-RASP, you can use 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:

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

In the above program fragment, 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:

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

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. We can choose, e.g., to consider as "better" the answer sets which satisfy the higher number of leftmost elements. In the last example, we would have that: producing an ice-cream with skim milk is the best solution; (a) producing 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 a 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.

In this paper, we introduce *conditional* preferences of various kinds. Assume, e.g., that one prefers skim milk when on a diet. The last rule above may become:

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

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. The above rule becomes equivalent to the couple of rules:

$$\begin{array}{l} \textit{ice\_cream}{:}1 > \textit{zabaglione}{:}2 \leftarrow \textit{skim\_milk}{:}2,\ \textit{egg}{:}2,\ \textit{sugar}{:}3. \\ \textit{ice\_cream}{:}1 > \textit{zabaglione}{:}2 \leftarrow \textit{whole\_milk}{:}2,\ \textit{egg}{:}2,\ \textit{sugar}{:}3. \end{array}$$

This generalizes to several p-lists, that we now call *cp-lists*, for "conditional" p-lists, like in the example below:

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

Assume now that one would like to add either vanilla or cinnamon, but this is not possible in case of allergy. The example above may become:

$$\begin{array}{l} \textit{ice\_cream}{:}1 > \textit{zabaglione}{:}2 \leftarrow (\textit{skim\_milk}{:}2 > \textit{whole\_milk}{:}2\ \textit{IF diet}), \\ \quad (\textit{vanilla}{:}1 > \textit{cinnamon}{:}1\ \textit{WHEN not allergy}),\ \textit{egg}{:}2,\ \textit{sugar}{:}3. \end{array}$$

If *allergy* holds, and then the *WHEN* condition is false, the p-list is ignored. This means that a *WHEN* cp-list specifies the additional level of preference which implies including or not the given p-list according to the condition.

Moreover, as we will see in Section 5, in both *IF* and *WHEN* cp-lists one may order the inner p-list according to the value of a binary predicate: in the above examples, among the ingredients listed in a p-list one will be able to prefer, e.g., the less caloric, the less expensive, the one with most vitamins, etc. This is in our opinion a significant improvements of wide applicability.

The plan of the paper is the following: in Sections 2 and 3 summarize the basic P-RASP syntax and semantics, without considering conditional preferences. In

Section 4 conditional preferences are introduced, and their semantic rendering in terms of plain P-RASP is discussed. In Section 5, a further extension is defined which allow preferences to be dynamically established according to the value of a binary predicate which determines an order among the alternatives. The semantic rendering of this extension is again in terms of plain P-RASP. Notice however that the translation is not straightforward and thus the new features cannot be considered *syntactic sugar*. Finally, we draw some conclusions.

2 P-RASP: Syntax

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 atoms 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 a writing 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 quantities can be either directly defined 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-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).

Below we introduce the possibility of expressing preferences:

Definition 1. *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 the* q_i*'s are distinct amount-atoms involving distinct resource-symbols. We say that the amount-atom* $q_i{:}a_i$ *has* degree of preference i *in the p-list.*

We now extend the definition of literal, including the new syntactic elements:

Definition 2. *A P-RASP* resource-literal *(r-literal) is either a program-literal or an amount-atom or a p-list.*

[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.

Therefore, we do not allow negation of amount-atoms. (See [6] 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.

Definition 3. *A* resource-proper-rule *has the form*

$$Idx: \quad H \leftarrow B_1, \ldots, B_k$$

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. Idx is of the form $[N_{1,1}\text{-}N_{1,2}, \ldots, N_{h,1}\text{-}N_{h,2}]$, with $h \geqslant 1$, and each $N_{j,\ell}$ is a variable or a positive integer number.

Intuitively, when all the $N_{j,\ell}$ s are integers, Idx denotes the union of h (possibly void) intervals in $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. It is intended to restrain the number of times the rule can be used: such number must be in Idx or the rule cannot be used at all. For the sake of generality, we admit that each $N_{j,\ell}$ is a variable. Then, after grounding (see below), each $N_{j,\ell}$ has to be instantiated to a positive integer.

A piece of notation: we will not write the list Idx when all $N_{j,\ell}$ s are intended to be the constant 1 (meaning that at most one use of the rule is admitted). Without loss of generality, in what follows we always assume $h = 1$ in resource-proper-rules. The treatment of the general case is essentially the same.

The following definition introduces the notion of resource-facts. Resource-facts that are not supposed to be iterable, since they are intended to model the fixed amount of resources that are available "from the beginning".

Definition 4. *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.*

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 5. *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.*

The grounding of a P-RASP program P is the set of all ground instances of rules of P, obtained through ground substitutions over the constants occurring in P. As it is well-known, ASP solvers produce the grounding of the given program as a first step, as they are able to find the answer sets of ground programs only.[2]

3 Semantics of P-RASP

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. For lack of space, here we have

[2] Work is under way both theoretically and practically to overcome at least partially this limitation. However, at present almost all ASP solvers perform the grounding.

to summarize many semantic aspects. The reader may refer to [6] and [7] for a full discussion. The rationale behind the proposed semantic definition is the following. On the one hand, we translate r-rules 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. On the other hand, an interpretation involves the allocation of actual quantities to amount-atoms. In fact, this allocation is one of the components of an interpretation: an answer set of a P-RASP program will model a resource-rule only if the rule is satisfied (in the usual way) as concerns its program-literals, and the correct amounts are allocated for the resource-atoms. A last component of an interpretation copes with the repeated firing of a rule: in case of several firings, the resource allocation must be iterated accordingly.

In order to define semantics of r-programs, we have to fix an interpretation for amount-symbols. This is done by choosing a collection Q of *quantities*, and the operations to combine and compare quantities. A natural choice is $Q = \mathbb{Z}$: thus, we consider given a mapping $\kappa : \mathcal{C}_R \to \mathbb{Z}$ that associates integers to amount-symbols. Positive (resp. negative) integers will be used to model produced (resp. consumed) amounts of resources (see [7] for a discussion about alternative options).

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[3] 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 disjunctive aspect of p-lists, we mark each resource amount with an integer index. Such indices are intended to model the degree of preference of the amount-atoms occurring in p-lists. So, any amount-atom will be represented as a pair in $\mathbb{N} \times Q$ that we call an *amount couple*. Then, for each p-list, the composing amount-atoms will be associated from left to right with successive indices starting from 1; for simple amount-atoms, the index will always be 0. For instance: for amount-atom *egg*:2 the representation is $\langle 0, 2\rangle$ where the first element is the degree of preference (which in this case is 0 as no preference is involved) and the second element is the quantity; for p-list *skim_milk*:2>*whole_milk*:2 the representation involves the couples $\langle 1, 2\rangle$ and $\langle 2, 2\rangle$, where one can see the increasing degree of preference. The representation can be extended to entire rules and to the entire program.

Given an amount couple $r = \langle n, x\rangle \in \mathbb{N} \times Q$, let $\mathit{degree}(r) = n$ and $\mathit{amount}(r) = x$. Notice that the amount can in principle be negative (e.g., if

[3] A multiset (or bag) is a generalization of a set, where repeated elements are allowed. Then, a member of a multiset can have more than one membership.

$Q = \mathbb{Z}$). We extend such a notation to sets and multisets of amount couples, as one expects: namely, if X is a multiset then $\mathit{degree}(X)$ is defined as the multiset $\{\!\!\{\, n \mid \langle n, x\rangle \text{ is in } X \,\}\!\!\}$. E.g., if $X = \{\!\!\{\, \langle 1,2\rangle, \langle 2,3\rangle, \langle 3,1\rangle, \langle 0,1\rangle, \langle 1,2\rangle \,\}\!\!\}$ then $\mathit{degree}(X)$ is $\{\!\!\{\, 1,2,3,0,1 \,\}\!\!\}$ and $\mathit{amount}(X)$ is $\{\!\!\{\, 2,3,1,1,2 \,\}\!\!\}$.

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

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

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.

Interpretation of P-RASP Programs. In what follows, we will apply a syntactical restriction on the form of the r-rules. Namely, we impose that each amount-atom cannot occur in more than one p-list within the same rule. (Clearly, a $q{:}a$ can occur in several p-lists of different rules.) Though this restriction is not strictly needed, for the sake of simplicity we focus on this simplified case.

We introduce now the notion of r-interpretation for r-programs. An interpretation of a (ground) r-program P must determine an allocation of amounts for all occurrences of such symbol in rules of P: in fact, below we introduce the first step of an interpretation, i.e., the actual assignment of amounts to the amount-atoms that occur in r-rules.

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 any resource cannot be consumed if it is not produced. Thus, we restrain to those allocations having (for each resource) a non-negative global balance. In fact,, we define a collection $\mathbb{S}_P$ of all potential allocations—for any single resource-symbol occurring in the program P (considered as a set of rules)— with the following role. Let q be a given 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 amounts to each occurrence of amount-atoms $q{:}a$ in γ. All such Fs must satisfy the only 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.

To interpret an r-program, we select a collection of elements of $\mathbb{S}_P$, one element for each resource-symbol in τ_R. More formally, an r-interpretation of a ground r-program P is defined by providing a mapping $\mu : \tau_R \to \mathbb{S}_P$. Such a function μ determines, for each resource-symbol $q \in \tau_R$, a mapping $\mu(q) \in \mathbb{S}_P$. In turn, each mapping $\mu(q)$ assigns to each rule $\gamma \in P$ a multiset $\mu(q)(\gamma)$ of quantities. The use of multisets allows us to handle multiple copies of the same amount-atom. Each of these copies must be taken into account, since it corresponds to a different amount of resource.

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.

Definition 6. *An* r-interpretation *for a (ground) r-program P is a triple $\mathcal{I} = \langle I, \mu, \xi \rangle$, with $I \subseteq \mathcal{B}(\Pi_P, \mathcal{C})$, $\mu : \tau_R \to \mathbb{S}_P$, and ξ a mapping $\xi : P \to \mathbb{N}^+$.*

Intuitively: I plays the role of a usual answer set assigning truth values to program-literals; μ describes an allocation of resources; ξ associates to each rule an integer representing the number of times the (iterable) rule is used. By little abuse of notation, we consider ξ to be defined also for program-rules and r-facts. For this kind of r-rules we assume the interval $[N_1\text{-}N_2] = [1\text{-}1]$ as implicitly specified in the rule definition, as a constraint on the number of firings.

Clearly, the firing of an r-rule (which may involve 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 $\widehat{\gamma}$.

Def. 7, to be seen, states that in order to be a model, an r-interpretation that allocates non-void amounts to the resource-symbols of γ, has to model the ASP-rules in $\widehat{\gamma}$. Some preliminary notion is in order.

We associate to each r-rule γ, the following set $\mathcal{R}(\gamma)$ of multisets. 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, taken in Q via the function κ, to the amount-symbol occurring in it. Notice that the quantities associated to amount-atoms occurring in the body of γ are negative, 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*.

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

where c_1 (resp. c_2) ranges on all possible choice functions for $\textit{setify}_h(\gamma)$ (resp. for $\textit{setify}_b(\gamma)$). In order to account for multiple firing of rules, we need to be able to "iterate" the allocation of quantities for a number n of times: to this aim, for any $n \in \mathbb{N}^+$ and $q \in \tau_R$, let

$$\mathcal{R}^n(\gamma) \;=\; \Big\{ \bigcup \{\!\!\{ X_1, \ldots, X_n \}\!\!\} \mid \{\!\!\{ X_1, \ldots, X_n \}\!\!\} \in \mathcal{FM}\big(\mathcal{R}(\gamma)\big) \Big\}$$

[4] To be precise, the assigned quantity corresponds to the negation, in Q, of the amount occurring in an amount-atom of the body. One may also specify negative *byproducts* in the body, which are produced and not consumed: in such a case, the assigned quantity will be positive (cf., [6]).

and

$$\mathcal{R}^n(q,\gamma) \;=\; \Big\{ \{\!\!\{ \langle i,v\rangle \mid \langle i,q,v\rangle \text{ is in } X \}\!\!\} \mid X \in \mathcal{R}^n(\gamma) \Big\}$$

Definition 7. *Let $\mathcal{I} = \langle I, \mu, \xi\rangle$ 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 $\gamma \in P$*

$$\Big(\forall\, q \in \tau_R\big(\mu(q)(\gamma) = \emptyset\big)\Big) \vee \Big(\forall\, q \in \tau_R\big(\mu(q)(\gamma) \in \mathcal{R}^{\xi(\gamma)}(q,\gamma)\big) \wedge \big(N_{1,1} \leqslant \xi(\gamma) \leqslant N_{1,2}\big)\Big)$$

– *I is a stable model for the ASP-program $\widehat{P}$, so defined*

$$\widehat{P} = \bigcup \left\{ \widehat{\gamma} \;\middle|\; \begin{array}{l} \gamma \textit{ is a program-rule in } P, \;\; \textit{or} \\ \gamma \textit{ is a resource-rule in } P \textit{ and } \exists\, q \in \tau_R \,\big(\mu(q)(\gamma) \neq \emptyset\big) \end{array} \right\}$$

The two disjuncts in the formula in Def. 7 correspond to the two cases: a) the rule γ is not fired, so null amounts are allocated to all its amount-symbols; b) the rule γ is actually fired $\xi(\gamma)$ times and all needed amounts are allocated (by definition this happens if and only if $\exists\, q \in \tau_R\, \big(\mu(q)(\gamma) \neq \emptyset\big)$ holds). 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 $\mathcal{R}^{\xi(\gamma)}(q,\gamma)$.

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* to compare answer sets. Such a criterion should impose an order on the collection of answer sets by reflecting the (preference degrees in the) p-lists. Any criterion $\mathcal{PC}$ 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.

In [7] we have formally considered for P-RASP two of the simpler criteria among the variety of alternative possible choices. As a first example, we have created an order among answer sets by directly exploiting the ordering of amount-atoms in the p-lists (i.e., their relative position). In a sense, this criterion has a "positional flavor": the answer sets that selects the highest possible number of leftmost elements (in the p-lists) are preferred. The second criterion brings into play the magnitude of the preference degrees. This can be done by considering the degrees 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.)

4 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. We will call *cp-*

lists (standing for conditional p-list) the construct for expressing conditional preferences in P-RASP.

Definition 8. *A cp-list 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, one of the resources occurring in r has to be consumed. If the firing occurs in correspondence of an answer set that satisfies $L_1, \ldots, L_m$, 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, if $L_1, \ldots, L_m$ does not hold, the r-rule containing the cp-list becomes equivalent to h r-rules, each containing exactly one of the $q_j{:}a_j$'s, 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:

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

where p, np, npq_j and pq_j (for each $j \in \{1, \ldots, h\}$) are fresh program atoms. In particular, p and np are used to discriminate between application of preference list and non-deterministic choice of one resource, respectively. In the latter case, the truth of each atom pq_j (for $j \in \{1, \ldots, h\}$) enables the firing of the jth rule listed at line (4). Note that at most one of such rules can be fired. In fact, rules at line (5) determine that at most one of the atoms pq_j (for $j \in \{1, \ldots, h\}$) can be true in any answer set of the program.

Consequently, the semantics of cp-lists is given in terms of that of p-lists.

If more than one cp-list occurs in a rule, then the above translation has to be generalized by introducing distinct fresh atoms for each cp-list.

A slightly modified translation must be used to handle multiple firing of a rule. Consider a rule of the form (assuming $1 \leqslant N_1 \leqslant N_2$ be natural numbers):

$$[N_1\text{-}N_2]: \quad H \leftarrow B_1, \ldots, B_k, (r\ \textbf{\textit{if}}\ L_1, \ldots, L_m).$$

Together with rules (1)–(2) we need the following substitutes for rules (3)–(5):

(6)		$np\_z \leftarrow not\ np\_nz,\ np.$	$np\_nz \leftarrow not\ np\_z,\ np.$
(7)		$u{:}N_2.$	
(8)		$low \leftarrow np\_z.$	$low \leftarrow v{:}N_1,\ np\_nz.$
(9)		$\leftarrow not\ low,\ np.$	
(10)	$[N_1\text{-}N_2]:$	$H \leftarrow B_1, \ldots, B_k,\ r,\ p.$	
(11)	$[1\text{-}N_2]:$	$H \leftarrow B_1, \ldots, B_k,\ q_j{:}a_j,\ np\_nz,\ u{:}1,\ v{:}\ {-1}.$	for $j \in \{1, \ldots, h\}$

where u, v are fresh resource symbols, while *low*, *np_z*, and *np_nz* are fresh atoms. The rationale in this translation is as follows. As before, atoms p and np discriminate, respectively, between the cases in which the preference list is used and those in which a non-deterministic choice of one resource is performed. Considering those answer sets in which np is true, the atom *np_z* (resp., *np_nz*) is used to distinguish the cases in which none (resp., some) of h rules in line (11) is fired. The auxiliary resource symbol u is used to bound the overall maximum number of firings of rules defined in line (11), because each of these firings consumes one instance of resource u, while rule (7) makes at most N_2 of such instances available. The lower bound N_1 on the number of firings is imposed through the balance on the (auxiliary) resource v. Notice that each firing of rules (11) actually produces one instance of such resource (because the amount is negative, cf., [6]). The atom *low* is then exploited to weed out those answer sets in which a low (but not null) number of firings of the rules at line (11) is performed, through the constraint (9).

Let us now introduce another form of cp-lists with a slightly different semantics. We want to model a preference that imposes use of resources only when a condition holds. More specifically, we introduce a conditional p-list of the form

$$(r\ \ \boldsymbol{when}\ L_1, \ldots, L_m)$$

With the following intended meaning: let such a cp-list occur in the body of a r-rule γ (the case of the head is analogous). Whenever γ is fired in correspondence of an answer set that satisfies all of the literals $L_1, \ldots, L_m$, the firing of the rule has to consume an amount of a resource from r. Which resource is determined by the preference expressed through the p-list. As before, the conjunction $L_1, \ldots, L_m$ needs not be satisfied in order to fire γ. In case some L_i does not hold, the firing can still be performed but this does not require any consumption of resources in r.

Also this extension of the P-RASP language can be treated by translating the rules involving cp-lists into regular r-rules. Again, the semantics of cp-lists is given in terms of p-lists. As an example consider the rule

$$[N_1\text{-}N_2]: \quad H \leftarrow B_1, \ldots, B_k, (r\ \ \boldsymbol{when}\ L_1, \ldots, L_m).$$

It is translated into this fragment of r-program:

$$
\begin{array}{rll}
 & p \leftarrow not\ np. & \\
 & np \leftarrow not\ p. & \\
 & \leftarrow np, L_1, \ldots, L_m. & \\
 & \leftarrow p, \overline{L_i}. & \text{for } i \in \{1, \ldots, m\} \\
[N_1\text{-}N_2]: & H \leftarrow B_1, \ldots, B_k,\ r,\ p. & \\
[N_1\text{-}N_2]: & H \leftarrow B_1, \ldots, B_k,\ np. &
\end{array}
$$

where, as before, p and np are fresh program atoms.

Both forms of cp-lists can be admitted in the same program. In translating rules involving more that one cp-list, the above described translations have to be generalized and combined to take into account the different combinations of truth values for the auxiliary atoms (e.g., p, np, etc.) associated to each cp-list[5].

5 Generalizing p-lists: expressing arbitrary preferences

In general, there might be cases in which (conditional) preferences are not expressible as a linear order on a set of resources. Moreover, the preferences might depend on specific contextual conditions that are not foreseeable in advance.

A simple example: for reaching my office, I prefer walking both to driving a car and catching a bus. But I have no preference between car and bus. This is a simple case of a preference order that, being not linear, cannot be modeled by a p-list.

We introduce now a generalization of p-lists, named *p-sets*, that allows one to use a partial order in expressing (collections of) p-lists. A p-set may occur in any place where a p-list does and is a writing of the form: $\{q_1{:}a_1, \ldots, q_k{:}a_k \mid p\}$ where p is a binary program predicate (defined elsewhere in the program).

Considering a specific answer set M, a particular extension is defined for the predicate p (namely, the set of pairs $\langle a, b\rangle$ such that $p(a, b)$ is true in M). Let X be the set of resource symbols $\{q_1, \ldots, q_k\}$. We consider the binary relation $R \subseteq X^2$ obtained by restricting to X the extension of p in M. R is interpreted as a preference relation over X: namely, for any $q_i, q_j \in X$ the fact that $\langle q_i, q_j\rangle \in R$ models a preference of q_i on q_j. The case of p-lists is a particular case of p-sets, obtained when R describes a total order.[6]

Notice that, in general, R needs not to be a partial order, e.g., for instance, it may involve cycles. In such cases we consider equivalent (w.r.t. preferences) those resource symbols that belongs to the same cycle in R. Moreover, there

[5] A naive way of proceeding consist in selecting a cp-list and in applying the described translation. This generates a number of new rules that involve the remaining cp-lists. Then one simply repeats the same step until no more cp-lists occur.

[6] Notice that, here, we are introducing a change in the syntax of RASP programs. Namely we are admitting resource symbols as arguments of program predicates.

might exist elements on X that are incomparable (cf., driving a car and catching a bus, in the above mentioned example).

Because of the presence of incomparable resources and equivalent resources, R can be seen as a representation of a collection of p-lists, one of them originating from an alternative total order on X compatible with R.

In order to take into account of all such possible total orders, a p-set in a P-RASP program is translated into a fragment of RASP as follows.

(12) $\quad dom_p(q_i). \quad num_p(i). \qquad$ for $i \in \{1, \ldots, k\}$
(13) $\quad cl_p(X,Y) \leftarrow p(X,Y), X \neq Y.$
(14) $\quad cl_p(X,Y) \leftarrow dom_p(X), dom_p(Y), dom_p(Z), cl_p(X,Z), cl_p(Z,Y), X \neq Y.$
(15) $\quad eq_p(T_1,T_2) \leftarrow cl_p(T_1,T_2), cl_p(T_2,T_1), dom_p(T_1), dom_p(T_2).$
(16) $\quad 1\{idx_p(T,N) : num_p(N)\}1 \leftarrow dom_p(T).$
(17) $\quad used_p(N) \leftarrow dom_p(T), idx_p(T,N), num_p(N).$
(18) $\quad \leftarrow dom_p(T), idx_p(T,N), num_p(N), N > 1, N_1 = N - 1, not\ used_p(N_1).$
(19) $\quad \leftarrow dom_p(T_1), dom_p(T_2), idx_p(T_1,N_1), idx_p(T_2,N_2), num_p(N_1), num_p(N_2),$
$\qquad cl_p(T_1,T_2), not\ cl_p(T_2,T_1), N_2 \leqslant N_1, T_1 \neq T_2.$
(20) $\quad \leftarrow not\ eq_p(T_1,T_2), dom_p(T_1), dom_p(T_2),$
$\qquad idx_p(T_1,N), idx_p(T_2,N), num_p(N), T_1 \neq T_2.$
(21) $\quad idx_p(T_2,N) \leftarrow eq_p(T_1,T_2), dom_p(T_1), dom_p(T_2), idx_p(T_1,N), num_p(N).$
(22) $\quad order_p(T,N) \leftarrow idx_p(T,N), not\ other_p(T,N), dom_p(T), num_p(N).$
(23) $\quad other_p(T,N) \leftarrow order_p(T_2,N), T \neq T_2, dom_p(T), dom_p(T_2), num_p(N).$

where: line (12) defines two domain predicates that enumerate the (relevant) arguments of p (the elements in X) and a collection of possible indices (as we will see, different indices represent different preference degrees), respectively. Rules at lines (13)–(14) evaluate the transitive closure of the relation R defined by p. Line (15) determines the equivalences between resource symbols. Rules at lines (16)–(18) index the elements of X with consecutive (possibly repeated) integers. Lines (19)–(21) restrict the possible indexing to those that do not violate the (closure of) the relation R: elements are assigned equal indices if and only if they are equally preferred (i.e., equivalent in R). Higher indices are assigned to less preferred resource symbols. Finally, rules (22)–(23) generate (compatibly with the extension of p) all admissible orders for X. Each of these orders admits a corresponding p-list. Such indexing of resource symbols can be used, in the context of a specific answer set, while applying a preference criterion, e.g., $\mathcal{PC}_1$ as described in Section 3 (page 9).

Let us consider the last example mentioned in the Introduction. Assume that, in making ice-cream or zabaglione, among some ingredients, e.g., chocolate, nuts and coconut, one would prefer the less caloric. This can be so formalized:

$ice\_cream{:}1 > zabaglione{:}2 \leftarrow (skim\_milk{:}2 > whole\_milk{:}2\ IF\ diet),$
$\qquad (vanilla{:}1 > cinnamon{:}1\ WHEN\ not\ allergy),$
$\qquad \{chocolate{:}1, nuts{:}1, coconut{:}1 \mid lesscaloric\},$
$\qquad egg{:}2, sugar{:}3.$
$lesscaloric(X,Y) \leftarrow calory(X,A), calory(Y,B), A < B.$
$calory(X,Y) \leftarrow ...$

That is, the preference among *chocolate*, *nuts* and *coconut*, i.e., a particular p-list, is determined depending on the extension of the predicate *lesscaloric*, which might be different for different answer sets and has to be established dynamically.

6 Conclusions

According to our study of related work (cf. [7]), the proposed approach dealing with resources and preferences exhibits novel features that might be of use in many cases. In this paper, we have further extended the approach by introducing the concept of conditional preferences (in previous work, there was just a hint on the "if" p-lists). These conditional preferences allow one to make either the preference or even the use of certain resources optional, according to given conditions. A futher extension allow preferences to be defined according to the value taken (at run-time) by a binary predicate which establishes the actual order. As future work, we mean to by introduce more complex conditions/constraints for parametric preferences.

RASP and basic P-RASP have been implemented (cf. [6]) by means of an inference engine able to work on top of answer set solvers. The solver for (P-)RASP acts as follows: a preliminary translation converts an r-program in ASP, by rendering its semantics (as outlined in Section 3), 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 other features such as budget policies for resource allocation) are encoded in the inference engine, also by exploiting optimization statement commonly supported by ASP-solvers such as `smodels` or `clasp`.

Since conditional preferences can be expressed by means of P-RASP program fragments, the implementation of RASP can be plainly completed so to process conditional preferences (some small modifications are due to marginal differences in syntax).

As regards an analysis of complexity of RASP, this is an issue still under investigation. Preliminary results indicate that the complexity of RASP is the same of that of LPOD [5]. In fact, observe that, given an r-rule, a solution can either provide the needed resources or not. This is a situation similar to the case of an LPOD program where the ordered disjunctions do not share any atom. The choice of preferred answer sets and the corresponding reasoning is also similar to LPOD. This means that, in principle, one formalism could be translated into the other. However, in [5], as well as in other approaches (such as [14], for instance), preferences are global, i.e., imposed all over the program, while in our case preferences are local to rules: the same amount-atoms might be ordered differently in different p-lists. Consequently, reflecting such a "locality" character by means of global preferences would originate a more complex translation making it harder to design an efficient implementation. On the other hand, providing a framework where resources and amounts are "first class" objects, certainly eases the programming activity by better reflecting the intuitive meaning that a programmer assigns to resources and quantities.

References

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

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

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

[4] G. Brewka. Answer sets and qualitative decision making. *Synthese*, 146(1-2):171–187, 2003.

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

[6] 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`.

[7] S. Costantini and A. Formisano. Modeling preferences on resource consumption and production in ASP. Technical Report 09, Dip. di Matematica e Informatica, Univ. di Perugia, 2008. Available in `www.dipmat.unipg.it/~formis/papers/report2008_09.ps.gz`.

[8] A. Dovier, A. Formisano, and E. Pontelli. A comparison of CLP(FD) and ASP solutions to NP-complete problems. In M. Gabbrielli and G. Gupta, editors, *Logic Programming, 21st International Conference, ICLP 2005, Proceedings*, volume 3668 of *LNCS*, pages 67–82. Springer, 2005.

[9] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proc. of the 5th Intl. Conference and Symposium on Logic Programming*, pages 1070–1080. The MIT Press, 1988.

[10] M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on AI*, 3(16):193–210, 1998.

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

[12] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562, 2006.

[13] V. Lifschitz. Answer set planning. In *Proc. of the 16th Intl. Conference on Logic Programming*, pages 23–37, 1999.

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

[15] T. Soininen, I. Niemelä, J. Tiihonen, and R. Sulonen. Representing configuration knowledge with weight constraint rules. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming (ASP'01): Towards Efficient and Scalable Knowledge*. AAAI Press, Menlo Park, 2001. Technical report SS-01-01.

[16] D. Van Nieuwenborgh and D. Vermeir. Ordered diagnosis. In *Proc. of LPAR'03*, pages 244–258, 2003.

[17] WASP–WP5 Report: Model applications and proofs-of-concept, 2005. Working Group on Answer Set Programming (WASP): Web site: `http://www.kr.tuwien.ac.at/projects/WASP/report.html`.