Modeling preferences and conditional preferences on resource consumption and production in ASP

Stefania Costantini

Dipartimento di Informatica, Università di L'Aquila
Via Vetoio, I-67010, L'Aquila, Italy

Andrea Formisano

Dipartimento di Matematica e Informatica, Università di Perugia
Via Vanvitelli 1, I-06123, Perugia, Italy

Abstract

In this paper, we extend our previous work on Resourced ASP, or for short RASP, where we have introduced the possibility of defining and using resources in ASP. In RASP, one can define resources with their amounts, where available resources can be used for producing other resources and the remaining amount, if any, can be used in a different way. In this paper, we introduce P-RASP (RASP with Preferences) where it is possible to express preferences about which resources should be either consumed or produced. Moreover, *conditional* preferences, of 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 of conditional preferences is given in terms of (non-conditional) preferences, though the translation is not straightforward and thus the new features are not *syntactic sugar*. Complexity of P-RASP is also discussed.

Key words: Answer set programming, quantitative reasoning, preferences, non-monotonic logic programming, language extensions

1. Introduction

As it is well-known, Answer Set Programming (ASP) is a form of logic programming based on the answer set semantics [16], where solutions to a given problem are represented in terms of selected models (answer sets) of the corresponding logic program [21]. 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, 2, 20, 24, 15] and the references therein).

Email addresses: stefcost@di.univaq.it (Stefania Costantini), formis@dipmat.unipg.it (Andrea Formisano)

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 [17] and Description Logics [3]. In recent work [11], an extension of ASP, called RASP (standing for Resourced ASP), has been proposed so as to support declarative reasoning on consumption and production of resources.

In this paper, we go further in this direction, by adding declarative *preferences* to the specification of production/consumption processes. In particular, in performing the same process (modeled through the firing of rules), one may prefer to produce and/or consume certain resources rather than others. 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, etc.

Let us briefly recall syntax and intended semantics of RASP programs through a simple example. We will then modify this example so as 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 of a certain resource is, respectively: available, in case of r-facts; produced, in case of r-rules. 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. 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*. The *amount-symbol* a specifies the quantity of resource that 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 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 2 to 4 cakes). Clearly, a suitable quantity of required resources must be available *for each* firing of a rule.

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

Usual ASP literals (possibly involving negation-as-failure) may occur in rules. Semantics of a RASP program is in fact 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 one the above rules by requiring, e.g., that ice-cream can be made only if there is a fridge and there is someone who is a good cook:[1]

[1] *Linguini* plus *Remy* makes a good cook (cf., *Ratatouille* by Pixar Animation Studios, 2007).

ice_cream:1 ← *egg*:2, *sugar*:2, *milk*:2, *fridge*, *a_cook_is_here*.
a_cook_is_here ← *is_here*(*remy*), *is_here*(*linguini*).
is_here(*remy*).
is_here(*linguini*).

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 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 [11]. In what follows, we develop a general and more expressive form of preferences on resource usage.

Recall the initial example and suppose you might prepare a cake either with plain 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:

cake:1 ← *egg*:3, *flour*:4, *sugar*:3.
cake:1 ← *egg*:3, *potato_flour*:3, *sugar*:3.

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

cake:1 ← *potato_flour*:3>*flour*:4, *egg*:3, *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:

cake:1 ← *egg*:3, *sugar*:3,
potato_flour:3>*flour*:4,
skim_milk:2>*whole_milk*:2.

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 also 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:

ice_cream:1>*zabaglione*:2 ← *egg*:2, *sugar*:3,
skim_milk:2>*whole_milk*:2.

The introduction of p-lists requires a notion 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.

P-RASP also provides *conditional* preferences of various kinds. Assume, e.g., that one prefers skim milk when on a diet. The last rule above may become:

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

If *diet* does not hold, then the preference list reduces to a disjunction, i.e. either skim milk or whole milk can be indifferently used. The above rule becomes equivalent to the couple of rules:

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

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

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

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:

$$\begin{aligned} \mathit{ice\_cream}{:}1{>}\mathit{zabaglione}{:}2 \leftarrow\ & \mathit{egg}{:}2, \mathit{sugar}{:}3, \\ & (\mathit{skim\_milk}{:}2{>}\mathit{whole\_milk}{:}2\ \ \textbf{\textit{pref\_when}}\ \ \mathit{diet}), \\ & (\mathit{vanilla}{:}1{>}\mathit{cinnamon}{:}1\ \ \textbf{\textit{only\_when}}\ \ \mathit{not\ allergy}). \end{aligned}$$

If *allergy* holds, and then the ***only_when*** condition is false, the p-list is ignored. This means that a ***only_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 6, in both ***pref_when*** and ***only_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 improvement of wide applicability.

The plan of the paper is the following: in Section 2 we review some closely related work in order to motivate our approach. Sections 3 and 4 summarize the basic P-RASP syntax and semantics. In Section 5, conditional preferences are introduced and their semantic rendering in terms of plain P-RASP is discussed. In Section 6, 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*. In Section 7 we discuss the complexity of our approach and give some hint on the implementation, where in Section 8 we discuss more widely the related work. Finally, we draw some conclusions.[2]

2. Previous Related Work on Preferences and Motivation

In Section 8 many existing approaches to preferences are mentioned and briefly discussed. In ASP however, the main distinction is among approaches that define priorities/preferences among atoms (facts), and typically introduce some form of priorities in the head of rules, and approaches that express instead priorities among rules. Below we briefly recall the two frameworks that are in our view the most relevant to the present work. Both of them introduce prioritized disjunction among atoms in the head of rules.

In the approach of [22], a preference, or priority, between two ground literals e_1, e_2 is expressed in the form $e_1 \prec e_2$. An answer set S_2 of a given program is preferable onto another answer set S_1 iff $S_2 \setminus S_1$ contains an element e_2 whose priority is higher than some element e_1 in $S_1 \setminus S_2$, and the latter does not contain another element e_3 whose priority is strictly higher that e_2. Then, preferred answer sets are a subset of the traditional ones, that can be seen as a special case corresponding to void priorities.

An interesting application is that of *priority with preconditions*. For instance, borrowing the example from [22], the situation where a person drinks tea or coffee, but (s)he prefers coffee to tea when sleepy, can be represented as follows:

$$tea \mid coffee.$$
$$tea \prec coffee \leftarrow sleepy.$$

Assuming that *sleepy* holds, this program has the preferred answer set $\{sleepy, coffee\}$.

In LPOD [7], one can write expressions such as $A \times B$ in the head of rules, where the new connective $\times$ stands for ordered disjunction. The expression intuitively stands for: if possible A, but if A is impossible then (at least) B. If there are several disjuncts, the first one represents the best preferred option, the second one represents the second best option, etc. The following is an example where a person who wishes to spend the evening out and has money, prefers to go to theatre, or else (if impossible) to go to the cinema, or else (if both previous options cannot be taken) to go to dine at a restaurant.

$$theatre \times cinema \times restaurant \leftarrow want\_to\_go\_out, have\_money.$$

For selecting the preferred answer set(s) of a program P, one obtains the possible split programs of P, where a split program P' is obtained from P by replacing each disjunctive rule by one of its options. Then, the answer sets of P are taken to be the answer

[2] It is worth noticing that this paper synthesizes and improves research partially presented in two workshop papers, namely [8, 9]. The full account of basic RASP can be found in [11].

sets of the split programs. To choose the preferred ones given that there may be several disjunctions, a notion of *degree of satisfaction* of disjunctive rules must be defined. This induces a partial ordering on answer sets. Preferred answer sets are those that satisfy all rules of P to the better degree.

The two formalisms have the same complexity (cf., Section 7) and this means that, in principle, one formalism could be translated into the other. However, as discusses in [7], despite the similarities a translation is far from being trivial in either direction. Moreover, handling such a translation in proving properties would make the task more complex. Along this line, instead of trying to adapt one of the above approaches for application in RASP, we decided to design an approach to preferences specifically intended for the RASP context. Notice, moreover, that resource consumption/production is local to each RASP rule as the same resource might be, e.g., consumed in various though independent contexts represented by different rules. Correspondingly, we have devised an approach where the preference on which resource to consume (if different choices are possible) is local to each rule. This allows a high flexibility in defining possible usage of resources also because, as seen in the above examples, preferences can be expressed conditionally. In Section 7, in order to study the complexity of P-RASP, we introduce a transformation into LPOD that should emphasize the validity of our choice as it should be apparent that the translation is far from clear.

3. P-RASP: Syntax

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

Let $\mathcal{A}(X, Y)$ denote the collection of all atoms of the form $p(t_1, \ldots, t_n)$, for $p \in X$ and $\{t_1, \ldots, t_n\} \subseteq Y$. Then, a *program atom* (i.e., an usual ASP atom) is an element of $\mathcal{A}(\Pi_P, C \cup \mathcal{V})$. An *amount-atom* is a writing of the form $q{:}a$ where a is an amount-term and $q \in \Pi_R \cup \mathcal{A}(\Pi_R, C \cup \mathcal{V})$. Let $\tau_R = \Pi_R \cup \mathcal{A}(\Pi_R, C)$. We call elements of τ_R *resource-symbols*. Some examples: in the two expressions $p{:}3$ and $q(2){:}b$, p and $q(2)$ are resource-symbols (with $p, q \in \Pi_R$ and $2 \in C$) aimed at defining two resources which are available in quantity 3 and b, resp., (with $3, b \in C_R$ 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. 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).[3] If $L = A$ (resp., $L = not\, A$) then $\overline{L}$ denotes *not* A (resp., A).

[3] We will only deal with negation-as-failure. Though, classical negation of program literals could be used

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 $q_1{:}a_1{>}\cdots{>}q_h{:}a_h$, with $h \geqslant 2$ and $q_1, \ldots, q_h$ pairwise distinct symbols. We say that $q_i{:}a_i$ has* degree of preference *i in the p-list.*

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

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

Therefore, we do not allow negation of amount-atoms (see [11] for a discussion about this point). Finally, we distinguish between *program-rules* (plain ASP rules, including the case of ASP *constraints*, i.e., rules with empty head) and *resource-rules* which differ from program rules in that they may contain amount-atoms.

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

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

where $B_1, \ldots, B_k$ ($k > 0$) are r-literals, H is either a program-atom, an amount-atoms, or a p-list, and at least one amount-atom occurs in γ. 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 are not supposed to be iterable, since they are intended to model the fixed amounts 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* rule *is either a program-rule or a resource-rule. An* r-program *is a finite set of rules.*

in (P-)RASP programs and treated as usually done in ASP.

The grounding of an r-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.[4]

Remark 6. *The kind of preference among alternative uses of resources expressed by p-lists has 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 related to 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 7. *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:*

$$
\begin{array}{l}
\mathit{main\_unit}(\mathit{server}){:}1 \leftarrow \mathit{cpu}{:}2,\ (\mathit{scsihd}{:}2{>}\mathit{eidehd}{:}2),\\
\qquad\qquad\qquad\qquad\quad \mathit{motherboard}{:}1,\ \mathit{ram\_module}{:}4.\\
\mathit{main\_unit}(\mathit{desktop}){:}1 \leftarrow \mathit{cpu}{:}1,\ (\mathit{eidehd}{:}2{>}\mathit{scsihd}{:}2),\\
\qquad\qquad\qquad\qquad\quad \mathit{motherboard}{:}1,\ \mathit{ram\_module}{:}2.\\
\mathit{cpu}{:}5.\\
\mathit{scsihd}{:}5. \qquad\qquad\qquad\qquad \mathit{eidehd}{:}9.\\
\mathit{motherboard}{:}7. \qquad\qquad\qquad\quad \mathit{ram\_module}{:}20.\\
\mathit{computer}(T){:}1 \leftarrow \mathit{main\_unit}(T){:}1,\ \mathit{monitor}{:}1,\ \mathit{mouse}{:}1,\\
\qquad\qquad\qquad\quad \mathit{keyboard}{:}1,\ \mathit{pc\_type}(T).\\
\mathit{pc\_type}(\mathit{server}).\\
\mathit{pc\_type}(\mathit{desktop}).\\
\mathit{monitor}{:}3. \qquad\quad \mathit{mouse}{:}8. \qquad\quad \mathit{keyboard}{:}4.
\end{array}
$$

Notice that completely antithetic orders are expressed in the bodies of the first two r-rules. However, both r-rules might be fired at the same time, since enough resources are available.

4. Semantics of P-RASP

Semantics of a (ground) r-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 to summarize many semantic aspects. The reader may refer to [11] for a full discussion. The rationale behind the proposed semantic definition is the following. On the one hand, we translate r-rules

[4] 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.

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 an r-program will model an r-rule only if it is satisfied (in the usual way) as concerns its program-literals, and the correct amounts are allocated for the amount-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 $\kappa : C_R \to \mathbb{Z}$ that associates integers to amount-symbols. Positive (resp. negative) integers will be used to model produced (resp. consumed) amounts of resources.

For the sake of simplicity, in what follows we will identify 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 disjunctive aspect of p-lists and to model the degrees of preference, we mark each amount-atom with an integer index. For each p-list its composing amount-atoms are associated, from left to right, with successive indices starting from 1. For single amount-atoms, the index will always be 0. So, any amount-atom will be represented as a pair in $\mathbb{N} \times Q$ that we call an *amount couple*. For example: an interpretation for *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 degree of preference and the second elements are the quantities. For single amount-atoms (in a head of an r-rule), such as *egg*:2, no preference is involved and a potential interpretation is $\langle 0, 2\rangle$.

Given an amount couple $r = \langle n, x\rangle$, let $degree(r) = n$ and $amount(r) = x$. Notice that the amount can in principle be negative (e.g., if $Q = \mathbb{Z}$). We extend such a notation to sets and multisets, as one expects: namely, if X is a multiset then $degree(X)$ is defined as the multiset $\{\!\{ n \mid \langle n, x\rangle \text{ is in } X \}\!\}$, and similarly for $amount(X)$. E.g., if $X = \{\!\{ \langle 1, 2\rangle, \langle 3, 1\rangle, \langle 1, 2\rangle \}\!\}$ then $degree(X)$ is $\{\!\{ 1, 3, 1 \}\!\}$ and $amount(X)$ is $\{\!\{ 2, 1, 2 \}\!\}$.

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 case.

An interpretation for an r-program P must determine an allocation of amounts for all occurrences of such amount symbols in P. We represent produced quantities (i.e., amount-atoms in heads) by positive values, while negative values model consumed amounts (i.e., amount-atoms in bodies). For each resource symbol q, the overall sum of quantities allocated to (produced and consumed) 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:

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

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.

Definition 8. *An* r-interpretation *for a (ground) r-program P is a triple $\mathcal{I} = \langle I, \mu, \xi \rangle$, with $I \subseteq \mathcal{B}(\Pi_P, C)$, $\mu : \tau_R \to \mathbb{S}_P$, 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 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.

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 $\widehat{\gamma}$. 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:

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

Def. 9, 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.

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. To be acceptable, an allocation has to reflect, for each p-list r in P, one of the admissible choices that r represents. In order to denote such admissible choices we need some further notation. Let ℓ be either an amount-atom or a p-list in an r-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. 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.

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*.[5] Vice versa, the quantities associated to amount-atoms occurring in the head are positive, as these resources are *produced*.

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

[5] 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., [11]).

where c_1 (resp. c_2) ranges on all possible choice functions for $setify_h(\gamma)$ (resp. for $setify_b(\gamma)$).

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

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

and

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

While $\mathcal{R}(\gamma)$ represents all the different ways of choosing one amount-atom from each p-list of γ, the collection $\mathcal{R}^n(\gamma)$ represents all the possible ways of making n times this choice (possibly, in different manners). Fixed a resource-symbol q, the set $\mathcal{R}^n(q, \gamma)$ extracts from each alternative in $\mathcal{R}^n(\gamma)$ the multiset of amount couples relative to q. Def. 9 exploits the set $\mathcal{R}^n(q, \gamma)$ to impose restrictions on the global resource balance, for each q.

Definition 9. *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 $\gamma \in P$*

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

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

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

The two disjuncts in the formula in Def. 9 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 $\exists\, q \in \tau_R\,(\mu(q)(\gamma) \neq \emptyset)$ 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. 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, 23].

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 $\mathcal{I}_1 = \langle I_1, \mu_1, \xi_1\rangle$ and $\mathcal{I}_2 = \langle I_2, \mu_2, \xi_2\rangle$ 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 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.) For any answer set $\mathcal{I} = \langle I, \mu, \xi\rangle$ let

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

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.

5. 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 10. *A cp-list is a writing of the form* $(r$ ***pref_when*** $L_1, \ldots, L_m)$*, where r is a p-list 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 an r-rule γ (the case of the head is analogous) is that whenever γ is fired, one of the resources occurring in $r = q_1{:}a_1 > \cdots > q_h{:}a_h$ 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 r. Otherwise, if any of the L_i is not satisfied, a non-deterministic choice is performed. (Hence the conjunction $L_1, \ldots, L_m$ does not need 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 amount-atoms $q_1{:}a_1, \ldots, q_h{:}a_h$, 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. The generic r-rule

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

is translated into this fragment of r-program:

(1)	$p \leftarrow not\ np.$	
(2)	$np \leftarrow not\ p.$	
(3)	$\leftarrow np, L_1, \ldots, L_m.$	
(4)	$\leftarrow p, \overline{L_i}.$	for $i \in \{1, \ldots, m\}$
(5)	$H \leftarrow B_1, \ldots, B_k, r, p.$	
(6)	$H \leftarrow B_1, \ldots, B_k, q_j{:}a_j, pq_j, np.$	for $j \in \{1, \ldots, h\}$
(7)	$npq_i \leftarrow np, pq_j.$	for $i, j \in \{1, \ldots, h\}, i \neq j$
(8)	$pq_j \leftarrow np, not\ npq_j.$	for $j \in \{1, \ldots, h\}$

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 (6). Note that at most one of such rules can be fired. In fact, rules at lines (7)–(8) 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):

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

Together with rules (1)–(4) we need the following substitutes for rules (5)–(8):

(9)		$np\_z \leftarrow not\ np\_nz,\ np.$
(10)		$np\_nz \leftarrow not\ np\_z,\ np.$
(11)		$u{:}N_2.$
(12)		$low \leftarrow np\_z.$
(13)		$low \leftarrow v{:}N_1,\ np\_nz.$
(14)		$\leftarrow not\ low,\ np.$
(15)	$[N_1\text{-}N_2]:$	$H \leftarrow B_1, \ldots, B_k,\ r,\ p.$
(16)	$[1\text{-}N_2]:$	$H \leftarrow B_1, \ldots, B_k,\ q_j{:}a_j,\ np\_nz,\ u{:}1,\ v{:}\ -1.$ 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 (16) is fired. The auxiliary resource symbol

u is used to bound the overall maximum number of firings of rules defined in line (16), because each of these firings consumes one instance of resource u, while rule (11) 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 (16) actually produces one instance of such a resource (because the amount is negative, cf., [11]). 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 (16) is performed, through the constraint (14).

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

$$(r \ \textbf{\textit{only\_when}} \ \ L_1, \ldots, L_m)$$

with this intended meaning: let such a cp-list occur in the body of an r-rule γ (the case of the head is analogous). If 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 $L_1, \ldots, L_m$ does not need to 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. The generic r-rule

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

is translated into this fragment of r-program (with p and np fresh atoms):

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

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. (A naïve way of proceeding consists 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. 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, 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 simple case of preference order, being not linear, cannot be modeled by p-lists.

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 does not need 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 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.

(17) $dom_p(q_i).\quad num_p(i).\qquad \text{for } i \in \{1, \ldots, k\}$

(18) $cl_p(X, Y) \leftarrow p(X, Y), X \neq Y.$

(19) $cl_p(X, Y) \leftarrow dom_p(X), dom_p(Y), dom_p(Z), cl_p(X, Z), cl_p(Z, Y), X \neq Y.$

(20) $eq_p(T_1, T_2) \leftarrow cl_p(T_1, T_2), cl_p(T_2, T_1), dom_p(T_1), dom_p(T_2).$

(21) $1\{idx_p(T, N) : num_p(N)\}1 \leftarrow dom_p(T).$

(22) $used_p(N) \leftarrow dom_p(T), idx_p(T, N), num_p(N).$

(23) $\leftarrow dom_p(T), idx_p(T, N), num_p(N), N > 1, N_1 = N - 1, not\ used_p(N_1).$

(24) $\leftarrow dom_p(T_1), dom_p(T_2), idx_p(T_1, N_1), idx_p(T_2, N_2), num_p(N_1), num_p(N_2),$
$\quad cl_p(T_1, T_2), not\ cl_p(T_2, T_1), N_2 \leqslant N_1, T_1 \neq T_2.$

(25) $\leftarrow not\ eq_p(T_1, T_2), dom_p(T_1), dom_p(T_2),$
$\quad idx_p(T_1, N), idx_p(T_2, N), num_p(N), T_1 \neq T_2.$

(26) $idx_p(T_2, N) \leftarrow eq_p(T_1, T_2), dom_p(T_1), dom_p(T_2), idx_p(T_1, N), num_p(N).$

(27) $order_p(T, N) \leftarrow idx_p(T, N), not\ other_p(T, N), dom_p(T), num_p(N).$

(28) $other_p(T, N) \leftarrow order_p(T_2, N), T \neq T_2, dom_p(T), dom_p(T_2), num_p(N).$

where: line (17) 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 (18)–(19)

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

evaluate the transitive closure of the relation R defined by p. Line (20) determines the equivalences between resource symbols. Rules at lines (21)–(23) index the elements of X with consecutive (possibly repeated) integers. Lines (24)–(26) 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 (27)–(28) 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 4 (page 12).

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:

$$\begin{array}{lll} \mathit{ice\_cream}{:}1 > \mathit{zabaglione}{:}2 & \leftarrow & (\mathit{skim\_milk}{:}2 > \mathit{whole\_milk}{:}2 \ \ \textbf{\textit{pref\_when}} \ \ \mathit{diet}), \\ & & (\mathit{vanilla}{:}1 > \mathit{cinnamon}{:}1 \ \ \textbf{\textit{only\_when}} \ \ \mathit{not\ allergy}), \\ & & \{\mathit{chocolate}{:}1, \mathit{nuts}{:}1, \mathit{coconut}{:}1 \mid \mathit{lesscaloric}\}, \\ & & \mathit{egg}{:}2, \mathit{sugar}{:}3. \\ \mathit{lesscaloric}(X,Y) & \leftarrow & \mathit{calory}(X,A), \mathit{calory}(Y,B), A < B. \\ \mathit{calory}(X,Y) & \leftarrow & ... \end{array}$$

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.

7. On Complexity and Implementation of P-RASP

In [11], credulous reasoning for plain RASP has been proved to be NP-complete. Namely, to determine whether there exists an answer set of a given ground r-program is an NP-complete problem. In particular, according to [11], a polynomial translation from RASP into ASP can be exhibited. In such an encoding different occurrences of amount-atoms are replaced by different plain (fresh) atoms. Then, a correspondence is established between the answer sets of r-programs and the answer sets of suitable ASP programs (see [11] for a detailed description).

Similarly, the analysis of the complexity of P-RASP can be made by establishing a relationship between P-RASP and LPOD [7].

Let us first consider P-RASP without conditional preferences.

In LPOD, one can define rules of the form:[7]

$$A_1 \times A_2 \ldots \times A_n \leftarrow \mathit{Body}.$$

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

[7]Without loss of generality, we can restrict ourselves to programs without classical negation.

be expressed only in the head of rules and have a global flavor, i.e., their scope is the entire program.

Apart from the notation, there is a similarity with p-lists when occurring in the head of r-rules. Moreover, any P-RASP program can be transformed into an LPOD program and vice versa. Roughly speaking, an LPOD program can be converted into a P-RASP program by substituting in each ordered disjunction of the form $A_1 \times A_2 \ldots \times A_n$, each A_i by $A_i{:}n$, where n is the number of occurrences of A_i in the bodies of rules. Similarly, any A_i in the body of a rule is replaced by $A_i{:}1$. This produces a P-RASP encoding of the given LPOD program. An answer set of the LPOD program can be obtained from an answer set $\mathcal{I}$ of its P-RASP encoding, by considering true an A_i if $\mathcal{I}$ allocates non-null amounts to $A_i{:}n$.

As regards the converse translation, we can complete the embedding introduced in [11] in order to deal with p-lists and obtain a translation from P-RASP to LPOD. Let us sketch the translation by focusing on the simpler case of r-rules containing just one p-list, say r, and no other amount-atoms. There are two cases:

- The P-RASP rule has the form $H \leftarrow L_1, \ldots, L_h,\ q_1{:}a_1 > \ldots > q_k{:}a_k$. Then, it is translated into the following rules:

$$
\begin{array}{ll}
s \leftarrow \mathit{not}\ ns. & \\
ns \leftarrow \mathit{not}\ s. & \\
s_{r,1} \times \ldots \times s_{r,k} \leftarrow L_1, \ldots, L_h,\ s. & \\
s \leftarrow s_{r,j}. & \text{for each } j \in \{1, \ldots, k\} \\
\leftarrow \mathit{not}\ p_r, s. & \\
p_r \leftarrow s_{r,j},\ q_j{:}a_j. & \text{for each } j \in \{1, \ldots, k\} \\
H \leftarrow L_1, \ldots, L_h,\ s. &
\end{array}
$$

  where s, ns, p_r, and all the $s_{r,j}$'s are fresh program atoms.

- The P-RASP rule has the form $q_1{:}a_1 > \ldots > q_k{:}a_k \leftarrow L_1, \ldots, L_h$. Then, it is translated as in the previous case, except that the last two lines of the above translation are replaced by the following one (observe that the negated amount $-a_j$ in the body or the rule denotes production instead of consumption of resources):

$$p_r \leftarrow s_{r,j},\ q_j{:}\ -a_j. \qquad \text{for each } j \in \{1, \ldots, k\}.$$

Such a translation can be generalized to handle an r-rule containing more than one p-list. In this case a slightly more complex translation is needed in order to ensure that either all or none of the ordered disjunctions are satisfied (i.e., either all or none of the auxiliary atoms p_r's are true). Notice that this translation also works for simple amount-atoms (that can be seen as p-lists involving just one option).

We may notice that, by applying this translation, the size of the resulting program is polynomially larger than the size of the original P-RASP program.

The program resulting from the above outlined steps does not contain any p-list, but it might contain both ordered disjunctions and amount-atoms. To get rid of these amount-atoms, we proceed by mimicking the embedding of RASP into ASP presented in [11]. This step completes the (polynomial) translation of the given program into

LPOD and allow us to establish a correspondence between the answer sets of the given P-RASP program and those of the target LPOD program. (The interested reader can also refer to [10] for a direct, but more involved, embedding of P-RASP into ASP.)

Complexity of credulous reasoning for LPOD depends on the chosen preference criterion for selecting preferred answer sets. Three main criteria for LPOD are introduced in [7]. Namely, the *cardinality*-, the *inclusion*-, and the *Pareto*- criterion. Credulous reasoning turns out to be Σ_P^2-complete in the first two cases, while stays in Δ_P^2 in the latter one.

Let us consider the PODS programs obtained by transforming P-RASP programs as described above. Observe that the criterion $\mathcal{PC}_1$ (cf., Section 4) corresponds to the cardinality preference criterion of [7]. Moreover, a Pareto-like criterion for P-RASP could be properly introduced. Then, in analogy to [7], we can inherit the same complexity results for credulous reasoning in P-RASP (without conditional preferences).

The approach of [22] has the same complexity of LPOD, which means that each of the three formalisms (LPOD, [22] and P-RASP) can in principle be translated into each other (but only as far as preferences are concerned, as neither [22] nor [7] deal with resources). However, if one considers the programming style, then P-RASP is significantly different from the mentioned approaches as they provide global preferences, i.e., imposed all over the program, while in P-RASP preferences are local to rules: i.e., the same amount-atoms might be ordered differently in different p-lists, cf., Example 7. 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. Therefore, we have chosen to provide an "autonomous" semantics which better reflects, in our opinion, the intuitive meaning that a programmer assigns to resources and quantities.

The translation into LPOD outlined above can be used also when conditional preferences are involved, by exploiting the rendering of cp-lists and p-sets in terms of p-lists, as described in Sections 5 and 6. However, such a naïve encoding of cp-lists (and p-sets) might introduce an exponential growth of the size of the resulting P-RASP program. The existence of a polynomial translation from P-RASP with conditional preferences into LPOD is still object of investigation.

As regards the implementation, a solver for (P-)RASP built on top of an existing ASP-solver is described in [11] (where completeness and correctness of the basic encoding are also shown). Essentially, a preliminary translation converts an r-program into ASP, by rendering the semantics presented in Section 4. 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.

8. Related Work

In RASP, we adopt the original intuition of linear logic, i.e., "Give as many A's as I might need and I will give you one B" in the context of the ASP semantics. Despite

of the limitations (e.g., finite domain) we stay within a decidable setting. For a comparison between RASP and the various approaches to resource-based reasoning, the interested reader can refer to [11]. Concerning preferences, we are not aware of approaches to preferences in linear logic. Thus, 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.

Concerning preferences in logic programming and non-monotonic reasoning, we briefly mention here some of the existing approaches (see [13] for a comprehensive treatment). As interesting attempts to introduce preference in (constraint) prolog-like logic programming, we mention [19, 18, 12].

Various forms of preferences have also been introduced in ASP (see [13]). Most of the proposed approaches are based on establishing priorities/preferences among rules. In [4], A-Prolog (a language of logic programs with the answer set semantics, cf., [14]) is enriched with ordered disjunction and preferences among rules are handled by means of a rule-naming mechanism. In *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* [22], a set of *priorities* determines preferences on literals: from priorities, a preference relation on answer sets is drawn. In [7], as already mentioned, 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 6 and Example 7.

9. Conclusions

According to our study of related work (cf. [11]), the proposed approach dealing with resources and preferences exhibits novel features that might be of use in many cases. Conditional preferences allow one to make either the preference or even the use of certain resources optional, according to given conditions. A further 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 also mean to by introduce more complex conditions/constraints for parametric preferences.

Acknowledgements

The authors would like to thank Wolfgang Faber, Georg Gottlob, Nicola Leone, and the anonymous referees for their useful suggestions and comments. This research has been partially supported by GNCS (Gruppo Nazionale per il Calcolo Scientifico).

References

[1] Hajnal Andréka, Mark Ryan, and Pierre-Yves Schobbens. Operators and laws for combining preference relations. *Journal of Logic and Computation*, 12(1):13–53, 2002.

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

[3] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.

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

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

[6] Gerhard Brewka. Complex preferences for answer set optimization. In *Proc. of KR'04*, pages 213–223, 2004.

[7] Gerhard Brewka, Ilkka Niemelä, and Tommi Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence*, 20(2):335–357, 2004.

[8] Stefania Costantini and Andrea Formisano. Conditional preferences in P-RASP. In *Proc. of LANMR'08*, 2008.

[9] Stefania Costantini and Andrea Formisano. Modeling preferences on resource consumption and production in ASP. In *Proc. of ASPOCP'08 Workshop of ICLP08*, 2008.

[10] Stefania Costantini and Andrea 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`.

[11] Stefania Costantini and Andrea Formisano. Answer set programming with resources. *Journal of Logic and Computation*, 2009. To appear. Draft available as Report-16/2008 of Dip. di Matematica e Informatica, Univ. di Perugia: `www.dipmat.unipg.it/~formis/papers/report2008_16.ps.gz`.

[12] Baoqiu Cui and Terrance Swift. Preference logic grammars: Fixed point semantics and application to data standardization. *Artificial Intelligence*, 138(1-2):117–147, 2002.

[13] James Delgrande, Torsten Schaub, Hans Tompits, and Kewen Wang. A classification and survey of preference handling approaches in nonmonotonic reas oning. *Computational Intelligence*, 20(12):308–334, 2004.

[14] Michael Gelfond. Representing knowledge in A-prolog. In Antonis C. Kakas and Fariba Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, volume 2408 of *Lecture Notes in Computer Science*, pages 413–451. Springer, 2002.

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

[16] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proc. of the 5th Intl. Conference and Symposium on Logic Programming*, pages 1070–1080. The MIT Press, 1988.

[17] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[18] Kannan Govindarajan, Bharat Jayaraman, and Surya Mantha. Preference queries in deductive databases. *New Generation Computing*, 19(1):57–86, 2000.

[19] Hai-Feng Guo and Bharat Jayaraman. Mode-directed preferences for logic programs. In *Proc. of ACM-SAC'05*, pages 1414–1418, 2005.

[20] Nicola Leone. Logic programming and nonmonotonic reasoning: From theory to systems and applications. In Chitta Baral, Gerhard Brewka, and John Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007*, page 1, 2007.

[21] Victor W. Marek and Miroslaw Truszczyński. *Stable logic programming - an alternative logic programming paradigm*, pages 375–398. Springer, 1999.

[22] Chiaki Sakama and Katsumi Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123(1-2):185–222, 2000.

[23] Tran Cao Son and Enrico Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 6(5):559–607, 2006.

[24] Miroslaw Truszczyński. Logic programming for knowledge representation. In Verónica Dahl and Ilkka Niemelä, editors, *Logic Programming, 23rd International Conference, ICLP 2007*, pages 76–88, 2007.

[25] Davy Van Nieuwenborgh and Dirk Vermeir. Preferred answer sets for ordered logic programs. *Theory and Practice of Logic Programming*, 6(1-2):107–167, 2006.