Extending and implementing RASP

Andrea Formisano and Davide Petturiti

Università di Perugia, Dipartimento di Matematica ed Informatica
formis@dipmat.unipg.it, davidepetturiti@gmail.com

Abstract. In previous work an extension of ASP, called RASP (standing for ASP with Resources), has been proposed. RASP supports declarative reasoning on production and consumption of (amounts of) resources. The approach combines stable model semantics with quantitative reasoning and relies on an algebraic structure to support computations and comparisons of amounts. The resulting framework also offered some form of preference reasoning on resources usage. In this paper we go further in this direction by introducing more expressive constructs to support complex preferences specification. The complexity of establishing the existence of an answer set, in such an enriched framework, is then shown to be NP-complete. A prototypical implementation of RASP has been realized. The tool, named *raspberry*, consists in a compiler that, given a ground RASP program, produces a pure ASP encoding suitable to be processed by commonly available ASP-solvers.

Key words: Answer set programming, quantitative reasoning, preferences, language extensions.

Introduction

Previous work [5, 4] proposed an extension of the Answer Set Programming (ASP) framework by explicitly introducing the notion of *resource*. Such an extension, named RASP (standing for ASP with Resources), supports both formalization and quantitative reasoning on consumption and production of amounts of resources. These are modeled by *amount-atoms* of the form $q\#a$, where q represents a specific type of resource and a denotes the corresponding amount. Resources can be produced or consumed (or declared available from the beginning). The processes that transform some amounts of resources into other resources are specified by *r-rules*, for instance, as in this simple example:

$$computer\#1 \leftarrow cpu\#1, harddisk\#2, motherboard\#1, ram\_module\#2.$$

where we model the fact that an instance of the resource computer can be obtained by "consuming" some other resources, in the indicated amounts.

In their most general form, r-rules might involve regular ASP literals together with amount-atoms. Semantics for RASP programs is given by combining stable model semantics with a notion of *allocation*. While stable models are used to deal with usual ASP literals, allocations are exploited to take care of amounts

and resources. Intuitively, an allocation assigns to each amount-atom a (possibly null) quantity. Quantities are interpreted in an auxiliary algebraic structure that supports comparisons and operations on amounts. Admissible allocations are those satisfying, for all resources, the requirement that one can consume only what has been produced. Clearly, alternative allocations might be possible, corresponding to different ways of using the same resources.

The RASP framework has been enriched in [5] so to support basic forms of preference specification on resource usage. The reader is referred to [5, 4] for a detailed comparison with previous approaches to preference reasoning, as well as for a discussion on the related works involving the notion of resource in logic programming frameworks.

The following simple example illustrates the way in which preferences are exploitable in RASP to specify different uses of the same resources.

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

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

Notice that some resources might be declared as available from the beginning. This is done by means of *r-facts*. As in the first line of the above program.

Computational complexity of RASP (without preferences) has been assessed in [4], by showing that the problem of establishing the existence of an answer set for a RASP program is NP-complete.

In this paper we further enrich the RASP language by introducing more expressive constructs to support complex preferences specification (Sect. 2). For all the proposed extensions we provide an encoding into ASP. As a consequence, we show that the enriched framework retains the same computational complexity. We also briefly report (Sect. 3) on a prototypical implementation of RASP. Sect. 1 recalls syntax and semantics of RASP (more details can be found in the appendix or in [4]).

1 A glimpse of RASP

In this section we briefly present the basic notions on RASP. For lack of space, here we have to summarize many aspects of RASP's semantics. The reader may refer to [4] (or to the appendix) for a much complete presentation.

The language of RASP. 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 symbols of variable. The elements of $\mathcal{C}_R$ are said *amount-symbols* (*a-symbols*, for short), while the elements of Π_R are said *resource-predicates* (*r-predicates*). A *program-term* (*p-term*) is either a variable or a constant symbol. An *a-term* is either a variable or an a-symbol.

Let $\mathcal{A}(X, Y)$ denote the collection of all atoms $p(t_1, \ldots, t_n)$, with $p \in X$ and $\{t_1, \ldots, t_n\} \subseteq Y$. Then, a *p-atom* is an element of $\mathcal{A}(\Pi_P, \mathcal{C} \cup \mathcal{V})$. An *r-term* is an element of $\Pi_R \cup \mathcal{A}(\Pi_R, \mathcal{C} \cup \mathcal{V})$. An *a-atom* is a writing of the form $q\#a$ where q is an r-term and a is an a-term. We call *resource-symbols* (*r-symbols*) the ground r-terms, i.e. the elements of $\tau_R = \Pi_R \cup \mathcal{A}(\Pi_R, \mathcal{C})$.

Some examples: in the two expressions $p\#3$ and $q(2)\#b$, p and $q(2)$ are r-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 $3, b \in \mathcal{C}_R$ a-symbols). Because the set of variables is not partitioned, the same variable may occur both as a p-term and as an a-term. Hence, we admit expressions such as $p(X)\#V$ where V, X are variables. In such cases, quantities are derived through instantiation.

Ground a-atoms contain no variables. A *program-literal* (*p-literal*, for short) L is a p-atom A or the negation *not* A of a p-atom (intended as negation-as-failure).[1] If $L = A$ (resp., $L = not\ A$) then $\overline{L}$ denotes *not* A (resp., A).

We generalize the notion of a-atom so to permit the description of preferences. A preference in resources usage is specified by means of a *preference-list* of a-atoms (*p-list*, for short). It is a writing of the form $q_1\#a_1 > \cdots > q_k\#a_k$, with $k \geqslant 1$. We say that $q_i\#a_i$ has *degree of preference* i in the p-list. Plainly, a-atoms are particular p-lists made of a single element (i.e., with $k = 1$).

An *r-literal* is either a p-literal or a p-list.

Notice that, we do not allow negation of a-atoms (cf., [4] for a discussion on this point). Finally, we distinguish between *p-rules* (plain ASP program rules, including the case of ASP *constraints*, i.e., rules with empty head) and *r-rules* which differ from p-rules in that they may contain p-lists (and hence, a-atoms).

An *r-rule* γ has the form

$$Idx: \quad H \leftarrow B_1, \ldots, B_m.$$

where $B_1, \ldots, B_m$ ($m > 0$) are r-literals, H is either a p-atom or a p-list, and at least one a-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, i.e., *fired*. Such a number must belong to *Idx* or the rule cannot be fired at all. 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.

Without loss of generality, in what follows we often assume $h = 1$ in r-rules.

An *r-fact* has the form $q\#a \leftarrow$, where $q\#a$ is a ground a-atom. It models a fixed amount of resource q that is available "from the beginning".

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

A *rule* is either a p-rule or an r-rule. An *r-program* is a finite set of rules.

The grounding of an r-program P is the set of all ground instances of rules (and facts) of P, obtained through ground substitutions over the constants occurring in P.[2]

Notice that in any r-program only a finite number of a-symbols of $\mathcal{C}_\mathcal{R}$ occurs, also because all r-facts must be ground. Hence, as far as a-atoms are concerned, a finite number of ground instances can be generated by the grounding process. This is because all instances of a-terms are among the instances of the terms occurring in p-atoms. A "smart" grounder for RASP would avoid generating instances of r-rule where variables occurring as a-terms are instantiated to constants of $\mathcal{C}_\mathcal{P}$ instead of constants of $\mathcal{C}_\mathcal{R}$. Such "wrong" instances are however both semantically and practically irrelevant (apart from the waste of space).

Semantics of RASP. Semantics of a (ground) r-program is determined by interpreting p-literals as usually done in ASP (namely, by exploiting stable model semantics) and a-atoms in an auxiliary algebraic structure Q. In principle, such a structure can be of full generality, provided that it supports operations and comparisons among amounts. For simplicity, we fix $Q = \mathbb{Z}$ as collection of *quantities* and consider given a mapping $\kappa : \mathcal{C}_R \to \mathbb{Z}$ that associates integers to a-symbols. Positive and negative integers will be used to model produced and consumed amounts, respectively. Moreover, to further simplify the treatment, we identify $\mathcal{C}_R$ with $\mathbb{Z}$ (and κ being the identity).

In the semantics for RASP proposed in [5, 4], r-rules are translated into fragments of a plain ASP program. As regards a-atoms, the notion of *allocation* is introduced to model the amounts of resources that are consumed/produced when an r-rule is *fired*. Intuitively, given a ground r-program P, an allocation for P is a mapping that assigns elements of Q to the a-symbols in P in such a way that for each r-symbol q, the overall sum of quantities allocated to (produced and consumed) a-atoms of the form $q\#a$ is not negative. A last component of an interpretation copes with the repeated firing of r-rules. (Clearly, multiple firings must be taken into account by the allocation.)

We have the following definition (see also Def. 2 in App. A):

Definition 1. *An* r-interpretation *for a (ground) r-program P is a triple $\mathcal{I} = \langle I, \mu, \xi \rangle$, where $I \subseteq \mathcal{A}(\Pi_P, \mathcal{C})$, μ is an allocation for P, and ξ is a mapping $\xi : P \to \mathbb{N}^+$.*

In Def. 1, I plays the role of a usual answer set assigning truth values to p-literals and ξ associates to each rule the number of times the r-rule is used.

An interpretation $\mathcal{I}$ for a ground r-program P determines which r-rules are fired. In particular, $\mathcal{I}$ is an *answer set* of P if:

- it satisfies all the p-rules in P and all the fired r-rules (in the usual way) as concerns their p-literals;

[2] As it is well-known, at present, almost all ASP solvers perform a preliminary grounding step as they are able to find the answer sets of ground programs only. Work is under way to overcome at least partially this limitation (cf., [6], for instance).

- for each p-list (and hence, a-atom) occurring in a fired r-rule, one of its a-atoms is selected;
- the correct amounts are allocated for all the selected a-atoms while null amounts are allocated for all other a-atoms;
- the global balance of each r-symbol is not negative (i.e., all consumed amounts have been also produced by rule firings or available from r-facts).

App. A provides a formal definition of answer set for RASP (Def. 3).

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

Clearly, different selections of a-atoms in p-lists originate different answer sets. To impose a preference order on such answer sets, any *preference criterion* can be used. Such a criterion should order the collection of answer sets by reflecting the (preference degrees in the) p-lists. Any criterion has to take into account that each rule determines a (partial) preference ordering on answer sets, and it should aggregate/combine all such "local" partial orders to obtain a global one. This yields the notion of *most preferred answer set* of an r-program. Simple criteria to rank the collection of answer sets are described in [5].

Complexity. Computational complexity of (ground) RASP has been assessed in [4] for the simplified case in which all p-lists are singletons (namely, a-atoms might occur in a program but no preferences are specified). In that case, NP-completeness of deciding whether an r-program has an answer set is shown by providing a polynomial translation of ground programs from RASP into ASP. A correspondence is then exhibited between the answer sets of the r-program and the answer sets of its ASP encoding. We will see in the sequel that the same result holds also when preferences are involved.

As regards the complexity of the problem of determining if a given literal is true in a most preferred answer set, different results are obtained depending on the specific preference criterion which is adopted. In [5] it is shown that complexity of credulous reasoning for RASP with preferences are in line with that of LPOD [3]. For instance, if a *Pareto*-like criterion is adopted, Σ_P^2-completeness of both RASP and LPOD is obtained.

2 Dealing with complex preferences

In this section we present a number of extensions of RASP that permit the specification of more complex forms of preference on resources usage. A first step in this direction has been made with the introduction of conditional p-lists in [5]. In what follows, we provide a generalization of such a proposal and establish a complexity result, thus settling an open problem raised in [5].

As a preliminarily step, we introduce two notions of "*compound*" resource. Namely, given the a-atoms $q_1\#a_1, \ldots, q_k\#a_k$, the occurrence of the writing $\{q_1\#a_1, \ldots, q_k\#a_k\}$ in an r-rule, simply denotes the set of all these a-atoms. This notation has to be intended conjunctively, i.e., all the a-atoms are simultaneously consumed/produced when the r-rule is fired.

Similarly, $\{q_1\#a_1;\cdots;q_k\#a_k\}$ denotes a collection of a-atoms whose production/consumption has to be intended disjunctively, i.e., when the r-rule is fired, exactly one of the listed a-atoms is non-deterministically chosen for consumption/production. Notice that, while the first notation is syntactic sugar, the second one introduces a novelty w.r.t. the language of [4, 5]. An ASP encoding of r-rules involving disjunctive compound resources can be designed. We do not describe here such an encoding, since it will be a particular cases of cp-lists (to be seen in Sect. 2.2).

2.1 Preferences between sets of a-atoms.

Let us slightly generalize the notion of p-list introduced in Sect. 1. Namely, we consider p-lists of the form $s_1>\cdots>s_k$, where each s_i is a compound resource, i.e., a set of a-atoms $\{q_{i,1}\#a_{i,1},\ldots,q_{i,k_i}\#a_{i,k_i}\}$. This form of p-list allows one to express preferences on set of a-atoms, instead of a single a-atoms. Hence, the intended meaning is that, for all j,ℓ, $j<\ell$, it is preferred to produce/consume all the resources in $\{q_{j,1}\#a_{j,1},\ldots,q_{j,k_j}\#a_{j,k_j}\}$ than all those in $\{q_{\ell,1}\#a_{\ell,1},\ldots,q_{\ell,k_\ell}\#a_{\ell,k_\ell}\}$. (Note that, by imposing $k_i=1$, for all $i\in\{1,\ldots,k\}$, we obtain p-lists as defined in Sect. 1.)

A second form of p-list involves compound resources of the kind $s_i=\{q_{i,1}\#a_{i,1};\cdots;q_{i,k_i}\#a_{i,k_i}\}$. In such cases, for all j,ℓ, $j<\ell$, it is preferred to produce/consume one of the resources in s_j, than one of those in s_ℓ. Situations in which both forms occur in the same p-list are handled coherently.

Let us now describe how an r-program P involving such kind of p-lists can be polynomially encoded into ASP. This encoding will show that adding preferences on resource usage in RASP does not affect its complexity. Let us focus on the case of a single p-list occurring in the body of a ground r-rule γ of P:

$$Idx:\quad H\leftarrow B_1,\ldots,B_m,s_1>\cdots>s_k$$

where, each for each $i\in\{1,\ldots,k\}$, $s_i=\{q_{i,1}\#a_{i,1},\ldots,q_{i,k_i}\#a_{i,k_i}\}$ and Idx is $[N_{1,1}\text{-}N_{1,2},\ldots,N_{h,1}\text{-}N_{h,2}]$, denoting a collection of disjoint integer intervals—i.e., $h\geqslant 1$, each $N_{j,b}\in\mathbb{N}^+$, and $N_{1,1}\leqslant N_{1,2}<N_{2,1}\leqslant N_{2,2}<\cdots<N_{h,1}\leqslant N_{h,2}$. (The cases of p-lists occurring in the head and/or involving disjunctive compound resources can be treated similarly.)

For each $i\in\{1,\ldots,k\}$ let aux_i be a fresh r-symbol not occurring elsewhere in the program. Then, we introduce the following r-rules:

$$(\gamma')\qquad Idx:\quad H\leftarrow B_1,\ldots,B_m,\ pl\#1\,,\ z\#\text{-}1.$$

$$(\gamma'')\qquad pl\#N_{h,2}.$$

$$(\gamma_i)\qquad [1\text{-}N_{h,2}]:\quad aux_i\#1\leftarrow q_{i,1}\#a_{i,1},\ldots,q_{i,k_i}\#a_{i,k_i},z\#1.\quad \text{for each } i\in\{1,\ldots,k\}$$

where pl and z are fresh r-symbols (note that an a-atom with negative amount in the body of an r-rule, actually denotes resource production, cf., [4, Sect. 6.2]). The rationale is as follows. The auxiliary resource z acts as a counter of the number of firings of γ': each time γ' is fired, an instance of z is produced and one instance of pl is consumed. Conversely, each resource aux_i is produced only if γ_i is fired and this can happen only by consuming one instance of z. Hence,

each firing of γ' corresponds to one firing of one of the γ_is. The r-fact γ'' ensures that γ' cannot be fired more than $N_{h,2}$ times.

Notice that γ'', γ'', and all of the γ_i do not involve preferences. Hence, their ASP encoding can be obtained through the translation introduced in [4]. We list here the relevant fragment of such encoding (the lack of space prevents us to provide the complete translation, the reader is referred to [4, Sect. 8.1], or to App. B, for a brief description):

$$
\begin{array}{lll}
(1) & r\_rule(n_{\gamma'}). & \\
(2) & firings(n_{\gamma'}, N_{i,1}..N_{i,2}). & \text{for } i \in \{1,\ldots,h\} \\
(3) & a\_atom(n_{\gamma'}, 0, pl, -1). & \\
(4) & a\_atom(n_{\gamma'}, 1, z, 1). & \\
(5) & \leftarrow \overline{B_i}, fired(n_{\gamma'}). & \text{for } i \in \{1,\ldots,m\} \\
(6) & H \leftarrow B_1, \ldots, B_m, fired(n_{\gamma'}). & \\
(7) & r\_rule(n_{\gamma''}). & \\
(8) & a\_atom(n_{\gamma''}, 0, pl, N_{h,2}). & \\
(9) & firings(n_{\gamma''}, 1). \qquad fired(n_{\gamma''}). & \\
(10) & r\_rule(n_{\gamma_i}). & \text{for } i \in \{1,\ldots,k\} \\
(11) & firings(n_{\gamma_i}, 1..N_{h,2}). & \text{for } i \in \{1,\ldots,k\} \\
(12) & a\_atom(n_{\gamma_i}, 0, aux_i, 1). & \text{for } i \in \{1,\ldots,k\} \\
(13) & a\_atom(n_{\gamma_i}, j, q_j, a_j). & \text{for } i \in \{1,\ldots,k\} \text{ and } j \in \{1,\ldots,k_i\} \\
(14) & a\_atom(n_{\gamma_i}, k_i + 1, z, -1). & \text{for } i \in \{1,\ldots,k\}
\end{array}
$$

By means of facts (1), (7), and (10), unique identifiers for r-rules are introduced. Facts *a_atom* lists all the resources involved in r-rules. The predicate *firings* is used to declare the admissible number of firings for each r-rule. Lines (5) and (6) relate the firing of a rule γ' to the satisfiability of its p-literals.

To complete the translation we impose a direct dependency between the uses of the resource pl and the resources $aux_1, \ldots, aux_k$. This is done by means of this fragment of ASP program (where *val* and *iter* act as domain predicates):

$$
\begin{array}{l}
auxres(n_{\gamma'}, pl). \\
res\_pl(n_{\gamma'}, pl, aux_i, i). \qquad \text{for } i \in \{1,\ldots,k\} \\
1\{use\_pl(n_{\gamma'}, aux_1, I, 1, pl), \ldots, use\_pl(n_{\gamma'}, aux_k, I, 1, pl)\}1 \leftarrow \\
\qquad use(n_{\gamma'}, 0, pl, -1 * NumFirings),\ iter(I), \\
\qquad I <= NumFirings, count(n_\gamma, NumFirings). \\
\\
res\_symb(Pl) \leftarrow auxres(G, Pl), r\_rule(G). \\
use(G, Pos, R, N) \leftarrow sum\_use\_pl(G, R, N), res\_pl(G, Pl, R, Grade), \\
\qquad r\_rule(G), val(N), a\_atom(G, Pos, Pl, 1), N! = 0. \\
sum\_use\_pl(G, R, N) \leftarrow N = sum\{A : use\_pl(G, R, Iter, Q), val(Q), iter(Iter)\} \\
\qquad res\_pl(G, Pl, R, Grade), r\_rule(G), val(N).
\end{array}
$$

Briefly, the facts in the first two lines associate each aux_i with the resource pl. The rule in the third line implements a correspondence between firings of γ' (i.e., uses of pl) and firings of γ_i (i.e., uses of aux_i). Being $NumFirings$ the number of times γ' is fired, this rule imposes that each time one instance of pl is used, (namely, once for each I, $1 \leqslant I \leqslant NumFirings$), exactly one instance of one resource among the aux_is is used (i.e., one of the facts $use\_pl(n_{\gamma'}, aux_i, I, 1, pl)$

is true). In turn, this forces one firing for the r-rule γ_i. (Notice the use of a cardinality constraint to force the truth of exactly one atom *use_pl*. The use of such a constraint could be avoided, but it allows a more succinct encoding. See, for instance, [2] for details on cardinality constraints and on how to surrogate them through usual ASP rules.) The r-rules in the last three lines extend the inference engine introduced in [4]. They are used to evaluate the balance of each resource occurring in the initial p-list. This is achieved by means of an aggregate literal (cf., [7, 9, 10]) that sums up the amounts of real resources (i.e., the q_is) corresponding to instances of the auxiliary resources (i.e., the aux_is).

Observe that the above sketched translation can be applied to treat each p-list in a program, independently from the treatment of the other p-lists. The resulting encoding involves the introduction of a number of ASP rules and literals which is polynomially bounded w.r.t. the length of the given r-program.

Consequently, by exploiting the completeness and soundness results of [4], we can establish a one-to-one correspondence between the answer set of an r-program and those of its ASP encoding. Hence, the complexity results mentioned in Sect. 1 are not affected by the presence of compound resources in p-lists.

2.2 Conditional p-lists

Conditional p-lists, called cp-lists, have been introduced in [5], where preferences between single a-atoms were considered. By proceeding in analogy to what done in Sect. 2.1, we introduce a generalization of cp-lists that admits preferences on compound resources. In this context, a cp-list is a writing of the form $(r \ \ \textbf{\textit{pref\_when}} \ \ L_1, \ldots, L_n)$, where r is a p-list, and $L_1, \ldots, L_n$ are p-literals.

The intended meaning of a cp-list occurring in the body of an r-rule ρ (the case of the head is analogous) is that, whenever ρ is fired, one of the (compound) resources s_i in $r = s_1 > \cdots > s_k$ has to be consumed. If the firing occurs in correspondence of an answer set that satisfies $L_1, \ldots, L_n$, then the choice is ruled by the preference expressed through r. Otherwise, if any of the L_i is not satisfied, a non-deterministic choice among the s_is is made. (Hence the conjunction $L_1, \ldots, L_n$ does not need to be satisfied in order to fire ρ.) More precisely, if $L_1, \ldots, L_n$ does not hold, the r-rule containing the cp-list becomes equivalent to k r-rules, each containing exactly one of the s_is, in place of the cp-list.

As before, let us focus on the case of a single cp-list occurring in the body of a ground r-rule ρ of P of the form:

$$Idx: \quad H \leftarrow B_1, \ldots, B_m, (s_1 > \cdots > s_k \ \ \textbf{\textit{pref\_when}} \ \ L_1, \ldots, L_n)$$

where each s_i and Idx are as before. As done in Sect. 2.1, ρ can be replaced by these r-rules (where p and np are fresh p-atoms, and pl is a fresh r-symbol):

$$\begin{array}{lll}
(\rho') & Idx: & H \leftarrow B_1, \ldots, B_m, \ pl\#1\,, z\#{-}1. \\
(\rho'') & Idx: & pl\#1 \leftarrow p, \ s_1 > \cdots > s_k, \ z\#1 \\
(\rho_i) & [1\text{-}N_{h,2}]: & pl\#1 \leftarrow np, \ s_i, \ z\#1. \qquad\qquad \text{for each } i \in \{1, \ldots, k\} \\
 & & p \leftarrow not \ \ np. \qquad\qquad np \leftarrow not \ \ p. \\
 & & \leftarrow np, L_1, \ldots, L_n. \\
 & & \leftarrow p, \overline{L_j}. \qquad\qquad \text{for each } j \in \{1, \ldots, n\}
\end{array}$$

Atoms p and np characterize the situations in which $L_1, \ldots, L_n$ are all satisfied or not, respectively. As before, z acts as a counter of the number of times ρ' is fired. Depending on the truth of p (resp., np), each firing of ρ' is forced to correspond to one firing of ρ'' (resp., of one among the ρ_is).

All the above rules, except ρ'', do not involve p-lists. Hence, we are left with the translation of ρ'', which can be completed as outlined in Sect. 2.1. As before, the ASP encoding can be generalized to treat r-rules involving more than one cp-list (even with different kinds of compound resources), always introducing a polynomially bounded number of (occurrences of) atoms.

Notice that, the effect of using a compound resource $\{q_1\#a_1; \cdots ; q_k\#a_k\}$ can be rendered by using the cp-list $(q_1\#a_1 > \cdots > q_k\#a_k \ \ \textbf{\textit{pref\_when}} \ \ p, not \ \ p)$, where p is any p-atom. Hence, its ASP encoding could be drawn from the one described so far. (A similar, approach is also viable when such a compound resource occurs within the scope of a cp-list.)

A second form of cp-list introduced in [5] is $(r \ \ \textbf{\textit{only\_when}} \ \ L_1, \ldots, L_n)$. In this case, the p-list is effective only when the condition holds. More specifically, if the r-rule 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 one of the resources in r. Which one is determined by the expressed preference. 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 in this case a generalization can be proposed to admit a p-list r involving compound resources. The ASP encoding of these enriched cp-lists largely coincides with the one described above. It is made of the r-rules ρ, ρ', together with the following one (in place of all ρ_is):

$$(\rho''') \qquad [1\text{-}N_{h,2}] : \quad pl\#1 \leftarrow np,\ z\#1.$$

We conclude by observing that also in this case it is immediate to verify that the introduction of cp-lists does not affect the complexity of RASP. Completeness and soundness of the encoding also follow from the analogous results in [4].

2.3 Expressing arbitrary preferences.

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

A simple example: next summer Jake would like to visit a foreign country. It might be that his sister Jill joins, but only if she does not get a job for the summer. Jack would like to go either to Brazil, France, Spain, Norway, or Iceland. He prefers Brazil the most. In case going to Brazil is not possible, he considers equally interesting visiting either France or Spain. The least preferred options are Norway and Iceland, but no preference is expressed between them, except in August, when Norway is preferred. This simple case of preference order, being not linear, cannot be modeled by p-lists.

P-sets are a generalization of p-lists that allows one to use any binary relation (not necessarily a partial order) in expressing (collections of alternative) p-lists.

A p-set may occur in any place where a p-list does and is a writing of the form:

$$\{q_1\#a_1, \ldots, q_k\#a_k \mid pred\}$$

where *pred* is a binary program predicate (defined elsewhere in the program).

Considering a specific answer set M, a particular extension is defined for the predicate *pred* (namely, the set of pairs $\langle a, b\rangle$ such that $pred(a, b)$ is true in M). Let X be the set of r-symbols $\{q_1, \ldots, q_k\}$. We consider the binary relation $R \subseteq X^2$ obtained by restricting to X the extension of *pred* 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.[3]

As mentioned, R does not need to be a partial order, e.g., for instance, it may involve cycles. In such cases, those resources that belong to the same cycle in R are considered equally preferable (e.g., France and Spain, in the above example). On the other hand, R might be a partial relation. So, there might exist elements on X that are incomparable (e.g., Norway and Iceland in July).

Because of the presence of incomparable resources and equivalent resources, R can be seen as a representation of a collection of p-lists, one for each possible total order on X compatible with R. In particular, in case of equally preferable options, a non-deterministic choice is made. Whereas, in case of incomparable options, one among the possible total ordering of these options is arbitrarily selected. In the above example, the extension of *pred* should include the pairs $\langle Brazil, France\rangle$, $\langle France, Spain\rangle$, $\langle France, Iceland\rangle$, $\langle France, Norway\rangle$, and $\langle Spain, France\rangle$ (plus $\langle Norway, Iceland\rangle$, if is it August). Consequently, the admissible linear orders, if it is not August, are:

$Brazil{>}Spain{>}Norway{>}Iceland.$ $\quad$ $Brazil{>}France{>}Norway{>}Iceland,$
$Brazil{>}Spain{>}Iceland{>}Norway,$ $\quad$ $Brazil{>}France{>}Iceland{>}Norway,$

while only the first two should be considered in August.

Let us consider the following r-rule η (where *Idx* is as before):

$$Idx: \quad H \leftarrow B_1, \ldots, B_m, \{q_1\#a_1, \ldots, q_k\#a_k \mid pred\}.$$

By introducing a fresh symbol *ps* (that is univocally associated with the p-set at hand), the r-rule η can be replaced by these r-rules:

$$\begin{array}{lll} (\eta') & Idx: & H \leftarrow B_1, \ldots, B_m,\ ps\#1. \\ (\eta'') & & ps\#N_{h,2}. \end{array}$$

These r-rules do not involve p-lists, hence they are treated as described in Sect. 2.1. To complete the translation we have to take into account each possible total order implicitly represented the extension of *pred* (i.e., its interpretation in the answer set at hand). To this aim the following ASP fragment is introduced to handle the specific p-set:

[3] Notice that, here, we are introducing a change in the syntax of RASP. Namely we are admitting r-symbols as arguments of p-literals.

(20) $dom(ps, q_i).\quad number(ps, i).\qquad \text{for } i \in \{1, \ldots, k\}$
(21) $pset\_name(ps) \leftarrow fired(n_\eta).$
(22) $trcl(ps, X, Y) \leftarrow pred(X, Y), X \neq Y, fired(n_\eta).$
(23) $1\{use\_pl(n_\eta, q_1, I, a_1, ps), \ldots, use\_pl(n_\eta, q_k, I, a_k, ps)\}1 \leftarrow$
$use(n_\eta, 0, ps, -1 * NumFirings),\ iter(I),$
$I <= NumFirings, count(n_\eta, NumFirings).$
(24) $res\_pl(n_\eta, ps, T, N) \leftarrow order(ps, T, N).$

In the above code, line (20) defines two domain predicates that enumerate the (relevant) arguments of *pred* (the elements in X) and a collection of possible indices (as we will see, different indices represent different preference degrees), respectively. The following rules extend the inference engine and are independent from the specific p-sets:

(30) $trcl(PS, X, Y) \leftarrow dom(PS, X), dom(PS, Y), dom(PS, Z),$
$trcl(PS, X, Z), trcl(PS, Z, Y), X \neq Y, pset\_name(PS).$
(31) $equiv(PS, T_1, T_2) \leftarrow trcl(PS, T_1, T_2), trcl(PS, T_2, T_1),$
$dom(PS, T_1), dom(PS, T_2), pset\_name(PS).$
(32) $1\{ord\_idx(PS, T, N) : number(PS, N)\}1 \leftarrow dom(PS, T), pset\_name(PS).$
(33) $used\_idx(PS, N) \leftarrow dom(PS, T), ord\_idx(PS, T, N),$
$number(Ps, N), pset\_name(PS).$
(34) $\leftarrow dom(Ps, T), ord\_idx(PS, T, N), number(PS, N),$
$N > 1, N_1 = N - 1, not\ \ used\_idx(PS, N_1), pset\_name(PS).$
(35) $\leftarrow dom(PS, T_1), dom(PS, T_2), ord\_idx(PS, T_1, N_1), ord\_idx(PS, T_2, N_2),$
$number(PS, N_1), number(PS, N_2), N_2 \leqslant N_1, T_1 \neq T_2,$
$trcl(PS, T_1, T_2), not\ \ trcl(PS, T_2, T_1), pset\_name(PS).$
(36) $\leftarrow not\ \ equiv(PS, T_1, T_2), dom(PS, T_1), dom(PS, T_2), ord\_idx(PS, T_1, N),$
$ord\_idx(PS, T_2, N), number(PS, N), T_1 \neq T_2, pset\_name(PS).$
(37) $ord\_idx(PS, T_2, N) \leftarrow equiv(PS, T_1, T_2), dom(PS, T_1), dom(PS, T_2),$
$ord\_idx(PS, T_1, N), number(PS, N), pset\_name(PS).$
(38) $order(PS, T, N) \leftarrow ord\_idx(PS, T, N), not\ \ other(PS, T, N), dom(PS, T),$
$number(PS, N), pset\_name(PS).$
(39) $other(PS, T, N) \leftarrow order(PS, T_2, N), T \neq T_2, dom(PS, T), dom(PS, T_2),$
$number(PS, N), pset\_name(PS).$

The rule at line (30), together with the one in line (22) (which appears in one instance for each specific p-set), evaluates the transitive closure of the relation R defined by *pred*. Line (31) determines the equivalences between r-symbols. Rules at lines (32)–(34) index the elements of X with consecutive (possibly repeated) integers. Lines (35)–(37) 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 r-symbols. Finally, rules (38)–(39) generate (compatibly with the extension of *pred*) all admissible orders for X. Each of these orders admits a corresponding p-list. Such indexing of r-symbols is used (in the context of a specific answer set) through the rules (23)–(24), seen before, analogously to what done for the translation of plain p-lists (cf., Sect. 2.1).

Example 2. Let us consider the example mentioned above. This is a fragment of r-program describing Jack's preferences on buying plane tickets:

$$
\begin{array}{l}
jack\_happy \leftarrow \{ticket(b)\#N, ticket(f)\#N, ticket(s)\#N, ticket(i)\#N, ticket(n)\#N \\
\qquad\qquad \mid jack\_pref\}, is\_summer, num\_of\_tickets(N). \\
jack\_pref(ticket(b), ticket(f)). \quad jack\_pref(ticket(f), ticket(i)). \\
jack\_pref(ticket(f), ticket(s)). \quad jack\_pref(ticket(s), ticket(f)). \\
jack\_pref(ticket(f), ticket(n)). \quad jack\_pref(ticket(n), ticket(i)) \leftarrow is\_august. \\
ticket(P)\#N \leftarrow money\#C, cost(P, C1), C = C1 * N, num\_of\_tickets(N). \\
cost(b, 9). \quad cost(f, 6). \quad cost(s, 5). \quad cost(i, 5). \quad cost(n, 6). \\
num\_of\_tickets(1) \leftarrow jill\_works. \quad num\_of\_tickets(2) \leftarrow not\ jill\_works.
\end{array}
$$

Here is another example involving different forms of preference specification:

Example 3. Assume that, in making a cake or some cookies, you might choose among different ingredients, and you have to consider some constraints due to possible allergy or diet. Note that the preference among *chocolate*, *nuts* and *coconut*, i.e., a particular p-list, is determined depending on the extension of the predicate *less_caloric*, which might be different for different answer sets and has to be established dynamically (through the predicate *calory*, defined elsewhere in the program).

$$
\begin{array}{l}
cake\#1 > cookie\#15 \leftarrow egg\#2, flour\#2, raisin\#4, \\
\qquad (\{aspartame\#1, skim\_milk\#6\} > \{sugar\#4, whole\_milk\#6\}\ \boldsymbol{pref\_when}\ diet), \\
\qquad (\{vanilla\#1; lemon\#2\} > cinnamon\#1\ \boldsymbol{only\_when}\ not\ allergy), \\
\qquad \{chocolate\#1, nuts\#1, coconut\#1 \mid less\_caloric\}. \\
less\_caloric(X, Y) \leftarrow calory(X, A), calory(Y, B), A < B. \\
calory(X, Y) \leftarrow ...
\end{array}
$$

3 Raspberry: a concrete implementation

The above-outlined translation from (ground) RASP programs into ASP has being implemented in a stand alone tool, named *raspberry.*[4] Raspberry, essentially consists in a compiler that, given an r-program, produces its pure ASP encoding. In turn, this encoding is joined to an ASP specification of an inference engine which performs the real reasoning on resources allocation (and that remains independent from the particular program at hand). The resulting ASP program is suitable to be processed by commonly available ASP-solvers (in particular, the input syntax for gringo and lparse are supported [1]).

The answer sets found by the ASP solver encode the solutions of the RASP problem.

As regards preference criteria (see [5]), the search for most preferred solutions is implemented by exploiting the optimization features offered, for example, by smodels and clasp.

[4] Actually, non-ground programs are also correctly translated, provided that all a-atoms are ground (i.e., variables occur only in p-literals). The treatment of generic non-ground r-rules is subject for future work.

The prototypical release of raspberry has been implemented in C++ and is available in `http://www.dipmat.unipg.it/~formis/raspberry`. Such a prototype is still under development, but covers all of the features described in this paper. As done in this paper, also in the implementation, we fix the algebraic structure Q (modeling amounts) to be the set $\mathbb{Z}$ of integer numbers. This is because commonly available ASP solvers offer operations on integers as built-in features. Refinements of the tool able to deal with other groups are a theme for future work. In the mentioned web page, together with the tool, some examples and a minimal documentation can be retrieved.

Since this concrete implementation strictly follows the translation described in this paper (which, in turn, is a refinement of the one described in [4]), both its correctness and its completeness derive from the results obtained in [4].

Concluding remarks

In this work, we started from RASP, an extension of ASP that supports reasoning about resources and their amounts as well as the specification, in form of rules, of those processes that produce and consume resources. We proposed several extensions of the RASP language in order to allow the specification of complex forms of preferences on resource allocation. We also shown that such new language constructs, far from being, in many cases, trivial syntactic sugar, do not imply increase in the computational complexity of the framework. In particular, the problem establishing the existence of an answer set for an r-program is still NP-complete. Consequently, the complexity of credulous reasoning for RASP, when such form of complex preferences are involved, is in line with that of LPOD [3].

The extension of (R)ASP we described is, to the best of our knowledge, an original proposal. This is so for, at least, two aspects. First, the kind of preference admitted in RASP have a *local scope*: each p-list (cp-list, or p-set) is seen in the context of a particular rule (which models a specific process in manipulating some resources). Consider, for instance, the r-program of Example 1, where completely antithetic orders are expressed in the two r-rules. Notice that both r-rules might be fired at the same time, since enough resources are available. 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.

A second novelty is represented by the possibility of dealing with non-linear preference orders in resource usage (cf., Sect. 2.3). In this sense, DLPOD [8] is a similar proposal, recently appeared in literature. In this case, pure ASP is considered and preferences are imposed on the truth of program literals occurring in heads of rules (in analogy to what done in LPOD [3]). (So, also in this proposal preferences have a global flavor.) These preferences, in turn, determine an ordering of the answer sets of a DLPOD program. In [8] non-linear preferences

are introduced by combining ordered and unordered disjunction in the same framework.

Clearly, RASP and DLPOD have different purposes and tend to solve different problems and consequently have different expressive power. In particular, notice that DLPOD has LPOD as special case. Hence, it is reasonable that RASP and DLPOD also differs as regards the computational complexity, for instance, of credulous reasoning. We shown that RASP is in line with LPOD while [8] conjectures Σ_P^3-completeness for DLPOD.

Acknowledgements The authors would like to thank Torsten Schaub and Sven Thiele for the support and the advices about the grounder gringo. The parser of raspberry plainly extends the one included in the distribution of gringo. This research has been partially supported by GNCS.

References

[1] Web references for some ASP solvers. ASSAT: `assat.cs.ust.hk`; Ccalc: `www.cs.utexas.edu/users/tag/ccalc`; Clasp: `potassco.sourceforge.net`; Cmodels: `www.cs.utexas.edu/users/tag/cmodels`; DeReS and aspps: `www.cs.uky.edu/ai`; DLV: `www.dbai.tuwien.ac.at/proj/dlv`; Smodels: `www.tcs.hut.fi/Software/smodels`.

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

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

[4] S. Costantini and A. 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`.

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

[6] A. Dal Palù, A. Dovier, E. Pontelli, and G. Rossi. GASP: ASP with lazy grounding. In *Proc. of LaSh08*, 2008.

[7] W. Faber, G. Pfeifer, N. Leone, T. Dell'Armi, and G. Ielpa. Design and implementation of aggregate functions in the DLV system. *Theory and Practice of Logic Programming*, To appear.

[8] P. Kärger, N. Lopes, A. Polleres, and D. Olmedilla. Towards logic programs with ordered and unordered disjunction. In *Proc. of ASPOCP'08 Workshop of ICLP08*, 2008.

[9] D. B. Kemp and P. J. Stuckey. Semantics of logic programs with aggregates. In *Prog. of the 1991 International Logic Programming Symposium*, pages 387–401, 1991.

[10] T. C. Son and E. Pontelli. A constructive semantic characterization of aggregates in answer set programming. *Theory and Practice of Logic Programming*, 7(3), 2007.

The material of the following sections is essentially drawn from [4, 5]. It has been recalled here for ease of the reader.

A A quick rush through the semantics of RASP

Semantics of a (ground) r-program is determined by interpreting p-literals as in ASP and a-atoms in an auxiliary algebraic structure that supports operations and comparisons. The rationale behind the proposed semantic definition is the following. On the one hand, we translate 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. On the other hand, an interpretation involves the allocation of actual quantities to a-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, relying on stable model semantics) as concerns its p-literals, and the correct amounts are allocated for the a-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 a-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 a-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 $\mathcal{C}_R$ with $\mathbb{Z}$ (and κ being the identity). This will not cause loss in the generality of the treatment.

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

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.

To deal with the disjunctive aspect of p-lists and to model the degrees of preference, we mark each a-atom with an integer index. For each p-list its composing a-atoms are associated, from left to right, with successive indices starting from 1. For single a-atoms, the index will always be 0. So, any a-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 a-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 RASP Programs. In what follows, we will apply a syntactical restriction on the form of the r-rules. Namely, we impose that each a-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., a-atoms in heads) by positive values, while negative values model consumed amounts (i.e., a-atoms in bodies). For each r-symbol q, the overall sum of quantities allocated to (produced and consumed) a-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 r-symbol occurring in P (considered as a set of rules)—is the following collection of mappings:

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

The rationale behind the definition of $\mathbb{S}_P$ is as follows. Let q be a fixed r-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 a-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 r-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 a-atom: each of them corresponds to a different amount of resource to be taken into account.

The following is a more precise definition of *r-interpretation* (cf.,Def. 1).

Definition 2. *An* r-interpretation *for a (ground) r-program P is a triple $\mathcal{I} = \langle I, \mu, \xi \rangle$, with $I \subseteq \mathcal{A}(\Pi_P, \mathcal{C})$, $\mu : \tau_R \to \mathbb{S}_P$, and ξ a mapping $\xi : P \to \mathbb{N}^+$.*

Intuitively: I plays the role of a usual answer set assigning truth values to p-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 p-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 p-literals satisfy the rule. We reflect the fact that the satisfaction of an r-rule γ depends on the truth of its p-literals by introducing a suitable fragment of ASP program $\widehat{\gamma}$. Let the r-rule γ have $L_1, \ldots, L_k$ as p-literals and $R_1, \ldots, R_h$ as a-atoms (or p-lists). The ASP-program $\widehat{\gamma}$ is so defined:

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

Def. 3, to be seen, states that in order to be a model, an r-interpretation that allocates non-void amounts to the r-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 a p-list (or, an a-atom) in r-rule γ. Let

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

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

Let $r\text{-}\mathit{head}(\gamma)$ and $r\text{-}\mathit{body}(\gamma)$ denote the multiset of p-lists occurring in the head and in the body of γ, respectively. To distinguish, in the representation, between a-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 a-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 a-atoms occurring in the body of γ are negative, as these resources are *consumed*. Vice versa, the quantities associated to a-atoms of the head are positive, as these resources are *produced*.

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

where c_1 (resp. c_2) ranges on all possible choice functions for $\mathit{setify}_h(\gamma)$ (resp. for $\mathit{setify}_b(\gamma)$).

To account for multiple firings, 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

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

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 a-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 an r-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. 3 exploits the set $\mathcal{R}^n(q, \gamma)$ to impose restrictions on the global resource balance, for each q.

Definition 3. *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*

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

The two disjuncts in Def. 3 correspond to the two cases: a) the rule γ is not fired, so null amounts are allocated to all its a-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 $\exists\, q \in \tau_R \left(\mu(q)(\gamma) \neq \emptyset\right)$ holds). Note 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 $\mathcal{R}^{\xi(\gamma)}(q, \gamma)$.

B An ASP encoding of RASP

Let $\Pi_{\mathcal{T}}$ be a set of fresh p-symbols with $\{\mathit{notfired}, \mathit{fired}, \mathit{use}, \mathit{r\_rule}, \mathit{a\_atom}\} \subseteq \Pi_{\mathcal{T}}$. Given a ground r-program S, let $S_{\mathcal{R}} \subseteq S$ be the set of r-rules in S and let $S_{\mathcal{P}} = S \setminus S_{\mathcal{R}}$ (i.e., the p-rules). For any set of ASP rules X, let $\mathit{atoms}(X)$ denote the set of all atoms occurring in X. For any ASP rule γ, let $\mathit{lits}^+(\gamma)$ (resp., $\mathit{lits}^-(\gamma)$) be the set of atoms occurring positively (resp., negatively) in the body of γ. Moreover, let $\mathit{head}(\gamma)$ be the set of atoms occurring in the head of γ.

A translation $\mathcal{T}$ from RASP into ASP is defined as follows. We start by univocally naming each r-rule γ in $S_{\mathcal{R}}$. This is done by introducing a fresh constant symbol r_γ (i.e., a constant not appearing elsewhere) and the fact:

$$\mathit{r\_rule}(r_\gamma). \tag{1}$$

Let the r-rule γ be of the form

$$q_0 \# a_0, \ldots, q_h \# a_h \leftarrow q_{h+1} \# a_{h+1}, \ldots, q_k \# a_k,\ L_1, \ldots, L_n.$$

for some $0 \leqslant h \leqslant k$, $n \geqslant 0$, and $(k-h)+n > 0$, with $L_1, \ldots, L_n$ p-literals. For each a-atom $q_i \# a_i$ in γ we introduce the fact:

$$\mathit{a\_atom}(r_\gamma, i, q_i, \hat{a_i}). \tag{2}$$

where $\hat{a_i} = a_i$ if $0 \leqslant i \leqslant h$ and $\hat{a_i} = -a_i$ if $h < i \leqslant k$. These facts represent in the ASP translation the a-atoms occurring in $S_{\mathcal{R}}$. The second argument of $\mathit{a\_atom}$ is needed in the ASP translation to distinguish among different occurrences of identical a-atoms of the r-rule. Recall, in fact, that multiple copies of the same a-atom must not be identified, since they corresponds to a different amount of resource. Keeping track of multiple copies of a-atoms reflects, in the translation into ASP code, the use of multisets in defining the semantics of r-programs.

As mentioned, the two disjoints of the formula in Def. 3 discriminate the two situations in which an r-rule γ is fired or not. These two situations are modeled in ASP through the two rules

$$\mathit{fired}(r_\gamma) \leftarrow \mathit{not}\ \ \mathit{notfired}(r_\gamma). \qquad \mathit{notfired}(r_\gamma) \leftarrow \mathit{not}\ \ \mathit{fired}(r_\gamma). \tag{3}$$

Whenever an r-rule γ is fired, all its resources are consumed/produced. We represent the fact that a certain amount a_i of a resource q_i (due to the a-atom $q_i \# a_i$ in γ), is actually used if and only if γ is fired, through the ASP rules (for each $i \in \{0, \ldots, k\}$):

$$\begin{array}{l} \mathit{use}(r_\gamma, i, q_i, \hat{a_i}) \leftarrow \mathit{fired}(r_\gamma), \mathit{a\_atom}(r_\gamma, i, q_i, \hat{a_i}). \\ \mathit{fired}(r_\gamma) \leftarrow \mathit{use}(r_\gamma, i, q_i, \hat{a_i}). \\ \mathit{notfired}(r_\gamma) \leftarrow \mathit{not}\ \ \mathit{use}(r_\gamma, i, q_i, \hat{a_i}). \end{array} \tag{4}$$

Finally, we impose that the firing of γ has to be enabled by the truth of the literals $L_1, \ldots, L_n$, through the ASP rules (for each $j \in \{1, \ldots, n\}$):

$$\mathit{aux}_{L_i,\gamma} \leftarrow \mathit{not}\ \ \mathit{aux}_{L_i,\gamma},\ \overline{L_i},\ \mathit{fired}(r_\gamma). \tag{5}$$

The translation $\mathcal{T}(\gamma)$ of γ is made of the rules (1) and (2)-(5):

$r\_rule(r_\gamma).$ $\quad$ $a\_atom(r_\gamma, i, q_i, \hat{a}_i).$

$fired(r_\gamma) \leftarrow not\ \ notfired(r_\gamma).$ $\quad$ $notfired(r_\gamma) \leftarrow not\ \ fired(r_\gamma).$

$use(r_\gamma, i, q_i, \hat{a}_i) \leftarrow fired(r_\gamma), a\_atom(r_\gamma, i, q_i, \hat{a}_i).$ $\quad$ $fired(r_\gamma) \leftarrow use(r_\gamma, i, q_i, \hat{a}_i).$

$notfired(r_\gamma) \leftarrow not\ \ use(r_\gamma, i, q_i, \hat{a}_i).$ $\quad$ $aux_{L_i,\gamma} \leftarrow not\ \ aux_{L_i,\gamma}, \overline{L_i}, fired(r_\gamma).$

where i and j range over $\{0, \ldots, k\}$ and $\{1, \ldots, n\}$, respectively.

The above described translation can to be slightly modified to treat r-rules having a p-atom as head (see [4] for the details). Facts are treated as r-rules that are supposed to be always fired. Hence, if γ is the r-fact $q\#a.$ then $\mathcal{T}(\gamma)$ is as follows:

$$\begin{array}{ll} r\_rule(r_\gamma). & a\_atom(r_\gamma, 0, q, a). \\ fired(r_\gamma). & use(r_\gamma, 0, q, a). \end{array} \tag{6}$$

By defining $\mathcal{T}(\gamma) = \{\gamma\}$ for all p-rules, the translation of an r-program S is defined as the ASP program $\mathcal{T}(S) = \bigcup_{\gamma \in S} \mathcal{T}(\gamma)$.

Let M be a model of the program $\mathcal{T}(S)$. For some r-symbols q, some of the atoms $use(r_\gamma, i, q, a)$, occurring in $\mathcal{T}(S)$, are true in M. These atoms are intended to represent the amounts of resources involved in fired r-rules. To take into account the constraints on global balance of the allocated amounts, we introduce the condition $pos(M)$:

$$pos(M) = \forall q \in \tau_{\mathcal{R}}\ \left(\sum \{\!\{ a \mid use(r_\gamma, i, q, a) \in M \}\!\} > 0 \right) \tag{7}$$

Such a condition can be imposed in the ASP encoding by means of a constraint involving an aggregate *sum*, as follows (cf., among others [9, 10, 7]):

$$\leftarrow sum\{A : use(Rule, I, Q, A)\} < 0,\ res\_symb(Q).$$

Moreover, for each r-symbol q occurring in $\mathcal{T}(P)$ a fact $res\_symb(q)$ is added to the encoding. Observe that we are introducing an aggregate literal in a constraint. Hence, no literal in $\mathcal{T}(S)$ is defined depending on such aggregate. This ensures that the resulting program is aggregate stratified. Stable model semantics can be smoothly extended to such class of programs by generalizing the notion of reduct of a program (cf., [9, 10, 7]).

An inference engine for RASP. The above outlined translation can be ameliorated, since the rules (3) and (4) can be factorized by exploiting variables. This allows us to design the core of an inference engine for reasoning on resource allocations and imposeing correct usage of resources:

$fired(Rule) \leftarrow not\ \ notfired(Rule),\ r\_rule(Rule).$

$notfired(Rule) \leftarrow not\ \ fired(Rule),\ r\_rule(Rule).$

$use(Rule, I, Res, Amount) \leftarrow fired(Rule), a\_atom(Rule, I, Res, Amount).$

$fired(Rule) \leftarrow use(Rule, I, Res, Amount).$

$notfired(Rule) \leftarrow not\ \ use(Rule, I, Res, Amount).$

The balance for each resource is evaluated by the following fragment of code. Observe the use of an ASP constraint involving an aggregate function to evaluate sums and to impose the condition (7):

$$res\_symb(Res) \leftarrow a\_atom(Rule, I, Res, Amount).$$
$$\leftarrow sum\{A : use(Rule, I, Q, A)\} < 0,\ res\_symb(Q).$$

An ASP-based solver for RASP would act as follows: first each r-rule of the RASP program is translated as previously explained (recall that p-rules are left unchanged); then, the rendering of all r-rules must be joined with the above ASP program that acts as an inference engine and performs the concrete reasoning activity on resource allocations; finally, the answer sets (if any) of the obtained ASP program are calculated by means of a standard ASP solver [1]. From each answer set M so computed, an answer set $\mathcal{I}_M$ of the original r-program can be extracted.