Strong Equivalence of RASP Programs

Stefania Costantini[1], Andrea Formisano[2], and David Pearce[3]

[1] Università di L'Aquila, Italy `stefania.costantini@univaq.it`
[2] Università di Perugia, Italy `formis@dmi.unipg.it`
[3] Universidad Politècnica de Madrid, Spain `david.pearce@upm.es`

Abstract. RASP is a recent extension of Answer Set Programming (ASP) that permits declarative specification and reasoning on consumption and production of resources. In this paper, we extend the concept of strong equivalence (which, as widely recognized, provides an important conceptual and practical tool for program simplification, transformation and optimization) from ASP to RASP programs and discuss its applicability, usefulness and implications in this wider context.

Introduction

Issues related to production, consumption and exchange of resources are at the basis od all human activities, and in particular of economy. "Intelligent" resource management becomes increasingly important in view of sustainability problems that mankind has to face.

In knowledge representation and reasoning, forms of *quantitative* reasoning are possible in Linear Logics [10] and Description Logics [1]. In logic programming, a number of Prolog-like logic programming languages based on linear logic have been proposed (cf. references and discussion in [4]). In Answer Set Programming (ASP for short), a form of resource treatment is described in [17, 16] to model product configuration problems. This framework is based on Weight Constraint Rules, which is a well-known construct encompassing default negation and disjunctive choices [12]. Weight Constraint Rules have a wide applicability in many applications and are able to express costs and limits on costs, where however they do not express directly resource consumption/production. Resources are rendered, in the action description language $\mathcal{CARD}$ [2], through multi-valued fluents and the use of resources is implicitly modeled by the changes in fluents' values caused by actions' executions. The approach emphasizes the use of resources in planning problems and the semantics is given in terms of transition systems (in the spirit of [9]).

RASP (standing for ASP with Resources) [4, 3, 5] is an extension of ASP supporting declarative reasoning on consumption and production of resources. Theoretical and practical results developed for ASP can be extended to RASP: in this paper in fact, we extend the concept of strong equivalence to RASP programs and discuss its usefulness.

Strong equivalence [13, 11], as widely recognized, provides an important conceptual and practical tool for program simplification, transformation and optimization. Even in the case where two theories are formulated in the same vocabulary, they may have the same answer sets yet behave very differently once they are embedded in some larger context. For a robust or modular notion of equivalence one should require that programs behave similarly when extended by any further programs. This leads to the concept of strong equivalence, where programs P_1 and P_2 are strongly equivalent if and only if for any S, $P_1 \cup S$ is equivalent to (has the same answer sets as) $P_2 \cup S$. It is easy to see that, whenever P_1 and P_2 are different formulations of a RASP program involving consumption and production of resources, their behaving equivalently in different contexts is of particular importance related to reliability in resource usage. For instance, a designer might be able to evaluate, in terms of strong equivalence, different though analogous processes for producing certain resources, so as to choose one rather than the other in terms of suitable criteria.

In order to extend the notion of strong equivalence to RASP, we need to reformulate the semantics of RASP programs as introduced in [4] (based on a notion of resource "allocation") in a form less elegant but similar to that of plain ASP programs. This is done in Section 2 after an introduction to RASP provided in Section 1. After that, the definition of strong equivalence developed in [11] can be fairly easily extended to RASP (Section 3). However, it turns out that RASP programs behave quite differently from ASP programs as far as strong equivalence is concerned, as interferences in resource usage among rules of P_1, P_2 and S can easily arise. Then, we will argue (Section 4) that a significant notion of strong equivalence for RASP programs requires to state some constraints on S. So done, strong equivalence becomes a more effective notion in the RASP context. Finally, in Section 5 we conclude and outline some possible future directions. We assume a reader to be familiar with both ASP and strong equivalence. The reader may refer for the former to [7] and to the references therein, and for the latter to [15]. An extended version of this papers including proofs (that are omitted here for lack of space) can be found in [6].

1 Background on RASP

RASP [4, 3, 5] is an extension of the ASP framework obtained by explicitly introducing the notion of *resource*. It 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:

$$\mathit{computer}\!:\!1 \leftarrow \mathit{cpu}\!:\!1, \mathit{hd}\!:\!2, \mathit{motherboard}\!:\!1, \mathit{ram\_module}\!:\!2.$$

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 may involve plain ASP literals together with amount-atoms. Semantics for RASP programs is given by combining answer set semantics with a notion of *allocation*. While answer sets 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. Thus, one has to choose a collection Q of *quantities*, the operations to combine and compare quantities, and a mapping that associates quantities to amount-symbols. Admissible allocations are those satisfying, for all resources, the requirement that one can consume only what has been produced. Alternative allocations might be possible. They correspond to different ways of using the same resources. A simple natural choice for Q is the set of integer numbers. In all the examples proposed in the rest of the paper, we implicitly make this choice.

Syntax and semantics of RASP were introduced in [4]. Various extensions, presented in [4] and in [3, 5] (which discuss preferences and complex preferences in RASP) are not considered here. An implementation of RASP is discussed in [5] and is available at `http://www.dmi.unipg.it/formis/raspberry/`.

RASP syntax is based upon partitioning the symbols of the underlying language into program symbols and resource symbols. Precisely, let $\langle \Pi, \mathcal{C}, \mathcal{V} \rangle$ be a structure where $\Pi = \Pi_{\mathcal{P}} \cup \Pi_{\mathcal{R}}$ is a set of predicate symbols such that $\Pi_{\mathcal{P}} \cap \Pi_{\mathcal{R}} = \emptyset$, $\mathcal{C} = \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{R}}$ is a set of constant symbols such that $\mathcal{C}_{\mathcal{P}} \cap \mathcal{C}_{\mathcal{R}} = \emptyset$, and $\mathcal{V}$ is a set of variable symbols. The elements of $\mathcal{C}_{\mathcal{R}}$ are said *amount-symbols*, while the elements of $\Pi_{\mathcal{R}}$ are said *resource-predicates*. The elements of $\mathcal{C}_{\mathcal{P}}$ and $\Pi_{\mathcal{P}}$ are constant and predicate symbols like in plain ASP. A *program-term* is either a variable or a constant symbol. An *amount-term* is either a variable or an amount-symbol. The second step is that of introducing amount-atoms in addition to plain ASP atoms, called program-atoms. Let $\mathcal{A}(X, Y)$ denote the collection of all atoms of the form $p(t_1, \ldots, t_n)$, with $p \in X$ and $\{t_1, \ldots, t_n\} \subseteq Y$. Then, a *program-atom* is an element of $\mathcal{A}(\Pi_{\mathcal{P}}, \mathcal{C} \cup \mathcal{V})$. Differently from program-atoms, each *amount-atom* explicitly denotes a resource and an amount. More precisely, an *amount-atom* is an expression of the form $q{:}a$ where $q \in \Pi_{\mathcal{R}} \cup \mathcal{A}(\Pi_{\mathcal{R}}, \mathcal{C} \cup \mathcal{V})$ and a is an amount-term. Let $\tau_{\mathcal{R}} = \Pi_{\mathcal{R}} \cup \mathcal{A}(\Pi_{\mathcal{R}}, \mathcal{C})$. We call elements of τ_R *resource-symbols*.

Expressions such as $p(X){:}V$ where V, X are variable symbols are allowed, as resources amounts can be either directly specified as constants or obtained via some kind of computation. Notice that the set of variables is not partitioned, as the same variable may occur both as a program term and as an amount-term. *Ground* amount- or program-atoms contain no variables. As usual, a *program-literal* L is a program-atom A or the negation $not\, A$ of a program-atom (intended as negation-as-failure).[4] A *resource-literal* is either a program-literal or an amount-atom. Notice that, for reasons discussed in [4], we do not admit negation of amount-atoms.

[4] In this paper we only deal with negation-as-failure. Nevertheless, classical negation of program literals could be used in RASP and treated as usually done in ASP.

Finally, we distinguish between plain rules and rules that involve amount-atoms. In particular, a *program-rule* is defined as a plain ASP rule. Besides program-rules we introduce resource-rules which differ from program rules (which are usual ASP rules) in that they may contain amount-atoms. A *resource-proper-rule* has the form $H \leftarrow B_1, \ldots, B_k$. where $B_1, \ldots, B_k$, are resource-literals and H is either a program-atom or a (non-empty) list of amount-atoms. If H is an amount-atom of the form $q{:}a$ where a is a constant and the body is empty then the rule is called a *resource-fact*. According to the definition then, the amount of an initially available resource has to be explicitly stated. As usual, we often denote the resource-fact $H \leftarrow$ simply by writing H.

In general, we admit several amount-atoms in the head of a rule where the case in which a rule γ has an empty head is admitted only if γ is a program-rule (i.e., γ is an ASP *constraint*). The list of amount-atoms composing the head of an resource-rule has to be understood conjunctively, i.e., as a collection of those resources that are all produced at the same time by *firing*, i.e. applying, the rule.

A *resource-rule* (*r-rule*, for short) can be either a resource-proper-rule or a resource-fact. A RASP program may involve both program rules and resource-rules, i.e., a *RASP-rule* (*rule*, for short) γ is either a program-rule or a resource-rule and a RASP program (*r-program*) is a finite multiset of RASP-rules (because in principle an r-rule may occur more than once in a program: in this case, each "copy" of the rule can be separately applied).

The ground version (or "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 customary, in what follows we will implicitly refer to the ground version of P. Intuitively, an interpretation of P is an answer set whenever it satisfies all the program rules in P and all the *fired* r-rules (in the usual way) as concerns their program-literals, and all consumed amounts either were available from resource-facts or have been produced by rule firings.

Example 1. Below is an example of a RASP program.

$g{:}2 \leftarrow q{:}4.$ $\qquad$ $h{:}1 \leftarrow q{:}1.$ $\qquad$ $f{:}3 \leftarrow q{:}2.$ $\qquad$ $q{:}4.$

This program has the following answer sets. The answer set $\{q{:}4, g{:}2, -q{:}4\}$, where we employ (consume) resource $q{:}4$ (as indicated by the notation $-q{:}4$) to *fire* (i.e., apply) the first rule and produce $g{:}2$, with no remainder (the full available amount of q is consumed). The answer set $\{q{:}4, f{:}3, h{:}1, -q{:}3\}$, where we employ (consume) part of resource $q{:}4$ (as indicated by the notation $-q{:}3$) to fire the second and third rule and produce $f{:}3$ and $h{:}1$. In this case, as we do not have consumed the full amount of q, there is a remainder $q{:}1$ that might have been potentially used elsewhere. We cannot produce $g{:}2$ together with $f{:}3$ and/or $h{:}1$ because the available quantity of q is not sufficient. But, we also have the answer sets $\{q{:}4\}$, $\{q{:}4, f{:}3, -q{:}2\}$ and $\{q{:}4, h{:}1, -q{:}1\}$ where all or part of resource $q{:}4$, though available, is left unconsumed.

Notice that the given program does not involve negation. In plain ASP we may have several answer sets only if the program involves negation, and, in particular, cycles on negation. In RASP, we may have several answer sets also in positive (i.e., definite) programs because of different possible allocations of available resources. Notice also that in the present setting each rule can be applied only once, i.e, we cannot for instance use the third rule several times to produce several items of h:1 according to the available qs. Actually, multiple "firings" of rules can be allowed by a suitable specification but we do not consider this extension here.

In [4] we introduce *politics* for resource usage, by extending the semantics so as to allow a programmer to state, for each rule, if the firing is either optional or mandatory. In the rest of this paper, to simplify the discussion, we make the assumption that the firing of rules is mandatory. In the above example, this politics excludes the answer sets $\{q{:}4\}$, $\{q{:}4, f{:}3, -q{:}2\}$ and $\{q{:}4, h{:}1, -q{:}1\}$. In the next section, we introduce a version of the semantics of RASP close to the one originally defined for plain ASP, i.e., in terms of a reduct (for ASP, the so-called Gelfond-Lifschitz reduct, or GL-reduct, introduced in [8]).

2 Reduct-based RASP Semantics

In order to extend the notion of strong equivalence to RASP, it is useful to devise a semantic definition for RASP as close as possible to the standard answer set semantics as defined in [8]. This will make it easier to extend to RASP the definitions and proofs provided in [11].

In fact, in [4], the formulation of RASP semantics is based on set theory and on a concept of "resource allocation". In this section, we propose a more "practical" version, close to standard ASP. This because, for the sake of conceptual clarity, we intend to define strong equivalence of RASP programs by extending step-by-step the original formalization of [11], which is heavily based upon the notion of reduct as introduced in [8]. We thus introduce for RASP an extended notion of reduct and propose an alternative RASP semantics accordingly, that we prove to be equivalent to the original one.

We base this new semantics on standardized-apart RASP programs, whose definition was originally introduced in [4] in order to evaluate RASP complexity (which turned out to be the same as plain ASP). This involves generating, from a ground r-program, a version where resource-predicates occurring in bodies of rules are associated to the rule where they occur. For the sake of simplicity and without loss of generality, we assume that a resource predicate, say q, may occur more than once in the body of the same rule only if the amounts are different. It may occur instead with either the same or different amounts in the head and body of several rules. Below, let a "program" be a RASP program. The answer sets for a program obtained with the formulation proposed in this section are called *reduct-answer sets* (for short, *r-answer sets*). Specifically, in order to cope with different instances of the same amount-atom, say $q{:}a$, occurring in the body of different rules, we "standardize apart" these occurrences, according to the following definition (reported from [4]).

Definition 1. *Let P be a ground r-program, and let $\gamma_1, ..., \gamma_k$ be the rules in P containing amount-atoms in their body. The standardized-apart version P_s of P*

is obtained from P by renaming each amount-atom $q{:}a$ in the body of γ_j, $j \leqslant k$ as $q^j{:}a$. The q^j's are called the standardized-apart versions of q, or in general standardized-apart resource-predicates.

Example 2. Let P be the following program.

$$\begin{array}{ll}
g{:}2 \leftarrow q{:}4. & a \leftarrow not\ b. \\
p{:}3 \leftarrow q{:}3, a. & b \leftarrow not\ a. \\
d{:}1 \leftarrow q{:}3, b. & c. \\
q{:}6 \leftarrow c. &
\end{array}$$

The standardized-apart version P_s of P is as follows.

$$\begin{array}{ll}
g{:}2 \leftarrow q^1{:}4. & a \leftarrow not\ b. \\
p{:}3 \leftarrow q^2{:}3, a. & b \leftarrow not\ a. \\
d{:}1 \leftarrow q^3{:}3, b. & c. \\
q{:}6 \leftarrow c. &
\end{array}$$

We let $\mathcal{A}_{P_s}$ be the set of all atoms (both program-atoms and amount-atoms) that can be built from predicate and constant symbols occurring in P_s. Notice that not only the same r-rule might occur more than once in the same program, but also the same amount-atom might occur more than once in the body of an r-rule.

Definition 2. *A candidate reduct-interpretation $\mathcal{I}_{P_s}$ for P_s is any multiset obtained from a subset of $\mathcal{A}_{P_s}$.*

Referring to Example 2, among the possible candidate interpretations are, e.g., $I_1 = \{p{:}3, q{:}6, q^2{:}3, a, c\}$ and $I_2 = \{p{:}3, q{:}6, q^1{:}4, q^3{:}3, a, c\}$. Standardized-apart amount-atoms occurring in a candidate r-interpretation represent resources that are *consumed*. Plain amount-atoms represent resources that have been produced, or were available from the beginning. For a candidate r-interpretation to be an admissible r-interpretation (or, simply, r-interpretation), consumption has not to exceed production.

Definition 3. *Given multi-set of atoms S and resource-predicate q (possibly standardized-apart) occurring in S, let $f(S)(q)$ be an amount-symbol obtained by summing the quantities related to the occurrences of q. If S contains $q{:}a_1$, $\ldots$, $q{:}a_k$ (or, respectively, $q^j{:}a_1$, $\ldots$, $q^j{:}a_k$) and $a = a_1 + \ldots + a_k$ we will have $f(S)(q) = a$ (or, respectively, $f(S)(q^j) = a$).*

Definition 4. *A candidate reduct-interpretation $\mathcal{I}_{P_s}$ is a reduct-interpretation (for short r-interpretation) if for every resource-predicate q occurring in $\mathcal{I}_{P_s}$, taken all its standardized-apart versions $q^{j_1}, \ldots, q^{j_h}$, $h \geq 0$, also occurring in $\mathcal{I}_{P_s}$, we have $f(\mathcal{I}_{P_s})(q) \geq \sum_{i=1}^{h} f(\mathcal{I}_{P_s})(q^{j_i})$.*

Referring again to Example 2, it is easy to see that I_1 is an r-interpretation, as 6 items of q are produced and just 3 are consumed, while I_2 is not, as 6 items of q are produced but 7 are supposed to be consumed.

We now establish whether an r-interpretation $\mathcal{I}_{P_s}$ is an r-answer set for P_s, and we will then reconstruct from it an r-answer set for P. To this aim, we introduce the following extension to the Gelfond-Lifschitz reduct.

Definition 5 (RASP-reduct). *Given a reduct-interpretation $\mathcal{I}_{P_s}$ for the standardized-apart version P_s of RASP program P, the RASP-reduct $cfp(P_s, \mathcal{I}_{P_s})$ is a RASP program obtained as follows.*

1. *For every standardized-apart amount-atom $A \in \mathcal{I}_{P_s}$, add A to P_s as a fact, obtaining $P_s^+(\mathcal{I}_{P_s})$;*
2. *Compute the GL-reduct of $P_s^+(\mathcal{I}_{P_s})$.*

Let $LM(T)$ be the Least Herbrand Model of theory T. In case T is a RASP program, in computing $LM(T)$ amount-atoms are treated as plain atoms. We may state the main definition:

Definition 6. *Given reduct-interpretation $\mathcal{I}_{P_s}$ for the standardized-apart version P_s of RASP program P, $\mathcal{I}_{P_s}$ is a reduct-answer set (r-answer set) of P_s if $\mathcal{I}_{P_s} = LM(cfp(P_s, \mathcal{I}_{P_s}))$*

Referring again to Example 2, the r-answer sets of P_s are:
$M_1 = \{q{:}6, g{:}2, q^1{:}4, a, c\}$, $M_2 = \{q{:}6, g{:}2, q^1{:}4, b, c\}$,
$M_3 = \{q{:}6, p{:}3, q^2{:}3, a, c\}$, $M_4 = \{q{:}6, d{:}1, q^3{:}3, b, c\}$.

Notice that M_1 and M_2 have the same resource consumption and production but from the even cycle on negation they choose a different alternative (a w.r.t. b). Producing $g{:}2$ excludes being able to produce $p{:}3$ or $d{:}1$ respectively, because the remaining quantity of q is not sufficient. In the terminology of previous section, $g{:}2$ is produced by *firing* the first r-rule, while the second and third ones remain *unfired.* Instead of producing $g{:}2$, one can produce either $p{:}3$ by firing the second r-rule (answer set M_3) if choosing the alternative a or $d{:}1$ by firing the third r-rule (answer set M_4) if choosing the alternative b.

It will be useful in the next sections to define the *set of unfired rules* of a RASP program P w.r.t. any of its answer sets (say M) as the subprogram of P consisting of all r-rules whose head is not in M. We define two RASP programs P_1 and P_2 to be *equivalent* if they have the same r-answer sets and to be *tightly equivalent* if they are equivalent and, for each of their answer sets (say M) they have the same set of unfired rules w.r.t. M.

We now have to prove the equivalence of the above-proposed semantic formulation with the one of [4]. We will prove in particular that there is a bijection between the set of the r-answer sets of the standardized-apart version P_s of RASP program P and the set of RASP answer sets as defined in [4]. To do that, we exploit the part of [4] where, for determining the complexity of RASP, a bijection is introduced between RASP answer sets and *admissible* answer sets (in the usual answer set semantics) of an ASP version of the given RASP program P, called *adapted program* (where admissible answer sets are those satisfying the constraint that resource production must exceed consumption). The adapted program augments a standardized-apart program so as to simulate the allocation of a resource to a rule. Below is the definition, reported from [4].

Definition 7. *The* adapted program P' *for P (corresponding to P_s) is obtained by adding to P_s for each $q^j{:}a$ occurring in the body of some r-rule of P_s and such that $q{:}b$ (for some b) occurs in the head of some r-rule of P_s the following pair of rules (where $no\_q^j{:}a$ is a fresh atom): $q^j{:}a \leftarrow$ not $no\_q^j{:}a$. $no\_q^j{:}a \leftarrow$ not $q^j{:}a$.*

So, to our aim it suffices to prove that a bijection exists between the answer sets of P_s and the admissible answer sets of P'. This in fact implies that there exists a bijection between the set of RASP answer sets of ground RASP program P and the set of r-answer sets of the standardized-apart version P_s of P.

In order to compare r-answer sets with admissible answer sets we have to make their form compatible. In particular, interpretations of an adapted program will thus contain, for resource q, either $q^j{:}a$ or $no\_q^j{:}a$ to signify availability or, respectively, unavailability of amount a of this resource for consumption by the j-th rule. Below we transform an r-interpretation into this form.

Definition 8. *Given an r-interpretation $\mathcal{I}_{P_s}$ for the standardized-apart version P_s of RASP program P, $ad(\mathcal{I}_{P_s})$ is obtained from $\mathcal{I}_{P_s}$ by adding $no\_q^j{:}a$ for every amount symbol $q^j{:}a$ which occurs in P_s but not in $\mathcal{I}_{P_s}$.*

We can now state the result we where looking for, i.e.,

Theorem 1. *Given an r-interpretation $\mathcal{I}_{P_s}$ for the standardized-apart version P_s of RASP program P, $\mathcal{I}_{P_s}$ is an r-answer set of P_s if and only if $ad(\mathcal{I}_{P_s})$ is an admissible answer set of the adapted program P' corresponding to P_s.*

and, consequently:

Corollary 1. *There exists a bijection between the set of RASP answer sets of ground RASP program P and the set of r-answer sets of the standardized-apart version P_s of P.*

At this stage, as we wish to obtain r-answer sets of P, we can get a more compact form by getting the global consumed/produced quantity of each resource q.

Definition 9. *Given an r-answer set $\mathcal{I}_{P_s}$ of P_s, an r-answer set $\mathcal{M}_P^R$ of P is obtained as follows. For each resource-predicate q occurring in $\mathcal{I}_{P_s}$:*
(i) replace its occurrences $q{:}b_1, \ldots, q{:}b_s$, $s > 0$, with $q{:}b$, for $b = b_1 + \ldots + b_s$;
(ii) replace its standardized-apart occurrences $q^{j_1}{:}a_1, \ldots, q^{j_k}{:}a_k$, $k \geq 0$, with $-q{:}a$, for $a = a_1 + \ldots + a_k$.

By some abuse of notation, we will often interchangeably mention r-answer sets or simply "answer sets" of P or of P_s. For Example 2, the r-answer sets of P are:
$M_1 = \{q{:}6, g{:}2, -q{:}4, a, c\}$, $M_2 = \{q{:}6, g{:}2, -q{:}4, b, c\}$,
$M_3 = \{q{:}6, p{:}3, -q{:}3, c\}$, $M_4 = \{q{:}6, d{:}1, -q{:}3, b, c\}$.

Notice that the above notation does not explicitly report about what is left. Actually, given each r-answer set we can establish which resources are still available after the production/consumption process by computing the difference, for each resource, between what has been produced and what has been consumed. For instance, in M_1 and M_2 we are left with $q{:}2$ and $g{:}2$, in M_3 with $q{:}3$ and $p{:}3$, and in M_4 with $q{:}3$ and $d{:}1$.

3 Strong Equivalence of RASP programs

In this section, we extend the standard notion of strong equivalence to RASP programs, taking as a basis the definitions that can be found in [11], which provides a characterization of strong equivalence of ground programs in terms of the propositional logic of here-and-there (HT-logic). We remind the reader that the logic of here-and-there is an intermediate logic between intuitionistic logic and classical logic. Like intuitionistic logic it can be semantically characterized by Kripke models, in particular using just two worlds, namely *here* and *there*, assuming that the *here* world is ordered before the *there* world. Accordingly, interpretations (HT-interpretations) are pairs (X, Y) of sets of atoms from given language L, such that $X \subseteq Y$. An HT-interpretation is total if $X = Y$. The intuition is that atoms in X (the *here* part) are considered to be true, atoms not in Y (the *there* part) are considered to be false, while the remaining atoms (from $Y \setminus X$) are undefined. A total HT-interpretation (Y, Y) is called an equilibrium model of a theory T, iff $(Y, Y) \models T$ and for all HT-interpretations (X, Y), such that $X \subset Y$, it holds that $(X, Y) \not\models T$. For an answer set program P, it turns out that an interpretation Y is an answer set of P iff (Y, Y) is an equilibrium model of P reinterpreted as an HT-theory.

We take as a basis the standardized-apart version P_s of RASP program P. Notice that we assume two RASP programs to have the same alphabet (an so to be potentially equivalent and strongly equivalent) if they are defined on the same set Π of predicate symbols. That is, for amount-atoms we allow in the two programs different amounts for the same resource-predicate. This makes the problem of strong equivalence of RASP programs different from the same problem for plain ASP programs.

To account for resource production and consumption, HT-logic must be extended as follows. The set of atoms must be augmented to admit amount-atoms, involving both plain and standardized-apart resource predicates. Like in the RASP semantics, we take for given the choice of an algebraic structure to represent amounts and support operations on them.

The satisfaction relation of HT-logic between an interpretation $I = \langle I^H, I^T \rangle$ and a formula F must be augmented to express that each resource can be produced and consumed in several fragments, but what counts is on the one hand that consumption does not exceed production, and on the other hand which is the total produced quantity. We thus add the following two new axioms (where w is the world, that can be either *here* or *there*).

The first new axiom "distributes" the available quantity of a resource q (obtained by summing the produced quantities) to the formulas that use it.

AR-1
$\langle I^H, I^T, w \rangle \models q^{j_1}{:}a_1 \wedge \ldots \wedge q^{j_k}{:}a_k$ where $k > 0$ and each $j_i > 0$ if
$\langle I^H, I^T, w \rangle \models q{:}b_1 \wedge \ldots \wedge q{:}b_s$, $s > 0$, and $b_1 + \ldots + b_s \geq a_1 + \ldots + a_k$

Notice that, for every HT-interpretation $\langle I_s^H, I_s^T \rangle$ of P_s, AR-1 ensures that the sets I_s^H and I_s^T are, in the terminology of previous section (Definition 4), r-interpretations of P_s.

The second new axiom "computes" the total produced quantity b of each resource q. To this aim, for each resource-predicate q we provisionally introduce a fresh corresponding resource-predicate q^T.

AR-2
$\langle I^H, I^T, w\rangle \models q^T{:}b$ if $\langle I^H, I^T, w\rangle \models q{:}a_1 \wedge \ldots \wedge q{:}a_k$ and
$\langle I^H, I^T, w\rangle \not\models q{:}a,\ a \neq a_1 \wedge \ldots \wedge a \neq a_k$, where $k > 0$ and $b = a_1 + \ldots + a_k$

In order to deal with different though equivalent production patterns that may occur in different RASP programs, we keep in HT-interpretations only the total quantities of produced resources.

Definition 10. *Given an HT-interpretation $\langle I_s^H, I_s^T\rangle$ of P_s, its* normalized version *is obtained by replacing in both I_s^H and I_s^T for each resource-predicate q the set of atoms $q^T{:}b, q{:}a_1, \ldots, q{:}a_k$ with $q : b$.*

In what follows, by some abuse of notation, by "HT-interpretation" we mean its normalized version (the same for HT-models). This stated, it is not difficult to suitably extend to RASP the results introduced in [11]. We introduce the following lemma, so that we can state the main theorem:

Lemma 1. *For any RASP program P and any set I of atoms, the HT-interpretation $\langle I, I\rangle$ is an equilibrium model of P iff I is an r-answer set of P_s.*

Theorem 2. *For any RASP programs P_1 and P_2, and for every RASP program S, $P_1 \cup S$ has the same answer sets of $P_2 \cup S$, i.e., P_1 is strongly equivalent to P_2, if and only if their standardized-apart versions P_{1s} and P_{2s} are equivalent in the extended logic of Here-and-There, i.e., have the same HT-models.*

Clearly, two RASP programs are candidates to be equivalent whenever the same total quantity of each resource q is produced. To ensure equivalence consumption must be performed in exactly the same way, which, as we will see, is a heavy limitation. In the rest of this section in fact, we discuss a number of examples to illustrate the subtleties of equivalence and strong equivalence of RASP programs, and emphasize the problems which arise. We will then address these problems in the subsequent section.

Example 3. The following two RASP programs are strongly equivalent.

P_1 :	P_2 :
$q{:}6 \leftarrow not\ c.$	$q{:}3 \leftarrow not\ c.$
	$q{:}3 \leftarrow not\ c.$

Their unique equilibrium model is $\langle I, I\rangle$ with $I = \{q : 6\}$ which is the unique r-answer set of both programs. The only difference between the two programs is that the same amount of resource q is produced in P_1 all in once, and in P_2 in two parts. This difference is taken into account by axiom AR-2.

Example 4. The following two RASP programs are, similarly to what would happen in plain RASP, not strongly equivalent.

P_1 :	P_2 :
$q{:}6.$	$q{:}6 \leftarrow not\ c.$

Their unique equilibrium model is $\langle I, I\rangle$ with $I = \{q : 6\}$ which is the unique r-answer set of both programs, but they are not strongly equivalent as can be seen by adding for instance fact c.

In the terms illustrated up to now, it is very difficult for RASP programs to be not only strongly equivalent, but even simply equivalent, as demonstrated by the following examples.

Example 5. The following two standardized-apart RASP programs are, differently from what would happen in plain ASP, not even equivalent. In fact, in the former program there is a number of plainly available resources, while in the latter producing $q{:}4$ requires consuming $p{:}2$.

P_1 : $\quad q{:}4. \quad p{:}2. \quad r{:}4. \quad c.$

Unique answer set: $\{c, q{:}4, p{:}2, r{:}4\}$

P_2 : $\quad q{:}4 \leftarrow p^1{:}2. \quad p{:}2. \quad r{:}4. \quad c.$

Unique answer set: $\{c, q{:}4, p{:}2, r{:}4, p^1{:}2\}$

Example 6. The following two standardized-apart RASP programs are not equivalent, despite that they produce and consume the same resources, though in a different way.

P_1 : $\quad q{:}1 \leftarrow p^1{:}2. \quad p{:}6. \quad r{:}1 \leftarrow p^2{:}3. \quad g{:}1.$

Unique answer set: $\{g{:}1, p{:}6, q{:}1, r{:}1, p^1{:}2, p^2{:}3\}$.

P_2 : $\quad q{:}1 \leftarrow p^1{:}3. \quad p{:}6. \quad r{:}1 \leftarrow p^2{:}2. \quad g{:}1.$

Unique answer set: $\{g{:}1, p{:}6, q{:}1, r{:}1, p^1{:}3, p^2{:}2\}$.

The two programs are however equivalent in terms of compact form of r-answer sets: according to Definition 9, one obtains for both programs the r-answer sets $\{g{:}1, p{:}6, q{:}1, r{:}1, -p{:}5\}$. Yet, even w.r.t. compact r-answer sets the two programs are not strongly equivalent: assume, e.g., to add rule $g{:}2 \leftarrow p{:}4$ (that, standardized-apart, becomes $g{:}2 \leftarrow p^3{:}4$). The additional rule "competes" with the original ones for the use of amounts of resource $p{:}6$. Thus, different choices for employing resource $p{:}6$ become possible. In fact, for the augmented former program the answer sets are:
$M_1 = \{g{:}1, p{:}6, q{:}1, r{:}1, p^1{:}2, p^2{:}3\}$ $M_2 = \{g{:}3, p{:}6, q{:}1, p^1{:}2, p^3{:}4\}$ i.e.,
in compact form, $Mr_1 = \{g{:}1, p{:}6, q{:}1, r{:}1, -p{:}5\}$ $Mr_2 = \{g{:}3, p{:}6, q{:}1, , -p{:}6\}$.
For the latter program they are instead:
$N_1 = \{g{:}1, p{:}6, q{:}1, r{:}1, p^1{:}3, p^2{:}2\}$ $N_2 = \{g{:}3, p{:}6, r{:}1, p^2{:}2, p^3{:}4\}$ i.e.,
in compact form, $Nr_1 = \{g{:}1, p{:}6, q{:}1, r{:}1, -p{:}5\}$ $Nr_2 = \{g{:}3, p{:}6, r{:}1, -p{:}6\}$.

In the two previous examples one may notice that given programs are equivalent w.r.t. what is produced, and differ w.r.t. resources that are consumed. A conceptual tool to recognize some kind of equivalence of the above programs is in order, as a designer would be enabled to assess their similarity and choose the pattern of production/consumption deemed more appropriate in the application at hand. In the next section we will propose a weaker notion of equivalence and strong equivalence than those introduced so far.

4 Strong Equivalence of RASP programs revisited

Below we introduce a compact form of HT-models where produced and consumed resources occur in their total quantities, similarly to r-answer sets introduced in Definition 9.

Definition 11. *Given an HT-model $\langle I_s^H, I_s^T \rangle$ of P_s, a* compact *HT-model (c-HT-model) $\langle I^H, I^T \rangle$ of P is obtained by replacing, for each resource-predicate q occurring in $\langle I_s^H, I_s^T \rangle$, its standardized-apart occurrences $q^{j_1}{:}a_1, \ldots, q^{j_k}{:}a_k$, $k \geq 0$, with $-q{:}a$, where $a = a_1 + \ldots + a_k$.*

The following definition makes a further simplification by eliminating consumed quantities from r-answers sets and c-HT-models.

Definition 12. *Given an r-answer set A of P or given a c-HT-model $\langle I_c^H, I_c^T \rangle$ of P_s, a* production *answer set (p-answer set) A' of P (resp., a* production *HT-model, or p-HT-model) $\langle I^H, I^T \rangle$ of P_s) is obtained by removing from A (resp., from I_c^H and I_c^T) all atoms of the form $-q{:}a$ for any q and a.*

In Example 6, the unique equilibrium c-HT-model of both programs is $\langle I, I \rangle$ with $I = \{g{:}1, p{:}6, q{:}1, r{:}1, -p{:}5\}$ where I is the unique r-answer set. The corresponding p-answer set is $I' = \{g{:}1, p{:}6, q{:}1, r{:}1\}$, and the unique equilibrium p-HT-model is $\langle I', I' \rangle$.

In Example 5, the unique r-answer set of P_1 is $M = \{c, q{:}4, p{:}2, r{:}4\}$, which coincides with the p-answer set. The unique equilibrium c- and p-HT-model is $\langle M, M \rangle$. For P_2, the unique r-answer set is $N = \{c, q{:}4, p{:}2, r{:}4, -p{:}2\}$, and the unique equilibrium c-HT-model is $\langle N, N \rangle$. The corresponding p-answer set is $N' = \{c, q{:}4, p{:}2, r{:}4\}$, and the unique equilibrium p-HT-model is $\langle N', N' \rangle$. Thus, in both Example 6 and Example 5 the two given programs are "equivalent" w.r.t. equilibrium p-HT-models (or, equivalently, p-answer sets). We formalize this notion of equivalence below.

Definition 13. *Two RASP theories (standardized-apart programs) are* equivalent on production *(p-equivalent) if their p-HT-models (and, consequently, their p-answer sets) coincide. They are* tightly equivalent on production *if they have the same set of unfired rules w.r.t. any of their p-answer sets. Two RASP theories are* strongly equivalent on production *(p-strongly equivalent) if, after adding whatever RASP theory S to both, they are still equivalent on production.*

This definition enlarges the set of RASP programs that can be considered to be equivalent. However, strong equivalence remains problematic.

Example 7. Consider the two programs of Example 5, that are equivalent on production. Assume to add rule: $r{:}1 \leftarrow p^4{:}3$. The c-HT-models and the p-HT-models remain unchanged, and thus the two programs are still p-equivalent. In fact the added rule cannot fire, not being available the needed quantity of p. Assume instead to add rule $r{:}1 \leftarrow p^4{:}2$, which is able to exploit the available resource amount $p{:}2$. In this case, for the augmented P_1 we get the unique r-answer set $\{c, q{:}4, p{:}2, r{:}1, -p{:}2\}$, but for the augmented P_2 the new rule "competes" with pre-existing ones on the use of $p{:}2$: thus, we get the two r-answer sets $\{c, p{:}2, r{:}4, r{:}1, -p{:}2\}$ and $\{c, q{:}4, p{:}2, r{:}4, -p{:}2\}$. Consequently, the updated programs are not p-equivalent.

We may notice that the problems that we have discussed arise when S is a *proper RASP program*, i.e., a RASP program containing amount-atoms. However, Example 4 points out that problems related to the addition of a plain ASP part must also be taken into account. Let a *plain ASP addition* be a RASP program fragment not containing amount-atoms. It is easy to see that:

Proposition 1. *A necessary condition for two RASP programs to be p-strongly equivalent is to be strongly equivalent w.r.t. any plain ASP addition (ASP-strongly-equivalent RASP programs, or ase-RASP programs).*

In order to make p-strong equivalence more widely possible, we may introduce the constraint that whenever the addition S is a proper RASP program, it does not compete on resources with the original program. This appears quite reasonable: in fact, if one intends to enlarge a production/consumption process, preservation of existing processes should be guaranteed. If not, it should be clear that one obtains a different process, with hardly predictable properties.

Definition 14. *A* rational addition *S to a RASP program P is RASP program such that in program $P \cup S$ rules of S do not consume resources produced by rules of P.*

It is easy to get convinced that:

Proposition 2. *A Proper RASP program S is a rational addition if and only if one of the following conditions holds.*

1. *Resources consumed in S are not available in P.*
2. *Resources consumed in S are produced in S.*
3. *Resources consumed in S are produced in $P \cup S$.*
4. *S does not consume resources.*

Referring to Example 5: rule $r{:}1 \leftarrow p^4{:}3$ is a rational addition of kind (1); proper RASP program $r{:}1 \leftarrow p^4{:}3.\ p{:}5.$ is a rational addition of kind (2); proper RASP program $r{:}1 \leftarrow p^4{:}3.\ p{:}5 \leftarrow c.$ is a rational addition of kind (3); rule $r{:}1 \leftarrow c$ is a rational addition of kind (4).

If we restrict p-strong equivalence to p-strong equivalence w.r.t. rational addition, then the notion becomes much more usable in practical cases. In particular for instance, programs P_1 and P_2 of Examples 6 and 5 are p-strongly equivalent w.r.t. rational addition. Notice however that p-strong equivalence between two programs does not guarantee that we are left with the same resources in the two cases. A subject of future work will be that of studying other forms of strong equivalence, e.g. w.r.t. what is left or more particularly what is left for a certain set of resources.

In tightly equivalent ase-RASP programs, resources possibly produced in $P \cup S$ would necessarily be employed in the same way, as the r-rules that might potentially fire are the same. In this case, we consider program atoms (plain ASP atoms) that may be derived in $P \cup S$ as resources that are available in unlimited quantity. If restricting ourselves to this class of programs, we obtain an interesting sufficient condition.

Proposition 3. *Any two tightly equivalent ase-RASP programs are p-strongly equivalent w.r.t. rational additions.*

5 Conclusions and Future Directions

In this paper, we have extended the notion of strong equivalence from answer set programs to RASP programs. We have seen that this notion takes a quite peculiar flavor in RASP, where strong equivalence (apart from trivial cases) can be ensured at the condition of imposing some requirements on the theory which is added to given one. In fact, interactions in resource usages between two components may be quite involved. Nonetheless, strong equivalence may help a designer to reason about different processes that are strongly equivalent in terms of resources that are produced: a process might be preferred for instance because it consumes less, or consumes smaller quantities of crucial resources. A relevant future direction of this work can be in fact that of introducing a notion strong equivalence of RASP programs taking into account preferences in resource usage, like e.g. those that can be expressed in [3]. Also, in the RASP context an extension of strong equivalence, i.e., synonymy [14], can find interesting applications. Synonymy extends strong equivalence in the sense that two theories may be equivalent even if they are expressed in different languages, if each is bijectively interpretable in the other. In RASP, this might allow one to compare processes formulated in seemingly different ways.

References

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

[2] S. Chintabathina, M. Gelfond, and R. Watson. Defeasible laws, parallel actions, and reasoning about resources. In E. Amir, V. Lifschitz, and R. Miller, editors, *Logical Formalizations of Commonsense Reasoning: Proceedings of CommonSense'07*. AAAI Press, Menlo Park, 2007. Technical report SS-07-05.

[3] S. Costantini and A. Formisano. Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of Algorithms in Cognition, Informatics and Logic*, 64(1):3–15, 2009.

[4] S. Costantini and A. Formisano. Answer set programming with resources. *Journal of Logic and Computation*, 20(2):533–571, 2010.

[5] S. Costantini, A. Formisano, and D. Petturiti. Extending and implementing RASP. *Fundamenta Informaticae*, 105(1-2):1–33, 2010.

[6] S. Costantini, A. Formisano, and D. Petturiti. Strong Equivalence for RASP Programs: Long Version, Tech. Rep. 02-2012 Dip. di Matematica e Informatica, Univ. di Perugia", Available at `http://www.dmi.unipg.it/formis/papers/report2012_02.ps.gz`.

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

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

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

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

[11] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.

[12] I. Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning LPNMR'99*, number 1730 in LNAI, pages 317–331. Springer-Verlag, 1999.

[13] D. Pearce. A new logical characterization of stable models and answer sets. In *Non-Monotonic Extensions of Logic Programming*, number 1216 in LNAI, pages 55–70. Springer, 1997.

[14] D. Pearce and A. Valverde. Synonymous theories in answer set programming and equilibrium logic. *Proc. of ECAI04, 16th Europ. Conf. on Art. Intell.*, pages 388–390, 2004.

[15] D. Pearce and A. Valverde. Quantified equilibrium logic and the first order logic of here-and-there. Technical report, Univ. Rey Juan Carlos, 2006. available at http://www.satd.uma.es/matap/investig/tr/ma06_02.pdf.

[16] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, number 1551 in LNCS, pages 305–319. Springer-Verlag, 1999.

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