Negation as a Resource: a novel view on Answer Set Semantics*

Stefania Costantini[1] and Andrea Formisano[2]

[1] DISIM, Università di L'Aquila, Italy stefania.costantini@univaq.it
[2] DMI, Università di Perugia, Italy formis@dmi.unipg.it

Abstract. In recent work, we provided a formulation of ASP programs in terms of linear logic theories. Answer sets were characterized in terms of maximal tensor conjunctions provable from such theories. In this paper, we propose a full comparison between Answer Set Semantics and its variation obtained by interpreting literals (including negative literals) as resources, which leads to a different interpretation of negation. We argue that this novel view can be of both theoretical and practical interest, and we propose a modified Answer Set Semantics that we call Resource-based Answer Set Semantics. One advantage is that of avoiding inconsistencies, so every program has a (possibly empty) resource-based answer set. This implies however the introduction of a different way of representing constraints.

Keywords: Answer Set Programming, Linear Logic, Default Negation

1 Introduction

In [1], we proposed a comparison between RASP and linear logic [2], where RASP [3, 4, 5] is a recent extension of the Answer Set Programming (ASP) framework obtained by explicitly introducing the notion of *resource*. As it is well-known, ASP is nowadays a well-established programming paradigm, with applications in many areas (see among many [6, 7, 8] and the references therein). RASP is a significant extension, supporting both formalization and quantitative reasoning on consumption and production of amounts of resources.

We proved in particular that RASP (and thus, ASP as a particular case) corresponds to a fragment of linear logic. This was done by providing a two-ways translation of RASP programs into a linear logic theory. The result implies that a RASP inference engine (such as Raspberry [5]) can be used for reasoning in this fragment. In defining the correspondence, we introduced a RASP and linear-logic modeling of default negation as understood under the answer set semantics. We meant in some sense to propose "yet another definition of answer set", in addition to those reported in [9].

In the present paper, we show that understanding default negation as a resource goes beyond, and leads to the definition of a generalization of the answers set semantics (for short AS, on which ASP is based), with some potential advantages. We provide a

* A short version of this paper will appear in the Proc. of LPNMR 2013. This research is partially supported by GNCS-13 project.

model-theoretic definition of the new semantics, that we call *Resource-based Answer Set Semantics*. In the new setting, there are no inconsistent programs, and basic odd cycles (similarly to basic even cycles in AS) are interpreted as exclusive disjunctions. Constraints must then be represented explicitly (while in ASP they are "simulated" via unary odd cycles). Therefore, what was before programs with constraints becomes a plain ASP program (under the extended semantics) augmented with a set of explicitly represented constraints. We argue that representing constraints separately can lead to more generality and to an improved elaboration-tolerance (in the sense of [10]). In the proposed approach, the "practical expressivity" in terms of knowledge representation is improved (as we demonstrate by means of significant examples), though unfortunately also the computational complexity increases.

The paper is organized as follows. In Sections 2 and 3, we provide the necessary background on linear logic and ASP. In Section 4, we specialize the method defined in [1] for RASP, so as to show that ASP can be defined as a fragment of linear logic. It is relevant to recall this formalization, because it makes it clear which is the motivation of the new notion of negation and of the generalized answer set semantics that we then propose. In Sections 5 and 6 the semantic extension is described, formalized and discussed. Finally, in Section 7 we conclude.

2 Background on Linear Logic

Linear logic [2] can be considered as a *resource sensitive* refinement of classical logic, since it intrinsically supports a natural accounting of resources. Intuitively speaking, in linear logic, two assumptions of a formula P are distinguished from a single assumption of it. Below we briefly review the basic traits of (a fragment of) linear logic, by recalling only the notions that will be used in the remaining part of the paper. For a comprehensive treatment we refer the reader to [11] and to the references therein.

In linear logic, contraction and weakening rules are not allowed: hence, while a statement of the form $P \rightarrow P \wedge P$ is valid in classical logic, this is not the case in linear logic. The point here can be explained by observing that in classical logic statements are assumed to express "static" properties, unchanging facts about the world. On the contrary, linear propositions are concerned with dynamic properties of finite amounts of resources (and the processes that use them). An example well-known in the literature [11, 12] may further clarify this point. Consider the following propositions/resources:

P : "*One dollar*"
Q : "*One pack of Camel*"
R : "*One pack of Marlboro*"

and the following axiomatization of a vending machine:

$$P \rightarrow Q$$
$$P \rightarrow R$$

In classical logic, one can derive that $P \rightarrow Q \wedge R$, but this makes little sense if we are assuming the mentioned interpretation of propositions as resources (and of implications as transformation processes, very much like in RASP).

One of the crucial features of linear logic is that it makes a neat distinction between two forms of conjunction that are not distinguished by classical logic. Namely, one of them intuitively means "I have both". This is said *multiplicative conjunction* and is written as $\otimes$. The other, the *additive conjunction* means "I have a choice" (and is written as &). Dually, there are two disjunctions. The multiplicative one, written $P \parr Q$ can be read as "if not P, then Q", and the additive disjunction $P \oplus Q$, that intuitively stands for the possibility of either P or Q, but we do not know which of the two. That is, it involves "someone else's choice".

Finally, we have linear implication $P \multimap Q$. It encodes a form of production process: it can be read as "Q can be derived using P exactly once". (Notice that, in such a process P is "consumed", so it cannot be used again.)

Linear negation $^\perp$ is the only negative operation in the logic. It is involutive (namely, $(P^\perp)^\perp$ and P can be safely identified) and, at the same time, it retains a constructive character. Notice that it acts as a sort of *transposition*: $P \multimap Q$ coincides with $Q^\perp \multimap P^\perp$. Moreover, the linear implication $P \multimap Q$ can be rewritten as $P^\perp \parr Q$.

In order to re-gain the full power of classical logic *exponential* operators, namely ! and its dual ?, are introduced. Intuitively, $!P$ means that we have how many P we want. These connectives reintroduce, in a more controllable way, contraction and weakening in the logical framework.

To better illustrate all these connectives, let us recall another example (taken from [12]). Suppose that for a fixed price of 5 Dollars a restaurant will provide a hamburger, a Coke, as many french fries as you like, onion soup or salad (your choice), and pie or ice cream (depending on availability, hence by someone else's choice). This is the menu:

For a fixed-Price Menu: 5 Dollars (D) you can have:
Hamburger (H)
Coke (C)
All the french fries (F) you can eat
One between Onion-Soup (O) or Salad (S)
Pie (P) or Ice-Cream (I) depending on availability

and its encoding in a linear logic formula:

$$(D \otimes D \otimes D \otimes D \otimes D) \multimap \big(H \otimes C \otimes !F \otimes (O \,\&\, S) \otimes (P \oplus I)\big)$$

Some further notions will be used in what follows. Let Xs and Ys denote *tensor products* of positive literals, e.g. formulas of the form $(P_1 \otimes \cdots \otimes P_n)$ (for $n > 0$). Then, *generalized Horn implications* are defined as follows:

- an Horn implication has the form: $X \multimap Y$.
- An $\oplus$-Horn implication has the form: $\big(X_1 \multimap (Y_1 \oplus Y_2)\big)$.
- An &-Horn implication has the form: $\big((X_1 \multimap Y_1) \,\&\, (X_2 \multimap Y_2)\big)$.

Notice that a formula of the last form, say $(P_1 \multimap Q_1) \,\&\, (P_2 \multimap Q_2)$, encodes a nondeterministical process where a choice is made between the two disjuncts (say $P_2 \multimap Q_2$) and then the (sub-)process encoded by the selected option is executed (in our case Q_2 is produced using P_2).

A formal proof system for linear logic can be formulated in terms of a Gentzen-style sequent calculus. A *sequent* is composed of two sequences of formulas separated by a turnstile ($\vdash$) symbol. The sequent $\Delta \vdash \Gamma$ asserts that the multiplicative conjunction of the formulas in Δ together imply the multiplicative disjunction of the formulas in Γ. In general, a *sequent calculus proof rule* consists of a set of hypothesis sequents and a single conclusion sequent. A full set of Gentzen-style sequent rules for linear logic can be found, for instance, in [13].

3 Background on Answer Set Semantics

In the answer set semantics (originally named "stable model semantics"), a (logic) program Π (cf., [14, 15]) is a collection of *rules* of the form $H \leftarrow L_1, \ldots, L_n.$ where H is an atom, $n \geqslant 0$ and each literal L_i is either an atom A_i or its *default negation not* A_i. The left-hand side and the right-hand side of rules are called *head* and *body*, respectively. A rule can be rephrased as $H \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n.$ where $A_1, \ldots, A_m$ can be called *positive body* and *not* $A_{m+1}, \ldots, not\ A_n$ can be called *negative body*. A rule with empty body ($n = 0$ is called a *fact*. A rule with empty head is a *constraint*, where a constraint is of the form $\leftarrow L_1, \ldots, L_n.$ and states that literals $L_1, \ldots, L_n$ cannot be simultaneously true.

Various extensions to the basic paradigm exist, that we do not consider here as they are not essential in the present context. We do not even consider "classical negation" (cf., [15]).

In the rest of the paper, whenever it is clear from the context, by "a (logic) program Π" we mean an answer set program (ASP program) Π, and we will implicitly refer to the "ground" version of Π. The ground version of Π is obtained by replacing in all possible ways the variables occurring in Π with the constants occurring in Π itself, and is thus composed of ground atoms, i.e., atoms which contain no variables. By "minimal model of Π" we mean a minimal model of Π intended as a classical logic theory, where $\leftarrow$ is intended as implication and *not* as negation in classical logic terms.

The answer sets semantics [14, 15] is a view of a logic program as a set of inference rules (more precisely, default inference rules), or, equivalently, a set of constraints on the solution of a problem: each answer set represents a solution compatible with the constraints expressed by the program. Consider simple program $\{q \leftarrow not\ p.\quad p \leftarrow not\ q.\}$. For instance, the first rule is read as "assuming that p is false, we can *conclude* that q is true." This program has two answer sets. In the first one, q is true while p is false; in the second one, p is true while q is false.

Unlike other semantics, a program may have several answer sets, or may have no answer set. Whenever a program has no answer sets, we will say that the program is *inconsistent*. Correspondingly, checking for consistency means checking for the existence of answer sets. The following program has no answer set: $\{a \leftarrow not\ b.\ b \leftarrow not\ c.\ c \leftarrow not\ a.\}$. The reason is that in every minimal model of this program there is a true atom that depends (in the program) on the negation of another true atom, which is strictly forbidden in this semantics, where every answer set can be considered as a self-consistent and self-supporting set of consequences of given program. The program $\{p \leftarrow not\ p.\}$ has no answer sets either as it is contradictory. Constraints of the form defined above can

be simulated by plain rules of the form $p \leftarrow not\ p, L_1, \ldots, L_n.$ where p is a fresh atom. Thus, consistency is related (as discussed at length in [16, 17]) to the occurrence of "odd cycles" (of which $p \leftarrow not\ p$ is the basic case, though odd cycles may involve any odd number of atoms) and how they are connected to other parts of the program. The reason is that, in principle, the negation *not A* of atom A is an assumption, that must be dropped whenever A can be proved, as answer sets are by definition non-contradictory.

Below is the specification of the Answer Set Semantics, reported from [14].

Definition 1 (The Gelfond-Lifschitz Operator). *Let I be a set of atoms and Π a program. A GL-transformation of Π modulo I is a new program Π/I obtained from Π by performing the following two reductions:*

1. *removing from Π all rules which contain a negative premise not A such that $A \in I$;*
2. *removing from the remaining rules those negative premises not A such that $A \notin I$.*

Π/I is a positive logic program, with Least Herbrand Model[1] *J. Let $\Gamma_\Pi(I) = J$.*

Answer sets are defined as follows.

Definition 2. *Let I be a set of atoms and Π a program. I is an answer set of Π iff $\Gamma_\Pi(I) = I$.*

It will be useful in what follows to report from [16] a simple property of Γ_Π.

Proposition 1. *Let M be a minimal model*[2] *of Π. Then, $\Gamma_\Pi(M) \subseteq M$.*

Answer sets are in fact minimal supported models, and non-empty answer sets form an anti-chain with respect to set inclusion.

In the ASP (Answer Set Programming) paradigm, each answer set is seen as a solution of given problem, encoded as an ASP program. To find these solutions, an ASP-solver is used. Several solvers have became available, see [19], each of them being characterized by its own prominent valuable features. The expressive power of ASP, as well as, its computational complexity have been deeply investigated (cf. e.g., [20]).

4 ASP and Linear Logic

In this section, we specialize the method defined in [1] for RASP, so as to show that ASP can be defined as a fragment of linear logic. In particular, we define a translation of ASP programs into a linear logic theory employing as connectives tensor product $\otimes$ (to express concomitant use/production of different resources), linear implication $\multimap$ (to model production processes), and additive conjunction & (to represent alternative/exclusive resource allocation). In well-known terminology, we adopt formulas belonging to the so-called Horn-fragment of linear logic. In [1] we treat the more general case of RASP, which manages resource production and consumption.

[1] Cf. [18] for the definition of Least Herbrand Model of a Horn logic program, due to Van Emden and Kowalski.

[2] The property holds for models in general, but minimal ones are those of interest here.

A positive ASP program Π (i.e., a program without default negation) can be transformed into a corresponding Linear Logic RASP Theory as follows (notice that the reverse translation is also possible, i.e., to transform a Linear Logic RASP Theory into a (R)ASP program). In particular, in the definition below each atom q in the body of the j-th rule of given program is renamed as q^j, where the q^j's are called the standardized-apart versions of q. Moreover, since the formalization passes through RASP, which considers atoms as resources, each standardized-apart atom q^j will stand for $q^j{:}1$ (In RASP terminology, a writing of the form $q{:}a$ denotes an *amount* a of the *resource* q.). The meaning is that, when using the body of a rule to derive the head, one uses one unit of each atom (seen as a resource) in the body[3]. Notice that, in Π, the truth of an atom might be used to prove several consequences (through different rules). As we mentioned before, linear logic provides the exponential connective $!A$, intuitively meaning that we can use as many occurrences of A as we want. However, exploiting this connective would bring us outside the finite propositional fragment of linear logic at hand. The devised method remains within the propositional fragment.

Definition 3. *Given a positive ASP program Π, the corresponding Linear Logic RASP Theory Σ_Π is obtained by applying, in sequence, the following rewritings.*

- *Standardize apart the atoms in the bodies of rules of Π. Namely, each occurrence of an atom A in the body of the j-th rule is replaced by $A^j{:}1$.*
- *For every atom A occurring as head of $h > 0$ rules in (the standardize apart version of) Π, let $A \leftarrow B_{i,1},\ldots,B_{i,\ell_i}$, for $i = 1,\ldots,h$, be such rules (with ℓ_i possibly null, if the corresponding rule is a fact). Replace these rules by the following linear implications (where the A_is are fresh atoms):*

$$\begin{array}{l} B_{i,1} \otimes \ldots \otimes B_{i,\ell_i} \multimap A_i \quad \textit{for } i = 1,\ldots,h \\ A_1 \,\&\, \ldots \,\&\, A_h \multimap A \end{array}$$

- *For each atom A, let $A^1{:}1, \ldots, A^m{:}1$ be its standardized apart versions, introduced as described earlier. Add to Σ_Π the linear implication $A \multimap A^1{:}1 \otimes \ldots \otimes A^m{:}1$.*
- *Replace in Σ_Π any linear implication $B_1 \otimes \ldots \otimes B_n \multimap H$ with the implication $B_1 \otimes \ldots \otimes B_n \multimap H \otimes B_{1R} \otimes \ldots \otimes B_{nR}$.*

Let us remark some aspects of the previous definition. Notice that through the second step of the translation, the body of each rule in Π, which is a conjunction of atoms, is turned into a tensor conjunction of atoms. The purpose of the linear implication $A_1 \,\&\, \ldots \,\&\, A_h \multimap A$ is that of enabling the derivation of A by either of the (translations of the) rules defining it. Clearly, the introduction of such an implication can be avoided in case A occurs as head of a single rule (in this case $h = 1$ and we can simply replace A_1 by A in the first linear implication). In what follows we will adhere to this convention whenever possible.

The linear implication $A \multimap A^1{:}1 \otimes \ldots \otimes A^m{:}1$ can be seen as an &-Horn implications with a unique conjunct. It models the fact that A is a resource available to any rule that may need to use it.

[3] RASP allows for arbitrary quantities, not needed here.

Notice, moreover, the introduction of a fresh atom B_{iR}s corresponding to each atom B_i, in the last step of the translation. These fresh atoms are called the *r-copies* of the B_is. They are produced just in order to keep a *record* of those resources that have been consumed. R-copies allow us to establish a correspondence between answer sets of Π and maximal tensor conjunctions provable from Σ_Π, where:

Definition 4. *Given linear logic theory Σ, a tensor conjunction of atoms $A_1 \otimes \ldots \otimes A_n$ ($n \geq 0$), is called* maximally provable *if it is provable from Σ, and for any atom B, the tensor conjunction $A_1 \otimes \ldots \otimes A_n \otimes B$ is not provable from Σ (we equivalently talk about a maximal tensor conjunction provable from Σ).*

Lemma 1. *Let Π be a positive ASP program Π, and Σ_Π be the corresponding* Linear Logic RASP Theory. *Every maximal tensor conjunction $\mathscr{A}$ provable from Σ_Π includes all the r-copies of facts of Σ_Π and of standardized-apart atoms occurring in the body of linear implications of Σ_Π that have been used for proving atoms in $\mathscr{A}$.*

As mentioned, the role of r-copies is to keep records of facts (intended as resources originally present in the program) and of intermediate conclusions used (as resources) in further inference. In a linear-logic setting in fact, resources which are consumed "disappear", thus we would not be able to establish a relation between provable tensor conjunctions and answer sets. Now in fact, we are able to state (neglecting, by abuse of notation, the syntactic distinction between an atom A and its r-copy A_R):

Theorem 1. *Let Π be a positive ASP program Π, and Σ_Π be the corresponding* Linear Logic RASP Theory. *$A_1 \otimes \ldots \otimes A_n$ is a maximal tensor conjunction provable from Σ_Π if and only if $\{A_1, \ldots, A_n\}$ is an answer set for Π.*

Let us now consider full ASP, where rule bodies involve negative literals. Assume there are n occurrences of *not A* in the body of rules of given program Π. To represent full RASP (and thus full ASP) we improve the transformation devised in Definition 3:

Definition 5. *Given ASP program Π, the corresponding Linear Logic RASP Theory Σ_Π is obtained by applying, in sequence, the following rewritings.*

- *For each atom A occurring negated in rules of Π, standardize apart each of its negated occurrences by replacing not A with not A^j:1, in the j-th rule.*
  Being not A^{j_1}:1, …, not A^{j_n}:1 all the occurrences introduced in this manner, add the (linear) fact not A:n to the translation of Π.
- *For each rule $A \leftarrow B_1, \ldots, B_\ell$ of Π. Let such rule be the j-th one; rewrite it as $A \leftarrow B_1, \ldots, B_\ell, not\ A^j{:}n$.*
  Let us denote by not $A^{k_1}{:}n, \ldots, not\ A^{k_s}{:}n$ all the atoms introduced in this manner.[4]
- *Apply the rewriting indicated in Definition 3 to the result of the previous steps.*
- *Finally, for each linear fact not A:n added to Σ_Π (cf., the first two steps), also add the &-Horn implications to the translation of Π:*

$$(not\ A{:}n \multimap not\ A^{k_1}{:}n)\,\&\,\ldots\,\&\,(not\ A{:}n \multimap not\ A^{k_s}{:}n)\,\&$$
$$(not\ A{:}n \multimap not\ A^{j_1}{:}1 \otimes \ldots \otimes not\ A^{j_n}{:}1)$$

[4] In case identical atoms would be introduced in the body in consequence of different steps of the translation, e.g., *not* A^k:1 and *not* A^k:1 might occur in the same rule if n equals 1 in the first step and *not A* already appeared in the ASP rule body, then further standardize apart these occurrences, e.g., as *not* A^{k1}:1 and *not* A^{k2}:1.

The intuitive meaning behind this translation is that the assumption *not A* is made available to every rule that intends to adopt it, unless A itself is provable. In which case the assumption becomes totally unavailable (as proving A consumes the full available quantity of the "resource" *not A*).

The transformation of Definitions 3 and 5 is clearly polynomial, as we add: (i) a new conjunct in the body (*not A* if the rule head is A) and new elements (r-copies) in the head of rules ; (ii) one &-Horn implication for each A occurring in the head of some rule; (iii) one linear implication for each atom defined via several rules; (iv) one &-Horn implication for each A occurring negatively in the body of some rule. Hence, we have:

Theorem 2. *Let Π be an ASP program, and Σ_Π the corresponding Linear Logic RASP Theory, obtained according to Definitions 3 and 5. Let $M = \{A_1, \ldots, A_n\}$ be an answer set for Π. Then, $A_1 \otimes \ldots \otimes A_n$ is a maximal tensor conjunction provable from Σ_Π.*

Note that the reverse result does not necessarily hold, because there are maximal tensor conjunctions that are not answer sets but are provable from Σ_Π. This is due (as discussed in [1]) to the lack of relevance of the answer set semantics (cf., [21]), but also to the locality of a proof-based system such as linear logic.

5 Negation as a Resource: a novel view on Answer Set Semantics

It is interesting to notice that the linear logic formulation we summarized in the previous section prevents contradictions. Consider for example the program $\Pi_1 = \{p \leftarrow not\ p.\}$. It is transformed into:

$$
\begin{array}{l}
not\ p^{11}{:}1 \otimes not\ p^{12}{:}1 \ \multimap p, \\
not\ p{:}1, \\
(not\ p{:}1 \ \multimap not\ p^{11}{:}1) \,\&\, (not\ p{:}1 \ \multimap not\ p^{12}{:}1)
\end{array}
$$

In the first rule, one occurrence of *not p* corresponds to the one originally present, the other one has been added as for proving p it is necessary to "absorb" the whole available quantity of *not p* (consider $n = 1$ in Definition 5). We can in fact verify that the singleton tensor conjunction p is by no means provable: in fact, it would require two units of *not p*, while just one is available. This does not lead to inconsistency, but simply to the impossibility to prove p.

Consider again program $\Pi = \{a \leftarrow not\ b.\ b \leftarrow not\ c.\ c \leftarrow not\ a.\}$ which is an "odd cycle" involving three atoms. In our formulation, Σ_Π is the following:

$$
\begin{array}{l}
not\ a^1{:}1 \otimes not\ b^1{:}1 \ \multimap a \\
not\ c^2{:}1 \otimes not\ b^2{:}1 \ \multimap b \\
not\ a^3{:}1 \otimes not\ c^3{:}1 \ \multimap c \\
not\ a{:}1 \\
not\ b{:}1 \\
not\ c{:}1 \\
(not\ a{:}1 \ \multimap not\ a^1{:}1) \,\&\, (not\ a{:}1 \ \multimap not\ a^3{:}1) \\
(not\ b{:}1 \ \multimap not\ b^1{:}1) \,\&\, (not\ b{:}1 \ \multimap not\ b^2{:}1) \\
(not\ c{:}1 \ \multimap not\ c^2{:}1) \,\&\, (not\ c{:}1 \ \multimap not\ c^3{:}1)
\end{array}
$$

From this linear logic theory we can prove the three maximal tensor conjunctions, namely, a, b and c. Assume, in fact, to try to prove a (the cases of b and c are of course analogous). Proving a uses resources *not* a^1:1 and *not* b^1:1. Therefore, after proving a, b cannot be proved because its own negation (i.e., *not* b^2:1) is not available: in fact, the &-Horn implication related to b generates (indifferently) only one of the two items, and has already been requested to produce *not* b^1:1 for proving a. In turn, c cannot be proved because *not* a^3:1 is not available, as the &-Horn implication related to a generates (indifferently) only one of the two items, and has already been requested to produce *not* a^1:1 for proving a. Then, Σ_Π behaves analogously to the GL-reduct as far as c is concerned, being *not a* unavailable once a has been proved. But it behaves in a more uniform way on b, in the sense that once *not b* has been used to prove a, it is no longer possible to prove b.

This means that the 3-atoms odd cycles is interpreted as an exclusive disjunction, exactly like the 2-atoms even cycle (such as $\{q \leftarrow not\ p.\ \ p \leftarrow not\ q.\}$) in AS. Therefore, in the generate-and-test perspective which is at the basis of the ASP programming methodology, our new view provides a new mean of easily generating the search space.

We call $\{a\}$, $\{b\}$, and $\{c\}$ *resource-based answer sets*, for which we provide below a logic programming characterization. The resource-based answers set for program $\{p \leftarrow not\ p.\}$ is the empty set.

The ternary cycle has many well-known interpretations in terms of knowledge representation, among which the following is an example:

$$\begin{array}{l}\{beach \leftarrow not\ mountain.\\ \ mountain \leftarrow not\ travel.\\ \ travel \leftarrow not\ beach.\}\end{array}$$

In our approach we would have exactly one of (indifferently) *beach*, *mountain*, or *travel*. Similarly for the program $\{work \leftarrow not\ tired.\ tired \leftarrow not\ sleep.\ sleep \leftarrow not\ work.\}$. Note that, in answer set programming, for defining the exclusive disjunction of three atoms one has to resort to the *extremal program* [22] $\{a \leftarrow not\ b, not\ c.\ b \leftarrow not\ c, not\ a.\ c \leftarrow not\ a, not\ b.\}$

There are other semantic approaches to managing odd cycles, such as for instance [23, 24] and [25, 26], with their own sound theoretical foundations, that can however be distinguished from the present one: in fact, the former proposals basically choose (variants of) the classical models, and the latter ones treat differently the unary and ternary cycles.

Below we provide a variation of the answer set semantics that defines resource-based answer sets.

Definition 6. *Let Π be a program and I a minimal model of Π. I is called a Π-based minimal model iff $\forall A \in I$, there exists a rule in Π with head A and positive body $C_1, \ldots, C_m$, $m \geq 0$, where $\{C_1, \ldots, C_m\} \subseteq I$.*

Definition 7. *Let Π be a program. M is a resource-based answer set of Π iff $M = \Gamma_\Pi(I)$, where I is a Π-based minimal model of Π.*

By Definition 7, there is a resource-based answer set for each Π-based classical minimal model. It is clear that answer sets are among resource-based answer sets. In

fact, as stated in Section 3 (Proposition 1), for any minimal model I it holds $\Gamma_\Pi(I) \subseteq I$: thus any answer set S, being a minimal model which is equal to $\Gamma_\Pi(S)$, fits as a particular case in the above definition. Therefore, some of the resource-based answer sets of Π are classical models (coinciding with its answer sets), while the others are subsets of the remaining Π-based minimal models (if any). Non-empty resource-based answer sets still form an anti-chain w.r.t. set inclusion.

We call the new semantics RAS semantics (Resource-Based Answer Set semantics), w.r.t. AS (Answer Set) semantics. Differently from answer sets, a (possibly empty) resource-based answer set always exists. Complexity of RAS semantics is however higher than complexity of AS semantics: in fact, [27] proves that deciding whether a set of formulas is a minimal model of a propositional theory is co-NP-complete. Clearly, checking whether a minimal model I is Π-based and computing $\Gamma_\Pi(I)$ has polynomial complexity. Then:

Proposition 2. *Given program Π, deciding whether a set of atom I is a resource-based answer set of Π is co-NP-complete.*

The previous result about the relation with linear logic (Theorem 2) extends to the new semantics. The proof, reported in [1] in the context of full RASP programs, remains essentially the same. The difference is that where in previous case one referred to answer sets, which implies that given program Π was supposed to be consistent, we are now able to refer to any ASP program. Then we have:

Theorem 3. *Let Π be an ASP program, and Σ_Π the corresponding Linear Logic RASP Theory, obtained according to Definitions 3 and 5. If $M = \{A_1, \ldots, A_n\}$ is a resource-based answer set for Π, then $A_1 \otimes \ldots \otimes A_n$ is a maximal tensor conjunction provable from Σ_Π.*

It remains to be explained why the new definition models the intuition, and how it applies to practical cases. In particular, given minimal model I of Π, it may be that $\Gamma_\Pi(I) \subset I$, i.e., $\Gamma_\Pi(I)$ is a proper subset of I and thus I is not an answer set, for only one reason. For atom A to belong to a Π-based minimal model I, there exists some rule in Π with head A. For A not to belong to $\Gamma_\Pi(I)$, so that $\Gamma_\Pi(I) \subset I$, each of the rules that could cause A to be in the model must have been canceled by step (1) of Γ_Π, as they include literal *not B* in their body, $B \in I$. Atoms belonging to $\Gamma_\Pi(I)$ are therefore those atoms in I that can be derived without such contradictions. As widely discussed in [16, 17], contradictions only arise in program fragments corresponding to *unbounded odd cycles*, i.e., odd cycles where no atom is bounded to be true/false (thus resolving the contradiction) by links with other parts of the program. Starting from Π-based minimal models however, $\Gamma_\Pi(I)$ provides for these cycles the "exclusive or" interpretation that we have proposed above.

Regarding general odd cycles involving k atoms, of the form $\{a_1 \leftarrow not\ a_2.\ a_2 \leftarrow not\ a_3 \ldots a_k \leftarrow not\ a_1.\}$, it is easy to see that each such cycle has k classical minimal Π-based models (it admits k classical minimal models, all of them Π-based as there are no positive conditions). Correspondingly, we obtain k resource-based answer sets, where we have $M_i = \{a_{i+d},$ with d even, $0 \leq d < k-1\}$. This fact can be verified by producing a translation into the corresponding Linear Logic RASP Theory analogous to the one performed above for unary and ternary cycles. Then, unfortunately, odd cycles

no longer model disjunction if $k > 3$, similarly to even cycles, which do not model disjunction if $k > 2$.

In resource-based answer set semantics, there are no inconsistent programs. Nevertheless, the new semantics is useful in knowledge representation not just to fix inconsistencies: rather, it depicts a more general scenario in many reasonable examples. Consider for instance the variation of the above program (inspired to examples proposed in [23, 24]):

$$\begin{array}{l} \mathit{beach} \leftarrow \mathit{not\ mountain}. \\ \mathit{mountain} \leftarrow \mathit{not\ travel}. \\ \mathit{travel} \leftarrow \mathit{not\ beach}, \mathit{passport\_ok}. \\ \mathit{passport\_ok} \leftarrow \mathit{not\ forgot\_renew}. \\ \mathit{forgot\_renew} \leftarrow \mathit{not\ passport\_ok}. \end{array}$$

This program has answer set $M_1 = \{\mathit{forgot\_renew}, \mathit{mountain}\}$, as *passport_ok* being false forces *travel* to be false, which in turn makes *mountain* true. The answer set semantics cannot cope with the case of the passport being ok, which is in fact excluded as this option determines no answer set. Instead, in resource-based answer set semantics we have, in addition to M_1, three other answer sets stating that, if the passport is ok, any choice is possible, namely we have $M_2 = \{\mathit{passport\_ok}, \mathit{mountain}\}$, $M_3 = \{\mathit{passport\_ok}, \mathit{beach}\}$, and $M_4 = \{\mathit{passport\_ok}, \mathit{travel}\}$. We may notice that the semantics is still a bit strong on this example on the side of the answer set, as one would say that not having *passport_ok* prevents traveling, but any other choice should be possible, while instead the *mountain* choice is forced.

A better formalization of the above example would be by means of the plain odd cycle, plus the even cycle concerning passport, plus the constraint

$$\leftarrow \mathit{not\ passport\_ok}, \mathit{travel}.$$

In the next section we will discuss how to introduce such a constraint, as a unary odd cycle is no longer usable to this purpose.

6 Modeling Constraints

In resource-based answer set semantics, constraints cannot be modeled in terms of odd cycles. Therefore, they have to be modeled explicitly. In particular, let assume a constraint $\mathscr{C}$ to be of the form $\leftarrow E_1, \ldots, E_n$. where the E_is are atoms[5]. This is with no loss of generality, as a constraint such as, for instance, $\leftarrow A, \mathit{not}\ B.$ can be reformulated as the program fragment $\leftarrow A, B'.$ $B' \leftarrow \mathit{not}\ B.$ Thus, an overall program $\Pi_{\mathscr{C}}$ can be seen as composed of answer set program Π plus a set $\{\mathscr{C}_1, \ldots, \mathscr{C}_v\}$ of constraints, and, possibly, an auxiliary program $\Pi_{\mathscr{C}}$, so that constraints can be defined on atoms belonging to either Π or $\Pi_{\mathscr{C}}$. We assume however that $\Pi_{\mathscr{C}}$ is stratified (i.e., it contains no cycles, cf. e.g., [28] for a formal definition) and that atoms of Π may occur in $\Pi_{\mathscr{C}}$ only in the body of rules (in the terminology of [16, 29], $\Pi_{\mathscr{C}}$ is a *top program* of Π).

[5] This limitation will be useful for the linear logic formulation (provided below).

Consider for instance $\Pi_{\mathscr{O}}$ to be composed of the following Π:

$$\{beach \leftarrow not\ mountain.\\ mountain \leftarrow not\ travel.\\ travel \leftarrow not\ beach.\\ hyperthyroidism.\}$$

plus the following $\Pi_{\mathscr{C}}$:

$$\{unhealthy \leftarrow beach, hyperthyroidism.\}$$

plus the constraint $\leftarrow unhealthy$.

The resulting theory will have resource-based answer sets $\{mountain, hyperthyroidism\}$, and $\{travel, hyperthyroidism\}$, while $\{beach, hyperthyroidism, unhealthy\}$ is excluded by the constraint. We now proceed to the formal definition.

Definition 8. *An Answer Set Theory $\mathscr{T}$ is a couple $\langle \Pi_{\mathscr{O}}, Constr \rangle$, with $\Pi_{\mathscr{O}} = \Pi \cup \Pi_{\mathscr{C}}$, where $\Pi_{\mathscr{C}}$ is a top program for Π, and where Constr is a set $\{\mathscr{C}_1, \ldots, \mathscr{C}_v\}$, $v \geq 0$, of constraints.*

Definition 9. *Given Answer Set Theory $\mathscr{T} = \langle \Pi_{\mathscr{O}}, Constr \rangle$, a resource-based Answer Set M for Π fulfills the constraints in Constr iff the answer set program Π' is consistent (in the sense of traditional answer set semantics), where Π' is obtained from $\Pi_{\mathscr{C}}$ by adding all atoms in M as facts, and all constraints in Constr as rules.*

Definition 10. *A* Resource-based Answer Set *M of Answer Set Theory $\mathscr{T} = \langle \Pi_{\mathscr{O}}, Constr \rangle$ is a resource-based answer set for Π that fulfills all constraints in Constr.*

It is easy to see that, in order to check that resource-based Answer Set M for Π fulfills the constraints, one can check consistency of Π' in a simple way, by: (i) computing (in polynomial time, cf., e.g., [20]) the unique answer set M'' of the stratified program Π'' obtained from $\Pi_{\mathscr{C}}$ by adding all atoms in M as facts, and then (ii) checking constraints on M'' by pattern-matching. Then, for constraints of the above simple form, we can conclude that:

Proposition 3. *Given Answer Set Theory $\mathscr{T}$, deciding about the existence of a resource-based answer set is a co-NP-complete problem.*

The partition of $\Pi_{\mathscr{O}}$ into Π and $\Pi_{\mathscr{C}}$ is not strictly necessary in the present context. In fact, one might simply check the constraints on $\Pi \cup \Pi_{\mathscr{C}}$. However, we choose to introduce the distinction because we believe that it may have a significance in terms of knowledge representation and elaboration-tolerance, in the sense of [10]. In fact, the same "generate" part (Π) can be customized by adding on top, as an independent layer, different "test" parts ($\Pi_{\mathscr{C}}$). Moreover, constraints might be generalized with respect to the simple form proposed above, for instance drawing inspiration from the discussion in [30, 31, 32], or also following the approach of *Answer Set Optimization* (cf. [33] and the references therein), which proposes constraints expressing complex preferences for choosing among answer sets.

For the sake of completeness, it may be interesting to illustrate the linear logic formalization of the full approach. To this aim, we have to resort to linear logic negation. A constraint $\mathscr{C} = \leftarrow E_1, \ldots, E_n.$ can in fact be represented in linear logic as $\mathscr{C}^L = {E_1}^{\perp} \parr \ldots \parr {E_n}^{\perp}$ where $\parr$ is the multiplicative disjunction, and $^{\perp}$ is linear logic negation, $A^{\perp}$ meaning "there is no proof for A".[6]

Thus, the overall linear logic theory would be $\Sigma_{\Pi_{\mathscr{O}}}$, and its resource-based answer sets should be matched against the constraints. Formally:

Definition 11. *Given resource-based answer set $M = \{A_1, \ldots, A_n\}$ for $\Pi_{\mathscr{O}}$, M is a resource-answer set for answer set theory $\mathscr{T} = \langle \Pi_{\mathscr{O}}, Constr \rangle$ where $Constr = \{\mathscr{C}_1, \ldots, \mathscr{C}_v\}$ iff tensor conjunction $A_1 \otimes \ldots \otimes A_n \otimes \mathscr{C}_1^L \otimes \ldots \otimes \mathscr{C}_v^L$ is provable from $\Sigma_{\Pi_{\mathscr{O}}}$.*

Notice that each constraint is provable whenever at least one of its disjuncts is not one of the A_i's. Then, in terms of equivalence between the logic programming and linear logic formulation, nothing really changes w.r.t. Theorem 3.

7 Concluding Remarks

In this paper, we have proposed an extension of the answer set semantics where ternary odd cycles are understood as exclusive disjunctions, similarly to binary even cycles. This extension stems from the interpretation of an answer set program as a linear logic theory, where default negation is considered to be a resource. The practical advantage is that there is more freedom in defining a search space, where constraints must however be defined in a separate "module" to be added to given answer set program.

Concerning implementation, which is of course a main future issue for this research, answer set solvers based on SAT appear to be good candidates for extension to the new setting. In fact, apart from checking for minimality of models (which is the part responsible for the additional complexity), they do not seem to need substantial modifications in order to cope with the new semantics, that thus might in principle be easily and quickly implemented.

References

[1] S. Costantini and A. Formisano, "RASP and ASP as a fragment of linear logic," *Journal of Applied Non-Classical Logics (JANCL)*, vol. 23, no. 1-2, pp. 49–74, 2013.

[2] J.-Y. Girard, "Linear logic," *Theoretical Computer Science*, vol. 50, pp. 1–102, 1987.

[3] S. Costantini and A. Formisano, "Answer set programming with resources," *Journal of Logic and Computation*, vol. 20, no. 2, pp. 533–571, 2010.

[4] 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*, vol. 64, no. 1, pp. 3–15, 2009.

[5] S. Costantini, A. Formisano, and D. Petturiti, "Extending and implementing RASP," *Fundamenta Informaticae*, vol. 105, no. 1-2, pp. 1–33, 2010.

[6] In fact, linear logic negation as such was used in [34] to model negation-as-failure in Prolog.

[6] C. Baral, *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.
[7] M. Truszczyński, "Logic programming for knowledge representation," in *Logic Programming, 23rd Intl. Conference, ICLP 2007* (V. Dahl and I. Niemelä, eds.), pp. 76–88, 2007.
[8] M. Gelfond, "Answer sets," in *Handbook of Knowledge Representation. Chapter 7*, Elsevier, 2007.
[9] V. Lifschitz, "Twelve definitions of a stable model," in *Proc. of the 24th Intl. Conference on Logic Programming* (M. Garcia de la Banda and E. Pontelli, eds.), vol. 5366 of *LNCS*, pp. 37–51, Springer, 2008.
[10] J. McCarthy, "Elaboration tolerance," in *Proc. of Common Sense'98*, 1998. Available at http://www-formal.stanford.edu/jmc/ elaboration.html.
[11] J.-Y. Girard, "Linear logic: Its syntax and semantics," in *Advances in Linear Logic* (J.-Y. Girard, Y. Lafont, and L. Regnier, eds.), Proc. of the 1993 Workshop on Linear Logic, pp. 1–42, Cambridge Univ. Press, 1995.
[12] P. Lincoln, "Linear logic," *ACM SIGACT News*, vol. 23, no. 2, pp. 29–37, 1992.
[13] M. I. Kanovich, "The complexity of horn fragments of linear logic," *Ann. Pure Appl. Logic*, vol. 69, no. 2-3, pp. 195–241, 1994.
[14] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proc. of the 5th Intl. Conference and Symposium on Logic Programming* (R. Kowalski and K. Bowen, eds.), pp. 1070–1080, The MIT Press, 1988.
[15] M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New Generation Computing*, vol. 9, pp. 365–385, 1991.
[16] S. Costantini, "Contributions to the stable model semantics of logic programs with negation," *Theoretical Computer Science*, vol. 149, no. 2, pp. 231–255, 1995.
[17] S. Costantini, "On the existence of stable models of non-stratified logic programs," *Theory and Practice of Logic Programming*, vol. 6, no. 1-2, 2006.
[18] J. W. Lloyd, *Foundations of Logic Programming*. Springer-Verlag, 1987.
[19] Web-references, "Some ASP solvers." Clasp: `potassco.sourceforge.net`; Cmodels: `www.cs.utexas.edu/users/tag/cmodels`; DLV: `www.dbai.tuwien.ac.at/proj/dlv`; Smodels: `www.tcs.hut.fi/Software/smodels`.
[20] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and expressive power of logic programming," *ACM Computing Surveys*, vol. 33, no. 3, pp. 374–425, 2001.
[21] J. Dix, "A classification theory of semantics of normal logic programs I-II.," *Fundam. Inform.*, vol. 22, no. 3, pp. 227–255 and 257–288, 1995.
[22] P. Cholewinski and M. Truszczynski, "Extremal problems in logic programming and stable model computation," in *JICSLP*, pp. 408–422, 1996.
[23] L. M. Pereira and A. M. Pinto, "Revised stable models - a semantics for logic programs," in *Progress in Artificial Intelligence, Proc. of EPIA 2005* (C. Bento, A. Cardoso, and G. Dias, eds.), vol. 3808 of *LNCS*, Springer, 2005.
[24] L. M. Pereira and A. M. Pinto, "Tight semantics for logic programs," in *Tech. Comm. of the 26th Intl. Conference on Logic Programming, ICLP 2010* (M. V. Hermenegildo and T. Schaub, eds.), vol. 7 of *LIPIcs*, pp. 134–143, 2010.
[25] M. Osorio and A. López, "Expressing the stable semantics in terms of the pstable semantics," in *Proc. of the LoLaCOM06 Workshop*, vol. 220 of *CEUR Workshop Proc.* , CEUR-WS.org, 2006.
[26] M. Osorio, J. A. N. Pérez, J. R. A. Ramírez, and V. B. Macías, "Logics with common weak completions," *J. Log. Comput.*, vol. 16, no. 6, pp. 867–890, 2006.
[27] M. Cadoli, "The complexity of model checking for circumscriptive formulae," *Inf. Process. Lett.*, vol. 44, no. 3, pp. 113–118, 1992.
[28] K. R. Apt and R. N. Bol, "Logic programming and negation: A survey," *J. Log. Program.*, vol. 19/20, pp. 9–71, 1994.

[29] V. Lifschitz and H. Turner, "Splitting a logic program," in *Proc. of ICLP'94, Intl. Conference on Logic Programming*, pp. 23–37, 1994.

[30] S. Costantini and A. Formisano, "Weight constraints with preferences in ASP," in *Proc. of Intl. Conf. on Logic Programming and Nonmonotonic Reasoning LPNMR'11*, vol. 6645 of *LNCS*, Springer-Verlag, 2011.

[31] S. Costantini and A. Formisano, "Nested weight constraints in ASP," *Fundamenta Informaticae*, vol. 124, no. 4, pp. 449–464, 2013.

[32] S. Costantini and A. Formisano, "Augmenting weight constraints with complex preferences," in *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, AAAI Press, 2011. Technical report SS-11-06.

[33] G. Brewka, I. Niemelä, and M. Truszczynski, "Answer set optimization," in *IJCAI-03, Proc. of the Eighteenth Intl. Joint Conference on Artificial Intelligence* (G. Gottlob and T. Walsh, eds.), pp. 867–872, Morgan Kaufmann, 2003.

[34] S. Cerrito, "A linear axiomatization of negation as failure," *Journal of Logic Programming*, vol. 12, no. 1&2, pp. 1–24, 1992.