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

Stefania Costantini
DISIM, Università di L'Aquila, Italy, stefania.costantini@univaq.it

Andrea Formisano
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. An advantage is that of avoiding inconsistencies, as every program has a (possibly empty) resource-based answer set. This implies however the introduction of a different way of representing constraints. We provide a characterization of the new semantics as a variation of the answer set semantics, and also in terms of Autoepistemic Logic. The latter characterization leads to a way of computing resource-based answer set via answer set solvers.

Keywords: Answer Set Programming, Default Negation, Linear Logic, Autoepistemic Logic

1. Introduction

In [14], we proposed a comparison between RASP and linear logic [24], where RASP [9, 8, 16] is a recent extension of the Answer Set Programming (ASP) framework obtained by explicitly introducing the notion of *resource*. ASP is nowadays a well-established programming paradigm, with applications in many areas (see among many [22, 36, 2, 46, 21, 28] and the references therein) and with many freely available efficient inference engines (usually called "answer set solvers" [48]).

RASP is a significant extension of ASP, supporting both formalization and quantitative reasoning on consumption and production of amounts of resources. In [14] we proved in particular that RASP

[*]A short version of this paper appeared in [12]. This research is partially supported by GNCS-13 and GNCS-14 projects.

(and thus, ASP as a particular case) corresponds to a fragment of linear logic. This was accomplished by providing a revertible translation of RASP programs into a linear logic theory. The result implies that a RASP inference engine (such as Raspberry [16]) might 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 and discussed in [29].

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 model-theoretic definition of the new semantics, that we call *Resource-based Answer Set Semantics* (RAS). Unlike the AS semantics, where programs can be *inconsistent*, i.e., may have no answer sets, under the new semantics every program admits resource-based answer sets. Consistency of answer set programs is related to the occurrence of *negative cycles*, i.e., cycles through negation, and to their connections to other parts of the program. Criticality is in particular related to the occurrence of "odd cycles", which are negative cycles involving an odd number of negated atoms (w.r.t. "even" cycles). Under the RAS semantics basic odd cycles, similarly to basic even cycles in AS, are interpreted as exclusive disjunctions, while in AS they are a potential cause of inconsistency. Constraints must however be suitably formalized, while in ASP they are "simulated" via unary odd cycles. Under the extended semantics we thus have programs augmented with a set of constraints. We argue that the proposed extended semantics may represent the understanding of a rational agent working on an incomplete or approximate description of a given situation, that the agent tries to interpret to the best of its currently available knowledge. We also argue that representing constraints separately can even lead to more generality and to an improved elaboration-tolerance (in the sense of [37]). In the proposed approach, the "practical expressivity" in terms of knowledge representation is improved, while computational complexity remains the same.

We provide a characterization of resource-based answer set semantics as a variation of the answer set semantics, which exploits a simple variant of Gelfond and Lifschitz's "Gamma" operator [22], though in an iterative way over a subdivision of the given program into layers, defined according to the principles introduced in [30]. We also provide a characterization in terms of autoepistemic logic, by extending the approach proposed in [35] for answer set programming. We show that the latter characterization provides a basis for the transposition of a given program Π into a new program Π' such that the answer sets of Π', that can be computed by any answer set solver, correspond to the resource-based answer sets of Π.

The paper is organized as follows. In Section 2 we provide the necessary background on ASP, and discuss some properties that will be useful for defining the new semantics. In Section 3 we shortly recall linear logic, and specialize the method defined in [14] 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 4, 5, and 6 the semantic extension is described, discussed, formalized and characterized under various perspectives. In Sections 7 and 8 we present some examples of application, and discuss how to integrate constraints into the new setting. Finally, in Section 9 we conclude.

2. Answer Set Semantics: Definition, Properties and Observations

The Answer Set Programming (ASP) paradigm is based upon the answer set semantics [22, 23], where an answer set program encodes a given problem, and each answer set of the program (if any exists) represents a solution of the problem. To find these solutions, an ASP-solver is used. Several solvers have became available, see [48], each of them being characterized by its own prominent valuable features. The expressive power of ASP and its computational complexity have been deeply investigated (cf. e.g., [17]). The reader may refer to [2, 46, 21, 28] about the ASP paradigm and its applications. In this section we concentrate on semantic aspects that will be needed in the rest of the paper: we recall in detail the definition of the answer set semantics, and discuss some of its properties. We refer to the definitions introduced or reported in [1, 32, 22, 23], among which the standard definitions of propositional general logic programs. We will sometimes re-elaborate definitions and terminology (without substantial change), in a way which is functional to the discussion proposed in subsequent sections.

In the answer set semantics [22, 23][1], an answer set program Π 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 *unit rule*, or *fact*. A rule with empty head, of the form $\leftarrow L_1, \ldots, L_n.$, is a *constraint*, and states that literals $L_1, \ldots, L_n$ cannot be simultaneously true. A positive logic program is a logic program including no negative literals and no constraints.

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., [23]), and we do not refer (at the moment) to the various useful programming constructs defined and added over time to the basic paradigm.

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. As it is customary in the ASP literature, we will implicitly refer to the "ground" version of Π, which 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.

Conceptually, the answer sets semantics 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 the simple program $\{q \leftarrow not\ p.\ \ p \leftarrow not\ q.\}$. 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

[1]The answer set semantics was originally denominated "stable model semantics".

self-supporting set of consequences of a given program. The program $\{p \leftarrow not\ p.\}$ has no answer sets either as it is considered to be 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.

As mentioned in the Introduction (cf. [6, 31, 7] for extensive discussions) consistency of answer set programs is related to the occurrence of *negative cycles*, i.e. cycles through negation, and to their connections to other parts of the program. Criticality is in particular related to the occurrence of "odd cycles", i.e. negative cycles involving an odd number of negations: $p \leftarrow not\ p$ is the basic case, though odd cycles may involve any odd number of negated atoms, either directly or through positive dependencies. This issue is further discussed in the rest of the paper.

Below is the formal specification of the answer set semantics, elaborated from [22]. Preliminarily, we remind the reader that the least Herbrand model of a positive logic program P can be computed (cf. [32]) by means of the T_P operator, that can be defined as follows (where the original definition is due to Van Emden and Kowalski). We then introduce the definition of reduct, the Γ operator and finally the definition of answer sets.

Given a positive program P, let

$$T_P(I) \quad = \quad \big\{A : \text{ there exists rule } A \leftarrow A_1, \ldots, A_m \text{ in } P \text{ where } \{A_1, \ldots, A_m\} \subseteq I\big\}$$

The T_P operator always has a unique least fixpoint, that for finite propositional programs is computable in a finite number of steps.

The following definition of (GL-)reduct, that will be often useful in what follows, is due to Gelfond and Lifschitz.

Definition 2.1. Let I be a set of atoms and Π a program. The reduct of Π modulo I is a new program, denoted as Π^I, obtained from Π by:

1. removing from Π all rules which contain a negative premise *not* A such that $A \in I$; and by

2. removing all negative premises from the remaining rules.

Notice that for each negative premise *not* A which is removed by step 2, it holds that $A \not\in I$: otherwise, the rule where it occurs would have been removed by step 1. We can see that Π^I is a positive logic program. Answer sets are defined as follows, via the GL-operator Γ.

Definition 2.2. [The Gelfond-Lifschitz (GL) Operator Γ] Let I be a set of atoms and Π a program. Let Π^I be the reduct of Π modulo I, and J be its least Herbrand model. We define $\Gamma_\Pi(I) = J$.

Definition 2.3. Let I be a set of atoms and Π a program. I is an *answer set* of Π iff $\Gamma_\Pi(I) = I$.

Non-empty answer sets form an anti-chain with respect to set inclusion. It will be useful in what follows to consider a simple property of Γ_Π (see [6]): if M is a (minimal) model of Π, then $\Gamma_\Pi(M) \subseteq M$. We also remind the reader of the following definition and properties.

Definition 2.4. Given a non-empty set of atoms I and a rule ρ of the form $A \leftarrow A_1, \ldots, A_n, not\ B_1, \ldots, not\ B_m$, we say that ρ is *supported* in I iff $\{A_1, \ldots, A_n\} \subseteq I$ and $\{B_1, \ldots, B_m\} \cap I = \emptyset$.

Definition 2.5. Given a program Π and a non-empty set of atoms I, we say that I is *supported* w.r.t. Π (or for short Π-supported) iff $\forall A \in I$, A is the head of a rule ρ in Π which is supported in I.

Some of the classical models of Π (interpreted in the obvious way as a classical first-order theory, i.e., where the comma stands for conjunction and the symbol $\leftarrow$ stands for implication) may be supported. Answer sets of Π, if any exists, are supported minimal classical models of the program. They however enjoy a stricter property, that we define below.

Definition 2.6. Given program Π and a set of atoms I, an atom $A \in I$ is *consistently supported* w.r.t. Π and I iff there exists a set S of rules of Π such that the following conditions hold (where we say that A is consistently supported via S):

1. every rule in S is supported in I;
2. exactly one rule in S has conclusion A;
3. A does not occur in the positive body of any rule in S;
4. every atom B occurring in the positive body of some rule in S is in turn consistently supported w.r.t. Π and I via the set of rules $S' \subseteq S$.

About point 3, notice that A cannot occur in the negative body of any rule in S either, since all such rules are supported in I.

Definition 2.7. Given a program Π and a set of atoms I, we say that I is a *consistently supported* set of atoms (w.r.t. Π) iff $\forall A \in I$, A is consistently supported w.r.t. Π and I. We say that I is a *maximal consistently supported* set of atoms (MCS, for short) iff there not exists $I' \supset I$ such that I' is consistently supported w.r.t. Π. We also say for short that I is an MCS for Π.

Observe that an MCS can be empty only if it is unique, i.e, only if no non-empty consistently supported set of atoms exists. The following proposition holds:

Proposition 2.1. Any answer set M of program Π is an MCS for Π.

However, maximal consistently supported sets of atoms are not necessarily answer sets. Take for instance the following program Π

$$p \leftarrow a. \quad a \leftarrow not\ p.$$

which admits no answer sets: the set of atoms $\{a\}$, which is not even a classical model of Π, is nonetheless an MCS for this program. Instead, the set of atoms $\{p\}$, which is indeed a classical minimal model of Π, is not an MCS for this program: in fact, it violates condition (1) of Definition 2.6, since atom p actually depends upon its own negation and thus the second rule is not supported in $\{p\}$.

We will now recall some properties of the answer set semantics which will be useful in what follows. The following definitions appear in [18], where Π^M is the reduct of program Π w.r.t. set of atoms M, as recalled in Definition 2.1.

Definition 2.8. The sets of atoms a single atom A depends upon, directly or indirectly, positively or negatively, is defined as $dependencies\_of(A) = \{B : A \text{ depends on } B\}$.

The former definition is provided with some approximation, as dependencies should be formally checked on the *dependency graph* [1] of the program at hand. Intuitively, B belongs to $\mathit{dependencies\_of}(A)$ if in the dependency graph there exists a nonempty directed path from A to B.

Definition 2.9. Given a program Π and an atom A, $\mathit{rel\_rul}(\Pi;A)$ is the set of relevant rules of Π with respect to A, i.e. the set of rules that contain an atom $B \in (\{A\} \cup \mathit{dependencies\_of}(A))$ in their heads.

The notions introduced by Definitions 2.8 and 2.9 for an atom A can be plainly generalized to any set X of atoms. Notice that, given an atom (or a set of atoms) X, $\mathit{rel\_rul}(\Pi;X)$ is a subprogram of Π. For example, if $\Pi = \{a.\ \ b \leftarrow a.\}$, then $\mathit{dependencies\_of}(a) = \emptyset$, $\mathit{dependencies\_of}(\{a,b\}) = \{a\}$, and $\mathit{rel\_rul}(\Pi;\{a,b\}) = \Pi$.

If we consider atom A and set S in Definition 2.6, we can see that $S \subseteq \mathit{rel\_rul}(\Pi;A)$.

As observed before, ASP programs may involve cycles which are used in the ASP programming paradigm for generating the search space and for modeling constraints. A program can be seen as divided into components, some of them involving cyclic dependencies. Since this will be of interest in the rest of the paper, we now introduce some notions about how an ASP program is, in general, structured. We provide some preliminary notions, and then the definition of decomposition.

Definition 2.10. An answer set program Π is *cyclic* if for every atom A occurring in the head of some rule ρ in Π, it holds that $A \in \mathit{dependencies\_of}(A)$. In particular, Π is *negatively cyclic* if some of these dependencies is negative.

A program Π in which there is no head A such that $A \in \mathit{dependencies\_of}(A)$ is called *acyclic*.

Notice that, by Definition 2.10 there exist programs that are neither cyclic nor acyclic, though involving cyclic and/or acyclic fragments as subprograms.

Moreover, an acyclic program has a unique answer set, coinciding with the set of true atoms of its well-founded model [47] (which is three-valued, and so it is composed of a set W^+ of true atoms and a set W^- of false atoms, all the other atoms having the truth value "undefined"). In general, the answer set semantics *extends* the well-founded semantics since, for every answer set M, $W^+ \subseteq M$.

The following definitions establish how program fragments can be either independent of or related to each other.

Definition 2.11. A subprogram Π_s of a given program Π is *self-contained* (w.r.t. Π) if the set X of atoms occurring (either positively or negatively) in Π_s is such that $\mathit{rel\_rul}(\Pi;X) \subseteq \Pi_s$.

Definition 2.12. Given two subprograms Π_{s_1}, Π_{s_2} of a program Π, Π_{s_2} is *on top* of Π_{s_1} if the set X_2 of atoms occurring in the head of some rule in Π_{s_2} is such that $\mathit{rel\_rul}(\Pi;X_2) \subseteq \Pi_{s_2} \cup \Pi_{s_1}$, and the set X_1 of atoms occurring (either positively or negatively) only in the body of rules of Π_{s_2} is such that $\mathit{rel\_rul}(\Pi;X_1) \subseteq \Pi_{s_1}$.[2]

Notice that, by Definition 2.12, if Π_{s_2} is *on top* of Π_{s_1}, then X_1 is a *splitting set* for Π in the sense of [30].

Definition 2.13. A program obtained as the union of a set of cyclic or acyclic programs, none of which is on top of another one, is called a *jigsaw* program.

[2]This notion was introduced in [6, 30].

Definition 2.14. Let Π be a program and Π_s a jigsaw subprogram of Π. Then, Π_s is *standalone* (w.r.t. Π) if it is self-contained (w.r.t. Π).

In what follows, we will often refer to a standalone program Π_s without mentioning the including program Π. In such cases Π will be identifiable from the context.

The following property, that comes as (an almost immediate) consequence of former definitions, will be very important in the rest of the paper. It states that a program can be divided into subprograms where a standalone one can be understood as the bottom *layer*, which is at the basis of a "tower" where each level is a jigsaw subprogram standing on top of lower levels. We omit the proof for lack of space.

Proposition 2.2. A non-empty answer set program Π can be seen as divided into a sequence of *components*, or layers, $C_1, \dots, C_n$, $n \geq 1$ where: C_1, which is called the *bottom* of Π, is a standalone program; each component C_i, for $i > 1$, is a jigsaw program which is on top of $C_{i-1} \cup \dots \cup C_1$.

The advantage of such a decomposition is that, by the *Splitting Theorem* introduced in [30], the computation of answer sets of Π can be divided into subsequent stages.

Proposition 2.3. Consider a non-empty ASP program Π, divided according to Proposition 2.2 into components $C_1, \dots, C_n$, $n \geq 1$. An answer set S of Π (if any exists) can be computed incrementally as follows:

step 0. Set $i = 1$.

step 1. Compute an answer set S_i of component C_i (for $i = 1$, this accounts to computing an answer set of the standalone bottom component).

step 2. Simplify program C_{i+1} by: (i) deleting all rules that have *not* B in their body, for some $B \in S_i$; (ii) deleting (from the body of the remaining rules) every literal *not* F where F does not occur in the head of rules of C_{i+1} and $F \notin S_i$, and every atom E with $E \in S_i$.[3]

step 3. If $i < n$ set $i = i + 1$ and go to step 1, else set $S = S_1 \cup \dots \cup S_n$.

All answer sets of Π can be generated via backtracking (from any possible answer set of C_1, combined with any possible answer set of simplified C_2, etc.). If no (other) answer set of Π exists, then at some stage step 1 will fail. An incremental computation of answer set have also been adopted in [19].

3. Relationship between Linear Logic and ASP

Linear logic, as it is well-known, can be considered as a *resource-sensitive* refinement of classical logic. Intuitively speaking, in linear logic two assumptions of a formula P are distinguished from a single assumption of it. *Multiplicative conjunction* or tensor product $\otimes$ between two formulas (intended as resources) intuitively means "I have both", while *additive conjunction* & means "I have a choice". Linear implication $P \multimap Q$ 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.)

[3]Notice that, due to the simplification, C_{i+1} becomes standalone.

In [14] we have shown that RASP (and thus ASP, which can be seen as a particular case of RASP) can be defined as a (propositional) fragment of linear logic [24, 26] by translating 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 program rules production processes), and additive conjunction & (to represent alternative/exclusive resource allocation). In well-known terminology, we adopted formulas belonging to the so-called Horn-fragment of linear logic.

In this section, we specialize to pure ASP the method defined in [14], in order to exploit the representation for introducing the new resource-based answer set semantics.

A positive ASP program Π (i.e., a program without default negation) can be transformed into a corresponding Linear Logic RASP Theory as follows (where the reverse translation is also possible). In particular, in the definition below each atom q in the body of the j-th rule of the 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 $q^j{:}1$, i.e. for a unary quantity of the "resource" corresponding to atom q.[4] 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) occurring in the body. In logic programming in fact, the truth of an atom can be used to prove several consequences (through different rules). 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. Instead, we devised a method which remains within this fragment. In particular, from an atom A we "produce" via a linear implication its standardized-apart "copies" that will be consumed by the linear logic counterpart of the rules which have A in their body.

Definition 3.1. 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):

$$\begin{aligned} &B_{i,1} \otimes \cdots \otimes B_{i,\ell_i} \multimap A_i \quad \text{for } i = 1, \ldots, h \\ &A_1 \,\&\, \cdots \,\&\, A_h \multimap A \end{aligned}$$

- 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 $A \multimap A^1{:}1 \otimes \cdots \otimes A^m{:}1$.

- Replace in Σ_Π any linear implication $D_1 \otimes \cdots \otimes D_n \multimap H$ with the linear implication $D_1 \otimes \cdots \otimes D_n \multimap H \otimes D_1^R \otimes \cdots \otimes D_n^R$.

[4]In RASP terminology, a writing of the form $q{:}a$ denotes an *amount a* of the *resource q*: RASP allows in fact for arbitrary quantities.

Let us illustrate 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. Connective $\leftarrow$ is turned into linear implication. The purpose of the linear implication $A_1 \& \cdots \& A_h \multimap A$ is that of enabling the derivation of A through the linear logic counterpart of either of its defining rules. 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 $A \multimap A^1{:}1 \otimes \cdots \otimes A^m{:}1$ models the fact that A is a resource available to any rule that may need to use it.

Notice the introduction of a fresh atom D_i^R corresponding to each atom D_i, in the last step of the translation. These fresh atoms are called the *r-copies* of the D_is. 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 of Π. R-copies in fact allow us to establish a correspondence between answer sets of Π and maximal tensor conjunctions provable from Σ_Π, where:

Definition 3.2. Given a linear logic theory Σ, a tensor conjunction of atoms $A_1 \otimes \cdots \otimes A_n$ $(n \geq 0)$, is called *maximally provable* if it is provable from Σ, and for any atom B, the tensor conjunction $A_1 \otimes \cdots \otimes A_n \otimes B$ is not provable from Σ (we equivalently talk about a maximal tensor conjunction provable from Σ).

Lemma 3.1. 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 $\mathscr{A}$.

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 a given program Π. To represent full RASP (and thus full ASP) we improve the transformation devised in Definition 3.1:

Definition 3.3. Given an 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* $A^{j_1}{:}1, \ldots,$ *not* $A^{j_n}{:}1$ all the occurrences introduced in this manner, add the (linear) fact *not* $A{:}n$ to the translation of Π. This to indicate that the total amount of resource *not* $A{:}n$ that is needed in order to provide unary items to every rule that need them is precisely n.

- 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.[5]

[5]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* $A^{k2}{:}1$.

- Apply the rewriting indicated in Definition 3.1 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:

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

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 this case the assumption becomes totally unavailable, as proving A consumes the full available quantity of the "resource" *not A*.

The transformation of Definitions 3.1 and 3.3 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.

We are able to state (neglecting, by abuse of notation, the syntactic distinction between an atom A and its r-copy A^R) the following

Theorem 3.1. Let Π be an ASP program, and Σ_Π the corresponding Linear Logic RASP Theory, obtained according to Definitions 3.1 and 3.3. Let $M = \{A_1, \ldots, A_n\}$ be an answer set for Π. Then, $A_1 \otimes \cdots \otimes A_n$ is a maximal tensor conjunction provable from Σ_Π.

Note that the reverse does not necessarily hold, because there are maximal tensor conjunctions provable from Σ_Π that are not answer sets. This is due (as discussed in [14]) to the lack of relevance of the answer set semantics (cf., [18]), but also to the locality of a proof-based system such as linear logic.

4. Negation as a Resource: a novel view on Answer Set Semantics

It is interesting to investigate how the linear logic formulation illustrated in previous section prevents contradictions. Consider for example the program $\Pi_1 = \{p \leftarrow not\ p.\}$, which is a unary odd cycle, and is (as discussed before) inconsistent under the answer set semantics. It is transformed into:

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

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 3.3). We can verify that the singleton tensor conjunction p is not provable: in fact, the derivation 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 the ternary odd cycle $\Pi = \{a \leftarrow not\ b.\ \ b \leftarrow not\ c.\ \ c \leftarrow not\ a.\}$ which is also inconsistent under the answer set semantics. In our formulation, Σ_Π is the following:

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

From this linear logic theory we can prove 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 no longer available: in fact, the &-Horn implication related to b generates (indifferently) only one of the two items, and one item has already been requested 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 one item has already been requested 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, proving b becomes no longer possible.

This implies that the 3-atoms odd cycles is interpreted in the linear logic view as an exclusive disjunction, exactly like the 2-atoms even cycle (such as $\{q \leftarrow not\ p.\quad p \leftarrow not\ q.\}$) in AS. We call $\{a\}$, $\{b\}$, and $\{c\}$ *resource-based answer sets*, for which we provide below both a logic programming and an autoepistemic logic characterization. The resource-based answer 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:

$$\begin{aligned}
\{&beach \leftarrow not\ mountain.\\
&mountain \leftarrow not\ travel.\\
&travel \leftarrow not\ beach.\}
\end{aligned}$$

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* [5] $\{a \leftarrow not\ b, not\ c.\ \ b \leftarrow not\ c, not\ a.\ \ c \leftarrow not\ a, not\ b.\}$.

We argue that resource-based answer set semantics is reasonable, and might possibly be adopted as a *proper extension* of the answer set semantics. As mentioned before, the answer set semantics *extends* the well-founded semantics (wfs). Wfs in fact assigns to a logic program Π a unique, three-valued model, where atoms that cannot be definitely established to be true or false are deemed to be *undefined*, while these atoms are instead assigned a truth value true/false in an answer set, if any exists. As also observed before, the answer set semantics selects those (two-valued) classical minimal models of a given program that are consistently supported (Definitions 2.6-2.7). Consistent support means that the support of atom A does not depend (directly or indirectly) neither upon A itself, nor upon the negation of another true atom, including itself. For *even cycles* such as $\{e \leftarrow not\ f.\ \ f \leftarrow not\ e.\}$, two answer sets can be found, namely $\{e\}$ and $\{f\}$, fulfilling all conditions. This extends to wider program including such cycles. For odd cycles (such as unary odd cycles of the form $\{p \leftarrow not\ p.\}$ and ternary odd cycles of the form $\{a \leftarrow not\ b.\ \ b \leftarrow not\ c.\ \ c \leftarrow not\ a.\}$) it is not possible to assign truth values to their composing atoms in classical models according to Definitions 2.6-2.7. This is the reason why under the answer set semantics a program including such cycles is inconsistent.[6] In a sense, the answer set semantics is still three-valued, as sometimes it is able to assign truth value to atoms, and sometimes (when the program is inconsistent) leaves all of them undefined. Resource-based answer set semantics instead is able to cope with any kind of cycle, by assigning truth value "true" to all atoms which enjoy consistent support. This is done by resorting to MCSs instead of classical models. For the unary odd cycle, p is deemed to be false because, were it true, it would depend upon the negation of a true atom (itself). The ternary odd cycle has the three resource-based answer sets $\{a\}$, $\{b\}$ and $\{c\}$. Taking for instance resource-based answer set $\{a\}$, atom b is deemed false to provide support, while atom c cannot possibly be supported in this set.

There are other semantic approaches to managing odd cycles, such as (among many) [41, 42] and [39, 40], with their own sound theoretical foundations. They 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.

5. RAS Semantics as a variation of the AS Semantics

Below we introduce some modifications to the original definition of the answer set semantics, so as to define resource-based answer set semantics. Some preliminary elaboration is needed.

As stated by Proposition 2.2, a non-empty answer set program Π can be seen as divided into a sequence of *components*, or layers, $C_1, \ldots, C_n$, $n \geq 1$ where: each C_i, $i \geq 1$, is the union of a set of cyclic or acyclic subprograms none of which is on top of the others; the bottom component C_1 is standalone; each component C_i, for $i > 1$, is on top of the lower layers. Based upon this decomposition, the answer sets of a given program can be computed incrementally (in a bottom-up fashion). Resource-based answer sets can be computed in a similar way. Therefore, we start by defining the notion of resource-based answer sets of a given standalone program (relying upon preliminary definitions in Section 2).

The semantic variation that we propose implies slight modifications in the definition of the T_P and the Γ operator, aimed at forbidding the derivation of atoms that necessarily depend upon their own negation.

The modified reduct, in particular, keeps track of negative literals which the "traditional" reduct would remove.

Definition 5.1. Let I be a set of atoms and let Π be a program. The *modified reduct* of Π modulo I is a new program, denoted as $\hat{\Pi}^I$, obtained from Π by removing from Π all rules which contain a negative premise $not\ A$ such that $A \in I$.

[6]Unless "handles" are provided from other parts of the program, see [6, 7, 31] for details.

Based upon the modified reduct, we can define a modified T_P which derives only those facts that do not depend (neither directly nor indirectly) on their own negation.

For simplicity, let us consider each rule of a program as reordered by grouping its positive and its negative literals, as follows:

$$A \leftarrow A_1,\ldots,A_m,\ not\ B_1,\ldots,not\ B_n$$

Moreover, let us define a *guarded atom* any expression of the form $A||G$ where A is an atom and $G = \{not\ D_1,\ldots,not\ D_\ell\}$ is a possibly empty collection of negative literals (we say that A is guarded by the D_is),

Definition 5.2. (Modified T_P)
Given a propositional program P, let

$$\begin{array}{rcl} T_P(I) & = & \Big\{A||G_1 \cup \cdots \cup G_r \cup \{not\ B_1,\ldots,not\ B_n\} : \text{ there exists a rule} \\ & & \quad A \leftarrow A_1,\ldots,A_r, not\ B_1,\ldots,not\ B_n \text{ in } P \\ & & \quad \text{such that } \{A_1||G_1,\ldots,A_r||G_r\} \subseteq I \\ & & \quad \text{and } \ not\ A \notin \{not\ B_1,\ldots,not\ B_n\} \cup G_1 \cup \cdots \cup G_r \Big\}. \end{array}$$

Moreover, for each $n \geq 0$, let T_P^n be the set of guarded atoms defined as follows:

$$\begin{array}{rcl} T_P^0 & = & \{A||\emptyset : \text{ there exists unit rule } A \leftarrow \text{ in } P\} \\ T_P^{n+1} & = & T_P(T_P^n) \end{array}$$

The *least contradiction-free Herbrand set* of P is the following set of atoms:

$$\hat{T}_P = \big\{A : A||G \in T_P^i \text{ for some } i \geq 0\big\}.$$

Notice that the set $\hat{T}_P$ does not necessarily coincide with the full least Herbrand model of the "traditional" reduct, as its construction excludes from the result those atoms that are guarded by their own negation. We can finally define a modified version of the Γ operator.

Definition 5.3. [Operator $\hat{\Gamma}$] Let I be a set of atoms and Π a program. Let $\hat{\Pi}^I$ be the modified reduct of Π modulo I, and J be its least contradiction-free Herbrand set. We define $\hat{\Gamma}_\Pi(I) = J$.

It is easy to see that

Proposition 5.1. Let I_1, I_2 be sets of atoms and Π a program. If $I_1 \subseteq I_2$ then $\hat{\Gamma}_\Pi(I_1) \supseteq \hat{\Gamma}_\Pi(I_2)$.

Proof (Sketch). The claim holds because, by Definition 5.1, a larger I leads to a potentially smaller modified reduct, since it may causes the removal of more rules. Consequently, this leads to a potentially smaller least contradiction-free Herbrand set. □

For reasons that will be clarified below, we need to consider potentially supported sets of atoms.

Definition 5.4. Let Π be a program, and let I be a set of atoms. I is Π-*based* iff for any $A \in I$ there exists rule ρ in Π with head A.

We need to state the following important result.

Theorem 5.1. Let Π be a standalone program, and let I be a non-empty Π-based set of atoms. Let $M = \hat{\Gamma}_\Pi(I)$. M is a consistently supported set of atoms for Π iff $M \subseteq I$.

Proof. Let M be a consistently supported set of atoms for Π. By definition, M is Π-based and provides consistent support for every $A \in M$ through a subset $\mathscr{S}$ of the rules of Π. Thus, M can be obtained as $\hat{\Gamma}_\Pi(I)$ for any of its supersets I such that, for each rule $\hat{\rho}$ in Π where $\hat{\rho} \notin \mathscr{S}$, we have that: either $\hat{\rho}$ is canceled when computing the modified reduct, or $\hat{\rho}$ cannot be applied by the modified T_P based upon the remaining rules. It can be $M = I$ in two cases: either $\forall \rho \in \Pi$, $\rho \in \mathscr{S}$ or for each $\hat{\rho} \notin \mathscr{S}$, there exists atom A in its negative body where $A \in M$, so that $\hat{\rho}$ is canceled when computing the modified reduct. Let $M = \hat{\Gamma}_\Pi(I)$, where I is a Π-based set of atoms, and let $M \subseteq I$. If $M \subseteq I$, this means that there exist atom $F \in I$, $F \notin M$. The reasons may be one or both of the following: (a) some or all rules in Π with conclusion F have been canceled by the modified reduct; the remaining rules cannot be applied in producing the set $\hat{T}_P$, because there is a dependency of F upon either itself (so no derivation is possible) or on its own negation. By construction in fact, rules that have not been canceled allow the modified T_P to derive all and only the consistently supported atoms, and therefore M is a consistently supported set of atoms for Π. $\square$

From the above theory and from Proposition 5.1 we can draw the following:

Corollary 5.1. Let Π be a standalone program, and let I be a non-empty Π-based set of atoms. Let $M = \hat{\Gamma}_\Pi(I)$. M is an MCS for Π iff $M \subseteq I$, and there is no proper subset I_1 of I such that $\hat{\Gamma}_\Pi(I_1) \subseteq I_1$.

The Corollary states that MCSs are the maximal sets M that can be obtained via $\hat{\Gamma}_\Pi(I)$ from some I, in the sense that any smaller I_1 produces via $\hat{\Gamma}_\Pi$ an M_1 that does not respect the condition stated by Theorem 5.1, and so it is not even a consistently supported set of atoms for Π. We are now ready to define resource-based answer sets of a standalone program.

Definition 5.5. Let Π be a standalone program, and let I be a Π-based set of atoms. $M = \hat{\Gamma}_\Pi(I)$ is a *resource-based answer set* of Π iff M is an MCS for Π.

It is easy to see that, from Proposition 2.1, the next two results follow.

Proposition 5.2. Any answer set of a standalone program Π is a resource-based answer set of Π.

Proposition 5.3. Let Π be an acyclic standalone program. Then, the unique answer set of Π is the unique resource-based answer set of Π.

These are consequences of the fact that consistent ASP programs are non-contradictory, and the modified T_P, in absence of contradictions (i.e., in absence of atoms necessarily depending upon their own negations), operates exactly like T_P. In case of acyclic programs, the unique answer set I is also the unique MCS as the computation of the modified reduct does not cancel any rule, and the modified T_P can thus draw the maximum set of conclusions, coinciding with I itself.

Being an MCS, a resource-based answer set can be empty only if it is the unique resource-based answer set. This can be the case if the modified reduct cancels all rules, like for program $\{p \leftarrow \mathit{not}\ p.\}$, or whether the modified T_P cannot draw any conclusion, like for the program $\{a \leftarrow b.\ \ b \leftarrow a.\}$.

A discussion is in order concerning the understanding of negation under the new semantics. For odd cycles like $\{a \leftarrow not\ b.\ \ b \leftarrow not\ c.\ \ c \leftarrow not\ a.\}$, it is often said that such a cycle is inconsistent because every atom depend upon its own negation. We should nevertheless consider that also in an even cycle such that $\{e \leftarrow not\ f.\ \ f \leftarrow not\ e.\}$ any atom anyway depends upon its own negation: however, in the latter case the answer set semantics states that one can conclude e by assuming *not* f, or in alternative conclude f by assuming *not* e. In fact, as remarked by Gelfond and Lifschitz, given an atom A, the expression *not* A is not to be interpreted as the classical negation of A, rather as the proposition "*not* A is assumed", or, following [20], "A is not believed". Thus, *not* A is a defeasible assumption that a rational agent may embrace in order to produce a rational account of a situation given the description that the agent is aware of (i.e., from the given program). Notice that all conclusions drawn through layers of default negation are in turn defeasible, so they are themselves assumptions, or beliefs. Such a default negation should in our opinion enjoy the following features:

- a rational agent cannot reasonably believe both A and *not* A;
- if a rational agent has assumed *not* B to (defeasibly) conclude some other fact A, then the agent cannot neither rationally assume *not* A nor it can rationally conclude B.
- a rational agent is not necessarily compelled to assume either C or *not* C for every fact C, unless this is strictly required to devise a credible belief state. The judgment about some C can thus remain suspended. This does not account to considering C "unknown" or "undefined", but simply accounts to not making assumptions about C.

So, no multi-valued logic is really needed. Rather, when an agent derives a two-valued belief set (e.g., an answer set or a resource-based answer set) from a program:

- atoms included in the belief set are either believed to be definitely true (i.e., those that are true w.r.t. the well-founded semantics), or have been rationally (though defeasibly) assumed to hold;
- atoms not included in the belief set are either believed to be definitely false (i.e., those that are false w.r.t. the well-founded semantics), or have been rationally assumed not to hold, or no judgment about them has been devised.
- Consider however that, when generalizing computation of resource-based answer set semantics to a general program composed of several layers, suspending the judgment on some atom at one level does not prevent taking a position at upper layers.

Let us examine simple odd cycle $\{p \leftarrow not\ p.\}$. While in AS it spoils consistency, in RAS it simply leads to no conclusion. In fact, as an agent should be prepared to deal with uncertain, incomplete, and partially incorrect definitions from which it should try to do its best: thus, such indecisive rules are better be ignored. More involved is the case of a intermediate positive conclusions, like in $\{p \leftarrow a.\ \ a \leftarrow not\ p.\}$. From the assumption p nothing is derived, leading to an empty $\hat{\Gamma}_\Pi(\{p\})$. The set $\{a\}$ is instead an MCS: from the assumption *not* p in fact, a can be (though defeasibly) derived via the second rule. The first rule, which derives a contradiction from a reasonable assumption, is simply ignored. If assigning any significant interpretation to p and a (e.g., ill and healthy, unpolite and nice, young and old, etc.) the reasonable stance of such a choice should be evident.

Let us consider again the odd cycle $\{a \leftarrow \textit{not } b.\ \ b \leftarrow \textit{not } c.\ \ c \leftarrow \textit{not } a.\}$ and let us examine the resource-based answer set $\{a\}$ (a similar reasoning can be done for $\{b\}$ and $\{c\}$). In particular, let us reconstruct the rational reasoning that can lead to derive this belief set:

- to defeasibly conclude a, *not b* can be assumed;
- deriving a prevents from assuming *not a*, and thus from concluding c;
- assuming *not b* prevents from concluding b, which implies suspending the judgment on c, i.e., avoiding to assume *not c*. Actually, there is no argument that compels the agent to make this assumption: in fact, the agent in the present situation does not derive c not because it has reasons to believe that c does not hold, but because c cannot be concluded based upon the assumptions already made.

Thus, the resource-based answer set $\{a\}$ is a belief set where a is defeasibly believed to hold based upon the assumption that *not b* holds, while the judgment on c is suspended.

In the next section we will show that such an understanding of default negation finds a formal justification not only in linear logic (cf., Section 3), but also in autoepistemic logic. This by re-intepreting the autoepistemic logic characterization of logic programs proposed first by Gelfond [21] and Konolige [27] and then refined by Marek and Truszczynski [33].

Below we illustrate the application of the modified definitions on a sample standalone program Π, including two odd cycles (one unary and the other one ternary) with internal positive dependencies.

$$\begin{aligned}
&q \leftarrow a.\\
&a \leftarrow \textit{not } q.\\
&e \leftarrow \textit{not } f.\\
&f \leftarrow g.\\
&g \leftarrow \textit{not } h.\\
&h \leftarrow \textit{not } e.
\end{aligned}$$

Let us consider those sets I of atoms which are Π-based. Some of such Is must be excluded as $\hat{\Gamma}_\Pi(I) \not\subseteq I$. These are, in particular, all singletons, e.g. $\{q\}$ where $\hat{\Gamma}_\Pi(\{q\}) = \{e, f, g, h\}$ as the modified reduct cancels only the second rule. If we consider $\{a\}$, we have $\hat{\Gamma}_\Pi(\{a\}) = \{a, e, f, g, h\}$ where the modified reduct cancels no rules, but notice that the modified T_P cannot derive q anyway as it depends (though indirectly) upon its own negation. Some other sets of atoms do not result in MCS as the modified reduct cancels too many rules. One of such sets is, for instance, $\{q, f, h, e\}$ where $\hat{\Gamma}_\Pi(\{q, f, h, e\}) = \emptyset$ because the modified reduct cancels all rules with negation in their bodies. Another one is $\{f, h, e\}$ where $\hat{\Gamma}_\Pi(\{f, h, e\}) = \{a\}$. The resource-based answer sets are actually $\{a, e\} = \hat{\Gamma}_\Pi(\{a, e, h\})$, $\{a, h\} = \hat{\Gamma}_\Pi(\{a, f, h\})$, and $\{a, f, g\} = \hat{\Gamma}_\Pi(\{a, e, f, g\})$.

We can now define resource-based answer sets of a generic program Π.

Definition 5.6. Consider a non-empty ASP program Π, divided according to Proposition 2.2 into components $C_1, \ldots, C_n$, $n \geq 1$. A resource-based answer set S of Π is defined as $M_1 \cup \cdots \cup M_n$ where M_1

is a resource-based answer set of C_1, and each M_i, $1 < i \leq n$, is a resource-based answer set of standalone component C'_i, obtained by simplifying C_i w.r.t. $M_1 \cup \cdots \cup M_{i-1}$, where the simplification consists in: (i) deleting all rules in C_i which have *not* B in their body, $B \in M_1 \cup \cdots \cup M_{i-1}$; (ii) deleting (from the body of remaining rules) every literal *not* D where D does not occur in the head of rules of C_i and $D \notin M_1 \cup \cdots \cup M_{i-1}$, and also every atom D with $D \in M_1 \cup \cdots \cup M_{i-1}$.[7]

The definition brings evident analogies to the procedure for answer set computation specified in Proposition 2.3. This program decomposition is under some aspects reminiscent of the one adopted in [19]. However, in general, resource-based answer sets are not models in the classical sense: rather, they are Π-supported sets of atoms which are the wider subsets of some classical model that fulfills non-contradictory support. We can prove, in fact, the following result:

Theorem 5.2. A set of atoms I is a resource-based answer set of Π iff it is an MCS for Π.

Proof. The proof is by induction over the decomposition of Π into components $C_1, \ldots, C_n$.
Base step: a resource-based answer set M_1 of C_1 is an MCS of C_1 by Theorem 5.1, as C_1 is standalone.
Induction step: suppose that $M_1 \cup \cdots \cup M_i$, $i < n$, is an MCS for $C_1 \cup \cdots \cup C_i$, where each M_j is a resource-based answer set of standalone component C'_j, defined according to Definition 5.6. We prove that $M_1 \cup \cdots \cup M_{i+1}$, $i+1 \leq n$, is an MCS for $C_1 \cup \cdots \cup C_{i+1}$. In fact: (i) M_{i+1} is an MCS for C'_{i+1}; (ii) according to Definition 5.6, C'_{i+1} is obtained from a component of Π by canceling rules involving negations not supported in $M_1 \cup \cdots \cup M_i$, and by simplifying literals which are supported in $M_1 \cup \cdots \cup M_i$. Therefore, since all literals occurring in M_{i+1} are supported in C_{i+1} and all literals which have been simplified are supported in the MCSs of lower-level components, we can conclude that $M_1 \cup \cdots \cup M_{i+1}$ is indeed an MCS for $C_1 \cup \cdots \cup C_{i+1}$. □

Resource-based answer sets still form (like answer sets) an anti-chain w.r.t. set inclusion, and answer sets (if any) are among the resource-based answer sets. Differently from answer sets, a (possibly empty) resource-based answer set always exists.

It can be observed that complexity remains the same. In fact, computation of resource-based answer sets of standalone programs does not differ (except for some computationally irrelevant details) from standard answer set computation. Selecting the MCSs implies checking inclusion among the sets computed via the modified Γ. For a generic program, a preliminary decomposition according to Proposition 2.2 is needed, that can however be performed in polynomial time. Therefore:

Proposition 5.4. Given a program Π, deciding whether there exists a resource-based answer set of Π is NP-complete.

This will be indeed confirmed in the next section, where we will show how to compute resource-based answer sets in ASP.

We now explain by means of an example why the incremental construction of resource-based answer sets is necessary. Let Π be the following:

$$a \leftarrow not\ p.$$
$$p \leftarrow not\ p.$$

[7]Notice that, due to the simplification, C'_i is standalone.

Suppose to apply Definition 5.5 directly to the entire program. We have to consider three Π-based non-empty sets of atoms, namely $I_1 = \{a\}$, $I_2 = \{p\}$ and $I_3 = \{a, p\}$. We have $\hat{\Gamma}_\Pi(I_1) = M_1 = \{a, p\}$, $\hat{\Gamma}_\Pi(I_2) = \hat{\Gamma}_\Pi(I_3) = M_2 = \emptyset$, where M_2 should be the unique resource-based answer set. However while p, which depends upon its own negation, must indeed be deemed to be false, instead a, depending upon the negation of an atom which has been assumed to be false, should be (though defeasibly) concluded true. Therefore, according to Definition 5.6, we divide Π into a standalone bottom component C_1, consisting of the last rule, with $M_1 = \emptyset$ as the unique resource-based answer set, and a top component C_2 consisting of the first rule $a \leftarrow$ *not* p. After simplification, C_2' reduces to fact a, with unique resource-based answer set $M_2 = \{a\}$ which coincides with the unique resource-based answer set of the overall program, thus meeting the intuition.

The previously stated result about the relation with linear logic (Theorem 3.1) extends to the new semantics. The proof, reported in [14] in the context of full RASP programs, can in act be easily extended. Then we have:

Theorem 5.3. Let Π be an ASP program, and Σ_Π the corresponding Linear Logic RASP Theory, obtained according to Definitions 3.1 and 3.3. If $M = \{A_1, \dots, A_n\}$ is a resource-based answer set for Π, then $A_1 \otimes \cdots \otimes A_n$ is a maximal tensor conjunction provable from Σ_Π.

Observation

If we consider general cycles of the form $\{a_1 \leftarrow \textit{not } a_2.\;\; a_2 \leftarrow \textit{not } a_3.\;\; \cdots\;\; a_k \leftarrow \textit{not } a_1.\}$ involving $k > 3$ atoms, it is easy to see that each such cycle has $\lfloor \frac{k}{2} \rfloor$ resource-based answer sets, where we have $M_i = \{a_{i+d}$, with d even, $0 \leq d < k-1\}$. Then, unfortunately, odd cycles no longer model exclusive disjunction if $k > 3$.

However, there is a further semantic variation where cycles of any arity model exclusive disjunction, thus potentially providing the semantic basis for a generalized choice operators. Precisely, let us consider cycles of the above form. Instead of looking for MCSs, i.e., for maximal consistently supported set of atoms, for such program fragments we should look for *minimal consistently supported sets* of atoms (briefly, *mCS*s). The mCSs are exactly k, each consisting in the singleton $\{a_j\}$, for each $1 \leq j \leq k$. In particular, a_j is supported by assuming *not* a_{j+1}, and by not making assumptions on the other atoms. Notice however that this adaptation is rather "ad hoc" in view of a choice operator, since it does not work in case of intermediate positive dependencies or of further literals occurring in rule bodies. This is however a potential alternative to adopting extremal programs [5]: to model exclusive disjunction, they define each involved atom in term of the negation of all the others.

6. Autoepistemic Logic characterization of Resource-based answer set semantics

In [35], several characterizations are proposed of (disjunctive) answer set semantics (also with classical negation) in terms of autoepistemic logic [33]. We will here consider the characterization provided in terms of *Reflexive autoepistemic logic* [44], which is closely related to the modal logic **SW5** (cf. [25, 34] for a detailed discussion of various modal logics, including **SW5**).

We remind the reader that (apart from the distribution axiom $\Box(\varphi \supset \psi) \supset (\Box\varphi \supset \Box\psi)$ characterizing normal modal logics) **SW5** is axiomatized through the following axioms, where in the autoepistemic setting the necessity operator $\Box$ is replaced by the operator L. The modal formula $L\varphi$ can be read as "φ is believed" or also "we assume φ" or "we assume that φ is true":

$$\begin{array}{ll} L\varphi \supset \varphi & (Ax1) \\ L\varphi \supset LL\varphi & (Ax2) \\ \neg L \neg L\varphi \supset (\varphi \supset L\varphi) & (Ax3) \end{array}$$

According to formula (16) in [35], an answer set programming rule[8] ρ of the form $A \leftarrow A_1, \dots, A_n, not\ B_1, \dots, not\ B_m$ can seen as standing for its modal transposition, or "modal image":

$$LA_1 \wedge \dots \wedge LA_n \wedge L\neg LB_1 \wedge \dots \wedge L\neg LB_m \supset LA \quad (MT93\,16)$$

Then, from the modal image of each rule one obtains the modal image of the entire given program Π.

To characterize resource-based answer sets, we propose a modified formulation. Precisely, we introduce for each atom A occurring in the head of some rule ρ the fresh auxiliary atom $\dot{A}$, that can be read as "A can be proved" or, equivalently, "one intends/is enabled to prove A". We propose the following transposition of an answer set program into a modal theory:

- each fact (unit rule) A of Π is simply transposed into its modal image

$$LA \qquad (Ae0)$$

- each rule ρ is transposed into the following couple of modal rules that form its modal image:

$$\begin{array}{ll} LA_1 \wedge \dots \wedge LA_n \wedge L\neg LB_1 \wedge \dots \wedge L\neg LB_m \supset L\dot{A} & (Ae1) \\ L\dot{A} \wedge \neg L \neg LA \supset LA & (Ae2) \end{array}$$

$(Ae1)$ modifies $(MT93\,16)$ in the sense that, based on the same premises, one concludes $L\dot{A}$. I.e., one concludes to believe to be enabled to prove A. $(Ae2)$ states that LA is derived only if $\dot{A}$ is believed, and one does not believe not to believe A: in fact, it would be quite singular to intend to prove something that one believes not to believe, which is the same (in this context) as believing/assuming it to be false. $(Ae2)$ follows the spirit of axiom $(Ax3)$ above, which is the axiom that characterizes **SW5**: $(Ax3)$ in fact states that in order to believe φ not only one has to prove it, but as a premise one must not believe not to believe φ. $(Ae2)$ states actually the same, though considering $L\dot{A}$ instead of directly considering LA.

Reflexive autoepistemic logic assigns to a set of modal formulas I (that in our case will be the modal image of a given answer set program Π under the transposition dictated by $(Ae0)$, $(Ae1)$, and $(Ae2)$) theories called *reflexive expansions*. Precisely, such a theory is defined as follows (where Cn indicates logical consequence as usually intended).

$$T = Cn(I \cup (\varphi \equiv L\varphi : \varphi \in T) \cup (\neg L\varphi : \varphi \notin T)) \qquad (Rae)$$

As proved in [43], a theory T is a reflexive expansion of I if and only if T is an **SW5** expansion of I. Therefore, a theory is intended here as a set of formulas that one can construct on the basis of I given the axioms of **SW5**.

[8]In [35], such a rule is assumed to admit classical negation and disjunction in the head, where in the present context we do not consider these improvements.

A theory T in a modal language is *stable* if it is closed under propositional provability and it is also closed under positive and negative introspection, i.e., $L\varphi \in T$ whenever $\varphi \in T$ (positive introspection) and $\neg L\varphi \in T$ whenever $\varphi \notin T$ (negative introspection). Reflexive (and thus **SW5**) expansions are stable by construction.

Given a reflexive expansion T of the modal image I of program Π, let $core(T)$ be the set of atoms $\{A : A \text{ occurs in } \Pi \text{ and } A \in T\}$. We can adapt to our setting Theorem 3.5 of [35].

Below, let I be the image of a given program Π under $(Ae0)$, $(Ae1)$, and $(Ae2)$, and let T be a reflexive expansion for I. Let us preliminarily state the following facts.

Fact 1. By definition, $core(T)$ does not include any of the auxiliary atoms $\dot{A}$.

Fact 2. I is a set of formulas composed of modal literals only. Thus, atoms occurring in $core(T)$ are necessarily obtained via the application of axiom $(Ax1)$ of **SW5** to formulas of the form LA where A is an atom.

Fact 3. Formulas of the form LA where A is an atom can in turn be obtained in two possible ways: (i) directly as the modal image of a fact of Π via $(Ax0)$, or (ii) as the result of the subsequent application of the two modal formulas which constitute the modal image via $(Ax1)$ and $(Ax2)$ of some rule ρ of Π with head A, by deriving first $\dot{A}$ and then LA.

Lemma 6.1. For every reflexive expansion T for the image I of Π under $(Ae0)$, $(Ae1)$, and $(Ae2)$, we have that $S = core(T)$ is Π-based.

Proof. By virtue of the above-stated facts we can conclude that in both cases (i) and (ii) listed in Fact 3, each atom $A \in S$ is the head of some rule ρ in Π (possibly a fact). $\square$

Lemma 6.2. For every reflexive expansion T for the image I of Π under $(Ae0)$, $(Ae1)$, and $(Ae2)$ we have that $S = core(T)$ is a consistently supported set of atoms for Π.

Proof. By Lemma 6.1, S is Π-based. By Fact 3, each $A \in S$ corresponds either to a fact or to the head of a rule ρ in Π transposed according to $(Ae1)$. In the former case, A is consistently supported in Π in the obvious way. In the latter case, for $L\dot{A}$ to be derived the conditions of the modal image of ρ must have been satisfied in T. This implies that, when transposed into plain literals via $(Ax1)$, they satisfy ρ as well. Thus, both the positive conditions and the conclusion A belong to S, whereas the negative conditions do not. Then, A is supported in Π and so are (by the analogous reasoning) the positive conditions of ρ. Therefore, we can conclude that each $A \in S$ is consistently supported in Π, and the result follows from Definition 2.7. $\square$

Proposition 6.1. For every reflexive expansion T for the image I of Π under $(Ae0)$, $(Ae1)$, and $(Ae2)$ we have that $S = core(T)$ is an MCS Π.

Proof. By Lemma 6.2, S is a consistently supported set of atoms for Π. It is in particular an MCS because the only way of preventing atom $A \in M$ from being in S is that $\neg L \neg LA$ does not hold in T, which implies that A is not supported. $\square$

From the above Lemmas and Proposition we obtain the following:

Corollary 6.1. For every reflexive expansion T for the image I of Π under $(Ae0)$, $(Ae1)$, and $(Ae2)$ $S = core(T)$ is a resource-based answer set for Π.

We are finally ready to prove the main result.

Theorem 6.1. Let S be a set of atoms. Then, S is a resource-based answer set of program Π if and only if $S = core(T)$, where T is a reflexive expansion for the image I of Π under $(Ae0)$, $(Ae1)$, and $(Ae2)$.

Proof. The 'only if' part is proved via Corollary 6.1. For the 'if' part, consider a resource-based answer set S of the given program Π, and consider the image I of Π. We have to prove that there exists a reflexive expansion T for I such that $S = core(T)$. Consider atom $A \in S$. Either A is a fact in Π, then by axiom $(Ae0)$ LA occurs in I, so A occurs in any reflexive expansion of I (by axiom $(Ax1)$ of **SW5**, or equivalently by positive introspection). Since S is consistently supported, then there exists rule ρ in Π with conclusion A and premises consistently supported in S. The modal image of such a rule is included in I. By the same kind of support each positive premise B will correspond to literal LB in any reflexive expansion of I, and each negative premise *not* C (where C does not belong to S) will correspond by negative introspection to literal $L\neg C$ in some reflexive expansion of I. Thus, A will be included in this reflexive expansion via the application of $(Ae1)$, $(Ae2)$, and $(Ax1)$. Since by Lemma 6.1 S is an MCS for Π, then no atom $D \notin S$ is allowed to belong to such a reflexive expansion. □

The autoepistemic characterization of resource-based answer set semantics leads to a method for computing resource-based answer sets of a given program Π by computing the answer sets of a modified program Π'.

In particular, following the principles underlying the definition of the autoepistemic image, each rule ρ of the form $A \leftarrow A_1,\dots,A_n, not\ B_1,\dots, not\ B_m$ of Π is represented in Π' by the following set of rules (except for facts, that are simply copied from Π to Π'). In these rules, A', A'' and noA'' are fresh atoms, where A' stands for $L\dot{A}$, and A'' stands for $\neg L\neg LA$. For simplicity, A and LA as well as *not* A and $L\neg LA$ are assumed to coincide (according however to the definition of reflexive expansion, where upward and downward reflection on the L operator are freely allowed).

$$\begin{aligned}
&A' \leftarrow A_1,\dots,A_n, not\ B_1,\dots, not\ B_m.\\
&A \leftarrow A', A''.\\
&\leftarrow A'', not\ A.\\
&A'' \leftarrow not\ noA''.\\
&noA'' \leftarrow not\ A''.
\end{aligned}$$

The first two rules correspond to axioms $(Ae1)$ and $(Ae2)$. The constraint models negative introspection, stating that one cannot assume both $\neg L\neg LA$ and $\neg A$, again in correspondence to the definition of reflexive autoepistemic expansions, which are required to be consistent. The even cycle has the role of generating the assumption $\neg L\neg LA$.

It is easy to see (the proof is omitted for space reasons) that, among the answer sets of Π, those which maximize the assumptions A'' coincide, after removing the fresh atoms, with the resource-based answer sets of Π. Thus, the resource-based answer sets of a given program can be computed by using an answer set solver on a new program whose size grows linearly w.r.t. the given one.

7. Practical Applications of Resource-based answer set semantics

In resource-based answer set semantics there are no inconsistent programs. Nevertheless, the new semantics can be useful in knowledge representation not just to fix inconsistencies: rather, it depicts a more general scenario in many practical cases. However, notice that constraints can no longer be modeled (as done under the answer set semantics) in terms of odd cycles. Therefore, they have to be modeled explicitly. We discuss in the next section how to formally introduce constraints. In this section, we take constraints for granted in order to discuss some examples of application of the new semantics.

Consider the three coloring of a graph. This is an NP-complete problem, and thus it constitutes a good testbed to understand how the new semantics copes with such cases. Given representation of nodes and arcs, two solutions are commonly found. One solution, as mentioned, is based upon extremal programs [5], which are composed of many nested cycles and are thus rather involved. Another solution is based upon a *cardinality constraint* [38, 45], which is translated into a substantial ASP subprogram. Under resource-based answer set semantics, the "look" of the RAS program can be the usual one, i.e.,

$$\begin{array}{l} color(X,red)|color(X,blue)|color(X,green) \leftarrow node(X). \\ \leftarrow edge(X,Y), possible\_color(C), color(X,C), color(Y,C). \end{array}$$

where however disjunction in our framework simply stands for the odd loop:

$$\begin{array}{l} color(X,red) \leftarrow node(X), not\ color(X,blue). \\ color(X,blue) \leftarrow node(X), not\ color(X,green). \\ color(X,green) \leftarrow node(X), not\ color(X,red). \end{array}$$

Therefore, resource-based answer set semantics provides an extra-freedom in generating the search space which can be usefully exploited in practical programming.

Consider now the following variation (inspired by examples introduced in [41, 42]) of the program discussed in Section 4:

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

This program has answer set $M_1 = \{forgot\_renew, mountain\}$, 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. So, from a description that appears to be reasonable, the only possible scenario is that, having forgotten to renew the passport, one must choose (for inscrutable reasons) to go to mountain.

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 $M_2 = \{passport\_ok, mountain\}$, $M_3 = \{passport\_ok, beach\}$, and $M_4 = \{passport\_ok, travel\}$. 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 again forced.

A more satisfactory formalization of the above example is in fact the following.

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

In this case we obtain (by the methods discussed in the next section) M_1, M_2, M_3, and M_4 but also $M_5 = \{\mathit{forgot\_renew}, \mathit{beach}\}$.

8. Modeling Constraints

To the extent of our discussion, each constraint $\leftarrow L_1, \dots, L_k$, $k > 0$, where each L_i is a literal, can be rephrased as simple constraint $\leftarrow H$, where H is a fresh atom, plus rule $H \leftarrow L_1, \dots, L_k$.[9] Thus, an overall program $\Pi_{\mathscr{O}}$ can be seen as composed of an answer set program Π plus a set $\{\mathscr{C}_1, \dots, \mathscr{C}_v\}$ 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) and that atoms of Π may occur in $\Pi_{\mathscr{C}}$ only in the body of rules, i.e., according to previous definitions, $\Pi_{\mathscr{C}}$ is on top of Π).

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

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

plus the following $\Pi_{\mathscr{C}}$: $\{\mathit{unhealthy} \leftarrow \mathit{beach}, \mathit{hyperthyroidism}.\}$ plus the constraint $\leftarrow \mathit{unhealthy}$. The resulting theory will have resource-based answer sets $\{\mathit{mountain}$, $\mathit{hyperthyroidism}\}$, and $\{\mathit{travel}$, $\mathit{hyperthyroidism}\}$, while $\{\mathit{beach}$, $\mathit{hyperthyroidism}$, $\mathit{unhealthy}\}$ is excluded by the constraint.

Below we proceed to the formal definitions concerning constraints.

Definition 8.1. An Answer Set Theory $\mathscr{T}$ is a couple $\langle \Pi_{\mathscr{O}}, \mathit{Constr} \rangle$, with $\Pi_{\mathscr{O}} = \Pi \cup \Pi_{\mathscr{C}}$, where $\Pi_{\mathscr{C}}$ is on top of Π, and where *Constr* is a set $\{\mathscr{C}_1, \dots, \mathscr{C}_v\}$, $v \geq 0$, of constraints.

Definition 8.2. Given an Answer Set Theory $\mathscr{T} = \langle \Pi_{\mathscr{O}}, \mathit{Constr} \rangle$, a resource-based answer set M for Π fulfills the constraints in *Constr* if and only if 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 8.3. A *resource-based answer set M* of Answer Set Theory $\mathscr{T} = \langle \Pi_{\mathscr{O}}, \mathit{Constr} \rangle$ is a resource-based answer set for Π that fulfills all constraints in *Constr*.

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

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., [17]) 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 8.1. Given an Answer Set Theory $\mathscr{T}$, deciding about the existence of a resource-based answer set is an 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 $\Pi' = \Pi \cup \Pi_{\mathscr{C}}$. In this alternative view:

Definition 8.4. Let Π' be a program, and $\{\mathscr{C}_1, \ldots, \mathscr{C}_k\}$ be a set of constraints, each $\mathscr{C}_i$ in the form $\leftarrow H_i$, $0 \leq i \leq k$ for some k. A resource-based answer set M for Π' is *admissible* if it fulfills all constraints, i.e., if for all $i < k$, $H_i \notin M$. M is *admissible* w.r.t. single constraint $\mathscr{C}_j$ if $H_j \notin M$.

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 [37]. In fact, the same "generate" part (Π) can be customized by adding on top, as independent layers, different "specialization" parts (represented as different $\Pi_{\mathscr{C}}$'s) and "test" parts (the constraints). Moreover, constraints might be generalized with respect to the simple form proposed above, for instance drawing inspiration from the discussion in [11, 13, 10], or also following the approach of *Answer Set Optimization* (cf. [3] 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 $\mathscr{C}^L = {E_1}^\perp \parr \cdots \parr {E_n}^\perp$ where $\parr$ is the multiplicative disjunction, and $^\perp$ is linear logic negation, $A^\perp$ meaning "there is no proof for A".[10]

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 8.5. Given a resource-based answer set $M = \{A_1, \ldots, A_n\}$ for $\Pi_{\mathscr{O}}$, if M is a resource-answer set for answer set theory $\mathscr{T} = \langle \Pi_{\mathscr{O}}, \mathit{Constr} \rangle$ where $\mathit{Constr} = \{\mathscr{C}_1, \ldots, \mathscr{C}_v\}$ then tensor conjunction $A_1 \otimes \cdots \otimes A_n \otimes \mathscr{C}_1^L \otimes \cdots \otimes \mathscr{C}_v^L$ 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 correspondence between the logic programming and the linear logic formulation, there are no essential differences w.r.t. Theorem 5.3.

9. Concluding Remarks

In this paper, we have proposed resource-based answer set semantics as an extension of the answer set semantics, where there are no inconsistent programs and more freedom is allowed in defining a search

[10]Notice that, linear logic negation as such was used in [4] to model negation-as-failure in Prolog.

space through odd and even cycles. However, in the new approach constraints have to be defined in a separate "module" to be added to the given answer set program.

We have shown that resource-based answer sets can be computed by any answer set solver on a modified version of the given program. For a real implementation, which is an important future issue for this research, answer set solvers based on SAT appear to be good candidates for extension to the new setting. In fact, 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.

The absence of inconsistency has allowed us to define (the foundations of) a top-down query answering procedure for answer set programs under the resource-based answer set semantics, where we also consider the issue of constraint checking. More details on this approach can be found in [15].

References

[1] Apt, K. R., Bol, R. N.: Logic Programming and Negation: A Survey, *Journal of Logic Programming*, **19/20**, 1994, 9–71.

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

[3] Brewka, G., Niemelä, I., Truszczyński, M.: Answer Set Optimization, *IJCAI-03, Proc. of the Eighteenth Intl. Joint Conference on Artificial Intelligence* (G. Gottlob, T. Walsh, Eds.), Morgan Kaufmann, 2003.

[4] Cerrito, S.: A Linear Axiomatization of Negation as Failure, *Journal of Logic Programming*, **12**(1&2), 1992, 1–24.

[5] Cholewinski, P., Truszczyński, M.: Extremal Problems in Logic Programming and Stable Model Computation, *Journal of Logic Programming*, **38**(2), 1999, 219–242.

[6] Costantini, S.: Contributions to the Stable Model Semantics of Logic Programs with Negation, *Theoretical Computer Science*, **149**(2), 1995, 231–255.

[7] Costantini, S.: On the existence of stable models of non-stratified logic programs, *Theory and Practice of Logic Programming*, **6**(1-2), 2006.

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

[9] Costantini, S., Formisano, A.: Answer Set Programming with Resources, *Journal of Logic and Computation*, **20**(2), 2010, 533–571.

[10] Costantini, S., Formisano, A.: Augmenting Weight Constraints with Complex Preferences, *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, AAAI Press, 2011, Tech. report SS-11-06.

[11] Costantini, S., Formisano, A.: Weight Constraints with Preferences in ASP, *Proc. of Intl. Conf. on Logic Programming and Nonmonotonic Reasoning LPNMR'11*, 6645, Springer-Verlag, 2011.

[12] Costantini, S., Formisano, A.: Negation as a Resource: A Novel View on Answer Set Semantics, *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013* (P. Cabalar, T. C. Son, Eds.), 8148, Springer, 2013.

[13] Costantini, S., Formisano, A.: Nested Weight Constraints in ASP, *Fundamenta Informaticae*, **124**(4), 2013, 449–464.

[14] Costantini, S., Formisano, A.: RASP and ASP as a Fragment of Linear Logic, *Journal of Applied Non-Classical Logics (JANCL)*, **23**(1-2), 2013, 49–74, Special Issue on Equilibrium Logic and Answer Set Programming in Honor of David Pearce.

[15] Costantini, S., Formisano, A.: Query Answering in Resource-Based Answer Set Semantics, *Proceedings of the 7th Workshop on Answer Set Programming and Other Computing Paradigms ASPOCP 2014* (D. Inclezan, M. Maratea, Eds.), 2014.

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

[17] Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming, *ACM Computing Surveys*, **33**(3), 2001, 374–425.

[18] Dix, J.: A Classification Theory of Semantics of Normal Logic Programs I-II., *Fundamenta Informaticae*, **22**(3), 1995, 227–255 and 257–288.

[19] Gebser, M., Gharib, M., Mercer, R. E., Schaub, T.: Monotonic Answer Set Programming, *Journal of Logic Computation*, **19**(4), 2009, 539–564.

[20] Gelfond, M.: On Stratified Autoepistemic Theories, *AAAI, Proceedings of the 6th National Conference on Artificial Intelligence* (K. D. Forbus, H. E. Shrobe, Eds.), 1987.

[21] Gelfond, M.: Answer Sets, in: *Handbook of Knowledge Representation. Chapter 7*, Elsevier, 2007.

[22] Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming, *Proc. of the 5th Intl. Conf. and Symposium on Logic Programming* (R. Kowalski, K. Bowen, Eds.), MIT Press, 1988.

[23] Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases, *New Generation Computing*, **9**, 1991, 365–385.

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

[25] Hughes, G., Cresswell, M.: *A companion to modal logic*, Methuen and Co. Ltd., London, 1984.

[26] Kanovich, M. I.: The Complexity of Horn Fragments of Linear Logic, *Ann. Pure Appl. Logic*, **69**(2-3), 1994, 195–241.

[27] Konolige, K.: On the Relation Between Default and Autoepistemic Logic, *Artif. Intell.*, **35**(3), 1988, 343–382.

[28] Leone, N.: Logic Programming and Nonmonotonic Reasoning: From Theory to Systems and Applications, *Logic Programming and Nonmonotonic Reasoning, 9th Intl. Conference, LPNMR 2007* (C. Baral, G. Brewka, J. Schlipf, Eds.), 2007.

[29] Lifschitz, V.: Twelve Definitions of a Stable Model, *Proc. of the 24th Intl. Conference on Logic Programming* (M. Garcia de la Banda, E. Pontelli, Eds.), 5366, Springer, 2008.

[30] Lifschitz, V., Turner, H.: Splitting a Logic Program, *Proc. of ICLP'94, Intl. Conference on Logic Programming*, 1994.

[31] Lin, F., Zhao, X.: On Odd and Even Cycles in Normal Logic Programs, *Proceedings of the Nineteenth National Conference on Artificial Intelligence AAAI 2004* (D. L. McGuinness, G. Ferguson, Eds.), AAAI Press / The MIT Press, 2004.

[32] Lloyd, J. W.: *Foundations of Logic Programming*, Springer-Verlag, 1987.

[33] Marek, V. W., Truszczyński, M.: Autoepistemic Logic, *Journal of the ACM*, **38**(3), 1991, 587–618.

[34] Marek, V. W., Truszczyński, M.: *Nonmonotonic logic, context-dependent reasoning*, Artificial intelligence, Springer, 1993.

[35] Marek, V. W., Truszczyński, M.: Reflective Autoepistemic Logic and Logic Programming, *LPNMR, Logic Programming and Non-monotonic Reasoning, Proceedings of the Second International Workshop* (A. Nerode, L. M. Pereira, Eds.), The MIT Press, 1993.

[36] Marek, V. W., Truszczynski, M.: Stable models and an alternative logic programming paradigm, in: *The Logic Programming Paradigm: A 25-Year Perspective* (K. R. Apt, D. S. Warren, M. Truszczynski, Eds.), 1st edition, Springer, 1999, ISBN 3540654631.

[37] McCarthy, J.: Elaboration tolerance, *Proc. of Commonsense'98*, 1998, Available at http://www-formal.stanford.edu/jmc/ elaboration.html.

[38] Niemelä, I., Simons, P., Soininen, T.: Stable model semantics of weight constraint rules, *Proc. of the 5th Intl. Conference on Logic Programming and Nonmonotonic Reasoning LPNMR'99*, number 1730 in LNAI, Springer-Verlag, 1999.

[39] Osorio, M., López, A.: Expressing the Stable Semantics in Terms of the Pstable Semantics, *Proc. of the LoLaCOM06 Workshop*, 220, CEUR-WS.org, 2006.

[40] Osorio, M., Pérez, J. A. N., Ramírez, J. R. A., Macías, V. B.: Logics with Common Weak Completions, *Journal of Logic Computation*, **16**(6), 2006, 867–890.

[41] Pereira, L. M., Pinto, A. M.: Revised Stable Models - A Semantics for Logic Programs, *Progress in Artificial Intelligence, Proc. of EPIA 2005* (C. Bento, A. Cardoso, G. Dias, Eds.), 3808, Springer, 2005.

[42] Pereira, L. M., Pinto, A. M.: Tight Semantics for Logic Programs, *Tech. Comm. of the 26th Intl. Conference on Logic Programming, ICLP 2010* (M. V. Hermenegildo, T. Schaub, Eds.), 7, 2010.

[43] Schwarz, G.: Autoepistemic Logic of Knowledge, *LPNMR, Logic Programming and Non-monotonic Reasoning, Proceedings of the First International Workshop* (A. Nerode, V. Subrahmanian, Eds.), The MIT Press, 1991.

[44] Schwarz, G.: Reflexive Autoepistemic Logic, *Fundamenta Informaticae*, **17**(1-2), 1992, 157–173.

[45] Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics, *Artificial Intelligence*, **138**(1-2), 2002, 181–234.

[46] Truszczyński, M.: Logic Programming for Knowledge Representation, *Logic Programming, 23rd Intl. Conference, ICLP 2007* (V. Dahl, I. Niemelä, Eds.), 2007.

[47] Van Gelder, A., Ross, K. A., Schlipf, J. S.: The Well-Founded Semantics for General Logic Programs, *Journal of the ACM*, **38**(3), 1991, 620–650.

[48] Web references of 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`.