Draft

RASP and ASP as a Fragment of Linear Logic

S. Costantini\textsuperscript{a}∗ and A. Formisano\textsuperscript{b}

\textsuperscript{a}Universit\`a di L'Aquila, Italy; \textsuperscript{b}Universit\`a di Perugia, Italy

(\textcopyright{}1.0 released January 2013)

RASP is a recent extension to Answer Set Programming (ASP) that permits declarative specification and reasoning on consumption and production of resources. ASP can be seen as a particular case of RASP. In this paper, we study the relationship between linear logic and RASP problem specification. We prove that RASP programs can be translated into (a fragment of) linear logic, and vice versa. In doing that, we introduce a linear logic representation of default negation as understood in Answer Set Programming. We are also able to establish a link between linear logic and HT-logic.

\textbf{Keywords:} Answer Set Programming; Resource Management; Linear Logic

Introduction

RASP (Costantini & Formisano, 2010, 2009; Costantini, Formisano, & Petturiti, 2010) is an extension of the Answer Set Programming (ASP) framework obtained by explicitly introducing the notion of resource. It supports both formalization and quantitative reasoning on consumption and production of amounts of resources.

In knowledge representation and reasoning, forms of quantitative reasoning are possible in Linear Logics (Girard, 1987) and Description Logics (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003). In this paper, we focus on the relationship between linear logic and logic programming. In logic programming in fact, a number of Prolog-like logic programming languages based on linear logic have been proposed (cf. references and discussion in (Costantini & Formisano, 2010)). For instance, (Cerrito, 1992) proposes a linear logic axiomatization of Prolog with negation as failure which takes procedural behavior of Prolog into account, while (Kanovich, 1994) studies the complexity of various Horn fragments of linear logic. (Miller, 1985) widely illustrates (providing a useful discussion and many references) how linear logic has been used and can be used to design new logic programming languages, also treating implementation issues and possible applications, while (Hodas & Miller, 1994) presents a specific (fully implemented) approach to linear logic programming. The approach of (Pym & Harland, 1994; Harland, Pym, & Winikoff, 1995) generalizes Prolog, by introducing rules based on linear logic that must (by default) be used exactly once, so as to model the consumption of resources.

In ASP, a form of resource treatment is described in (Soininen, Niemelä, Tikhonen, & Sulonen, 2001; Soininen & Niemelä, 1999) to model product configuration problems. This framework is based on Weight Constraint Rules, which is a well-known construct

∗Corresponding author. Email: stefania.costantini@univaq.it
encompassing default negation and disjunctive choices (Niemelä, Simons, & Soininen, 1999). Weight Constraint Rules have a wide applicability in many applications and are able to express costs and limits on costs, where however they do not express directly resource consumption/production. Resources are rendered, in the action description language \textit{CARD} (Chintabathina, Gelfond, & Watson, 2007), through multi-valued fluents and the use of resources is implicitly modeled by the changes in fluents’ values caused by actions’ executions. The approach emphasizes the use of resources in planning problems and its semantics is provided in terms of transition systems (in the spirit of (Gelfond & Lifschitz, 1998)). Compared to these approaches RASP, as discussed at some length in (Costantini & Formisano, 2010), has its own distinguished features, as it allows for explicit representation of resource amounts, and in RASP one can model via ASP-like rules processes that produce resources via consumption of available ones.

Below we propose a comparison between RASP and linear logic. We will prove in particular that RASP corresponds to a fragment of linear logic. This implies that a RASP inference engine (such as Raspberry, that we have defined and prototypically implemented (Costantini et al., 2010)) can be used for reasoning in this fragment. Since ASP is a proper subset of RASP (an ASP program is a particular case of a RASP program) our result also implies that ASP corresponds to a fragment of linear logic. In fact, we introduce a RASP and linear-logic modeling of default negation as understood under the answer set semantics. Through RASP, we also establish a connection between HT-logic (the propositional logic of here-and-there, cf. (Lifschitz, Pearce, & Valverde, 2001) and the references therein), that provides a characterization of answer sets and of strong equivalence of ground ASP programs, and linear logic. Ultimately then, this work can be seen as proposing (with some limitation, discussed below) yet another characterization of answer sets, in addition to those reported in (Lifschitz, 2008). In particular we propose a novel, resource-based interpretation of ASP default negation.

The paper is organized as follows. In Sections 2- 4 we provide the necessary background on ASP, RASP and linear logic. We assume the reader to familiar with standard concepts regarding Logic Programming (cf. e.g., (Lloyd, 1987; Apt & Bol, 1994)). In Section 5 we discuss how RASP syntax and semantics can be rendered in a fragment of linear logic, considering first positive RASP program and then general RASP programs. In Section 6, we shortly discuss how previous results can also be seen in connection to HT-logic, and in Section 7 we conclude.

1. Background on Answer Set Semantics

Under the answer set semantics (originally named 'stable model semantics', cf. (Gelfond & Lifschitz, 1988, 1991)), a program \( \Pi \) is a collection of rules of the form

\[ H ← L_1, \ldots, L_m. \]

where \( H \) is an atom, \( m ≥ 0 \) and each literal \( L_i \) is either an atom \( A_i \) or its default negation \( \text{not} A_i \). Connective \( ← \) is often indicated with \( :− \), which is the symbol adopted in practical programming systems. 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. (Gelfond & Lifschitz, 1991)). The left-hand side and the right-hand side of rules are called head and body, respectively. A rule with empty body is called a \textit{fact}. A rule with empty head is a \textit{constraint}, where a constraint of the form

\[ ← L_1, \ldots, L_n. \]
states that literals $L_1, \ldots, L_n$ cannot be simultaneously true.

In the rest of the paper, whenever it is clear from the context, by “a (logic) program $\Pi$” we mean an answer set program (ASP program) $\Pi$, and we will implicitly refer to the “ground” version of $\Pi$. The ground version of $\Pi$ is obtained by replacing in all possible ways the variables occurring in $\Pi$ with the constants occurring in $\Pi$ itself, and is thus composed of ground atoms, i.e., atoms which contain no variables.

The answer sets semantics is a view of logic programs as sets of inference rules (more precisely, default inference rules). Alternatively, one can see a program as a set of constraints on the solution of a problem, where each answer set represents a solution compatible with the constraints expressed by the program. Consider the simple program

\[
\{q ← \text{not} p. \ p ← \text{not} q.\}
\]

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 (or stability) means checking for the existence of answer sets. Answer sets are minimal supported models of the first-order-logic theory corresponding to given program (where $←$ is interpreted as implication and not as classical-logic negation), and form an anti-chain.

Given set $I$ of atoms, $I$ is an answer set of $\Pi$ if it coincides with the Least Herbrand Model (cf. (Lloyd, 1987)) of the GL-reduct\(^1\) of $\Pi$ with respect to $I$.

**Definition 1.** Given a set of atoms $I$ and an ASP program $\Pi$, the GL-reduct $\Pi^I$ is the definite program obtained $\Pi$ from as follows:

1. remove each rule $r$ of $\Pi$ such that for any $A ∈ I$, the literal not$A$ occurs in the body of $r$; and then
2. remove each literal not$A$ from the remaining rules.

For example, program \(\{a ← \text{not} b. \ b ← \text{not} c. \ c ← \text{not} a.\}\) has no answer set. The reason is that in every minimal model (in the classical sense) of this program there is a true atom that depends (in the program) on the negation of another true atom. This 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. Program \(\{p ← \text{not} p.\}\) has no answer sets either, as it contradictory. It is usefully exploited because constraints, of the form shown above, can be simulated by means of plain rules of the form

\[
p ← \text{not} p, L_1, \ldots, L_n.
\]

where $p$ is a fresh atom.

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 (Web References of some ASP solvers, 2013), 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. The interested reader can refer, for instance, to (Dantsin, Eiter, Gottlob, & Voronkov, 2001). The reader can also see (Baral, 2003) and (Dovier, Formisano, & Pontelli, 2009), among others, for a presentation of ASP as a tool for declarative problem-solving.

---

\(^1\)The GL-reduct is named after Gelfond and Lifschitz, the scholars who introduced this semantics.
2. RASP in a Nutshell

In this section, we report, borrowing from our previous work (Costantini & Formisano, 2010, 2009; Costantini et al., 2010), RASP syntax and semantics, also providing some examples.

In RASP, resources are modeled by amount-atoms of the form $q:a$, where $q$ represents a specific type of resource and $a$ denotes the corresponding amount. Resources can be produced or consumed (or declared available from the beginning).

The processes that transform some amounts of resources into other resources are specified by $r$-rules, for instance, as in this simple example:

$$
\text{computer} : 1 \leftarrow \text{cpu} : 1, \text{hd} : 2, \text{motherboard} : 1, \text{ram}_{\text{module}} : 2.
$$

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

In their most general form, $r$-rules may involve plain ASP literals together with amount-atoms. Semantics for RASP programs is given by combining answer set semantics with a notion of allocation. While answer sets are used to deal with usual ASP literals, allocations are exploited to take care of amounts and resources. Intuitively, an allocation assigns to each amount-atom a (possibly null) quantity. Quantities are interpreted in an auxiliary algebraic structure that supports comparisons and operations on amounts. Thus, one has to choose a collection $Q$ of quantities, the operations to combine and compare quantities, and a mapping that associates quantities to amount-symbols. Admissible allocations are those satisfying, for all resources, the requirement that one can consume only what has been produced. Clearly, alternative allocations might be possible. They correspond to different ways of using the same resources. A simple natural choice for $Q$ is the set of integer numbers. In all the examples proposed in the rest of the paper, we implicitly make this choice.

Syntax and semantics of RASP were introduced in (Costantini & Formisano, 2010). Various extensions, presented in (Costantini & Formisano, 2010) and in (Costantini & Formisano, 2009; Costantini et al., 2010) (which discuss preferences and complex preferences in RASP) are not considered here. An implementation of RASP is discussed in (Costantini et al., 2010) and is available at http://www.dmi.unipg.it/formis/raspberry/.

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

Expressions such as $p(X):V$ where $V, X$ are variable symbols are allowed, as resources amounts can be either directly specified as constants or obtained via some kind of computation. Notice that the set of variables is not partitioned, as the same variable may occur both as a program term and as an amount-term. Ground amount- or program-
atoms contain no variables. As usual, a program-literal $L$ is a program-atom $A$ or the negation $\text{not } A$ of a program-atom (intended as negation-as-failure).\(^1\) A resource-literal is either a program-literal or an amount-atom.

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

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

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

The ground version (or “grounding”) of an r-program $P$ is the set of all ground instances of rules of $P$, obtained through ground substitutions over the constants occurring in $P$. As customary, in what follows we will implicitly refer to the ground version of $P$.

Intuitively, an interpretation of $P$ is an answer set whenever it satisfies all the program rules in $P$ and all the fired r-rules (in the usual way) as concerns their program-literals, and all consumed amounts either were available from resource-facts or have been produced by rule firings. In what follows, whenever it should be clear from the context we will often talk about a “resource” to refer to the available amount of that resource or to part of it.

Let us intuitively illustrate, by means of a simple example, the intended RASP-semantics and the notions of RASP program, allocation, and answer set. In the following section we will provide the formal definitions.

**Example 1.** Consider the following RASP program $P$:

- $r:3 \leftarrow g:1, s:1$.
- $p:2 \leftarrow g:1$.
- $g:2 \leftarrow q:4$.
- $h:1 \leftarrow q:1$.
- $f:3 \leftarrow q:2$.
- $s:2$.
- $q:4$.

The two facts, $s:2$, and $q:4$, model the resources that are available from the beginning. The five rules model five different production processes that, when fired, can produce some new resources by consuming (part of the) available ones. Plainly, different rules might compete because they might consume the very same resources (for instance, consider the last three rules: they all use amounts of the resource $q$, but the available amount

---

1In this paper we only deal with negation-as-failure. Nevertheless, classical negation of program literals could be used in RASP and treated as usually done in ASP.
of $q$ does not suffice for firing all these rules). In such cases, alternatives are possible, corresponding to different allocations of available resources and different sets of fired rules. Consequently, each different allocation originates a different answer set for the program. In particular, the program $P$ has two answer sets. In fact, one possibility consists in firing the third rule and, in this way, consuming all the available 4 units of resource $q$. This produces 2 units of resource $g$ (i.e., $g:2$). Then, by using one unit of available resource $s:2$ and one unit of the just produced resource $g:2$, we can fire the first rule to produce $r:3$. To sum up, we can represent the final balance of resources by writing:

\{s:1,p:2,r:3\}

A more informative way of denoting this answer set is the following:

\{q:4, −q:4, s:2, −s:1, g:2, −g:2, p:2, r:3\}

where both produced (or available) and consumed amounts are explicitly indicated. For instance, $s:2$ denotes a produced amount while $−s:1$ denotes a consumption.

A second answer set of $P$ is:

\{q:4, −q:3, s:2, f:3, h:1\}

(or, in compact notation, \{q:1, s:2, f:3, h:1\}) where we consumed part of resource $q:4$ (as indicated by the notation $−q:3$) to fire the fourth and fifth rule so as to produce $h:1$ and $f:3$. In this case, as we do not have consumed the full amount of $q$, there is a remainder $q:1$ that might have been potentially used elsewhere.

Observe that, we cannot produce $g:2$ together with $f:3$ and/or $h:1$ because the available quantity of $q$ is not sufficient. Consequently, in this case we cannot produce $r:3$ as happens in the first answer set.

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

Consider again the program $P$ of Example 1. Other answer sets for $P$ exist. They corresponding to cases where all or part of available resources, though available, is left unconsumed. In \{q:4, s:2\} no resource is used, so no rule is enabled to fire. In the answer sets (using the extended notation) \{q:4, s:2, f:3, −q:2\} and \{q:4, s:2, h:1, −q:1\} only part of resource $q:4$ is used while $s:2$ is left intact. Notice that, if $q:4$ is not consumed to produce $g:2$, it is impossible to consume $s:2$. In fact, a unit of $s$ is required only by the first rule which however also requires a unit of $q$.

In (Costantini & Formisano, 2010) we introduce politics for resource usage, by extending the semantics so as to allow a programmer to state, for each rule, if the firing is either optional or mandatory.

Remark 1. In the rest of this paper, we adopt one of the possible politics described in (Costantini & Formisano, 2010). Namely, we make the assumption that the firing of rules is mandatory, i.e., that whenever an available amount of some resource can potentially be used it is actually used.

3. Reduct-based RASP Semantics

In this section, we introduce a version of the semantics of RASP close to the one originally defined for plain ASP, i.e., in terms of a reduct (for ASP, the GL-reduct, recalled
in Section 1). Then, in Section 5, we will identify a connection between RASP and linear logic and find a linear logic formalization of default negation.

The reduct-based semantics of RASP is defined on standardized-apart RASP programs, whose definition was originally introduced in (Costantini & Formisano, 2010) in order to evaluate RASP complexity (which turned out to be the same as plain ASP). It involves generating, from a ground r-program, a version where resource-symbols occurring in bodies of rules are associated to the rule where they occur. For the sake of simplicity and without loss of generality, we assume that a resource symbol, say $q$, may occur more than once in the body of the same rule only if the amounts are different. It may occur instead with either the same or different amounts in the head and body of several rules. Below, let a “program” be a RASP program. The answer sets for a program whose definition was originally introduced in (Costantini & Formisano, 2010) in linear logic and find a linear logic formalization of default negation.

**Definition 2.** Let $P$ be a ground r-program, and let $\gamma_1, \ldots, \gamma_k$ be the rules in $P$ containing amount-atoms in their body. The standardized-apart version $P_s$ of $P$ is obtained from $P$ by renaming each amount-atom $qa$ in the body of $\gamma_j$, $j \leq k$ as $q^ja$. The $q^j$'s are called the standardized-apart versions of $q$, or, in general, standardized-apart resource-symbols.

Given amount atom $qa$, $q^ja$ is called a standardized-apart version of $qa$, or more generally a standardized-apart amount atom related to the resource symbol $q$.

**Example 2.** Let $P$ be the following program.

$$
\begin{align*}
g:2 & \leftarrow q:4. & a & \leftarrow \neg b. \\
p:3 & \leftarrow q:3, a. & b & \leftarrow \neg a. \\
d:1 & \leftarrow q:3, b. & c. \\
q:6 & \leftarrow c.
\end{align*}
$$

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

$$
\begin{align*}
g:2 & \leftarrow q^1:4. & a & \leftarrow \neg b. \\
p:3 & \leftarrow q^2:3, a. & b & \leftarrow \neg a. \\
d:1 & \leftarrow q^3:3, b. & c. \\
q:6 & \leftarrow c.
\end{align*}
$$

We let $A_P$ be the set of all atoms (both program-atoms and amount-atoms) that can be built from predicate and constant symbols occurring in $P$. Notice that the same r-rule might occur more than once in the same program.

**Definition 3.** A candidate reduct-interpretation $I_P$ for $P_s$ is any multiset obtained from a subset of $A_P$.

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

**Definition 4.** Given multi-set of atoms $S$ and resource-symbol $q$ (possibly standardized-apart) occurring in $S$, let $f(S)(q)$ be an amount-symbol obtained by summing the quantities related to the occurrences of $q$. If $S$ contains $qa_1, \ldots, qa_k$ (or, respectively, $q^ja_1, \ldots, q^ja_k$) and $a = a_1 + \ldots + a_k$ we will have $f(S)(q) = a$ (or, respectively, $f(S)(q^j) = a$).
Definition 5. A candidate reduct-interpretation $I_P$ is a reduct-interpretation (for short $r$-interpretation) if for every resource-symbol $q$ occurring in $I_P$, taken all its standardized-apart versions $q^i, \ldots, q^h$, $h \geq 0$, also occurring in $I_P$, we have $f(I_P)(q) \geq \sum_{i=1}^{h} f(I_P)(q^i)$.

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

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

Definition 6 (RASP-reduct). Given a reduct-interpretation $I_P$, for the standardized-apart version $P_s$ of RASP program $P$, the RASP-reduct $\text{cfp}(P_s, I_P)$ is a RASP program obtained as follows.

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

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

Definition 7. Given reduct-interpretation $I_P$, for the standardized-apart version $P_s$ of RASP program $P$, $I_P$ is a reduct-answer set ($r$-answer set) of $P_s$ if $I_P = LM(\text{cfp}(P_s, I_P))$

Referring again to Example 2 (recall that we are assuming that firing of rules is mandatory, cf., Remark 1), the $r$-answer sets of $P_s$ are:

$M_1 = \{q:6, g:2, q^1:4, a, c\}$, $M_2 = \{q:6, g:2, q^1:4, b, c\}$, $M_3 = \{q:6, p:3, q^2:3, a, c\}$, $M_4 = \{q:6, d:1, q^3:3, b, c\}$.

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

We conclude this section by recalling that a different manner of defining semantics for RASP programs has been proposed in (Costantini & Formisano, 2010). In (Costantini, Formisano, & Pearce, 2012) we proved the equivalence of the above-reported reduct-based semantics with the original semantics defined in (Costantini & Formisano, 2010). In particular, the following result holds:

Proposition 1. There exists a bijection between the set of RASP answer sets of ground RASP program $P$ and the set of reduct-answer sets of the standardized-apart version $P_s$ of $P$.

4. Background on Linear Logic

Linear logic (Girard, 1987) can be considered as a resource sensitive refinement of classical logic, since it intrinsically supports a natural accounting of resources. This is so because in linear logic, in opposition to classical and intuitionistic logic calculi, con-
traction and weakening rules are dropped. Intuitively speaking, in linear logic, two assumptions of a formula $P$ are distinguished from a single assumption of it. We could say that, while classical logic is about truth and intuitionistic logic is about construction of proofs, linear logic is more concerned with resources and their usage (Scedrov, 1993, 1995). This informal perspective immediately emphasizes a relationship between (propositional) linear logic and RASP.

In what follows we will 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 interested reader to (Girard, 1995; Lincoln, 1992), among others.

As mentioned, in linear logic contraction and weakening rules are not allowed. Hence while a statement of the form $P \rightarrow P \land P$ 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 (Girard, 1995; Lincoln, 1992) 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 $P \rightarrow Q \land R$, 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 $P \otimes Q$ 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 $\bot$ is the only negative operation in the logic. It is involutive (namely, $(P^\bot)^\bot$ 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 $Q^\bot \multimap P^\bot$. Moreover, the linear implication $P \multimap Q$ can be rewritten as $P^\bot \otimes Q$.

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.

Too better illustrate all these connectives, let us recall another example (taken from (Lincoln, 1992)). Suppose for 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:

- One dollar
- One pack of Camel
- One pack of Marlboro

The vending machine can be axiomatized as follows:

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

In classical logic, one can derive that $P \rightarrow Q \land R$, 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).
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:

\[
(D \otimes D \otimes D \otimes D \otimes D) \rightarrow (H \otimes C \otimes F \otimes (O \& S) \otimes (P \oplus I))
\]

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 \((P_1 \otimes \ldots \otimes P_n)\) (for \(n > 0\)). Then, generalized Horn implications are defined as follows:

- An Horn implication has the form: \(X \rightarrow \circ Y\).
- An \(\oplus\)-Horn implication has the form: \((X_1 \rightarrow \circ (Y_1 \oplus Y_2))\).
- An \&-Horn implication has the form: \(((X_1 \rightarrow \circ Y_1) \& (X_2 \rightarrow \circ Y_2))\).

Notice that a formula of the last form, say \((P_1 \rightarrow \circ Q_1) \& (P_2 \rightarrow \circ Q_2)\), encodes a non-deterministic process where a choice is made between the two disjuncts (say \(P_2 \rightarrow \circ Q_2\)) 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 \(\Delta\) together imply the multiplicative disjunction of the formulas in \(\Gamma\). In general, a sequent calculus proof rule consists of a set of hypothesis sequents and a single conclusion sequent. It is displayed as below:

\[
\begin{array}{l}
\text{Hypothesis}_1 \quad \text{Hypothesis}_2 \quad \ldots \quad \text{Hypothesis}_n \\
\hline
\text{Conclusion}
\end{array}
\]

A full set of Gentzen-style sequent rules for linear logic can be found, for instance, in (Kanovich, 1994; Lincoln, 1992). For the sake of conciseness, we recall in Table 1 only those proof rules that will be exploited in the following sections. Notice that we restrict our treatment to the inference rules of intuitionistic linear logic. These are obtained from the rules of full linear logic by imposing the restriction that the right-hand side of each sequent is made of a single formula. Focusing on the intuitionistic fragment of linear logic is justified by the following result (cf., (Kanovich, 1994)):

**Lemma 1.** For any \(\Delta\) consisting of generalized Horn implications, and any tensor products \(X, Y\), a sequent of the form \(X, \Delta \vdash Y\) is derivable in linear logic if and only if it is derivable in intuitionistic linear logic.

5. **RASP and Linear Logic**

In this section, we discuss how RASP syntax and semantics can be rendered in a fragment of linear logic. We first consider RASP programs without negation, and then generalize to the full case. In fact, we may notice that the linear logic treatment of negation according to the answer set semantics is not specific of RASP, but constitutes an issue in itself.
Table 1. A fragment of Gentzen-style sequent calculus for (intuitionistic) linear logic. (Where \( \Delta \) denotes a sequence of formulas, \( X, Y, Z \) and \( W \) denote tensor products of positive literals, and \( P, Q, R \) denote generalized Horn implications.)

Table 2. Proofs of \( X \otimes Z, (X \rightarrow oY) \vdash Y \otimes Z \) and of \( \Sigma_1, \Sigma_2, (X \rightarrow oY) \vdash Z \otimes W \).

### 5.1 Positive RASP Programs

Let us consider for now “pure” positive RASP programs, intending positive RASP programs entirely consisting of amount atoms. Below we propose a translation of a pure RASP program into a linear logic theory employing as connectives tensor product \( \otimes \), linear implication \( \rightarrow o \), and additive disjunction \( & \). In well-known terminology, we basically adopt formulas belonging to the so-called Horn-fragment of linear logic, subsets of which are proved in (Kanovich, 1994) to be NP-complete. Our use of additive connective \( & \), that as we will see is needed to model the possible different resource allocations, determines a fragment to the best of our knowledge not fully characterized in terms of complexity. (For instance, (Kanovich, 1994) among its many results, does not specifically mention the combination that we are going to adopt.)

The translation from RASP into linear logic relies on the observation that resource production in RASP corresponds to provability in linear logic. In this frame of mind, a resource unit in RASP can be modeled by an atomic formula in linear logic. Moreover, the firing of a rule in RASP is rendered by the use, in a linear logic derivation, of a linear implication. Before introducing the formal definitions leading to the translation, let us describe the rationale behind such “encoding” of RASP into linear logic by means of a simple working example. Consider again the RASP program \( P \) of Example 1:
As mentioned, a fact represents an initially available amount of units of resource. Hence, it should be translated into a set of linear atomic formulas, one for each unit of available resource. For instance, a fact such as $q:4$, should be translated into four copies of the linear atomic formula $q$. From them, it is immediate to derive formula $q \otimes q \otimes q \otimes q$.

A similar translation can be done for $s:2$, yielding $s \otimes s$.

In what follows, for the sake of conciseness we will still employ the notation of amount atoms (as in $s:2$), where however each of them stands for the tensor conjunction of as many items as the quantity indicates (as in $s \otimes s$).

The translation of rules exploits linear implication to render the transformation process of consumed resources into produced ones. This translation must rely on the standardized-apart version of the program. This is needed in order to take into account the distribution of available resources among rules that need them, and of possible alternative usages. Hence, any rule $H \leftarrow \Sigma$ is translated as $\Sigma - \circ \circ H$, where by abuse of notation we still call these formulas “rules”, where $H$ is the “head” and $\Sigma$ the “body” (possibly empty, then we will talk of “facts”). In our working example we have this standardized-apart version of $P$:

\[
\begin{align*}
  r:3 & \leftarrow g:1, s:1, \\
  p:2 & \leftarrow g:1, \\
  g:2 & \leftarrow q:4, \\
  h:1 & \leftarrow q:1, \\
  f:3 & \leftarrow q:2, \\
  s:2 & \leftarrow q \\
  q:4 & \leftarrow \\
\end{align*}
\]

which originates this translation:

\[
\begin{align*}
  & (g:1 \otimes s:1) - \circ r:3, \\
  & g:2 - \circ p:2, \\
  & q:4 - \circ g:2, \\
  & q:1 - \circ h:1, \\
  & q:2 - \circ f:3, \\
  & s:2, \\
  & q:4 \\
\end{align*}
\]

To take into account possible resource distribution, the $\&$ connective needs however to come into play, to be used in $\&$-Horn implications of the form

\[
((X_1 - \circ Y_1) \& (X_2 - \circ Y_2))
\]

which, as mentioned before, mean non-deterministic choice. I.e., when encountering such a formula during a proof/process (as linear logic is often employed to model various kinds of processes), one has to make a choice between the two Horn implications $(X_1 - \circ Y_1)$ and $(X_2 - \circ Y_2)$, and, after one of them is selected, this implication is performed as next step. In our setting, we will adopt a specific form where $X_1 = X_2$, i.e. we will adopt
only &-Horn implications of the form

\(((X \rightarrow Y_1) \& (X \rightarrow Y_2))\)

that we call Uniform &-Horn implications. In particular, a satisfactory translation \(\Sigma_P\) of \(P\) will be the following:

\[
g^1:1 \otimes s^1:1 \rightarrow r:3, \\
g^2:1 \rightarrow p:2, \\
q^3:4 \rightarrow q^2:2, \\
(q:2 \rightarrow (g^1:1 \otimes g^2:1)), \\
q^4:1 \rightarrow h:1, \\
q^5:2 \rightarrow f:3, \\
s:2, \\
(s:2 \rightarrow s^2:2), \\
q^4, \\
(q:4 \rightarrow q^3:4) \& (q:4 \rightarrow (q^4:1 \otimes q^5:2)) \& (q:4 \rightarrow q^4:1) \& (q:4 \rightarrow q^5:2))
\]

That is, if a resource, say e.g. \(q\), is (potentially) produced in quantity \(n\) (i.e., \(q:n\) is either a fact or the head of a rule), then it can be “allocated” by an &-Horn implication in all possible ways, given the “requests” posed by the various rules. In particular, each conjunct of the &-conjunction is of the form

\[(q:n \rightarrow (q^1:j_1 \otimes \ldots \otimes q^h:j_h))\]

where the \(q^h:j_h\)’s all occur in the body of some rule, and \(n_{j_1} + \ldots + n_{j_h} = n_c \leq n\) (notice that there may be a reminder \(q:r\) of \(q\) if \(n_c < n\), with \(r = n - n_c\); however, we do not consider the remainder explicitly here). So, &-Horn implications must be added to allow for all possible consumption patterns of available resources. Notice the particular cases of \(s:2\) and \(g:2\), for which only one consumption pattern is possible in this program, so the corresponding &-Horn implication has only one conjunct. However, there is a remainder for \(s\) while \(g\) is (potentially) fully consumed. Below, we call “premise” of an &-Horn implication the part before (to the left of ) the \(\rightarrow\), and “conclusions” the parts after (to the right of) each \(\rightarrow\). In fact, in our setting, as said before and formally defined below, the premise is always the same, and represents the presumed allotment of a certain resource, where the conjuncts, which are linear implications, are distinguished by their different conclusions, which represent the possible distributions of the resource itself.

Notice that there is a last aspect to consider when defining the final translation \(\Sigma_P\) of \(P\). Reduct-answer sets of \(P\) contain not only resource atoms corresponding to resources which have been produced. Rather, as shown earlier, they report also resources available from the beginning (facts of \(P\)) and standardized-apart atoms which have been consumed. Below, we intend to establish a correspondence between reduct-answer sets of \(P\) and sets of atoms provable from \(\Sigma_P\). In \(\Sigma_P\) however, whenever a resource is consumed it “disappears”, so one for instance from the above sample program is no longer able to prove \(q:4\) after having consumed it to prove \(g:2\). In fact, \(g:2\) is proved from \(q:4\) via the creation (by a &-Horn implication) of the standardized-apart version \(q^3:4\), which is “generated” and consumed. Since however \(q:4\), \(g:2\) and \(q^3:4\) occur together in some reduct-answer set of \(P\), in order to establish a direct correspondence, we have to keep records of resources that are consumed. Therefore, we make the following convention.

**Remark 2.** In translating into linear logic a RASP program \(P\), every amount atom \(A\) occurring in the head of a \(\rightarrow\) implication and every standardized-apart atom \(A\) occurring in the conclusions of some &-Horn implication implicitly stands for the tensor
conjunction \( A \otimes A_R \) where \( A_R \) is a fresh atom identical to \( A \), \( R \) standing for "record". \( A_R \) is called the r-copy of \( A \).

Observe that r-copies of amount atoms cannot be consumed by rules of \( P \), as they not occur in the body of linear implications. For the sake of simplicity, in what follows we often omit r-copies. Hence, the \( A_R \)'s are left invisible though being implicitly present.

Notice that, if the amounts \( a_1, \ldots, a_z \) of any resource \( p \) are producible by \( z \) rules, each one with head \( p \alpha_i \) (for \( i \leq z \)), then, any number of these rules might actually fire and produce the related resource amount. Thus, it is necessary to add to \( \Sigma \) for each non-empty subset of resource items potentially producible, i.e., with premise one set of \( \land \)-Horn implications. Instead, there must be at least one \( \land \)-Horn implication for each non-empty subset of resource items potentially producible, i.e., with premise \( p \alpha_1 \otimes \ldots \otimes p \alpha_z \), where \( \{p \alpha_1, \ldots, p \alpha_z\} \subseteq \{p \alpha_1, \ldots, p \alpha_z\} \). The conclusions will account for possible distribution patterns of potentially producible quantity \( a_1 + \ldots + a_z \).

Formally,

**Definition 8.** Given a set of amount atoms \( S = \{q \alpha_1, \ldots, q \alpha_r\} \) (for \( r > 0 \)) where \( q \) is a (possibly standardized-apart) resource symbol. Then,

- the total amount \( a(S) \) for (or, related to) this set is the sum of all quantities, namely, \( a(S) = a_1 + \ldots + a_r \);
- the corresponding tensor conjunction \( T_S \) is \( q \alpha_1 \otimes \ldots \otimes q \alpha_r \). The total amount \( a(S) \) for \( S \) is also called the total amount for \( T_S \).

**Definition 9.** Given two sets of amount atoms \( S_1 \) and \( S_2 \), their total amounts \( a(S_1) \) and \( a(S_2) \) and their corresponding tensor conjunctions \( T_{S_1} \) and \( T_{S_2} \), the linear implication \( T_{S_1} \rightarrow T_{S_2} \) is congruous if \( a(S_1) \geq a(S_2) \).

In our setting in fact, a tensor conjunction of amount atoms can imply another one only if the total amount of the former is sufficient for the latter.

**Definition 10.** Given set of amount atoms \( S \) and \( k \) sets of amount atoms \( S_1, \ldots, S_k \) (\( k > 0 \)), and their corresponding tensor conjunctions \( T_{S_1}, T_{S_2}, \ldots, T_{S_k} \), such that all the linear implications \( T_S \rightarrow T_{S_i} \), \( i \leq k \), are all congruous, the \( \land \)-Horn implication \( (T_S \rightarrow T_{S_1}), \ldots, (T_S \rightarrow T_{S_k}) \) is called a congruous \( \lor \)-Horn implication.

**Definition 11.** Given resource symbol \( p \) occurring in pure positive RASP program \( P \) in the head of \( h \) rules, i.e., corresponding to amount atoms \( q \alpha_1, \ldots, q \alpha_h \) each one head of a rule, the total potential amount of \( p \) is \( tpa(p) = a_1 + \ldots + a_h \).

This amount is said to be "potential", as rules with \( p \) in the head will not necessarily be all applicable, so it may be the case when none or only part of this amount will actually be produced.

**Definition 12.** Given resource symbol \( p \) occurring in pure positive RASP program \( P \), a congruous \( \land \)-Horn implication for \( p \) (or, related to \( p \)) is a congruous \( \land \)-Horn implication where \( S \) is composed of amount atoms related to \( p \) where \( tpa(p) \geq a(S) \), and each of the \( S_1, \ldots, S_k \) is composed of standardized-apart amount atoms related to \( p \) and occurring in the body of rules of \( P \).

A congruous \( \land \)-Horn implication for \( p \) represents a way of distributing resources that are potentially produced by rules to other rules that need to use them. Notice that there will be one such implication for each production pattern. Production patterns are related to which rules are applicable or not.

**Remark 3.** Given resource symbol \( p \) occurring in pure positive RASP program \( P \) in the head of \( h \) rules (i.e., corresponding to set of amount atoms \( S^q = \{q \alpha_1, \ldots, q \alpha_h\} \) each one head of a rule), one tensor conjunction and one congruous \( \land \)-Horn implication for \( p \) can be specified, for each non-empty subset \( S \) of \( S^q \). Each \( S \) correspond in fact to the
hypothesis that just a subset of the rules with \( p \) in the head will be actually applicable.

A pure positive RASP program \( P \) can be transformed into a corresponding Linear Logic RASP Theory as follows (notice that the reverse is also possible, i.e., transform a Linear Logic RASP Theory into a pure positive RASP program).

**Definition 13.** Given pure positive standardized-apart RASP program \( P \), the corresponding Linear Logic RASP Theory \( \Sigma_P \) is obtained as follows:

- For every rule \( A \leftarrow B \) occurring in \( P \), where \( A \) is an amount atom and \( B \) is a (possibly empty) conjunction of amount atoms, add to \( \Sigma_P \) the linear implication \( B \rightarrow A \) (or simply add formula \( A \) if \( B \) is empty);
- For every resource symbol \( p \) occurring in \( P \), add all the possible congruous \&-Horn implications related to \( p \).

Notice that addition of \&-Horn implications determines in the worst case an exponential growth of the size of \( \Sigma_P \) with respect to the size of \( P \), as all possible allocations must be considered. This transposes into proof-theoretic terms what is done in model-theoretic terms in the semantics of RASP. In pure RASP, an exponential number of interpretations in fact arises from possible resource allocations.

According to Definitions 6 and 7, computing a reduct-answer set for positive RASP programs \( P \) given a reduct-interpretation \( I \) for the standardized-apart version \( P_s \) of \( P \) accounts simply to:

- add to \( P_s \) as a fact every standardized-apart amount-atom \( A \in I \), thus obtaining \( P_s^+(I) \);
- compute the Least Herbrand Model of \( P_s^+(I) \).

We remind the reader that the Least Herbrand Model of logic program \( Q \) can be computed by iterating operator \( T^\uparrow_Q \) (cf. e.g., (Lloyd, 1987)), where in particular:

\[
T^0_Q = \{ A \mid A. \text{is a fact in } Q \} \\
T^{n+1}_Q = \{ A \mid A \leftarrow B_1, \ldots, B_n \text{ is a rule in } Q, \text{ and } \{B_1, \ldots, B_n\} \subseteq T^n_Q \}
\]

The Least Herbrand Model of \( Q \) is obtained as the fixpoint \( T^\uparrow_Q \) of operator \( T_Q \) (in the propositional case however, this fixpoint is reached within a finite number of steps, i.e., there exists \( k \) such as \( T^\uparrow_Q = T^k_Q \)).

This is of interest for proving the main result, that will establish a correspondence between reduct-answer sets of \( P_s \) and maximal tensor conjunction provable from \( \Sigma_P \), where:

**Definition 14.** Given linear logic theory \( \Sigma \), a tensor conjunction of atoms \( A_1, \ldots, A_n, n \geq 0 \) is called maximally provable if it is provable from \( \Sigma \), and for any atom \( B \), the tensor \( A_1, \ldots, A_n \otimes B \) is not provable from \( \Sigma \) (we equivalently talk about a maximal tensor conjunction provable from \( \Sigma \)).

It is easy to see that:

**Lemma 2.** Let \( P_s \) be a positive standardized-apart RASP program and \( \Sigma_P \) be the corresponding Linear Logic RASP Theory. Every maximal tensor conjunction \( A \) provable from \( \Sigma_P \) includes all the \( r \)-copies of facts of \( \Sigma_P \) and of standardized-apart atoms occurring in the body of linear implications of \( \Sigma_P \).

In fact, these \( r \)-copies are added to \( P_s \) but cannot be consumed by rules of \( P_s \), and thus result provable (cf., Remark 2). Were such an \( r \)-copy, say \( A \), not in \( A \), then \( A \) would not be maximal.

**Remark 4.** In what follows, for uniformity of notation with reduct-answer sets of \( P_s \), in mentioning atoms composing maximal tensor conjunctions provable from \( \Sigma_P \), by abuse
of notation we omit the subscript \( R \). So, in maximal tensor conjunctions an \( r \)-copy looks like a duplicate of its “original”.

**Theorem 3.** Let \( P_s \) be a positive standardized-apart RASP program and let \( \Sigma_{P_s} \) be the corresponding Linear Logic RASP Theory. \( A_1 \otimes \ldots \otimes A_n \) is a maximal tensor conjunction provable from \( \Sigma_{P_s} \) if and only if \( \{ A_1, \ldots, A_n \} \) is a reduct-answer set for \( P_s \).

**Proof.** If part

Let \( A = A_1 \otimes \ldots \otimes A_n \) be a maximal tensor conjunction provable from \( \Sigma_{P_s} \). We prove that \( M = \{ A_1, \ldots, A_n \} \), i.e., the set composed of the atoms occurring in the given tensor conjunction \( A \), is a reduct-answer set for \( P_s \). We first prove that \( M \) is a reduct interpretation for \( P_s \). Let us in fact consider the subsets of \( \{ A_1, \ldots, A_n \} \) consisting of amount atoms (both standardized-apart or not) related to each single resource symbol \( p \), of the form \( M^p = \{ p:a_1, \ldots, p:a_s, p^t_1:a_{f_1}, \ldots, p^t_s:a_{f_s} \} \), \( s \geq 1, t \geq 0 \). By definition of \( \Sigma_{P_s} \), standardized apart amount atoms in \( M^p \) (if any) have been proved via an \&-Horn implication related to \( p \), with premise \( p:a_1 \otimes \ldots \otimes p:a_s \) and with tensor conjunction \( (p^t_1:a_{f_1} \otimes \ldots \otimes p^t_s:a_{f_s}) \) as one of the conclusions. By Definitions 8–10, we have \( a_1 + \ldots + a_s \geq a_{f_1} + \ldots + a_{f_s} \). This fulfills the condition stated in Definition 5, and then \( M \) is a reduct interpretation for \( P_s \).

Now, we have to prove that \( M \) is the Least Herbrand Model of the version \( P_s' \) of \( P_s \) obtained by adding as facts all standardized-apart amount atoms occurring in \( M \). I.e., we have to prove that all atoms in \( M \), and no others, belong to this Least Herbrand Model. We can proceed by induction.

**Base step:** \( T^0_{P_s} \subseteq M \). Since all facts of \( P_s' \) are in its Least Herbrand Model, we prove that all facts of \( P_s' \) are in \( M \). Part of the facts of \( P_s' \) (specifically, those which do not occur in \( P_s \)) are exactly, as said before, the standardized-apart atoms occurring in \( M \). Each fact \( A \) in \( P_s \) correspond to fact \( A \) in \( \Sigma_{P_s} \). In both cases, by Lemma 2 the \( r \)-copies of these atoms (that, as said before, are duplicates of the original) are in every maximal tensor conjunction \( A \) provable from \( \Sigma_{P_s} \). So, also all facts in \( P_s' \) actually belong to \( M \).

**Inductive step:** Let us assume that \( T^k_{P_s} \subseteq M \), and let us prove that this implies \( T^{k+1}_{P_s} \subseteq M \). For non-standardized-apart atom \( A \in M \) (the standardized-apart atoms have been considered in the Base step) to belong to \( T^{k+1}_{P_s} \), there must be a rule \( A \leftarrow B_1, \ldots, B_n \) in \( P_s \) (as \( P_s' \) differs from \( P_s \) only for facts), that corresponds to a linear implication \( B_1 \otimes \ldots \otimes B_n \leftarrow A \) in \( \Sigma_{P_s} \), where \( \{ B_1, \ldots, B_n \} \subseteq T^k_{P_s} \). Since by inductive hypothesis \( T^k_{P_s} \subseteq M \), then \( A \), that by the applicability of \( (R \leftarrow o) \) is provable from \( \Sigma_{P_s} \), belongs to \( M \). This concludes the inductive proof.

So far however, we have proved that \( T^1_{P_s} \subseteq M \). Assume however that there exists \( A \in M \) where \( A \) does not occur in \( T^1_{P_s} \). \( A \) cannot be a standardized-apart amount atom, as they all belong to \( T^0_{P_s} \). Then, the only way to prove \( A \) in \( \Sigma_{P_s} \) is via a linear implication with head \( A \) and tensor conjunction \( \Sigma \) as the body, where atoms in \( \Sigma \) are in turn provable. Since \( A \) is maximal, it must then include the atoms composing \( \Sigma \), that are by definition also in \( M \). Therefore, \( A \) is derivable in \( P_s \) via a corresponding rule \( A \leftarrow \Sigma \), and thus must belong to \( T^k_{P_s} \) for some \( k \), and consequently to \( T^{k+1}_{P_s} \).

**Only-if part**

Let \( M = \{ A_1, \ldots, A_n \} \) be a reduct-answer set for \( P_s \). We prove that the tensor conjunction \( A = A_1 \otimes \ldots \otimes A_n \) is a maximal tensor conjunction provable from \( \Sigma_{P_s} \). Since \( M \) is obtained as the fixpoint of operator \( T_{P_s} \) whose definition is reported above, we can proceed by induction.

**Base step:** atoms in \( T^0_{P_s} \) are provable from \( \Sigma_{P_s} \). \( T^0_{P_s} \) includes the facts of \( P_s \), which are also facts in \( \Sigma_{P_s} \). \( T^0_{P_s} \) also includes facts of \( P_s' \setminus P_s \), which are the standardized apart
amount atoms. For each resource symbol \( p \), let the related standardized-apart atoms related to \( p \) be \( p^{f_1}:a_{f_1}, \ldots, p^{f_t}:a_{f_t} \), \( t > 0 \). Since a reduct-answer set for \( P_s \) is a reduct interpretation for \( P_s \), then \( M \) must contain amount atoms \( p:a_1, \ldots, p:a_s \), \( s > 0 \), where by Definitions 8–10, we have \( a_1 + \ldots + a_s \geq a_{f_1} + \ldots + a_{f_t} \). Then, when transforming \( P_s \) into \( \Sigma_P \), an \&-Horn implication related to \( p \) is added, with premise \( p:a_1 \otimes \ldots \otimes p:a_s \) and with tensor conjunction \((p^{f_1}:a_{f_1} \otimes \ldots \otimes p^{f_t}:a_{f_t})\) as one of the conclusions. Thus, \( p^{f_1}:a_{f_1}, \ldots, p^{f_t}:a_{f_t} \) are provable from \( \Sigma_P \) via this \&-Horn implication, by choosing this conjunct. In both cases, r-copies of facts of \( P_s \) are provable in \( \Sigma_P \), without spoiling the possibility of using these facts in subsequent proofs.

**Inductive step:** Let us assume that all atoms in \( T^P_{\mathcal{P}_s} \) are provable from \( \Sigma_P \), and let us prove that this implies that all atoms in \( T^P_{\mathcal{P}_s+1} \) are provable from \( \Sigma_P \). For non-standardized-apart atom \( A \in M \) (the standardized-apart atoms have been considered in the Base step) to belong to \( T^P_{\mathcal{P}_s+1} \), there must be a rule \( A \leftarrow B_1, \ldots, B_n \) in \( P_s \) (as \( P_s' \) differs from \( P_s \) only for facts). When \( \Sigma_P \) is constructed, a corresponding linear implication \( B_1 \otimes \ldots \otimes B_n \rightarrow A \) is added. Thus, as the \( B_i \)s are, by inductive hypothesis, provable from \( \Sigma_P \) (more precisely, their r-copies are provable), \( A \) is provable as well via Rule \((R \rightarrow \rightarrow)\) of the linear inference system that we adopt (cf., Table 1). This concludes the inductive proof.

Positive RASP programs including plain ASP positive literals are easily managed. Whenever a rule in a RASP program has a plain ASP atom \( A \) as its head, then \( A \) is not a resource with limited usage, but is instead unlimitedly available. 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 Horn propositional fragment of linear logic. What we really need is to simulate a sort of “bounded exponentiation”. Namely, in place of !\( A \), we introduce a finite number of copies of \( A \). Such number depend on the number of occurrences of \( A \) in the program. More specifically, we devise the following method:

**Definition 15.** Given positive standardized-apart RASP program \( P_s \), the corresponding Linear Logic RASP Theory \( \Sigma_P \) is obtained as follows:

- in \( P_s \), for each \( A \), standardize-apart also the occurrences of \( A \) appearing in the body of rules, say, in rules \( j_1, \ldots, j_m \). So, obtain a new version \( P'_s \) of \( P_s \) where the \( m \) occurrences of \( A \) are replaced by \( A^{j_1}:1, \ldots, A^{j_m}:1 \) (i.e., \( A^{j_i}:1 \) occurring in the rule \( j_i \)).
- transform \( P'_s \) into \( \Sigma_P \), by considering \( A \) like all the other atoms (i.e., transforming rules with head \( A \) into corresponding linear implications with head \( A \));
- add to \( \Sigma_P \) the linear implication \( A \rightarrow A^{j_1}:1 \otimes \ldots \otimes A^{j_m}:1 \) (which can be seen as an \&-Horn implication with a unique conjunct).

As before, r-copies are left implicit, therefore, \( A \) stands for \( A \otimes A_R \) and each of the \( A^{j_i}:1 \)'s stands for \( A^{j_i}:1 \otimes A_R^{j_i}:1 \).

Therefore, we reduce the case of positive RASP programs to the case of pure positive RASP programs. It is easy to see that, if obtaining \( \Sigma_P \) with the procedure dictated in the above definition, Theorem 3 can be improved as follows:

**Theorem 4.** Let \( P_s \) be a standardized-apart RASP program and let \( \Sigma_P \) be the corresponding Linear Logic RASP Theory. \( A_1 \otimes \ldots \otimes A_n \) is a maximal tensor conjunction provable from \( \Sigma_P \) if and only if \( \{A_1, \ldots, A_n\} \) is a reduct-answer set for \( P_s \).

**Proof.** The proof develops along the lines of the proof of Theorem 3, thanks to the treatment of ASP atoms introduced by Definition 15.
Notice that the above results allow logician to adopt a RASP solver (cf. (Costantini et al., 2010)) to program in a fragment of linear logic. For the sake of simplicity, the programmer might just write down the basic linear implications, while the related and Horn implications might be generated automatically.

**Example 3.** Let us consider the program of Example 1 and its translation in linear logic, as described earlier (see page 13). Table 3 shows a linear logic derivation of a tensor product from such a translation, corresponding to the reduct answer set \{q:4, q:1, q:2, h:1, f:3, s:2\}.

### 5.2 Full RASP

Recall that, by the definition of RASP, it is not allowed to negate amount atoms. Hence, for every literal not A occurring in a RASP program, A is a plain ASP atom and not an amount atom. Then, to model via linear logic negation in RASP, it suffices to model negation in ASP.

The representation that we have devised derives from the observations and theory developed in (Costantini, 2006) (where a reader can also find references to significant related work on these aspects). The discussion in (Costantini, 1995, 2006) can be very shortly summarized as follows: in general, an answer set of program \(P\) is a classical minimal model of \(P\) (interpreted as a first-order theory in the obvious way) where no true atom \(A\) necessarily depends upon the negation (in ASP terms) of another true atom, and ultimately of itself (i.e., there must be ways of supporting \(A\) that do not involve not\(A\)). The existence of answer set ("consistency" of the program) is thus determined by cycles on negation, or "negative cycles". The simplest example is the program composed by the single rule \(p \leftarrow \text{not } p\), that can be seen in classical terms as \(p \lor \text{not } p\): the unique classical model \(\{p\}\) is not an answer set, because in the answer set program the only way for potentially supporting \(p\) is via its own negation \(\text{not } p\). Thus, consistency is related (as discussed at length in (Costantini, 2006)) to the occurrence of odd cycles (of which \(p \leftarrow \text{not } p\) is the basic case, though odd cycles may involve any odd number of atoms) and how they are related to other parts of the program. The reason is that, in the answer
set semantics, the negation \( \text{not } A \) of an atom \( A \) is an assumption, that must be dropped whenever \( A \) can be proved, as answer sets are by definition non-contradictory.

Assume there are \( n \) occurrences of \( \text{not } A \) in the body of rules of given program \( P \). To represent full RASP (and thus full ASP) we improve the transformation devised in Definitions 13–15 as follows.

**Definition 16.** Given standardized-apart RASP program \( P_s \), the corresponding Linear Logic RASP Theory \( \Sigma_{P_s} \) is obtained by the following steps:

- In \( P_s \), for each \( A \), standardize apart also the occurrences of \( \text{not } A \) appearing in the body of rules, say, in rules \( j_1, \ldots, j_m \), so, obtain a new version \( P'_s \) of \( P_s \) where \( \text{not } A^{i_1}:1, \ldots, \text{not } A^{i_r}:1 \) occur in rule bodies in place of \( \text{not } A \) (with \( \text{not } A^{i_1}:1 \) occurring in the \( j_i \)-th rule).
- Add to a first version of \( \Sigma_{P_s} \) the new fact \( \text{not } A:n \), unless \( A \) is a fact.
- In \( P_s \), add to each non-unit rule \( A \leftarrow \Sigma \) the new conjunct corresponding to the standardized-apart version of not \( A:n \), i.e., add not \( A^{k}:n \) to the body of the \( k \)-th rule. Let not \( A^{k_1}:n, \ldots, A^{k_s}:n \), \( s > 0 \), be the added instances. In case multiple identical copies arise in a body, in consequence of such addition, for instance, not \( A^{k_1}:1 \) and not \( A^{k_2}:1 \) occur in the same rule, then further standardize apart into not \( A^{k_1}:1 \) and not \( A^{k_2}:1 \).
- For each fact not \( A:n \) added to \( \Sigma_{P_s} \), add also the uniform \&-Horn implication:
  \[
  (\text{not } A:n \rightarrow \text{not } A^{k_1}:n) \& \cdots \& (\text{not } A:n \rightarrow \text{not } A^{k_s}:n) \&
  (\text{not } A:n \rightarrow \text{not } A^{k_1}:1, \otimes \cdots \otimes \text{not } A^{k_s}:1)
  \]
  thus obtaining a second version of \( \Sigma_{P_s} \).
- On the modified \( P_s \) and on the second version of \( \Sigma_{P_s} \), obtained as specified above, follow for the rest the steps as indicated in Definitions 13–15 to obtain the final \( \Sigma_{P_s} \).

The meaning is that the assumption \( \text{not } A \) is available to every rule that intends to adopt it unless \( A \) is provable, in which case the assumption becomes unavailable (as proving \( A \) consumes the full quantity of \( \text{not } A \)). This prevents contradictions.

**Example 4.** Consider the above simple program \( p \leftarrow \text{not } p \). It is transformed into:
\[
\text{not } p^{11}:1 \otimes \text{not } p^{12}:1, \text{not } p^{13}:1, (\text{not } p:1 \rightarrow \text{not } p^{11}:1) \& (\text{not } p:1 \rightarrow \text{not } p^{12}:1)
\]

In the first rule, one occurrence of \( \text{not } p \) corresponds to the one originally present in the program, the other one has been added as for proving \( p \), it is necessary to consume the whole available quantity of \( \text{not } p \). We can in fact verify that \( \{ p \} \) cannot be an answer set, as the singleton tensor conjunction \( p \) is by no means provable. In fact, it would require two units of \( \text{not } p \), while just one is available. This to signify that trying to prove \( p \) in the first place “absorbs” \( \text{not } p \), that cannot be further used to perform the proof, thus avoiding the contradiction.

In the rest of this section, we will follow the method introduced in previous section, i.e. trying to establish a relationship between answer sets of \( P_s \), of the form \( \{ A_1, \ldots, A_n \} \), and maximal tensor conjunctions, of the form \( A_1 \otimes \cdots \otimes A_n \), provable from \( \Sigma_{P_s} \). Here we do not however obtain a full equivalence, due to the lack of relevance of the answer set semantics (Dix, 1995b, 1995a), that cannot easily be addressed in a proof-theoretic system. If one would add, for instance, to the (inconsistent) program of Example 4, the rule \( p \leftarrow a \), and the fact \( a \), thus obtaining a consistent program with unique answer set \( \{ a, p \} \), from the resulting linear logic theory one would be able to prove the
corresponding maximal tensor conjunction $a \otimes b$. However, if one added just fact $a$, the singleton tensor conjunction $a$ would be a provable maximal tensor conjunction, but not an answer set, as the resulting program would still be inconsistent. Lack of relevance in fact implies that it is not sufficient to prove something, e.g., $a$, locally, by means of “relevant rules”, i.e., for $a$, rules with conclusion $a$ and rules upon which $a$ depends indirectly. This because local provability does not guarantee global consistency.

Consider now another simple program, consisting in an even loop: $a \leftarrow \neg b . b \leftarrow \neg a .$, with answer sets $\{a\}$ and $\{b\}$. It is transformed into:

$$
\begin{align*}
\neg a^1 : 1 \otimes \neg b^1 : 1 & \rightarrow a, \\
\neg b^2 : 1 \otimes \neg a^2 : 1 & \rightarrow b, \\
\neg a & : 1, \\
(\neg a : 1 & \rightarrow \neg a^1 : 1) \& (\neg a : 1 & \rightarrow \neg a^2 : 1) \\
\neg b & : 1, \\
(\neg b : 1 & \rightarrow \neg b^1 : 1) \& (\neg b : 1 & \rightarrow \neg b^2 : 1)
\end{align*}
$$

Then, singleton conjunction $a$ corresponding to answer set $\{a\}$ is actually provable, by using both resources $\neg a^1 : 1$ and $\neg b^1 : 1$, which are obtained by selecting the first conjunct in both the $\&$-Horn implications. Singleton conjunction $b$ corresponding to answer set $\{b\}$ is provable in a similar way, by selecting the second conjunct. However, notice that $a \otimes b$ is not provable, and in fact $\{a, b\}$ is not an answer set. According to linear logic in fact, proving $a \otimes b$ requires a separate proof for both $a$ and $b$, where however resources, once used, are no longer available. So, proving the first conjunct “drains” all available resources, and therefore the proof of the second one fails, leading to failure of the whole proof. Notice that most proof-theoretic systems have problems with such cycles, and even a naive translation to linear logic would lead to erroneously proving both $a$ and $b$.

We proceed now to the main result. We remind the reader that a maximal tensor conjunction is composed of atoms, so it does not contain negative literals.

**Lemma 5.** Let $P_s$ be a standardized-apart RASP program, and let $\Sigma_{P_s}$ be the corresponding linear logic RASP theory, obtained according to Definitions 13–16. Let $M = \{A_1, \ldots, A_n\}$ be a reduct-answer set for $P_s$. Let $P'_s = cfp(P_s, M)$, and let $\Sigma_{P'_s}$ be the corresponding linear logic RASP theory, obtained according to Definitions 13–15. Maximal tensor conjunction $A = A_1 \otimes \ldots \otimes A_n$ is provable from $\Sigma_{P_s}$ if and only if it is provable from $\Sigma_{P'_s}$.

**Proof.** Let us examine how $P'_s$ is obtained from $P_s$ according to Definition 6, so as to investigate how this affects provability in $\Sigma_{P_s}$ and $\Sigma_{P'_s}$ respectively. The procedure dictated by Definition 6 is the following.

1. For every standardized-apart amount-atom $A \in M$, add $A$ to $P_s$ as a fact, obtaining program $P^+_s$. By definition of both $\Sigma_{P_s}$ and $\Sigma_{P'_s}$, standardized-apart atoms are generated by $\&$-Horn implications which make them available to the linear implications that use them (i.e., that have them in their body), which is equivalent to adding these atoms as fact, as done in $cfp(P_s, M)$. The $\&$-Horn implications with an atom (and not a negative literal) as premise are identical in both $\Sigma_{P_s}$ and $\Sigma_{P'_s}$. So, any standardized-apart amount-atom belonging to a maximal tensor conjunction provable from $\Sigma_{P'_s}$ belongs to a corresponding maximal tensor conjunction provable from $\Sigma_{P_s}$, and vice versa, by generating $A$ by means of the same conjunct of the same $\&$-Horn implication.

2. Compute the GL-reduct of $P^+_s$. Let us take into consideration the two points that compose the definition of GL-reduct, considering that there is a biunivocal correspondence between rules of $P_s$ and linear implications in $\Sigma_{P'_s}$; in particular,
each linear implication of $\Sigma_P$ with $\text{not } A:1$ in the body corresponds to a rule $r$ in $P_s$ with $\text{not } A$ in the body, and vice versa.

(a) For every $A \in M$, remove all rules $r_1, \ldots, r_v$ ($v \geq 0$, as there can be no such rule) where $\text{not } A$ occurs in the body. So, no such rule occurs in $P'_s$, and no corresponding linear implication occurs in $\Sigma_P$. However, as $A \in M$ then (by Theorem 3) $A$ occurs in $\mathcal{A}$ and is provable in $\Sigma_P$ via a linear implication with head $A$, that by Definition 16 also occurs in $\Sigma_P'$, though with the additional literal $\text{not } A:n$, which means that the full available amount of $\text{not } A$ is consumed to prove $A$. Consumption of $\text{not } A:n$ to prove $A$ is thus actually equivalent to removing from $\Sigma_P$ all linear implications with $A^k:1$ in the body (for some $k$), as they cannot be used in proving their head, being this resource certainly not available. This is in turn equivalent to not having, in $\Sigma_P$, the corresponding linear implications, as it is the case for $\Sigma_P'$.

(b) For every $A \notin M$, cancel $\text{not } A$ from the body of all rules where it occurs. If $A \notin M$, then $A$ is not provable from $P'_s$ and $\Sigma_P'$, and (by Definition 16) not from $\Sigma_P$, either. Thus, in $\Sigma_P$, resource $\text{not } A:n$ is not consumed. Therefore, by Definition 16 it is divided into standardized-apart fragments of the form $\text{not } A^k:1$ (for some $k$), which are made freely available to every linear implication that has them in the body and thus might potentially use them. This is equivalent to canceling literals $\text{not } A^k:1$ from these linear implications (though without affecting what can be proved), as these resources are granted. In turn, this is equivalent to canceling literals $\text{not } A$ from rules in $P_s$, which accounts to considering $\text{not } A$ true and thus “available” by default, and thus to not having these literals in the corresponding linear implications in $\Sigma_P'$.

Therefore, we have demonstrated that whatever can be derived in $\Sigma_P$ can also be derived in $\Sigma_P'$, and vice versa, by using corresponding linear implications, and thus by using the same resources. This concludes the proof.  

The point of Lemma 5 above is to establish a relationship between maximal tensor conjunctions provable from $\Sigma_P$, which is a linear logic RASP theory corresponding to RASP program $P_s$ which includes negation, and maximal tensor conjunctions provable from $\Sigma_P'$, which is the linear logic RASP theory corresponding to its reduct $P'_s$ which does not include negation and characterizes (by Definition 7) reduct-answer sets of $P_s$. Therefore, we can resort to Theorem 3 to conclude that:

**Theorem 6.** Let $P_s$ be a standardized-apart RASP program, and let $\Sigma_P$ be the corresponding linear logic RASP theory, obtained according to Definitions 13–16. Let $M = \{A_1, \ldots, A_n\}$ be a reduct-answer set for $P_s$. Then, $A_1 \otimes \ldots \otimes A_n$ is a maximal tensor conjunction provable from $\Sigma_P$.

The converse of Theorem 6 of does not hold in general. This is so because, as exemplified before, there are maximal tensor conjunctions that are not answer sets but are provable from $\Sigma_P$, which instead is not the case for positive programs. It is our aim to address this aspect in future work.

The above result suggests that a RASP inference engine might be used as an inference engine for a significant fragment of linear logic, which includes default negation, understood according to the answer set semantics. Most parts of $\Sigma_P$ can be generated in a standard way by a system, starting from plain $\text{not } \circ$ implications, by following the formal definitions.
5.3 Observations on Complexity

We may notice that the translation between RASP programs and linear logic theories of the form seen above works both ways. In particular, a Linear Logic RASP theory can be transformed into a RASP program. It is easy to see that the transformation is polynomial, and that the size of the resulting RASP program is smaller, because resource allocation is left to the underlying inference engine, rather than being performed via &-Horn implications. Precisely, in such a transformation linear implications become RASP rules, and &-Horn implications are simply dropped. So, since the RASP framework has been proved to be NP-complete (like plain ASP, cf. (Costantini & Formisano, 2010)), this translation allows us to make the following observation of the fragment of linear logic that we have adopted. Actually, it is a Horn fragment, where both the head of linear implications and the premise of &-Horn implications can be a tensor conjunction of copies of one and the same atom, where default negation of atoms is also present.

**Theorem 7.** The decision problem for the linear logic fragment consisting of Linear Logic RASP theories is NP-complete.

This result is of some significance, since full Multiplicative-Additive propositional Linear Logic without exponentials (MALL) is P-SPACE complete (cf. e.g., (Lincoln, Mitchell, Scedrov, & Shankar, 1990)).

6. Extended Here-and-There Logic for RASP programs

Strong equivalence, introduced in (Pearce, 1997; Lifschitz et al., 2001), is widely recognized to provide an important conceptual and practical tool for program simplification, transformation and optimization. Even in the case where two theories are formulated in the same vocabulary, they may have the same answer sets yet behave very differently once they are embedded in some larger context. For a robust or modular notion of equivalence one should require that programs behave similarly when extended by any further programs. This leads to the concept of strong equivalence, where programs \( P_1 \) and \( P_2 \) are strongly equivalent if and only if for any \( S \), \( P_1 \cup S \) is equivalent to (has the same answer sets as) \( P_2 \cup S \).

In (Costantini et al., 2012), we have extended (in a joint work with David Pearce) the notion of strong equivalence to RASP programs, that as seen above involve production and consumption of resources. This extension is important because it is easy to see that, whenever \( P_1 \) and \( P_2 \) are different formulations of a RASP program involving consumption and production of resources, their behaving equivalently in different contexts is of particular importance related to reliability in resource usage. For instance, a designer might be able to evaluate, in terms of strong equivalence, different though analogous processes for producing certain resources, so as to choose one rather than the other in terms of suitable criteria.

Strong equivalence of ground ASP programs has been characterized in (Lifschitz et al., 2001) in terms of the propositional logic of here-and-there, or HT-logic (cf. (Lifschitz et al., 2001) and the references therein). In particular, the logic of here-and-there is an intermediate logic between intuitionistic logic and classical logic. Like intuitionistic logic, it can be semantically characterized by Kripke models, though using just two worlds, namely here and there, where the here world is ordered before the there world. Accordingly, interpretations (HT-interpretations) are pairs \((X,Y)\) of sets of atoms from given language \( L \), such that \( X \subseteq Y \). An HT-interpretation is total if \( X = Y \). The intuition is that atoms in \( X \) (the here part) are considered to be true, atoms not in \( Y \) (the there part) are considered to be false, while the remaining atoms (from \( Y \setminus X \)) are
undefined. A total HT-interpretation \((Y,Y)\) is called an equilibrium model of a theory \(T\), iff \((Y,Y)\models T\) and for all HT-interpretations \((X,Y)\), such that \(X \subset Y\), it holds that \((X,Y)\not\models T\). For an answer set program \(P\), it turns out that an interpretation \(Y\) is an answer set of \(P\) iff \((Y,Y)\) is an equilibrium model of \(P\) reinterpreted as an HT-theory.

In (Costantini et al., 2012), we have extended (in a joint work with David Pearce) the propositional logic of here-and-there to RASP programs. Namely, taking as a basis the standardized-apart version \(P_s\) of RASP program \(P\) to account for resource production and consumption, HT-logic has been extended to RASP as follows. It is necessary to introduce amount-atoms, involving both plain and standardized-apart resource symbols. Like in the RASP semantics, we take for given the choice of an algebraic structure to represent amounts and support operations on them. The satisfaction relation of HT-logic between an interpretation \(I = \langle I^H, I^I \rangle\) and a formula \(F\) has been augmented so as to express that each resource can be produced and consumed in several fragments, but what counts is, on the one hand, that consumption does not exceed production, and, on the other hand, which is the total produced quantity. This was obtained by adding two new axioms: the first one (called AR-1) “distributes” the total available quantity of a resource \(q\) to the formulas that use it; the second one (called AR-2) “computes” the total produced quantity of each resource \(q\). This stated, we generalized the results of (Lifschitz et al., 2001), by proving the following:

**Theorem 8.** For any RASP program \(P\) and any set \(I\) of atoms, the HT-interpretation \(\langle I, I \rangle\) is an equilibrium model of \(P\) iff \(I\) is a reduct-answer set of the standardized-apart version \(P_s\) of \(P\).

The above theorem extends to RASP programs the characterization of strong equivalence as provided in (Lifschitz et al., 2001). This notion takes however a quite peculiar flavor in RASP, where strong equivalence (apart from trivial cases) can be ensured only at the condition of imposing some requirements on the theory which is added to given one.

In the present context, it is interesting to notice that we can relate linear logic and HT-logic characterizations of ASP and RASP. In fact, from Theorem 6 and Theorem 8 it immediately follows the following:

**Corollary 9.** For any RASP (and in particular, for any ASP) program \(P\), let \(P_s\) be the corresponding standardized-apart RASP program, and let \(\Sigma_P\) be the corresponding linear logic RASP theory, obtained according to Definitions 13–16. For every equilibrium model \(\langle I, I \rangle\) of \(P\), \(I = \{A_1, \ldots, A_n\}\), we have that \(A_1 \otimes \ldots \otimes A_n\) is a maximal tensor conjunction provable from \(\Sigma_P\).

The above result establishes a connection between two non-classical logics that have proved to be of particular importance in the logic programming realm.

### 7. Concluding Remarks

In this paper, we have proved the equivalence between Resourced Answer Set Programming (of which plain Answer Set Programming is a particular case) and a fragment of linear logic. The result is of theoretical interest, as it establishes a relationship between a proof-theoretic system (linear logic) and a model-theoretic approach (Answer Set Programming, where the answer sets are supposed to represent the solutions of the problem at hand). The concept of resource has proved crucial for modeling in linear logic default negation as understood under the Answer Set Semantics. The fragment of linear logic that we have defined has its own interest, because it is basically an empowered Horn fragment (different from those treated at length in (Kanovich, 1994)) allowing for
a default negation that linear logic does not provide, though still remaining within an NP-complete framework. Our treatment of negation is distinct from the one presented in (Cerrito, 1992) for dealing with Prolog-like negation-as-failure: in this approach, pure Prolog programs are translated into linear logic in the obvious way (by exploiting tensor conjunction, linear implication and linear negation), but the resulting linear logic theories are processed by a linear logic system, called LL, which is able to cope with Prolog procedural aspects.

A comparison between the answer set semantics and linear logic has been attempted in (Osorio, Arrazola-Ramírez, & Palacios-Pérez, 2002) and (Palacios-Pérez, 2006). In the former, a characterization of stable models is provided for augmented programs formed by a linear part (for modeling consumable resources) plus rules for representing knowledge. They adopt an embedding of intuitionistic logic into linear logic, providing results which characterize stable models as provability in intuitionistic logic. The latter work refines these results in terms of the logic of constructible duality $N_{cd}$, which is a conservative extension of the paraconsistent version of Nelson’s logic of constructible falsity (cf. the references in (Palacios-Pérez, 2006)). Both approaches refer to full linear logic, while we stay within a propositional NP-complete framework.

For future developments, we intend to examine the various Horn-fragments of linear logic studied at depth in the literature (cf., e.g., (Kanovich, 1994)) so as to identify possible extensions to RASP and ASP, in the perspective of modeling for instance parallelism. We also mean to extend the concept of strong equivalence of RASP programs that we have introduced in (Costantini et al., 2012) to the linear logic setting, where such a concept has not been explored so far. This extension may be based upon the connection that we have established in Section 6. The way we have proposed of understanding negation might open a perspective for query-answering systems for answer set programs: in fact, by considering subsequent queries to be conjuncts of a tensor conjunction, each query might be answered within the answer set(s) identified by previous ones. This line could be a further subject of future work, though the above-mentioned problem of the lack of relevance of the answer set semantics should be coped with.

Acknowledgements

The authors, and especially Stefania, wish to acknowledge a long-termed acquaintance and friendship with David Pearce. We have been able to appreciate (also in developing joint work) not only the exceptional scientific and intellectual abilities of David’s, but his human qualities that are by no means inferior to the scientific ones. Discussing with David about any topic is pure pleasure, and we always learnt something valuable. David is able to carefully consider the opponent’s arguments, value the positive over the negative aspects, and identify the possibilities and implications. These are, we believe, among the qualities that make David such an outstanding researcher. Stefania is deeply grateful for the friendship David demonstrated to herself and her family in the aftermath of the earthquake that struck L’Aquila, where Stefania lived, on April 6, 2009. David insisted upon inviting them to Spain, accompanied them around and prepared delicious dinners (David’s gentle wife is also to be thanked for all this). Thank you David, and greetings for your (first) 60 years!

References


