Logic Programs with Propositional Connectives and Aggregates

PAOLO FERRARIS
Google Inc.

Answer set programming (ASP) is a logic programming paradigm that can be used to solve complex combinatorial search problems. Aggregates are an ASP construct that plays an important role in many applications. Defining a satisfactory semantics of aggregates turned out to be a difficult problem, and in this paper we propose a new approach, based on an analogy between aggregates and propositional connectives. First, we extend the definition of an answer set/stable model to cover arbitrary propositional theories; then we define aggregates on top of them both as primitive constructs and as abbreviations for formulas. Our definition of an aggregate combines expressiveness and simplicity, and it inherits many theorems about programs with nested expressions, such as theorems about strong equivalence and splitting.

Categories and Subject Descriptors: I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods; F.1.4 [Mathematical Logic and Formal Languages]: Mathematical logic—Logic and constraint programming

General Terms: Languages, Theory

Additional Key Words and Phrases: Aggregates, answer sets, aggregates, logic programming, Stable models

1. INTRODUCTION

Answer set programming (ASP) is a logic programming paradigm that can be used to solve complex combinatorial search problems ([Marek and Truszczyński 1999], [Niemelä 1999]). ASP is based on the stable model semantics [Gelfond and Lifschitz 1988] for logic programs: programming in ASP consists in writing a logic program whose stable models (also called answer sets) represent the solution to our problem. ASP has been used, for instance, in planning [Dimopoulos et al. 1997; Lifschitz 1999], model checking [Liu et al. 1998; Heljanko and Niemelä 2001], product configuration [Soininen and Niemelä 1998], logical cryptanalysis [Hietalahti et al. 2000], workflow specification [Trajcevski et al. 2000; Koksal et al. 2001], reasoning about policies [Son and Lobo 2001], wire routing problems [Erdem et al. 2000] and phylogeny reconstruction problems [Erdem et al. 2003].

The stable models of a logic program are found by systems called answer set solvers. Answer set solvers can be considered the equivalent of SAT solvers — systems used to find the models of propositional formulas — in logic programming.
On the other hand, it is much easier to express, in logic programming, recursive definitions (such as reachability in a graph) and defaults. Several answer set solvers have been developed so far, with smodels\(^1\) and dlv\(^2\) among the most popular. As in the case of SAT solvers, answer set solver competitions — where answer set solvers are compared to each others in terms of performance — are planned to be held regularly.\(^3\)

An important construct in ASP are aggregates. Aggregates allow, for instance, to perform set operations such as counting the number of atoms in a set that are true, or summing weights the weights of the atoms that are true. We can, for instance, express that a node in a graph has exactly one color by the following cardinality constraint:

\[
1 \leq \{c(node, color_1), \ldots, c(node, color_m)\} \leq 1.
\]

As another example, a weight constraint of the form

\[
3 \leq \{p = 1, q = 2, r = 3\}
\]

intuitively says that the sum of the weights (the numbers after the “=” sign) of the atoms from the list \(p, q, r\) that are true is at least 3.

Aggregates are a hot topic in ASP not only because of their importance, but also because there is no standard understanding of the concept of an aggregate. In fact, different answer set solvers implement different definitions of aggregates: for instance, smodels implements cardinality and weight constraints [Niemelä and Simons 2000], while dlv implements aggregates as defined by Faber, Leone and Pfeifer (2005) (we call them FLP-aggregates). Unfortunately, constructs that are intuitively equivalent to each other may actually lead to different stable models. In some sense, no current definition of an aggregate can be considered fully satisfactory, as each of them seems to have properties that look unintuitive. For instance, it is somehow puzzling that, as noticed in [Ferraris and Lifschitz 2005b], weight constraints

\[
0 \leq \{p = 2, p = -1\} \quad \text{and} \quad 0 \leq \{p = 1\}
\]

are semantically different from each other (may lead to different stable models). Part of this problem is probably related to the lack of mathematical tools for studying properties of programs with aggregates, in particular for reasoning about the correctness of programs with aggregates.

This paper addresses the problems of aggregates mentioned above by (i) giving a new semantics of aggregates that, we argue, is more satisfactory than the existing alternatives, and (ii) providing tools for studying properties of logic programs with aggregates.

Our approach is based on a relationship between two directions of research on extending the stable model semantics: the work on aggregates, mentioned above, and the work on “propositional extensions” (see Figure 1). The latter makes the syntax of rules more and more similar to the syntax of propositional formulas.

\(^1\)http://www.tcs.hut.fi/Software/smodels/
\(^2\)http://www.dbai.tuwien.ac.at/proj/dlv/
\(^3\)http://dtai.cs.kuleuven.be/events/ASP-competition/index.shtml
In disjunctive programs, the head of each rule is a (possibly empty) disjunction of atoms, while in programs with nested expressions the head and body of each rule can be any arbitrary formula built with connectives AND, OR and NOT. For instance,

\[-(p \vee \neg q) \leftarrow p \vee \neg \neg r\]

is a rule with nested expressions. Programs with nested expressions are quite attractive especially relative to point (ii) above, because many theorems about properties of logic programs have been proved for programs of this kind. For instance, the splitting set theorem [Lifschitz and Turner 1994; Erdoğan and Lifschitz 2004] simplifies the task of computing the stable models of a program/theory by breaking it into two parts. Work on strong equivalence [Lifschitz et al. 2001] allows us to modify a program/theory with the guarantee that stable models are preserved (more details in Section 2.5).

Nested expressions have already been used to express aggregates: [Ferraris and Lifschitz 2005b] showed that each weight constraint can be replaced by a nested expression, preserving its stable models. As a consequence, theorems about nested expressions can be used for programs with weight constraints. It turns out, however, that nested expressions are not sufficiently general for defining a semantics for aggregates that overcomes the unintuitive features of the existing approaches. For this reason, we extend the syntax of rules with nested expressions, allowing implication in every part of a “rule”, and not only as the outermost connective. (We understand a rule as an implication from the body to the head). A “rule” is then an arbitrary propositional formula, and a program an arbitrary propositional theory. Our new definition of a stable model, like many other definitions of a stable model (in particular, for the languages in Figure 1), is based on the process of constructing a reduct. The process that we use looks very different from the
others, and in particular for programs with nested expressions. Nevertheless, it
turns out that in application to programs with nested expressions, our definition is
equivalent to the one from [Lifschitz et al. 1999]. This new definition of a stable
model also turns out to closely related to equilibrium logic [Pearce 1997], a logic
based on the concept of a Kripke-model in the logic of here-and-there. Also, we
will show that many theorems about programs with nested expressions extend to
arbitrary propositional theories.

On top of arbitrary propositional formulas, we give our definition of an aggregate.
Our extension of the semantics to aggregates treats aggregates in a way similar to
propositional connectives. Aggregates can be viewed either as primitive constructs
or as abbreviations for propositional formulas; both approaches lead to the same
concept of a stable model. The second view is important because it allows us
to use theorems about stable models of propositional formulas in the presence of
aggregates. As an example of application of such theorems, we use them to prove
the correctness of an ASP program with aggregates that encodes a combinatorial
auction problem.

Syntactically, our aggregates can occur in any part of a formula, even nested
inside each other. (The idea of “nested aggregates” is not new: for instance [Pelov
et al. 2007] allows the nesting of aggregates, and the proof of Theorem 3(a) in [Ferr-
raris 2007a] involves “nested weight constraints”.) In our definition of an aggregate
we can have, in the same program/theory, many other kinds of constructs, such as
choice rules and disjunction in the head, while other definitions allow only a subset
of them. Our aggregates seems not to exhibit the unintuitive behaviours of other
definitions of aggregates.

It also turns out that a minor syntactical modification of programs with FLP-
aggregates allows us to view them as a special kind of our aggregates. (The new
picture of extensions is shown in Figure 2.) Consequently, we also have a “proposi-
tional” representation of FLP-aggregates. We use this fact to compare them with other aggregates that have a characterization in terms of nested expressions. (As we said, [Ferraris and Lifschitz 2005b] showed that weight constraints can be expressed as nested expressions, and also [Pelov et al. 2003] implicitly defined PDB-aggregates in terms of nested expressions.) We will show that all characterizations of aggregates are essentially equivalent to each other when the aggregates are monotone or antimonotone and without negation, while there are differences in the other cases.\(^4\)

The paper is structured as follows. We start, in the next section, with the new definition of a stable model for propositional theories, their properties and comparisons with previous definitions of stable models and equilibrium logic. In Section 3 we present our aggregates, their properties and the comparisons with other definitions of aggregates. The conclusions are in Section 4, and the proofs of our theorems are in Section 5.

Preliminary reports on some results of this paper were published in [Ferraris 2005].

2. STABLE MODELS OF PROPOSITIONAL THEORIES

2.1 Definition

Usually, in logic programming, variables are allowed. As in most definitions of a stable model (see, for instance, [Gelfond and Lifschitz 1988]), we assume that the variables have been replaced by constants in a process called “grounding”. Even if, strictly speaking, atoms can have a more general form, we consider them as propositional as the definitions of a stable model don’t depend on the form of the atoms. The name “propositional” associated to formulas and theories also stresses the arbitrary nesting of propositional connectives, and the fact that aggregates — introduced in Section 3 — are not allowed by this syntax.

(Propositional) formulas are built from atoms and the 0-place connective \(\perp\) (false), using the connectives \(\land\), \(\lor\) and \(\rightarrow\). Even if our definition of a stable model below applies to formulas with any set of propositional connectives, we will consider \(\top\) as an abbreviation for \(\perp \rightarrow \perp\), a formula \(\neg F\) as an abbreviation for \(F \rightarrow \perp\) and \(F \leftrightarrow G\) as an abbreviation for \((F \rightarrow G) \land (G \rightarrow F)\). This will keep notation for other sections simpler. We will see later in this section why we chose such abbreviations and not others.

A (propositional) theory is a set of formulas. As usual in logic programming, truth assignments will be viewed as sets of atoms; we will write \(X \models F\) to express that a set \(X\) of atoms satisfies a formula \(F\), and similarly for theories.

An implication \(F \rightarrow G\) can be also written as a “rule” \(G \leftarrow F\), so that traditional programs, disjunctive programs and programs with nested expressions (reviewed in Section 2.2) can be seen as special cases of propositional theories.\(^5\)

We will now define when a set \(X\) of atoms is a stable model of a propositional theory \(\Gamma\). For the rest of the section \(X\) denotes a set of atoms.

\(^4\)The important role of monotonicity in aggregates has already been shown, for instance, in [Faber et al. 2004].

\(^5\)Traditionally, conjunction is represented in a logic program by a comma, disjunction by a semi-colon, and negation as failure as \textit{not}.
The reduct \( F^X \) of a propositional formula \( F \) relative to \( X \) is obtained from \( F \) by replacing each maximal subformula not satisfied by \( X \) with \( \bot \). That is, recursively,

\[
- \bot^X = \bot;
- \text{for every atom } a, \text{ if } X \models a \text{ then } a^X \text{ is } a; \text{ otherwise it is } \bot; \text{ and}
- \text{for every formulas } F \text{ and } G \text{ and any binary connective } \odot, \text{ if } X \models F \odot G \text{ then } (F \odot G)^X \text{ is } F^X \odot G^X, \text{ otherwise it is } \bot.
\]

This definition of reduct is similar to a transformation proposed in [Osorio et al. 2005, Section 4.2].

For instance, if \( X \) contains \( p \) but not \( q \) then

\[
\begin{align*}
(p \leftarrow \neg q)^X &= (p \leftarrow (q \rightarrow \bot))^X = p \leftarrow (\bot \rightarrow \bot) = p \leftarrow \top \\
(q \leftarrow \neg p)^X &= (q \leftarrow (p \rightarrow \bot))^X = \bot \leftarrow \bot \\
((p \rightarrow q) \lor (q \rightarrow p))^X &= \bot \lor (\bot \rightarrow p)
\end{align*}
\]

\( (2) \)

The reduct \( \Gamma^X \) of a propositional theory \( \Gamma \) relative to \( X \) is \( \{ F^X : F \in \Gamma \} \). A set \( X \) of atoms is a stable model of \( \Gamma \) if \( X \) is a minimal set satisfying \( \Gamma^X \).

For instance, let \( \Gamma \) be the theory consisting of

\[
\begin{align*}
p &\leftarrow \neg q \\
q &\leftarrow \neg p
\end{align*}
\]

\( (3) \)

Theory \( \Gamma \) is actually a traditional program, a logic program in the sense of [Gelfond and Lifschitz 1988] (more details in the next section). Set \( \{ p \} \) is a stable model of \( \Gamma \); indeed, by looking at the first two lines of (2) we can see that \( \Gamma^\{p\} \) is \( \{ p \leftarrow \top, \bot \leftarrow \bot \} \), which is satisfied by \( \{ p \} \) but not by its unique proper subset \( \emptyset \). It is easy to verify that \( \{ q \} \) is the only other stable model of \( \Gamma \). Similarly, it is not difficult to see that \( \{ p \} \) is the only stable model of the theory

\[
(p \rightarrow q) \lor (q \rightarrow p)
\]

\( (4) \)

(\( p \) is the reduct relative to \( \{ p \} \) is \( \{ \bot \lor (\bot \rightarrow p), p \} \)).

Recall that we have decided to treat \( \top, \neg F \) and \( F \leftrightarrow G \) as abbreviations for \( \bot \rightarrow \bot, F \rightarrow \bot \) and \( (F \rightarrow G) \land (G \rightarrow F) \) respectively. Such abbreviations clearly capture the meaning of the three connectives \( \top, \neg \) and \( \leftrightarrow \) in classical logic, but also in terms of the reduct. Indeed, it is not hard to see that the following three clauses for a reduct

\[
\begin{align*}
\top^X &= \top \\
(\neg F)^X &= \begin{cases} 
\neg (F^X), & \text{if } X \models \neg F, \\
\bot, & \text{otherwise.}
\end{cases}
\end{align*}
\]

\( (5) \)

hold independently if we consider the three connectives as primitive or we use the abbreviations above. As a consequence stable models of a theory are the same in both cases.
### Table 3. Syntax of “propositional” logic programs. Each $a, a_1, \ldots, a_m$ ($m \geq 0$) denotes an atom, and each $l_1, \ldots, l_n$ ($n \geq 0$) a literal — an atom possibly prefixed by $\neg$. A nested expression is any propositional formula that contains no implications other than negations or $\top$.

<table>
<thead>
<tr>
<th>kind of rule</th>
<th>syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>traditional</td>
<td>$a \leftarrow l_1 \wedge \cdots \wedge l_n$</td>
</tr>
<tr>
<td>disjunctive</td>
<td>$a_1 \lor \cdots \lor a_m \leftarrow l_1 \wedge \cdots \wedge l_n$</td>
</tr>
<tr>
<td>with nested expressions</td>
<td>$F \leftarrow G$ ($F$ and $G$ are nested expressions)</td>
</tr>
</tbody>
</table>

Fig. 3. Syntax of “propositional” logic programs. Each $a, a_1, \ldots, a_m$ ($m \geq 0$) denotes an atom, and each $l_1, \ldots, l_n$ ($n \geq 0$) a literal — an atom possibly prefixed by $\neg$. A nested expression is any propositional formula that contains no implications other than negations or $\top$.

On the other hand, if we consider $F \leftrightarrow G$ as an abbreviation for $(F \land G) \lor (\neg F \land \neg G)$ then (5) doesn’t hold, and stable models could be different. Indeed, in that case $(p \leftrightarrow q)^{[p,q]}$ would be $(p \land q) \lor \bot$, which is not even classically equivalent to $p \leftrightarrow q$, and $\{p,q\}$ would be a stable model of $p \leftrightarrow q$ while it is not if we treat $\leftrightarrow$ as a primitive connective or we use the other abbreviation. This shows a general principle about the stable model semantics, that classically equivalent transformations may change the stable models of a theory. We will see several other examples of such “unallowed” transformations later in Section 2.3.

Finally, a note about a second kind of negation in propositional theories, used in answer set programming for knowledge representation. In [Ferraris and Lifschitz 2005a, Section 3.9], atoms were divided into two groups: “positive” and “negative”, so that each negative atom has the form $\neg a$, where $a$ is a positive atom. Symbol $\sim$ is called “strong negation”, to distinguish it from the connective $\neg$, which is called negation as failure.\(^6\) A stable model of a theory with this second kind of negations must satisfy an additional condition: it cannot contain both a positive atom $a$ and its “negation” $\sim a$. This condition can be expressed by simply adding a formula

$$\neg(a \land \sim a)$$ (6)

to a theory for each negative atom $\sim a$ occurring in the theory. To keep notation simple for the rest of the paper we will consider positive atoms only.

### 2.2 Relationship with previous definitions of a stable model

As mentioned in the introduction, a theory is the extension of traditional programs [Gelfond and Lifschitz 1988], disjunctive programs [Gelfond and Lifschitz 1991] and programs with nested expressions [Lifschitz et al. 1999] (see Figure 2). We want to compare the definition of a stable model from the previous section with the definitions in the three papers cited above.

The syntax of a traditional rule, disjunctive rule and rule with nested expressions are shown in Figure 3. We understand an empty conjunction as $\top$ and an empty disjunction as $\bot$, so that traditional and disjunctive rules are also rules with nested expressions. The part before and after the arrow $\leftarrow$ are called the head and the body of the rule, respectively. When the body is empty (or $\top$), we can denote the whole rule by its head. A logic program is a set of rules. If all rules in a logic program are traditional then we say that the program is traditional too, and similarly for the other two kinds of rules.

\(^6\)Strong negation was introduced in the syntax of logic programs in [Gelfond and Lifschitz 1991]. In that paper, it was called “classical negation” and treated not as a part of an atom, but rather as a logical operator.
For instance, (3) is a traditional program as well as a disjunctive program and a program with nested expressions. On the other hand, (4) is not a logic program of any of those kinds, because of the first formula that contains implications nested in a disjunction.

For all kinds of programs described above, the definition of a stable model is similar to ours for propositional theories: to check whether a set \(X\) of atoms is a stable model of a program \(\Pi\), we (i) compute the reduct of \(\Pi\) relative to \(X\), and (ii) verify if \(X\) is a minimal model of such reduct. On the other hand, the way in which the reduct is computed is different. We consider the definition from [Lifschitz et al. 1999], as the definitions from [Gelfond and Lifschitz 1988, [1991]] are essentially its special cases.

The reduct \(\Pi^X\) of a program \(\Pi\) with nested expressions relative to a set \(X\) of atoms is the result of replacing, in each rule of \(\Pi\), each maximal subformula of the form \(\neg F\) with \(\top\) if \(X \models \neg F\), and with \(\bot\) otherwise. Set \(X\) is a stable model of \(\Pi\) if it is a minimal model of \(\Pi^X\).

For instance, if \(\Pi\) is (3) then the reduct \(\Pi^\{p\}\) is

\[
\begin{align*}
p &\leftarrow \top \\
q &\leftarrow \bot,
\end{align*}
\]

while \(\Pi^\emptyset\) is

\[
\begin{align*}
p &\leftarrow \top \\
q &\leftarrow \bot,
\end{align*}
\]

The stable models of \(\Pi\) — based on this definition of the reduct — are the same ones that we computed in the previous section using the newer definition of a reduct: \(\{p\}\) and \(\{q\}\). On the other hand, there are differences in the value of the reducts: for instance, we have just seen that \(\Pi^\emptyset\) is classically equivalent to \(\{p, q\}\), while \(\Pi^\emptyset = \{\bot, \bot\}\). However, some similarities between these definitions exist. For instance, negations are treated essentially in the same way: a nested expression \(\neg F\) is transformed into \(\bot\) if \(X \models \neg F\), and into \(\top\) otherwise, under both definitions of a reduct.

The following proposition states a more general relationship between the new definition and the 1999 definition of a reduct.

**Proposition 2.1.** For any program \(\Pi\) with nested expressions and any set \(X\) of atoms, \(\Pi^X\) is equivalent, in the sense of classical logic,

\(\neg \models \bot\), if \(X \not\models \Pi\), and

\(-\to \models \bot\), if \(X \models \Pi\) by replacing all atoms that do not belong to \(X\) by \(\bot\), otherwise.

**Corollary 2.2.** Given two sets of atoms \(X\) and \(Y\) with \(Y \subseteq X\) and any program \(\Pi\) with nested expressions, \(Y \models \Pi^X\) iff \(X \models \Pi\) and \(Y \models \Pi^X\).  

---

7 We underline the set \(X\) in \(\Pi^X\) to distinguish this definition of a reduct from the one from the previous section.

From the corollary above, one of the main claims of this paper follows, that our definition of a stable model is an extension of the definition for programs with nested expressions.

**Proposition 2.3.** For any program \( \Pi \) with nested expressions, the collections of stable models of \( \Pi \) according to our definition and according to [Lifschitz et al. 1999] are identical.

2.3 Relationship with classical logic

Most of the following considerations about stable models are valid for other definitions of a stable model.

Even if the syntax is the same of classical logic, the stable model semantics is a nonmonotone language: indeed, stable models of a theory may arbitrarily change by adding new formulas to it, and not only decrease in its numbers. For instance, \( \{\neg p \rightarrow q\} \) has stable model \( \{q\} \), while \( \{\neg p \rightarrow q,p\} \) has stable model \( \{p\} \).

As the name suggests, a stable model of a propositional theory \( \Gamma \) is a model — in the sense of classical logic — of \( \Gamma \). Indeed, it follows from the easily verifiable fact that, for each set \( X \) of atoms, \( X \models \Gamma^X \iff X \models \Gamma \). The converse doesn’t hold: for instance, model \( \{p,q\} \) of (4) is not stable.

We saw at the end of Section 2.1 that formulas that are equivalent in classical logic may have different stable models. As another example, the disjunctive program

\[ p \lor q \leftarrow \top \quad (7) \]

has two stable models \( \{p\} \) and \( \{q\} \), while

\[ p \leftarrow \neg q \quad (8) \]

has stable model \( \{p\} \) only. The intuitive meaning of connectives in the stable model semantics is different from classical logic. For instance, implication can be understood as “justifies”, and negation (usually called “default negation” or “negation as failure”) as “there is no proof of”. Also disjunction expresses a condition somehow stronger than classical disjunction. For instance, (7) can be read as “there is a justification for either \( p \) or \( q \)”. On the other hand, (8) can be read as “there is a justification for \( p \) if there is no proof for \( q \)”. With such intuitive meaning the stable models of (7) and (8) makes sense.

Another pair of formulas that are equivalent in classical logic but not in the stable model semantics are \( p \) and \( \neg \neg p \). Indeed, their intuitive meanings are “\( p \) is justified” and “\( p \) cannot be false” respectively. When we write those two formulas as 1-rule programs

\[ p \leftarrow \top \]

and

\[ \bot \leftarrow \neg p \]

we have stable model \( \{p\} \) for the first program and no stable model for the second.

Finally, another useful construct in answer set programming has the form \( a \lor \neg a \), and is called a “choice formula”. It intuitively means “either \( a \) is justified or there is no proof for \( a \)”. As a difference from classical logic, formulas of this kind cannot be dropped or replaced by \( \top \). In [Ferraris 2007b] choice rules are introduced in the
body of rules in a translation from causal theories (another knowledge representation language presented in [McCain and Turner 1997]) into ASP programs. The most known usage of a formula \(a \lor \neg a\) is in the head of a rule, where it expresses that we are free to include \(a\) or not in our stable models. For instance, program

\[ p \lor \neg p \leftarrow \top \]

has two stable models \(\emptyset\) and \(\{p\}\). Choice formulas allow to express choice rules — a syntactical construct defined in the same paper that introduced weight constraints ([Niemelä and Simons 2000]) — in logic programs with nested expressions as explained in [Ferraris and Lifschitz 2005b].

Choice formulas can also be seen as a “bridge” between classical logic and the stable model semantics.

**Proposition 2.4.** Let \(\Gamma\) be a theory with signature \(\sigma\). A set \(X\) of atoms subset of \(\sigma\) is a model of \(\Gamma\) iff it is a stable model of \(\Gamma \cup \{a \lor \neg a : a \in \sigma\}\).

Proposition 2.7 below will give some characterizations of transformations of formulas that preserves stable models. Notice that classically equivalent transformations can be applied to the reduct of a theory, as the sets of atoms that are minimal don’t change.

### 2.4 Relationship with Equilibrium Logic

Equilibrium logic [Pearce 1997, 1999] is defined in terms of Kripke models in the logic of here-and-there, a logic intermediate between intuitionistic and classical logic.

The logic of here-and-there is a 3-valued logic, where an interpretation (called an HT-interpretation) is represented by a pair \((X, Y)\) of sets of atoms where \(X \subseteq Y\). Intuitively, atoms in \(X\) are considered “true”, atoms not in \(Y\) are considered “false”, and all other atoms (that belong to \(Y\) but not \(X\)) are “undefined”.

An HT-interpretation \((X, Y)\) satisfies a propositional formula \(F\) (symbolically, \((X, Y) \models F\)) based on the following recursive definition (\(a\) stands for an atom):

\[
\begin{align*}
-(X, Y) &\models a \text{ iff } a \in X, \\
-(X, Y) &\not\models \bot, \\
-(X, Y) &\models F \land G \text{ iff } (X, Y) \models F \text{ and } (X, Y) \models G, \\
-(X, Y) &\models F \lor G \text{ iff } (X, Y) \models F \text{ or } (X, Y) \models G, \\
-(X, Y) &\models F \rightarrow G \text{ iff } (X, Y) \models F \text{ implies } (X, Y) \models G, \text{ and } Y \text{ satisfies } F \rightarrow G \text{ in classical logic.}
\end{align*}
\]

An HT-interpretation \((X, Y)\) satisfies a theory if it satisfies all the elements of the theory. Two propositional formulas are equivalent in the logic of here-and-there if they are satisfied by the same HT-interpretations.

Equilibrium logic defines when an HT-interpretation \((X, X)\) is an equilibrium model of a theory \(\Gamma\). HT-interpretation \((X, X)\) is an equilibrium model of \(\Gamma\) if \((X, X) \models \Gamma\) and, for all proper subsets \(Z\) of \(X\), \((Z, X) \not\models \Gamma\).

A relationship between the concept of a model in the logic of here-and-there, and satisfaction of the reduct exists.
Proposition 2.5. For any propositional formula \( F \) and any HT-interpretation \((X, Y)\), \((X, Y) \models F \) iff \( X \models F^Y \).

Next proposition compares the concept of an equilibrium model with the new definition of a stable model.

Proposition 2.6. A set of atoms is a stable model of a theory \( \Gamma \) iff \((X, X)\) is an equilibrium model of \( \Gamma \).

This proposition offers another way of proving Proposition 2.3, as [Lifschitz et al. 2001] showed that the equilibrium models of a program with nested expressions are the stable models of the same program in the sense of [Lifschitz et al. 1999].

2.5 Properties of propositional theories

This section shows how several theorems about logic programs with nested expressions can be extended to propositional theories.

2.5.1 Strong equivalence. Two theories \( \Gamma_1 \) and \( \Gamma_2 \) are strongly equivalent if, for every theory \( \Gamma \), \( \Gamma_1 \cup \Gamma \) and \( \Gamma_2 \cup \Gamma \) have the same stable models.

Proposition 2.7. For any two theories \( \Gamma_1 \) and \( \Gamma_2 \), the following conditions are equivalent:

(i) \( \Gamma_1 \) is strongly equivalent to \( \Gamma_2 \),

(ii) \( \Gamma_1 \) is equivalent to \( \Gamma_2 \) in the logic of here-and-there, and

(iii) for each set \( X \) of atoms, \( \Gamma_1^X \) is equivalent to \( \Gamma_2^X \) in classical logic.

The equivalence between (i) and (ii) is essentially Lemma 4 from [Lifschitz et al. 2001] about equilibrium logic. The equivalence between (i) and (iii) is similar to Theorem 1 from [Turner 2003] about nested expressions, but simpler and more general. Notice that (iii) cannot be replaced by

(iii') for each set \( X \) of atoms, \( \Gamma_1^X \) is equivalent to \( \Gamma_2^X \) in classical logic,

not even when \( \Gamma_1 \) and \( \Gamma_2 \) are programs with nested expressions. Indeed, \( \{ p \leftarrow \neg p \} \) is strongly equivalent to \( \{ \bot \leftarrow \neg p \} \), but \( \{ p \leftarrow \neg p \}^\emptyset = \{ p \leftarrow \top \} \) is not classically equivalent to \( \{ \bot \leftarrow \neg p \}^\emptyset = \{ \bot \leftarrow \top \} \).

Replacing, in a theory \( \Gamma \), a (sub)formula \( F \) with a formula \( G \) is guaranteed to preserve strong equivalence iff \( F \) is strongly equivalent to \( G \). Indeed, strong equivalence between \( F \) and \( G \) is clearly a necessary condition: take \( \Gamma = \{ F \} \). It is also sufficient because — as in classical logic — replacements of formulas with equivalent formulas in the logic of here-and-there preserves equivalence in the same logic.

As the logic of here-and-there is intermediate between intuitionistic and classical logic, most classically equivalent transformations are strongly equivalent. This includes, for instance, distributivity, absorption and DeMorgan laws. Please refer to [Lifschitz et al. 2001] for more details. A list of some of the most-common equivalences in classical logic that are not strongly equivalent transformation is the following.

\[
F \vee \neg F \equiv \top \quad F \equiv \neg \neg F \quad F \vee G \equiv \neg F \rightarrow G \quad F \rightarrow G \equiv \neg G \rightarrow \neg F
\]
\[
F \leftrightarrow G \iff (F \land G) \lor (\neg F \land \neg G).
\]

Cabalar and Ferraris [[2007]] showed that any theory is strongly equivalent to a logic program with nested expressions. That is, a theory can be seen as a different way of writing a logic program. This shows that the concept of a stable model for theories is not too different from the concept of a stable model for a logic program.

2.5.2 Other properties. To state several propositions below, we need the following definitions. Recall that an expression of the form \(\neg F\) is an abbreviation for \(F \rightarrow \bot\), and equivalences are the conjunction of two opposite implications. An occurrence of an atom in a propositional formula is positive if it is in the antecedent of an even number of implications. An occurrence is strictly positive if such number is 0, and negative if it odd.\(^8\) For instance, in a formula \((p \rightarrow r) \rightarrow q\), the occurrences of \(p\) and \(q\) are positive, the one of \(r\) is negative, and the one of \(q\) is strictly positive.

The following proposition is an extension of the property that in each stable model of a program, each atom occurs in the head of a rule of that program [Lifschitz 1996, Section 3.1]. An atom is an head atom of a theory \(\Gamma\) if it has a strictly positive occurrence in \(\Gamma\).\(^9\)

**Proposition 2.8.** Each stable model of a theory \(\Gamma\) consists of head atoms of \(\Gamma\).

A rule is called a constraint if its head is \(\bot\). In a logic program, adding constraints to a program \(\Pi\) removes the stable models of \(\Pi\) that don’t satisfy the constraints. A constraint can be seen as a formula of the form \(\neg F\), a formula that doesn’t have head atoms. Next proposition generalizes the property of logic programs stated above to theories.

**Proposition 2.9.** For every two theories \(\Gamma_1\) and \(\Gamma_2\) such that \(\Gamma_2\) has no head atoms, a set \(X\) of atoms is a stable model of \(\Gamma_1 \cup \Gamma_2\) iff \(X\) is a stable model of \(\Gamma_1\) and \(X \models \Gamma_2\).

The following two propositions are generalizations of propositions stated in [Ferraris and Lifschitz 2005b] in the case of logic programs. We say that an occurrence of an atom is in the scope of negation when it occurs in a propositional formula \(\neg F\).

**Proposition 2.10 Lemma on Explicit Definitions.** Let \(\Gamma\) be any theory, and \(Q\) a set of atoms not occurring in \(\Gamma\). For each \(q \in Q\), let \(\text{Def}(q)\) be a propositional formula that doesn’t contain any atoms from \(Q\). Then \(X \mapsto X \setminus Q\) is a 1–1 correspondence between the stable models of \(\Gamma \cup \{\text{Def}(q) \rightarrow q : q \in Q\}\) and the stable models of \(\Gamma\).

**Proposition 2.11 Completion Lemma.** Let \(\Gamma\) be any theory, and \(Q\) a set of atoms that have positive occurrences in \(\Gamma\) only in the scope of negation. For each \(q \in Q\), let \(\text{Def}(q)\) be a propositional formula such that all negative occurrences of atoms from \(Q\) in \(\text{Def}(q)\) are in the scope of negation. Then \(\Gamma \cup \{\text{Def}(q) \rightarrow q : q \in Q\}\) and \(\Gamma \cup \{\text{Def}(q) \leftrightarrow q : q \in Q\}\) have the same stable models.

---

\(^8\)The concept of a positive and negative occurrence of an atom should not be confused by the concept of a “positive” and “negative” atom mentioned at the end of Section 2.1.

\(^9\)In case of programs with nested expressions, it is easy to check that head atoms are atoms that occur in the head of a rule outside the scope of negation \(\neg\).

The following proposition is essentially a generalization of the splitting set theorem from [Lifschitz and Turner 1994] and [Erdoğan and Lifschitz 2004], which allows to break logic programs/theories into parts and compute the stable models separately. A formulation of this theorem has also been stated in [Ferraris and Lifschitz 2005a] in the special case of theories consisting of a single formula.

**Proposition 2.12 Splitting Set Theorem.** Let $\Gamma_1$ and $\Gamma_2$ be two theories such that no atom occurring in $\Gamma_1$ is a head atom of $\Gamma_2$. Let $S$ be a set of atoms containing all head atoms of $\Gamma_1$ but no head atoms of $\Gamma_2$. A set $X$ of atoms is a stable model of $\Gamma_1 \cup \Gamma_2$ iff $X \cap S$ is a stable model of $\Gamma_1$ and $X$ is a stable model of $(X \cap S) \cup \Gamma_2$.

### 2.6 Computational complexity

Since the concept of a stable model is equivalent to the concept of an equilibrium model, checking the existence of a stable model of a theory is a $\Sigma^P_2$-complete problem as for equilibrium models [Pearce et al. 2001]. Notice that the existence of a stable model of a disjunctive program is already $\Sigma^P_2$-hard [Eiter and Gottlob 1993, Corollary 3.8].

The existence of a stable model for a traditional program is a NP-complete problem [Marek and Truszczyński 1991]. The same holds, more generally, for logic programs with nested expressions where the head of each rule is an atom or $\bot$. (We call programs of this kind nondisjunctive). We may wonder if the same property holds for arbitrary sets of formulas of the form $F \rightarrow a$ and $F \rightarrow \bot$. The answer is negative: the following lemma shows that as soon as we allow implications in formulas $F$ then we have the same expressivity — and then complexity — as disjunctive rules.

**Lemma 2.13.** Rule

$$l_1 \land \cdots \land l_m \rightarrow a_1 \lor \cdots \lor a_n$$

$(n > 0, m \geq 0)$ where $a_1, \ldots, a_n$ are atoms and $l_1, \ldots, l_m$ are literals, is strongly equivalent to the set of $n$ implications $(i = 1, \ldots, n)$

$$(l_1 \land \cdots \land l_m \land (a_1 \rightarrow a_i) \land \cdots \land (a_n \rightarrow a_i)) \rightarrow a_i.$$  \hfill (9)

**Proposition 2.14.** The problem of the existence of a stable model of a theory consisting of propositional formulas of the form $F \rightarrow a$ and $F \rightarrow \bot$ is $\Sigma^P_2$-complete.

We will see, in Section 3.5, that the conjunctive terms in the antecedent of (9) can equivalently be replaced by aggregates of a simple kind, thus showing that allowing aggregates in nondisjunctive programs increases their computational complexity.

### 3. AGGREGATES

#### 3.1 Syntax and semantics

A formula with aggregates is defined recursively as follows:

— atoms and $\bot$ are formulas with aggregates$^{10}$,

---

$^{10}$Recall that $\top$ is an abbreviation for $\bot \rightarrow \bot$

—propositional combinations of formulas with aggregates are formulas with aggregates, and
—any expression of the form
\[ \text{op}\langle\{F_1 = w_1, \ldots, F_n = w_n\}\rangle \prec N \] (10)
where
—op is (a symbol for) a function from multisets of real numbers to \( \mathbb{R} \cup \{-\infty, +\infty\} \) (such as sum, product, min, max, etc.),
—\( F_1, \ldots, F_n \) are formulas with aggregates, and \( w_1, \ldots, w_n \) are (symbols for) real numbers ("weights"),
—\( \prec \) is (a symbol for) a binary relation between real numbers, such as \( \leq \) and =, and
—\( N \) is (a symbol for) a real number,
is a formula with aggregates.

A theory with aggregates is a set of formulas with aggregates. A formula of the form (10) is called an aggregate.

The intuitive meaning of an aggregate is explained by the following clause, which extends the definition of satisfaction of propositional formulas to arbitrary formulas with aggregates. For any aggregate (10) and any set \( X \) of atoms, let \( W_X \) be the multiset \( W \) consisting of the weights \( w_i \) (1 \( \leq \) i \( \leq \) n) such that \( X \models F_i \); we say that \( X \) satisfies (10) if \( \text{op}(W_X) \prec N \). For instance,
\[ \text{sum}\langle\{p = 1, q = 1\}\rangle \neq 1 \] (11)
is satisfied by the sets of atoms that satisfy both \( p \) and \( q \) or none of them.

As usual, we say that \( X \) satisfies a theory \( \Gamma \) with aggregates if \( X \) satisfies all formulas in \( \Gamma \). We extend the concept of classical equivalence to formulas/theories with aggregates.

We extend the definition of a stable models of propositional theories (Section 2) to cover aggregates, in a very natural way. Let \( X \) be a set of atoms. The reduct \( F^X \) of a formula \( F \) with aggregates relative to \( X \) is again the result of replacing each maximal subformula not satisfied by \( X \) with \( \bot \). That is, it is sufficient to add a clause relative to aggregates to the recursive definition of a reduct: for an aggregate \( A \) of the form (10),
\[ A^X = \begin{cases} 
\text{op}\langle\{F_1^X = w_1, \ldots, F_n^X = w_n\}\rangle \prec N, & \text{if } X \models A, \\
\bot, & \text{otherwise}. 
\end{cases} \]

This is similar to the clause for binary connectives:
\[ (F \otimes G)^X = \begin{cases} 
F^X \otimes G^X, & \text{if } X \models F \otimes G, \\
\bot, & \text{otherwise}. 
\end{cases} \]

The rest of the definition of a stable model remains the same: the reduct \( \Gamma^X \) of a theory \( \Gamma \) with aggregates is \( \{F^X : F \in \Gamma\} \), and \( X \) is a stable model of \( \Gamma \) if \( X \) is a minimal model of \( \Gamma^X \).

Consider, for instance, the theory \( \Gamma \) consisting of one formula
\[ \text{sum}\langle\{p = -1, q = 1\}\rangle \geq 0 \rightarrow q. \] (12)
Set \( \{q\} \) is a stable model of \( \Gamma \). Indeed, since both the antecedent and consequent of (12) are satisfied by \( \{q\} \), \( \Gamma^{\{q\}} \) is
\[
\text{sum}(\{\bot = -1, q = 1\}) \geq 0 \rightarrow q.
\]
The antecedent of the implication above is satisfied by every set of atoms, so the whole formula is equivalent to \( q \). Consequently, \( \{q\} \) is the minimal model of \( \Gamma^{\{q\}} \), and then a stable model of \( \Gamma \).

3.2 Aggregates as propositional formulas

A formula/theory with aggregates can also be seen as a propositional formula/theory by identifying (10) with the formula
\[
\bigwedge_{I \subseteq \{1, \ldots, n\} \setminus \{\text{op}(\{w_i: i \in I\}) \neq N\}} ((\bigwedge_{i \in I} F_i) \rightarrow (\bigvee_{i \in \mathcal{T}} F_i)),
\]
where \( \mathcal{T} \) stands for \( \{1, \ldots, n\} \setminus \) I, and \( \neq \) is the negation of \( \leq \).

For instance, if we consider aggregate (11), the conjunctive terms in (13) correspond to the cases when the sum of weights is 1, that is, when \( I = \{1\} \) and \( I = \{2\} \). The two implications are \( q \rightarrow p \) and \( p \rightarrow q \) respectively, so that (11) is
\[
(q \rightarrow p) \land (p \rightarrow q).
\]
Similarly,
\[
\text{sum}(\{p = 1, q = 1\}) = 1
\]
is
\[
(\top \rightarrow (p \lor q)) \land ((p \land q) \rightarrow \bot),
\]
which is strongly equivalent to
\[
(p \lor q) \land \neg(p \land q).
\]

Even though (15) can be seen as the negation of (11), the negation of (16) is not strongly equivalent to (14) (although they are classically equivalent). This shows that it is generally incorrect to “move” a negation from a binary relation symbol (such as \( \neq \)) in front of the aggregate as the unary connective \( \neg \), and vice versa.

Next proposition shows that this understanding of aggregates as propositional formulas is equivalent to the semantics for theories with aggregates of the previous section. Two formulas with aggregates are classically equivalent to each other if they are satisfied by the same sets of atoms.

**Proposition 3.1.** Let \( A \) be an aggregate of the form (10) and let \( G \) be the corresponding formula (13). Then
(a) \( G \) is classically equivalent to \( A \), and
(b) for any set \( X \) of atoms, \( G^X \) is classically equivalent to \( A^X \).

Treating aggregates as propositional formulas allows us to apply many properties of propositional theories presented in Section 2.5 to theories with aggregates also. Consequently, we have the concept of an head atom, of strong equivalence, we can use the completion lemma and so on. We will use several of those properties
to prove Proposition 3.3 below. In the rest of the paper we will often make no
distinctions between the two ways of defining the semantics of aggregates discussed
here.

Notice that replacing, in a theory, an aggregate of the form (10) with a for-
mula that is not strongly equivalent to the corresponding formula (13) may lead
to different stable models. This shows that there is no other way (modulo strong
equivalence) of representing our aggregates as propositional formulas.

3.3 Monotone Aggregates

An aggregate $\text{op}\langle\{F_1 = w_1, \ldots, F_n = w_n\}\rangle \prec N$ is monotone if, for each pair
of multisets $W_1, W_2$ such that $W_1 \subseteq W_2 \subseteq \{w_1, \ldots, w_n\}$, $\text{op}(W_2) \prec N$ is true
whenever $\text{op}(W_1) \prec N$ is true. The definition of an antimonotone aggregate is
similar, with $W_1 \subseteq W_2$ replaced by $W_2 \subseteq W_1$.

For instance, $\text{sum}\langle\{p = 1, q = 1\}\rangle > 1$ (17)
is monotone, and

$\text{sum}\langle\{p = 1, q = 1\}\rangle < 1.$ (18)
is antimonotone. An example of an aggregate that is neither monotone nor anti-
monotone is $(11)$.

PROPOSITION 3.2. For any aggregate $\text{op}\langle\{F_1 = w_1, \ldots, F_n = w_n\}\rangle \prec N$, formula (13) is strongly equivalent to

$$\bigwedge_{I \subseteq \{1, \ldots, n\}} \bigvee_{i \in I} (F_i)$$ (19)

if the aggregate is monotone, and to

$$\bigwedge_{I \subseteq \{1, \ldots, n\}} \neg \bigwedge_{i \in I} F_i$$ (20)

if the aggregate is antimonotone.

In other words, if $\text{op}(S) \prec N$ is monotone then the antecedents of the implications
in (13) can be dropped. Similarly, in case of antimonotone aggregates, the conse-
quents of these implications can be replaced by $\bot$. In both cases, (13) is turned
into a nested expression, if $F_1, \ldots, F_n$ are nested expressions.

For instance, aggregate (17) is normally written as formula

$$(p \lor q) \land (p \rightarrow q) \land (q \rightarrow p).$$

Since the aggregate is monotone, it can also be written, by Proposition 3.2, as
nested expression

$$(p \lor q) \land q \land p,$$

which is strongly equivalent to $q \land p$. Similarly, aggregate (18) is normally written
as formula

$$((p \land q) \rightarrow \bot) \land (p \rightarrow q) \land (q \rightarrow p);$$
since the aggregate is nonmonotone, it can also be written as nested expression

\[-(p \land q) \land \neg p \land \neg q,

which is strongly equivalent to \(\neg p \land \neg q\).

On the other hand, if an aggregate is neither monotone nor antimonotone, it may be not possible to find a nested expression strongly equivalent to (13), even if \(F_1, \ldots, F_n\) are nested expressions. This is the case for (11). Indeed, the formula (13) corresponding to (11) is (14), whose reduct relative to \(\{p, q\}\) is (14). Consequently, by Proposition 2.7, for any formula \(G\) strongly equivalent to (14), \(G^{\{p,q\}}\) is classically equivalent to (14). On the other hand, the reduct of nested expressions are essentially AND-OR combinations of atoms, \(\top\) and \(\bot\) (negations either become \(\bot\) or \(\top\) in the reduct), and no formula of this kind is classically equivalent to (14).

In some uses of ASP, aggregates that are neither monotone nor antimonotone are essential, as discussed in the next section.

### 3.4 Example

We consider the following variation of the combinatorial auction problem [Baral and Uyan 2001], which can be naturally formalized using an aggregate that is neither monotone nor antimonotone.

Joe wants to move to another town and has the problem of removing all his bulky furniture from his old place. He has received some bids: each bid may be for one piece or several pieces of furniture, and the amount offered can be negative (if the value of the pieces is lower than the cost of removing them). A junkyard will take any object not sold to bidders, for a price. The goal is to find a collection of bids for which Joe doesn’t lose money, if there is any.

Assume that there are \(n\) bids, denoted by atoms \(b_1, \ldots, b_n\). We express by the formulas

\[b_i \lor \neg b_i\]  \hspace{1cm} (21)

(1 \leq i \leq n) that Joe is free to accept any bid or not. Clearly, Joe cannot accept two bids that involve the selling of the same piece of furniture. So, for every such pair \(i, j\) of bids, we include the formula

\[-(b_i \land b_j)\]  \hspace{1cm} (22)

Next, we need to express which pieces of the furniture have not been given to bidders. If there are \(m\) objects we can express that an object \(i\) is sold by bid \(j\) by adding the rule

\[b_j \rightarrow s_i\]  \hspace{1cm} (23)

to our theory.

Finally, we need to express that Joe doesn’t lose money by selling his items. This is done by the aggregate

\[\text{sum}(\{b_1 = w_1, \ldots, b_n = w_n, -s_1 = -c_1, \ldots, -s_m = -c_m\}) \geq 0,\]  \hspace{1cm} (24)

where each \(w_i\) is the amount of money (possibly negative) obtained by accepting bid \(i\), and each \(c_i\) is the money requested by the junkyard to remove item \(i\). Note that (24) is neither monotone nor antimonotone.

We define a solution to Joe’s problem as a set of accepted bids such that
(a) the bids involve selling disjoint sets of items, and
(b) the sum of the money earned from the bids is greater than the money spent
giving away the remaining items.

Proposition 3.3. The mapping from a set \( X \) of atoms to the set \( \{ i : b_i \in X \} \) is a 1–1 correspondence between the stable models of the theory consisting of
formulas (21)–(24) and a solution of Joe’s problem.

3.5 Computational Complexity

Since theories with aggregates generalize disjunctive programs, the problem of the
existence of a stable model of a theory with aggregates clearly is \( \Sigma^p_2 \)-hard.\(^{11}\) We
need to check in which class of the computational hierarchy this problem belongs.

Even if propositional formulas corresponding to aggregates can be exponentially
larger than the original aggregate, it turns out that (by treating aggregates as
primitive constructs) the computation is not harder than for propositional theories.

Proposition 3.4. If, for every aggregate, computing \( \text{op}(W) < N \) requires poly-
nomial time then the existence of a stable model of a theory with aggregates is a
\( \Sigma^p_2 \)-complete problem.

For a nondisjunctive program with nested expressions the existence of a stable
model is NP-complete. If we allow nonnested aggregates in the body, for instance
by allowing rules

\[
A_1 \land \cdots \land A_n \rightarrow a
\]

\((A_1, \ldots, A_n \text{ are aggregates and } a \text{ is an atom or } \bot)\) then the complexity increases
to \( \Sigma^p_2 \). This follows from Lemma 2.13, since, in (9), each formula \( l_i \) is the “proposi-
tional” representation of \( \text{sum}(\{ l_i = 1 \}) \geq 1 \); similarly, each \( a_j \rightarrow a_i \) is the
“propositional” representation of \( \text{sum}(\{ a_j = -1, a_i = 1 \}) \geq 0 \).

However, if we allow monotone and antimonotone aggregates only — even nested
— in the antecedent, we are in class NP.

Proposition 3.5. Consider theories with aggregates consisting of formulas of
the form

\[
F \rightarrow a,
\]

where \( a \) is an atom or \( \bot \), and \( F \) contains monotone and antimonotone aggregates
only, no equivalences and no implications other than negations. If, for every ag-
gregate, computing \( \text{op}(W) < N \) requires polynomial time then the problem of the
existence of a stable model of theories of this kind is an NP-complete problem.

Similar results have been independently proven in [Calimeri et al. 2005] for FLP-
aggregates.

3.6 Other Formalisms

Figure 4 already shows that there are several differences between the various defi-
nitions of an aggregate. We analyze that more in details in the rest of this section.

\(^{11}\)We are clearly assuming weight not to be arbitrary real numbers but to belong to a countable
subset of real numbers, such as integers or floating point numbers.
Fig. 4. Properties of definitions of programs with aggregates, in the case in which the head of each rule is an atom. We limit the syntax of our aggregates to the syntax allowed by the other formalisms. The complexity is relative to the problem of the existence of a stable model. The minimality property holds when stable models of a theory/program is a minimal model of the theory/program.

3.6.1 Programs with weight constraints. Weight constraints are aggregates defined in [Niemelä and Simons 2000] and implemented in answer set solver smodels. We simplify the syntax of weight constraints and of programs with weight constraints for clarity, without reducing its semantical expressivity.

Weight constraints are expressions of the form
\[ N \leq \{ l_1 = w_1, \ldots, l_m = w_m \} \] (25)
and
\[ \{ l_1 = w_1, \ldots, l_m = w_m \} \leq N \] (26)
where
- \( N \) is (a symbol for) a real number,
- each of \( l_1, \ldots, l_n \) is a (symbol for) a literal, and \( w_1, \ldots, w_n \) are (symbols for) real numbers.

An example of a weight constraint is (1).

The intuitive meaning of (25) is that the sum of the weights \( w_i \) for all the \( l_i \) that are true is not lower than \( N \). For (26) the sum of weights is not greater than \( N \).

Often, \( N_1 \leq S \leq N_2 \) are written together as \( N_1 \leq S \leq N_2 \). If a weight \( w \) is 1 then the part “\( = w \)” is generally omitted. If all weights are 1 then a weight constraint is called a cardinality constraint.

A rule with weight constraints is an expression of the form
\[ a \leftarrow C_1 \land \cdots \land C_n \] (27)
where \( a \) is an atom or \( \bot \), and \( C_1, \ldots, C_n \) \((n \geq 0)\) are weight constraints.

Finally, a program with weight constraints is a set of rules with weight constraints. Rules/programs with cardinality constraints are rules/programs with weight constraints containing cardinality constraints only.

Programs with cardinality/weight constraints can be seen as a generalization of traditional programs, by identifying each literal \( l \) in the body of each rule with cardinality constraint \( 1 \leq \{ l \} \).

The definition of a stable model from [Niemelä and Simons 2000] requires first the elimination of negative weights from weight constraints. This is done by replacing each term \( l_i = w_i \) where \( w_i \) is negative with \( \overline{l_i} = -w_i \) (\( \overline{l_i} \) is the literal complementary to \( l_i \)) and increasing the bound by \(-w_i\). For instance,
\[ 0 \leq \{ p = 2, q = -1 \} \]
is rewritten as

$$1 \leq \{p = 2, \neg q = 1\}.$$ \hfill (28)

Then [Niemelä and Simons 2000] proposes a definition of a reduct and of a stable model for programs with weight constraints without negative weights. For this paper, we prefer showing a translational, equivalent semantics of such programs from [Ferraris and Lifschitz 2005b], that consists in replacing each weight constraint $C$ with a nested expression $[C]$, preserving the stable models of the program: if $C$ is (25) then $[C]$ is ($I \subseteq \{1, \ldots, n\}$)

$$\bigvee_{I : N \leq \sum_{i \in I} w_i} (\bigwedge_{i \in I} l_i)$$ \hfill (28)

and if $C$ is (26) then $[C]$ is

$$\neg \bigvee_{I : N < \sum_{i \in I} w_i} (\bigwedge_{i \in I} l_i).$$ \hfill (29)

It turns out that the way of understanding a weight constraint $C$ of this paper is not different from $[C]$ when all weights are nonnegative.

**Proposition 3.6.** In presence of nonnegative weights only, $[N \leq S]$ is strongly equivalent to $\text{sum}(S) \geq N$, and $[S \leq N]$ is strongly equivalent to $\text{sum}(S) \leq N$.

From this proposition, Propositions 2.3 and 2.7 of this paper, and Theorem 1 from [Ferraris and Lifschitz 2005b] it follows that our concept of an aggregate captures the concept of weight constraints defined in [Niemelä and Simons 2000] when all weights are nonnegative. It also captures the absence of the minimality property of its stable models: for instance,

$$p \leftarrow \{\neg p\} \leq 0$$

has stable models $\emptyset$ and $\{p\}$ in both formalisms.

When we consider negative weights, however, such correspondence doesn’t hold. For instance,

$$p \leftarrow 0 \leq \{p = 2, p = -1\},$$ \hfill (30)

according to [Niemelä and Simons 2000], has no stable models, while

$$p \leftarrow \text{sum}(\{p = 2, p = -1\}) \geq 0$$ \hfill (31)

has stable model $\{p\}$. An explanation of this difference can be seen in the preprocessing proposed by [Niemelä and Simons 2000] that eliminates negative weights. For us, weight constraint $0 \leq \{p = 2, p = -1\}$, and the result $1 \leq \{p = 2, \neg p = 1\}$ of eliminating its negative weight, are semantically different.\footnote{The fact that the process of eliminating negative weights is somehow unintuitive was already mentioned in [Ferraris and Lifschitz 2005b] with the same example proposed in this section.} Surprisingly, under the semantics of [Niemelä and Simons 2000], $0 \leq \{p = 2, p = -1\}$ is different from $0 \leq \{p = 1\}$. In fact,

$$p \leftarrow 0 \leq \{p = 1\}$$ \hfill (32)
has stable model \( \{ p \} \), the same of (31), while (30) has none. Notice that summing weights that are all positive or all negative preserves stable models under both semantics.

The preliminary step of removing negative weights can be seen as a way of making weight constraints either monotone or antimonotone. This keeps the problem of the existence of a stable model in class NP, while we have seen in Section 3.5 that, under our semantics, even simple aggregates with the same intuitive meaning of 
\[ 0 \leq \{ p = 1, q = -1 \} \] bring the same problem to class \( \Sigma^P_2 \).

3.6.2 PDB-aggregates. A PDB-aggregate is an expression of the form (10), where \( F_1, \ldots, F_n \) are literals. A program with PDB-aggregates is a set of rules of the form

\[ a \leftarrow A_1 \land \cdots \land A_m, \]

where \( m \geq 0, a \) is an atom and \( A_1, \ldots, A_m \) are PDB-aggregates.

As in the case of programs with weight constraints, a program with PDB-aggregates is a generalization of a traditional program, by identifying each literal \( l \) in the bodies of traditional programs by aggregate
\[ \text{sum}\{ l = 1 \} \geq 0. \]

A semantics for such aggregates was proposed in [Denecker et al. 2001], based on the approximation theory [Denecker et al. 2002]. But the first characterization of PDB-aggregates in terms of stable models is from [Pelov et al. 2003]. [Son et al. 2007] independently proposed a similar semantics.

\[ 13 \] [Pelov et al. 2003] doesn’t explicitly mention nested expressions.

\[ 14 \] [Pelov et al. 2003] doesn’t explicitly mention nested expressions.
It can be shown, using strong equivalent transformations (see Proposition 2.7) that the disjunction of such nested expressions can be rewritten as $\neg p \lor q$.

In case of monotone and antimonotone PDB-aggregates and in the absence of negation as failure, the semantics of Pelov et al. is equivalent to ours.

**Proposition 3.7.** For any monotone or antimonotone PDB-aggregates $A$ of the form (10) where $F_1, \ldots, F_n$ are atoms, $A_{tr}$ is strongly equivalent to (13).

The claim above is generally not true when either the aggregates are not monotone or antimonotone, or when some formula in the aggregate is a negative literal. Relatively to aggregates that are neither monotone nor antimonotone, the semantics of [Pelov et al. 2003] seems to have the same unintuitive behaviour of [Niemelä and Simons 2000]: for instance, according to [Pelov et al. 2003], (31) has no stable models while

$$p \leftarrow \text{sum}(\{p = 1\}) \geq 0$$

has stable model $\{p\}$.

To illustrate the problem with negative literals, consider the following II:

$$p \leftarrow \text{sum}(\{q = 1\}) < 1$$  \hspace{1cm} (33)

$$q \leftarrow \neg p$$

and II':

$$p \leftarrow \text{sum}(\{\neg p = 1\}) < 1$$  \hspace{1cm} (34)

$$q \leftarrow \neg p$$

Intuitively, the two programs should have the same stable models. Indeed, the operation of replacing $q$ with $\neg p$ in the first rule of II should not affect the stable models since the second rule “defines” $q$ as $\neg p$: it is the only rule with $q$ in the head. However, under the semantics of [Pelov et al. 2003], II has stable model $\{p\}$ only and II' has stable model $\{q\}$ also. Under our semantics, both (33) and (34) have stable models $\{p\}$ and $\{q\}$.

Note that already the first rule of (34) has different stable models under the two semantics. Under ours, they are $\emptyset$ and $\{p\}$. According to [Pelov et al. 2003], only the empty set is a stable model; it couldn’t have both stable models because stable models as defined in [Pelov et al. 2003] have the minimality property.

Here is another example — provided by [Pelov et al. 2007] — for which different semantics of aggregates produce different stable models.$^{15}$

$$r \leftarrow \text{sum}(\{p = 1, q = 1\}) \neq 1$$

$$p \leftarrow r$$

$$q \leftarrow r$$

$$p \leftarrow q$$

$$q \leftarrow p.$$

(35)

Set $\{p, q, r\}$ is a stable model of (35) under our semantics, but not under the semantics of PDB-aggregates. Here is an intuitive reason why it should be a stable

$^{15}$We present it here in our syntax and, in particular, grounded. We thank one of the reviewers for mentioning this example to us.

model: the last two rules make the aggregate in the body of the first rule trivially true, so that \( r \) is justified. It then follows that \( p \) and \( q \) are justified as well by second and third rule.

One of the main argumentations of why \( \{p, q, r\} \) should not be a stable model of (35) is that the atoms \( p, q \) and \( r \) should not be recursively defined by the rules of the program. In our reasoning above this doesn’t happen: in fact we didn’t need to mention a rule more than once.

Let’s analyze the interaction between the first and the last two rules of (35) in a more formal way. The last two rules express that \( p \) and \( q \) are equivalent to each other. In essentially every semantics of stable models (including the one of PDB-aggregates) they allow to replace — in other rules — occurrences of \( q \) with \( p \) and viceversa preserving the stable models. In particular, we can rewrite the first rule of (35) as

\[
\begin{align*}
    r &\leftarrow \text{sum}^\ast(\{p = 1, p = 1\}) \neq 1
\end{align*}
\]

without modifying its stable models.

Now it is clear that the value of the aggregate above doesn’t depend on \( p \), so we would expect \( r \) to be justified by this rule. Our semantics follow this intuitive reasoning: in fact, the aggregate can be seen as formula \((p \rightarrow p) \land (p \rightarrow p)\) (strongly equivalent to \( \top \)), and this leads to stable model \( \{p, q, r\} \). According to [Pelov et al. 2003], however, aggregate \( \text{sum}^\ast(\{p = 1, p = 1\}) \neq 1 \) is not trivially true: it stands for nested expression \((p \land p) \lor (\neg p \land \neg p)\), which cannot be simplified into \( \top \) in the semantics of nested expressions.

Interestingly, if we rewrite aggregate \( \text{sum}^\ast(\{p = 1, p = 1\}) \neq 1 \) as \( \text{sum}^\ast(\{p = 2\}) \neq 1 \) then the modified program has stable model \( \{p, q, r\} \) under both semantics.

3.6.3 FLP-aggregates. An FLP-aggregate is an expression of the form (10) where each of \( F_1, \ldots, F_n \) is a conjunction of literals. A program with FLP-aggregates is a set of rules of the form

\[
\begin{align*}
    a_1 \lor \cdots \lor a_n &\leftarrow A_1 \land \cdots \land A_m \land \neg A_{m+1} \land \cdots \land \neg A_p
\end{align*}
\]

(36)

where \( n \geq 0, 0 \leq m \leq p, a_1, \ldots, a_n \) are atoms and \( A_1, \ldots, A_p \) are FLP-aggregates.

A program with FLP-aggregates is a generalization of a disjunctive program, by identifying each atom \( a \) in the bodies of disjunctive rules by aggregate \( \text{sum}^\ast(\{a = 1\}) \geq 1 \).

The semantics of [Faber et al. 2004] defines when a set of atoms is a stable model for a program with FLP-aggregates. The definition of satisfaction of an aggregate is identical to ours. The reduct, however, is computed differently. The reduct \( \Pi^X \) of a program \( \Pi \) with FLP-aggregates relative to a set \( X \) of atoms consists of the rules of the form (36) such that \( X \) satisfies its body. Set \( X \) is a stable model for \( \Pi \) if \( X \) is a minimal set satisfying \( \Pi^X \).

For instance, let \( \Pi \) be the FLP-program

\[
\begin{align*}
    p &\leftarrow \text{sum}^\ast(\{p = 2\}) \geq 1.
\end{align*}
\]

The only stable model of \( \Pi \) is the empty set. Indeed, since the empty set doesn’t satisfy the aggregate, \( \Pi^\emptyset = \emptyset \), which has \( \emptyset \) as the unique minimal model; we can conclude that \( \emptyset \) is a stable model of \( \Pi \). On the other hand, \( \Pi^{\{p\}} = \Pi \) because \( \{p\} \)
Paolo Ferraris

satisfies the aggregate in \( \Pi \). Since \( \emptyset \models \Pi \), \( \{p\} \) is not a minimal model of \( \Pi^{(p)} \) and then it is not a stable model of \( \Pi \).

This definition of a reduct is different from all other definitions of a reduct described in this paper (and also from many other definitions), in the sense that it may leave negation \( \neg \) in the body of a rule. For instance, the reduct of \( a \leftarrow \neg b \) relative to \( \{a\} \) is the fact \( a \) according to the other (more usual) definitions, whereas in the theory of FLP-aggregates, the reduct doesn’t modify the rule. On the other hand, this definition of a stable model is equivalent to the definition of a stable model in the sense of [Gelfond and Lifschitz 1991] (and successive definitions) when applied to disjunctive programs.

Next proposition shows a relationship between our concept of an aggregate and FLP-aggregates. An FLP-program is positive if, in each formula (36), \( p = m \).

**Proposition 3.8.** The stable models of a positive FLP-program under our semantics are identical to its stable models in the sense of [Faber et al. 2004].

The proposition doesn’t apply to arbitrary FLP-aggregates as negation has different meanings in the two semantics. In case of [Faber et al. 2004], \( \neg (op(S) \prec N) \) is essentially the same as \( op(S) \not\prec N \), while we have seen, in Section 3.2, that this fact doesn’t always hold in our semantics. The difference in meaning can be seen in the following example. Program

\[
\begin{align*}
p & \leftarrow \neg q \\
q & \leftarrow \text{sum} \langle \{p = 1\} \rangle \leq 0
\end{align*}
\]

has two stable models \( \{p\} \) and \( \{q\} \) according to both semantics. However, if we replace \( q \) in the first rule with the body of the second (\( q \) is “defined” as \( \text{sum} \langle \{p = 1\} \rangle \leq 0 \) by the second rule), we get program

\[
\begin{align*}
p & \leftarrow \neg (\text{sum} \langle \{p = 1\} \rangle \leq 0) \\
q & \leftarrow \text{sum} \langle \{p = 1\} \rangle \leq 0
\end{align*}
\]

which, according to [Faber et al. 2004], has only stable model \( \{q\} \). We find it unintuitive.

It is the first rule of (38) that has a different meaning in the two semantics. The rule alone has different stable models: according to [Faber et al. 2004], its only stable models is \( \emptyset \). Under our semantics, the stable models are \( \emptyset \) and \( \{p\} \). As they don’t have the minimality property, there is no program with FLP-aggregates that has such stable models under [Faber et al. 2004].

As any program with FLP-aggregates can be easily rewritten as a positive program with FLP-aggregates, our definition of an aggregate essentially generalizes the one of [Faber et al. 2004].

4. CONCLUSIONS

We have proposed a new definition of a stable model — for proposition theories — that is simple, very general, and that inherits several properties from logic programs with nested expressions. On top of that, we have defined the concept of an aggregate, both as an atomic operator and as an abbreviation for a formula. We should say that we chose our syntax of aggregates based on what is most used in

answer set programming, i.e. operations on numbers. The syntax and semantics of aggregates can be easily extended to allow arbitrary “truth table” relations (aggregates of this kind are sometimes called “abstract”) as for instance in [Son et al. 2007].

In the same way as we added aggregates to formulas in the stable models semantics, it is possible to extend equilibrium logic with aggregates, by adding a clause for satisfaction of aggregates in the logic of here-and-there. Unfortunately we don’t know how to extend derivation systems for the logic of here-and-there to aggregates. Many theorems of this paper about propositional theories with aggregates would be valid under the extended equilibrium logic.

In our semantics stable models don’t have the minimality property, which is considered by several researchers an important property of logic programs. The fact that for programs with FLP-aggregates — that can be mapped into our semantics — such property holds suggests that there is a subset of propositional theories that allows some nesting of implications for which the minimality condition holds. This is a subject for further studies.

A definition of loop formulas for finite propositional theories has been provided in [Ferraris et al. 2006], and extended to aggregates in [Lee and Meng 2009]. In this last paper, definitions of loop formulas for several semantics of aggregates are compared to each other. This represents another point of view for studying the differences between the various semantics of aggregates. The definition of loop formulas for our aggregates turns out to be among the simplest.

We consider propositional aggregates an interesting research topic as there is no agreement on the concept of a ground aggregate. Extending the syntax of first-order formula under the stable model semantics of [Ferraris et al. 2007] to allow aggregates is work in progress.

We hope that this very general framework may be useful in the heterogeneous world of aggregates in answer set programming.

5. PROOFS
5.1 Proof of Propositions 2.1 and 2.3

We first need the recursive definition of reduct for programs with nested expressions from [Lifschitz et al. 1999]. The reduct $F^X$ of a nested expression $F$ relative to a set $X$ of atoms, as follows:

\[\neg a^X = \bot, \quad \perp^X = \perp \quad \text{and} \quad \top^X = \top,\]

\[\neg (F \land G)^X = F^X \land G^X \quad \text{and} \quad (F \lor G)^X = F^X \lor G^X,\]

\[\neg \neg F^X = \begin{cases} \top, & \text{if } X \models F, \\ \bot, & \text{otherwise}, \end{cases}\]

Then the reduct $(F \leftarrow G)^X$ of a rule $F \leftarrow G$ with with nested expression is defined as $F^X \leftarrow G^X$, and the reduct $\Pi^X$ of a program with nested expressions as the union of the reduct of its rules.

Lemmas 5.1. For any formulas $F_1, \ldots, F_n$ ($n \geq 0$), any set $X$ of atoms, and any connective $\otimes \in \{\lor, \land\}$, $(F_1 \otimes \cdots \otimes F_n)^X$ is classically equivalent to $F_1^X \otimes \cdots \otimes F_n^X$. 

Proof. Case 1: \( X \models F_1 \land \cdots \land F_n \). Then, by the definition of reduct, \((F_1 \land \cdots \land F_n)^X = F_1^X \land \cdots \land F_n^X \). Case 2: \( X \not\models F_1 \land \cdots \land F_n \). Then \((F_1 \land \cdots \land F_n)^X = \bot \); moreover, one of \( F_1, \ldots, F_n \) is not satisfied by \( X \), so that one of \( F_1^X, \ldots, F_n^X \) is \( \bot \). The case of disjunction is similar.

**Lemma 5.2.** The reduct \( F^X \) of a nested expression \( F \) is equivalent, in the sense of classical logic, to the nested expression obtained from \( F^X \) by replacing all atoms that do not belong to \( X \) by \( \bot \).

**Proof.** The proof is by structural induction on \( F \).

—When \( F \) is \( \bot \) or \( \top \) then \( F^X = F = F^X \).

—For an atom \( a \), \( a^X = a \). The claim is immediate.

—Let \( F \) be a negation \( \neg G \). If \( X \models G \) then \( F^X = \bot = F^X \); otherwise, \( F^X = \neg \bot = \top = F^X \).

—For \( F = G \otimes H (\otimes \in \{\lor, \land\}) \), \( F^X \) is \( G^X \otimes H^X \), and, by Lemma 5.1, \( F^X \) is equivalent to \( G^X \otimes H^X \). The claim now follows by the induction hypothesis.

**Proposition 2.1.** For any program \( \Pi \) with nested expressions and any set \( X \) of atoms, \( \Pi^X \) is equivalent, in the sense of classical logic,

—to \( \bot \), if \( X \not\models \Pi \), and

—to the program obtained from \( \Pi^X \) by replacing all atoms that do not belong to \( X \) by \( \bot \), otherwise.

**Proof.** If \( X \not\models \Pi \) then clearly \( \Pi^X \) contains \( \bot \). Otherwise, \( \Pi^X \) consists of formulas \( F^X \rightarrow G^X \) for each rule \( G \leftarrow F \in \Pi \), and consequently for each rule \( G^X \leftarrow F^X \in \Pi^X \). Since each \( F \) and \( G \) is a nested expression, the claim is immediate by Lemma 5.2.

**Proposition 2.3.** For any program \( \Pi \) with nested expressions, the collection of stable models of \( \Pi \) according to our definition and according to [Lifschitz et al. 1999] are identical.

**Proof.** If \( X \not\models \Pi \) then clearly \( \Pi^X \) contains \( \bot \), and also \( X \not\models \Pi^X \) (a well-known property about programs with nested expressions), so \( X \) is not a stable model under either definitions. Otherwise, by Corollary 2.2, the two reducts are satisfied by the same subsets of \( X \). Then \( X \) is a minimal set satisfying \( \Pi^X \) iff it is a minimal set satisfying \( \Pi^X \), and, by the definitions of a stable models \( X \) is a stable model of \( \Pi \) either for both definitions or for none of them.

### 5.2 Proof of Proposition 2.4

**Lemma 5.3.** For any theory \( \Gamma \) and any set \( X \) of atoms, \( X \models \Gamma^X \) iff \( X \models \Gamma \).

**Proof.** Reduct \( \Gamma^X \) is obtained from \( \Gamma \) by replacing some subformulas that are not satisfied by \( X \) with \( \bot \).

**Proposition 2.4.** Let \( \Gamma \) be a propositional theory with signature \( \sigma \). A set \( X \) of atoms subset of \( \sigma \) is a model of \( \Gamma \) iff it is a stable model of \( \Gamma \cup \{a \lor \neg a : a \in \sigma\} \).

Proof. Let $\Delta$ be $\Gamma \cup \{a \lor \neg a : a \in \sigma\}$. First of all we notice that the reduct of a formula $a \lor \neg a$ relative to a set $X$ of atoms is $a \lor \bot$ if $a \in X$, and $\bot \lor \neg \bot$ otherwise. Consequently if $X \subseteq \sigma$ then $\Delta^X$ is classically equivalent to $\Gamma^X \cup X$. It is easy to see that $\Delta^X$ is not satisfied by any proper subset of $X$, but satisfied by $X$ iff $X \models \Gamma^X$. It follows that $X$ is a stable model of $\Delta$ iff $X \models \Gamma$ by Lemma 5.3.

5.3 Proofs of Propositions 2.5 and 2.6

**Proposition 2.5.** For any formula $F$ and any HT-interpretation $(X, Y)$, $(X, Y) \models F$ iff $X \models F^Y$.

**Proof.** It is sufficient to consider the case when $\Gamma$ is a singleton $\{F\}$, where $F$ contains only connectives $\land$, $\lor$, $\rightarrow$ and $\bot$. The proof is by structural induction on $F$.

- $F$ is $\bot$. $X \not\models \bot$ and $(X, Y) \not\models \bot$.
- $F$ is an atom $a$. $X \models a^Y$ iff $Y \models a$ and $X \models a$. Since $X \subseteq Y$, this means iff $X \models a$, which is the condition for which $(X, Y) \models a$.
- $F$ has the form $G \land H$. $X \models (G \land H)^Y$ iff $X \models G^Y \land H^Y$ by Lemma 5.1, and then iff $X \models G^Y$ and $X \models H^Y$. This is equivalent, by induction hypothesis, to say that $(X, Y) \models G$ and $(X, Y) \models H$, and then that $(X, Y) \models G \land H$.
- The proof for disjunction is similar to the proof for conjunction.
- $F$ has the form $G \rightarrow H$. $X \models (G \rightarrow H)^Y$ iff $X \models G^Y \rightarrow H^Y$ and $Y \models G \rightarrow H$, and then iff
  $$X \models G^Y \text{ implies } X \models H^Y, \text{ and } Y \models G \rightarrow H.$$ 
  This is equivalent, by the induction hypothesis, to
  $$(X, Y) \models G \text{ implies } (X, Y) \models H, \text{ and } Y \models G \rightarrow H,$$
  which is the definition of $(X, Y) \models G \rightarrow H$.

\[ \square \]

**Proposition 2.6.** A set of atoms is a stable model of a theory $\Gamma$ iff $(X, X)$ is an equilibrium model of $\Gamma$.

**Proof.** HT-interpretation $(X, X)$ is an equilibrium model of $\Gamma$ iff

$(X, X) \models \Gamma$ and, for all proper subsets $Y$ of $X$, $(Y, X) \not\models \Gamma$.

In view of Proposition 2.5, this is equivalent to the condition

$$X \models \Gamma^X \text{ and, for all proper subsets } Y \text{ of } X, Y \not\models \Gamma^X.$$ 

which means that $X$ is a stable model of $\Gamma$.

\[ \square \]

5.4 Proofs of Propositions 2.7–2.9

**Proposition 2.7.** For any two theories $\Gamma_1$ and $\Gamma_2$, the following conditions are equivalent:

(i) $\Gamma_1$ is strongly equivalent to $\Gamma_2$,
(ii) $\Gamma_1$ is equivalent to $\Gamma_2$ in the logic of here-and-there, and
(iii) for each set $X$ of atoms, $\Gamma_1^X$ is equivalent to $\Gamma_2^X$ in classical logic.

**Proof.** We will prove the equivalence between (i) and (ii) and between (ii) and (iii). We start with the former. Lemma 4 from [Lifschitz et al. 2001] tells that, for any two theories, the following conditions are equivalent:
(a) for every theory $\Gamma$, theories $\Gamma_1 \cup \Gamma$ and $\Gamma_2 \cup \Gamma$ have the same equilibrium models, and
(b) $\Gamma_1$ is equivalent to $\Gamma_2$ in the logic of here-and-there.

Condition (b) is identical to (ii). Condition (a) can be rewritten, by Proposition 2.6, as
(a') for every theory $\Gamma$, theories $\Gamma_1 \cup \Gamma$ and $\Gamma_2 \cup \Gamma$ have the same stable models, which means that $\Gamma_1$ is strongly equivalent to $\Gamma_2$.

It remains to prove the equivalence between (ii) and (iii). Theory $\Gamma_1$ is equivalent to $\Gamma_2$ in the logic of here-and-there iff, for every set $Y$ of atoms, the following condition holds:

$$(\text{for every } X \subseteq Y, (X, Y) \models \Gamma_1 \text{ iff } (X, Y) \models \Gamma_2).$$

This condition is equivalent, by Proposition 2.5, to

$$(\text{for every } X \subseteq Y, X \models \Gamma_1^Y \text{ iff } X \models \Gamma_2^Y).$$

Since $\Gamma_1^Y$ and $\Gamma_2^Y$ contain atoms from $Y$ only (the other atoms are replaced by $\bot$ in the reduct), this last condition expresses equivalence between $\Gamma_1^Y$ and $\Gamma_2^Y$. \qed

**Lemma 5.4.** For any theory $\Gamma$, let $S$ be a set of atoms that contains all head atoms of $\Gamma$. For any set $X$ of atoms, if $X \models \Gamma$ then $X \cap S \models \Gamma^X$.

**Proof.** It is clearly sufficient to prove the claim for $\Gamma$ that is a singleton \{F\}. The proof is by induction on $F$.

—If $F = \bot$ then $X \not\models F$, and the claim is trivial.
—For an atom $a$, if $X \models a$ then $a^X = a$, but also $a \in S$, so that $X \cap S \models a^X$.
—If $X \models G \land H$ then $X \models G$ and $X \models H$. Consequently, by induction hypothesis, $X \cap S \models G^X$ and $X \cap S \models H^X$. It remains to notice that $(G \land H)^X = G^X \land H^X$.
—The case of disjunction is similar to the case of conjunction.
—If $X \models G \land H$ then $(G \Rightarrow H)^X = G^X \Rightarrow H^X$. Assume that $X \cap S \models G^X$. Consequently $G^X \not\models \bot$ and then $X \models G$. It follows that, since $X \models G \Rightarrow H$, $X \models H$. Since $S$ contains all head atoms of $H$, the claim follows by the induction hypothesis.

\qed

**Proposition 2.8.** Each stable model of a theory $\Gamma$ consists of head atoms of $\Gamma$.

**Proof.** Consider any theory $\Gamma$, the set $S$ of head atoms of $\Gamma$, and a stable model $X$ of $\Gamma$. By Lemma 5.3, $X \models \Gamma$, so that, by Lemma 5.4, $X \cap S \models \Gamma^X$. Since $X \cap S \subseteq X$ and no proper subset of $X$ satisfies $\Gamma^X$, it follows that $X \cap S = X$, and consequently that $X \subseteq S$. \qed

PROPOSITION 2.9. For every two propositional theories $\Gamma_1$ and $\Gamma_2$ such that $\Gamma_2$ has no head atoms, a set $X$ of atoms is a stable model of $\Gamma_1 \cup \Gamma_2$ iff $X$ is a stable model of $\Gamma_1$ and $X \models \Gamma_2$.

PROOF. If $X \models \Gamma_2$ then $\Gamma_1^X$ is satisfied by every subset of $X$ by Lemma 5.4, so that $(\Gamma_1 \cup \Gamma_2)^X$ is classically equivalent to $\Gamma_1^X$; then clearly $X$ is a stable model of $\Gamma_1 \cup \Gamma_2$ iff it is a stable model of $\Gamma_1$. Otherwise, $\Gamma_1^X$ contains $\bot$, and $X$ cannot be a stable model of $\Gamma_1 \cup \Gamma_2$. \qed

5.5 Proofs of Propositions 2.10 and 2.12
We start with the proof of Proposition 2.12. Some lemmas are needed.

LEMMA 5.5. If $X$ is a stable model of $\Gamma$ then $\Gamma^X$ is equivalent to $X$.

PROOF. Since all atoms that occur in $\Gamma^X$ belong to $X$, it is sufficient to show that the formulas are satisfied by the same subsets of $X$. By the definition of a stable model, the only subset of $X$ satisfying $\Gamma^X$ is $X$. \qed

LEMMA 5.6. Let $S$ be a set of atoms that contains all atoms that occur in a theory $\Gamma_1$ but does not contain any head atoms of a theory $\Gamma_2$. For any set $X$ of atoms, if $X$ is a stable model of $\Gamma_1 \cup \Gamma_2$ then $X \cap S$ is a stable model of $\Gamma_1$.

PROOF. Since $X$ is a stable model of $\Gamma_1 \cup \Gamma_2$, $X \models \Gamma_1$, so that $X \cap S \models \Gamma_1$, and, by Lemma 5.3, $X \cap S \models \Gamma_1^{X \cap S}$. It remains to show that no proper subset $Y$ of $X \cap S$ satisfies $\Gamma_2$. Let $S'$ be the set of head atoms of $\Gamma_2$, and let $Z$ be $X \cap (S' \cup Y)$. We will show that $Z$ has the following properties:

(i) $Z \cap S = Y$;
(ii) $Z \subseteq X$;
(iii) $Z \models \Gamma_2^X$.

To prove (i), note that since $S'$ is disjoint from $S$, and $Y$ is a subset of $X \cap S$,

$$Z \cap S = X \cap (S' \cup Y) \cap S = X \cap Y \cap S = (X \cap S) \cap Y = Y.$$  

To prove (ii), note that set $Z$ is clearly a subset of $X$. It cannot be equal to $X$, because otherwise we would have, by (i),

$$Y = Z \cap S = X \cap S;$$

this is impossible, because $Y$ is a proper subset of $X \cap S$. Property (iii) follows from Lemma 5.4, because $X \models \Gamma_2$, and $S' \cup Y$ contains all head atoms of $\Gamma_2$.

Since $X$ is a stable model of $\Gamma_1 \cup \Gamma_2$, from property (ii) we can conclude that $Z \not\models (\Gamma_1 \cup \Gamma_2)^X$. Consequently, by property (iii), $Z \not\models \Gamma_2^X$. Since all atoms that occur in $\Gamma_1$ belong to $S$, $\Gamma_2^X = \Gamma_1^{X \cap S}$, so that $Z \not\models \Gamma_1^{X \cap S}$. Since all atoms that occur in $\Gamma_1^{X \cap S}$ belong to $S$, it follows that $Z \cap S \not\models \Gamma_1^{X \cap S}$. By property (i), we conclude that $Y \not\models \Gamma_1^{X \cap S}$. \qed

PROPOSITION 2.12 (Splitting Set Theorem). Let $\Gamma_1$ and $\Gamma_2$ be two theories such that no atom occurring in $\Gamma_1$ is a head atom of $\Gamma_2$. Let $S$ be a set of atoms containing all head atoms of $\Gamma_1$ but no head atoms of $\Gamma_2$. A set $X$ of atoms is a stable model of $\Gamma_1 \cup \Gamma_2$ iff $X \cap S$ is a stable model of $\Gamma_1$ and $X$ is a stable model of $(X \cap S) \cup \Gamma_2$. 

Proof. We first prove the claim in the case when $S$ contains all atoms of $\Gamma$. If $X \cap S$ is not a stable model of $\Gamma$ then $X$ is not a stable model of $\Gamma_1 \cup \Gamma_2$ by Lemma 5.6. Now suppose that $X \cap S$ is a stable model of $\Gamma$. Then, by Lemma 5.5, $\Gamma_1^{X \cap S}$ is equivalent to $X \cap S$. Consequently,

$$
(\Gamma_1 \cup \Gamma_2)^X = \Gamma_1^X \cup \Gamma_2^X = \Gamma_1^{X \cap S} \cup \Gamma_2^X \leftrightarrow (X \cap S) \cup \Gamma_2^X = (X \cap S)^X \cup \Gamma_2^X = (X \cap S) \cup \Gamma_2 \leftrightarrow (X \cap S) \cup \Gamma_2
$$

We can conclude that $X$ is a stable model of $\Gamma_1 \cup \Gamma_2$ iff $X$ is a stable model of $\Gamma_2 \cup (X \cap S)$.

The most general case remains. Let $S_1$ be the set of all atoms in $\Gamma$ (the value of $S$ for which we have already proved the claim). In view of the special case described above, it is sufficient to show that, for any set $S$ of atoms that respects the hypothesis conditions,

$X \cap S_1$ is a stable model of $\Gamma$ and $X$ is a stable model of $(X \cap S_1) \cup \Gamma_2$  \hspace{1cm} (39)

holds iff

$X \cap S$ is a stable model of $\Gamma_1$ and $X$ is a stable model of $(X \cap S) \cup \Gamma_2$. \hspace{1cm} (40)

Assume (39). Sets $S$ and $S_1$ differ only for sets of atoms that are not head atoms of $\Gamma$. Consequently, since $X \cap S_1$ is a stable model of $\Gamma$, it follows from Proposition 2.8 that $X \cap S_1 = X \cap S$. We can then conclude that (40) follows from (39).

The proof in the opposite direction is similar. \(\square\)

Lemma 5.7. Let $\Gamma$ be a theory, and let $Y$ and $Z$ be two disjoint sets of atoms such that no atom of $Z$ is an head atoms of $\Gamma$. Let $\Gamma'$ a theory obtained from $\Gamma$ by replacing occurrences of atoms of $Y$ with $\top$ and occurrences of atoms of $Z$ with $\bot$. Then $\Gamma \cup Y$ and $\Gamma' \cup Y$ have the same stable models.

Proof. Atoms of $Z$ are not head atoms of $\Gamma \cup Y$. Consequently, by Proposition 2.8, every stable model of $\Gamma \cup Y$ is disjoint from $Z$. It follows, by Proposition 2.9, that $\Gamma \cup Y$ has the same stable models of

$$
\Gamma \cup Y \cup \{\neg a : a \in Z\}.
$$

Similarly, $\Gamma' \cup Y$ has the same stable models of

$$
\Gamma' \cup Y \cup \{\neg a : a \in Z\}.
$$

It is a known property that the two theories above are equivalent to each other in intuitionistic logic, and then in the logic-of-here-and-there. Consequently, by Proposition 2.7, they are strongly equivalent to each other, and we can conclude that they have the same stable models. \(\square\)

Proposition 2.10. Let $\Gamma$ be any propositional theory, and $Q$ a set of atoms not occurring in $\Gamma$. For each $q \in Q$, let $\text{Def}(q)$ be a formula that doesn’t contain any atoms from $Q$. Then $X \mapsto X \setminus Q$ is a 1–1 correspondence between the stable models of $\Gamma \cup \{\text{Def}(q) \rightarrow q : q \in Q\}$ and the stable models of $\Gamma$.

Proof. Let $\Gamma_2$ be $\{\text{Def}(q) \rightarrow q : q \in Q\}$. Since $Q$ contains all head atoms of $\Gamma_2$ but no atom occurring in $\Gamma$ then, by the splitting set theorem (Proposition 2.12),
(“s.m.” stands for “a stable model”)

\[ X \text{ is s.m. of } \Gamma \cup \Gamma_2 \iff X \setminus Q \text{ is s.m. of } \Gamma \text{ and } X \text{ is s.m. of } (X \setminus Q) \cup \Gamma_2. \quad (41) \]

Clearly, if \( X \) is a stable model of \( \Gamma \cup \Gamma_2 \) then \( X \setminus Q \) is a stable model of \( \Gamma \), which proves one of the two directions of the 1–1 correspondence in the claim. Now take any stable model \( Y \) of \( \Gamma \). We need to show that there is exactly one stable model \( X \) of \( \Gamma \cup \Gamma_2 \) such that \( X \setminus Q = Y \). In view of (41), it is sufficient to show that

\[ Z = Y \cup \{ q \in Q : Y \models \text{Def}(q) \} \]

is the only stable model \( X \) of \( \Gamma \cup \Gamma_2 \), and that \( Z \setminus Q = Y \). This second condition can be easily verified. Now consider \( Y \cup \Gamma_2 \). By Lemma 5.7, \( Y \cup \Gamma_2 \) has the same stable models of

\[ Y \cup \{ \text{Def}(q)' \rightarrow q : q \in Q \}, \]

where \( \text{Def}(q)' \) is obtained from \( \text{Def}(q) \) by replacing all occurrences of atoms in it with \( \top \) if the atom replaced belongs to \( Y \), and with \( \bot \) otherwise. This theory can be further simplified into theory \( Z \). Indeed, \( \text{Def}(q)' \) doesn’t contain atoms, and then it is strongly equivalent to \( \top \) or \( \bot \). In particular, if \( Y \models \text{Def}(q) \) then \( \text{Def}(q)' \) is strongly equivalent to \( \top \), and then \( \text{Def}(q)' \rightarrow q \) is strongly equivalent to \( q \). Otherwise, \( \text{Def}(q)' \) is strongly equivalent to \( \bot \), and then \( \text{Def}(q)' \rightarrow q \) is strongly equivalent to \( \bot \). As \( Z \) is a set of atoms, it is easy to verify that its only stable model is \( Z \) itself. \( \Box \)

5.6 Proof of Proposition 2.11

In order to prove the Completion Lemma, we will need the following lemma.

Lemma 5.8. Take any two sets \( X, Y \) of atoms such that \( Y \subseteq X \). For any formula \( F \) and any set \( S \) of atoms,

(a) if each positive occurrence of an atom from \( S \) in \( F \) is in the scope of negation and \( Y \models F^X \) then \( Y \setminus S \models F^X \), and

(b) if each negative occurrence of an atom from \( S \) in \( F \) is in the scope of negation and \( Y \setminus S \models F^X \) then \( Y \models F^X \).

Proof. — If \( X \not\models F \) then \( F^X = \bot \), and the claim is trivial. This covers the case in which \( F = \bot \).

— If \( X \models F \) and \( F \) is an atom \( a \) then claim (b) holds because if \( a \in Y \setminus S \) then \( a \in Y \). For claim (a), if \( a \notin S \) and \( a \in Y \) then \( a \in Y \setminus S \).

— If \( X \models F \) and \( F \) is a conjunction or a disjunction, the claim is almost immediate by Lemma 5.1 and induction hypothesis.

— The case in which \( X \models F \) and \( F \) has the form \( G \rightarrow H \) remains. Clearly, \((G \rightarrow H)^X = G^X \rightarrow H^X\). Case 1: If \( G \rightarrow H \) is a negation (that is, \( H = \bot \)) then, since \( X \models F \), \( X \not\models G^X \) and then \( F^X = \top \), and the claims clearly follows.

Case 2: \( H \neq \bot \). We describe a proof of claim (a). The proof for (b) is similar. Assume that no atom from \( S \) has positive occurrences in \( G \rightarrow H \) outside the scope of the negation, that \( Y \models G^X \rightarrow H^X \), and that \( Y \setminus S \models G^X \). We want to prove that \( Y \setminus S \models H^X \). Notice that no atom from \( S \) has negative occurrences in \( G \) outside the scope of negation; consequently, by the induction hypothesis (claim
\[ \{ q \rightarrow \text{Def}(q) : q \in Q \} \]

Proof. Let \( \Gamma_1 = \Gamma \cup \{ \text{Def}(q) \rightarrow q : q \in Q \} \) and let \( \Gamma_2 = \Gamma_1 \cup \{ q \rightarrow \text{Def}(q) : q \in Q \} \). We want to prove that a set \( X \) of atoms is a stable model of both theories or for none of them. Since \( \Gamma_1^X \subseteq \Gamma_2^X \), \( \Gamma_2^X \) entails \( \Gamma_1^X \). If the opposite entailment holds also then we clearly have that \( \Gamma_2^X \) and \( \Gamma_1^X \) are satisfied by the same subsets of \( X \), and the claim immediately follows. Otherwise, for some \( Y \subseteq X \), \( Y \neq \Gamma_2^X \) and \( Y = \Gamma_1^X \). First of all, that means that \( X \models \Gamma_1 \), so that \( \Gamma_1^X \) is equivalent to

\[ \Gamma^X \cup \{ \text{Def}(q)^X \rightarrow q : q \in Q \cap X \}. \]

Secondly, set \( Y \) is one of the sets \( Y' \) having the following properties:

1. \( Y' \setminus Q = Y \setminus Q \), and
2. \( Y' \models \text{Def}(q)^X \rightarrow q \) for all \( q \in Q \cap X \).

Let \( Z \) be the intersection of such sets \( Y' \), and let \( \Delta = \{ q \rightarrow \text{Def}(q)^X : q \in Q \cap X \} \).

Set \( Z \) has the following properties:

- \( Z \subseteq Y \),
- \( Z \models \Gamma_1^X \), and
- \( Z \models \Delta \).

Indeed, claim (a) holds since \( Y \) is one of the elements \( Y' \) of the intersection. To prove (b), first of all, we observe that \( Z \setminus Q = Y \setminus Q \), so that, by (a), there is a set \( S \subseteq Q \) such that \( Z = Y \setminus S \); as \( Y \models \Gamma^X \) and \( \Gamma \) has all positive occurrences of atoms from \( S \subseteq Q \) in the scope of negation, it follows that \( Z \models \Gamma^X \) by Lemma 5.8(a). It remains to show that, for any \( q \), if \( Z \models \text{Def}(q)^X \) then \( q \in Z \). Assume that \( Z \models \text{Def}(q)^X \). Then, since \( \text{Def}(q) \) has all negative occurrences of atoms from \( Q \) in the scope of negation, and since all \( Y' \) whose intersection generate \( Z \) are superset of \( Z \) with \( Y' \setminus Z \subseteq Q \), all those \( Y' \) satisfy \( \text{Def}(q)^X \) by Lemma 5.8. By property (ii), we have that \( q \in Y' \) for all \( Y' \), and then \( q \in Z \).

It remains to prove claim (c). Take any \( q \in Z \) that belongs to \( Q \cap X \). Set \( Y' = Z \setminus \{ q \} \) satisfies condition (i), but it cannot satisfy (ii), because sets \( Y' \) that satisfy (i) and (ii) are supersets of \( Z \) by construction of \( Z \). Consequently, \( Y' \neq \text{Def}(q)^X \). Since all positive occurrences of atom \( q \) in \( \text{Def}(q) \) are in the scope of negation and \( Y' = Z \setminus \{ q \} \), we can conclude that \( Z \neq \text{Def}(q)^X \) by Lemma 5.8 again.

Now consider two cases. If \( X \neq \Gamma_2 \) then clearly \( X \) is not a stable model of \( \Gamma_2 \). It is not a stable model of \( \Gamma_1 \) as well. Indeed, since \( X \models \Gamma_1 \), we have that, for some

\[ \text{ACM Transactions on Computational Logic, Vol. V, No. N, 20YY.} \]
which is equivalent to $\Sigma^{\text{compl}}$. The same problem for the (larger) class of arbitrary theories is also in $\Sigma^P_2$ by (b). $X$ is not a stable model of $\Gamma_1$.

In the other case ($X \models \Gamma_2$) it is not hard to see that $\Gamma^X_2$ is equivalent to $\Gamma^X \cup \Delta$. We have that $Z \models \Gamma^X_1$ by (b), and then $Z \models \Gamma^X_2$ by (c). Since $Y \not\models \Gamma^X_2$, $Z \not\models Y$. On the other hand, $Z \subseteq Y \subseteq X$ by (a). This means that $Z$ is a proper subset of $X$ that satisfies $\Gamma^X_1$ and $\Gamma^X_2$, and we can conclude that $X$ is not an stable model of any of $\Gamma_1$ and $\Gamma_2$. □

5.7 Proof of Lemma 2.13 and Proposition 2.14

**Lemma 2.13.** Rule

$$l_1 \land \cdots \land l_m \rightarrow a_1 \lor \cdots \lor a_n$$

(42)

$(n > 0, m \geq 0)$ where $a_1, \ldots, a_n$ are atoms and $l_1, \ldots, l_m$ are literals, is strongly equivalent to the set of $n$ implications $(i = 1, \ldots, n)$

$$(l_1 \land \cdots \land l_m \land (a_1 \rightarrow a_i) \land \cdots \land (a_n \rightarrow a_i)) \rightarrow a_i.$$  \hspace{1cm} (43)

**Proof.** Let $F$ be (42) and $G_i$ $(i = 1, \ldots, n)$ be (43). We want to prove that $F$ is strongly equivalent to $\{G_1, \ldots, G_n\}$ by showing that $F^X$ is classically equivalent to $\{G^X_1, \ldots, G^X_n\}$. Let $H$ be $l_1 \land \cdots \land l_m$.

**Case 1:** $X \not\models H$. Then the antecedents of $F$ and of all $G_i$ are not satisfied by $X$. It is then easy to verify that the reducts of $F$ and of all $G_i$ relative to $X$ are equivalent to $\top$. **Case 2:** $X \models H$ and $X \not\models F$. Then clearly $F^X = \bot$. But, for each $i$, $G^X_i$ is $\bot$: indeed, since $X \not\models F$, $X \not\models a_i$ for all $i = 1, \ldots, n$. It follows that the consequent of each $G_i$ is not satisfied by $X$, but the antecedent is satisfied, because $X \models H$ and in each implication $a_j \rightarrow a_i$ in $G_i$, the antecedent is not satisfied.

**Case 3:** $X \models H$ and $X \models F$. This means that all $G_i$ belong to $X$. Assume, for instance, that $a_1, \ldots, a_p$ (with $0 < p \leq n$) belong to $X$, and $a_{p+1}, \ldots, a_n$ don’t. Then $F^X$ is equivalent to $H^X \rightarrow (a_1 \lor \cdots \lor a_p)$. Now consider formula $G_i$. If $i > p$ then the consequent $a_i$ is not satisfied by $X$, but also the antecedent is not: it contains an implication $a_1 \rightarrow a_i$; consequently $G^X_i$ is $\bot$. On the other hand, if $i \leq p$ then the consequent $a_i$ is satisfied by $X$, as well as each implication $a_j \rightarrow a_i$ in the antecedent of $G_i$. After a few simplifications, we can rewrite $G^X_i$ as $$(H^X \land (a_1 \rightarrow a_i) \land \cdots \land (a_p \rightarrow a_i)) \rightarrow a_i.$$ It is not hard to see that this formula is classically equivalent to

$$(H^X \rightarrow (a_1 \lor \cdots \lor a_p))$$ which is equivalent to $F^X$, so that the claim easily follows. □

**Proposition 2.14.** The problem of the existence of a stable model of a theory consisting of formulas of the form $F \rightarrow a$ and $F \rightarrow \bot$ is $\Sigma^P_2$-hard.

**Proof.** The problem is in class $\Sigma^P_2$ because, as mentioned in Section sec:prop-compl, the same problem for the (larger) class of arbitrary theories is also in $\Sigma^P_2$ [Pearce et al. 2001]. Hardness remains to be proven.
In view of Lemma 2.13, we can transform a disjunctive program into a theory consisting of formulas of the form \( F \leftarrow a \), with the same stable models and in polynomial time. Consequently, as the existence of a stable model of a disjunctive program is \( \Sigma^P_2 \)-hard by [Eiter and Gottlob 1993], the same holds for theories as in the statement of this proposition. 

5.8 Proof of Propositions 3.1 and 3.2

For the proof of these propositions, we define an extended aggregate to be either an aggregate of the form (10), or \( \perp \). It is easy to see, that, for each aggregate \( A \) of the form (10) and any set \( X \) of atoms, \( A^X \) is an extended aggregate. We also define, for any extended aggregate \( A \), \( \hat{A} \) as

— the formula (13) if \( A \) has the form (10), and
— \( \perp \), otherwise.

**Lemma 5.9.** For any extended aggregate \( A \), \( \hat{A} \) is classically equivalent to \( A \).

**Proof.** The case \( A = \perp \) is trivial. The remaining case is when \( A \) is an aggregate. Consider any possible conjunctive term \( H_I \) (where \( I \subseteq \{1, \ldots, n\} \)) of \( \hat{A} \):

\[
( \bigwedge_{i \in I} F_i ) \rightarrow ( \bigvee_{i \in \bar{I}} F_i ).
\]

For each set \( X \) of atoms there is exactly one set \( I \) such that \( X \not\models H_I \): the set \( I_X \) that consists of the \( i \)'s such that \( X \models F_i \). Consequently, for every set \( X \) of atoms,

\[
X \models \hat{A} \text{ iff } H_{I_X} \text{ is not a conjunctive term of } \hat{A} \text{ iff }
\]

\[
\text{op}(\{ w_i : i \in I_X \}) \prec N \text{ iff } X \models A.
\]

**Lemma 5.10.** For any aggregate \( A \) and any set \( X \) of atoms, \( \hat{A}^X \) is classically equivalent to \( \hat{A} \).

**Proof.** Case 1: \( X \not\models A \). Then \( \hat{A}^X = \perp = \perp \). On the other hand, by Lemma 5.9, \( X \not\models \hat{A} \) so that \( \hat{A} = \perp \) also. Case 2: \( X \models A \). Then \( A \) is an aggregate, and, by the definition of a reduct, \( \hat{A}^X \) is

\[
\bigwedge_{I \subseteq \{1, \ldots, n\} : \text{op}(\{ w_i : i \in I \}) \not\prec N} ((\bigwedge_{i \in I} F_i^X) \rightarrow (\bigvee_{i \in \bar{I}} F_i^X)).
\]

On the other hand, \( \hat{A}^X \) is classically equivalent, by Lemma 5.1, to

\[
\bigwedge_{I \subseteq \{1, \ldots, n\} : \text{op}(\{ w_i : i \in I \}) \not\prec N} ((\bigwedge_{i \in I} F_i) \rightarrow (\bigvee_{i \in \bar{I}} F_i)^X).
\]

Notice that, since \( X \models \hat{A} \) by Lemma 5.9, all implications in the formula above are satisfied by \( X \). Consequently, \( A^X \) is classically equivalent to

\[
\bigwedge_{I \subseteq \{1, \ldots, n\} : \text{op}(\{ w_i : i \in I \}) \not\prec N} ((\bigwedge_{i \in I} F_i)^X \rightarrow (\bigvee_{i \in \bar{I}} F_i)^X),
\]
and then, by Lemma 5.1 again, to (44). □

**Proposition 3.1.** Let \( A \) be an aggregate of the form (10) and let \( G \) be the corresponding formula (13). Then

(a) \( G \) is classically equivalent to \( A \), and

(b) for any set \( X \) of atoms, \( G^X \) is classically equivalent to \( A^X \).

**Proof.** Part (a) is immediate from Lemma 5.9, as \( G = \bigwedge \). For part (b), we need to show that \( A^X \) is classical equivalent to \( A^X \). By Lemma 5.10, \( A^X \) is classically equivalent to \( A^X \). It remains to notice that \( A^X \) is classically equivalent to \( A^X \) by Lemma 5.9. □

**Lemma 5.11.** For any aggregate \( \text{op}(\{F_1 = w_1, \ldots, F_n = w_n\}) \prec N \), formula (13) is classically equivalent to

\[
\bigwedge_{I \subseteq \{1, \ldots, n\} : \text{op}(\{w_i : i \in I\}) \not\prec N} \big( \bigvee_{i \in I} F_i \big) \tag{45}
\]

if the aggregate is monotone, and to

\[
\bigwedge_{I \subseteq \{1, \ldots, n\} : \text{op}(\{w_i : i \in I\}) \not\prec N} \big( \neg \bigwedge_{i \in I} F_i \big)
\]

if the aggregate is antimonotone.

**Proof.** Consider the case of a monotone aggregate first. Let \( G \) be (13), and \( H \) be (45). It is easy to verify that \( H \) entails \( G \). The opposite direction remains. Assume \( G \), and we want to derive every conjunctive term

\[
\bigvee_{i \in \overline{I}} F_i \tag{46}
\]

in \( H \). For every conjunctive term \( D \) of the form (46) in \( H \), \( \text{op}(\{w_i : i \in I\}) \not\prec N \). As the aggregate is monotone then, for every subset \( I' \) of \( I \), \( \text{op}(\{w_i : i \in I'\}) \not\prec N \), so that the implication

\[
\big( \bigwedge_{i \in I'} F_i \big) \rightarrow \big( \bigvee_{i \in \overline{I}} F_i \big)
\]

is a conjunctive term of \( H \) for all \( I' \subseteq I \). Then, since \( \overline{I} = I \cup (I \setminus I') \), ("\( \Rightarrow \)" denotes entailment, and "\( \Leftrightarrow \)" equivalence)

\[
H \Rightarrow \bigwedge_{I' \subseteq I} \left( \left( \bigwedge_{i \in I'} F_i \right) \rightarrow \left( \bigvee_{i \in \overline{I}} F_i \right) \right)
\]

\[
\Leftrightarrow \bigwedge_{I' \subseteq I} \left( \left( \bigwedge_{i \in I'} F_i \right) \land \bigwedge_{i \in I' \setminus I} \neg F_i \rightarrow \left( \bigvee_{i \in \overline{I}} F_i \right) \right)
\]

\[
\Leftrightarrow \left( \bigvee_{I' \subseteq I} \left( \left( \bigwedge_{i \in I'} F_i \right) \land \bigwedge_{i \in I' \setminus I} \neg F_i \right) \right) \rightarrow D.
\]
The antecedent of the implication is a tautology: for each interpretation \( X \), the disjunctive term relative to \( I' = \{ i \in I : X \models F_i \} \) is satisfied by \( X \). We can conclude that \( H \) entails \( D \).

The proof for antimonotone aggregates is similar. □

**Proposition 3.2.** For any aggregate \( \text{op} \{ F_1 = w_1, \ldots, F_n = w_n \} \), formula (12) is strongly equivalent to

\[
\bigwedge_{I \subseteq \{1, \ldots, n\} : \text{op}(\{ w_i : i \in I \}) \not\prec N} (\bigvee_{i \in I} F_i)
\]

if the aggregate is monotone, and to

\[
\bigwedge_{I \subseteq \{1, \ldots, n\} : \text{op}(\{ w_i : i \in I \}) \not\prec N} (\neg \bigwedge_{i \in I} F_i)
\]

if the aggregate is antimonotone.

**Proof.** Consider the case of a monotone aggregate first. Let \( G \) be (13), and \( H \) be (45). In view of Proposition 2.7, it is sufficient to show that \( G^X \) is equivalent to \( H^X \) in classical logic for all sets \( X \). If \( X \not\models H \) then also \( X \not\models G \) by Lemma 5.11, so that both reducts are \( \bot \). Otherwise (\( X \models H \)), by the same lemma, \( X \models G \). Then, by Lemma 5.10, \( G^X \) is classically equivalent to (44). On the other hand, it is easy to verify, by applying Lemma 5.1 to \( H^X \) twice, that \( H^X \) is classically equivalent to

\[
\bigwedge_{I \subseteq \{1, \ldots, n\} : \text{op}(\{ w_i : i \in I \}) \not\prec N} (\bigvee_{i \in I} F_i^X).
\]

The claim now follows from Lemma 5.11.

The reasoning for nonmonotone aggregates is similar. □

**5.9 Proof of Proposition 3.3**

Let \( \Gamma \) be the theory consisting of formulas (21)–(24).

**Lemma 5.12.** For any stable model \( X \) of \( \Gamma \), \( X \) contains an atom \( s_i \) iff \( X \) contains an atom \( b_j \) such that \( b_j \) involves selling object \( i \).

**Proof.** Consider \( \Gamma \) as a propositional theory. We notice that

—formulas (23) can be strongly equivalently grouped as \( m \) formulas (\( i = 1, \ldots, m \))

\[
( j=1,\ldots,m \colon \text{object } i \text{ is part of bid } j ) \rightarrow s_i,
\]

and

—no other formula of \( \Gamma \) contains atoms of the form \( s_i \) outside the scope of negation.

Consequently, by the Completion Lemma (Proposition 2.11), formulas (23) in \( \Gamma \) can be replaced by \( m \) formulas (\( i = 1, \ldots, m \))

\[
( j=1,\ldots,m \colon \text{object } i \text{ is part of bid } j ) \leftrightarrow s_i.
\]

preserving the stable models. It follows that every stable model of \( \Gamma \) must satisfy formulas (47), and the claim immediately follows. □
Proposition 3.3. The mapping from a set $X$ of atoms to the set $\{i : b_i \in X\}$ is a 1-1 correspondence between the stable models of the theory consisting of formulas (21)–(24) and a solution of Joe’s problem.

Proof. Take any stable model $X$ of $\Gamma$. Since $X$ satisfies rules (22) of $\Gamma$, condition (a) is satisfied. Condition (b) is satisfied as well, because $X$ contains exactly all atoms $s_i$ sold in some bids by Lemma 5.12, and since $X$ satisfies aggregate (24) that belongs to $\Gamma$.

Now consider a solution of Joe’s problem. This determines which atoms of the form $b_i$ belongs to a possible corresponding stable model $X$. Consequently, Lemma 5.12 determines also which atoms of the form $s_j$ belong to $X$, reducing the candidate stable models $X$ to one. We need to show that this $X$ is indeed a stable model of $\Gamma$. The reduct $\Gamma^X$ consists of (after a few simplifications)

(i) all atoms $b_i$ that belong to $X$ (from (21)),
(ii) $\top$ from (22) since (a) holds,
(iii) (by Lemma 5.12) implications (23) such that both $b_j$ and $s_i$ belong to $X$, and
(iv) the reduct of (24) relative to $X$.

Notice that (i)–(iii) together are equivalent to $X$, so that every proper subset of $X$ doesn’t satisfy $\Gamma^X$. It remains to show that $X \models \Gamma^X$. Clearly, $X$ satisfies (i)–(iii). To show that $X$ satisfies (iv) it is sufficient, by Lemma 5.3 (consider (24) as a propositional formula), to show that $X$ satisfies (24): it does that by hypothesis (b). □

5.10 Proof of Propositions 3.4 and 3.5

Lemma 5.13. If, for every aggregate, computing $\text{op}(W) \prec N$ requires polynomial time then

(a) checking satisfaction of a theory with aggregates requires polynomial time, and
(b) computing the reduct of a theory with aggregates requires polynomial time.

Proof. Part (a) is easy to verify by structural induction. Computing the reduct essentially consists of checking satisfaction of subexpressions of each formula of the theory. Each check doesn’t require too much time by (a). It remains to notice that each formula with aggregates has a linear number of subformulas. □

Proposition 3.4. If, for every aggregate, computing $\text{op}(W) \prec N$ requires polynomial time then the existence of a stable model of a theory with aggregates is a $\Sigma^p_2$-complete problem.

Proof. Hardness follows from the fact that theories with aggregates are a generalization of propositional theories. To prove inclusion, consider that the existence of a stable model of a theory $\Gamma$ is equivalent to satisfiability of:

exists $X$ such that for all $Y$, if $Y \subseteq X$ then $Y \models \Gamma^X$ iff $X = Y$

It remains to notice that, in view of Lemma 5.13, checking (for any $X$ and $Y$)

if $Y \subseteq X$ then $Y \models \Gamma^X$ iff $X = Y$

requires polynomial time. □
function verifyAS(\(\Gamma, X\))
    if \(X \not\models \Gamma\) then return false
    \(\Delta := \{F^X \rightarrow a : F \rightarrow a \in \Gamma \text{ and } X \models a\}\)
    \(Y := \emptyset\)
    while there is a formula \(G \rightarrow a \in \Delta\) such that \(Y \models G\) and \(a \not\in Y\)
        \(Y := Y \cup \{a\}\)
    end while
    if \(Y = X\) then return true
    return false

Fig. 5. A polynomial-time algorithm that checks stable models of special kinds of theories

**Lemma 5.14.** Let \(F\) be a formula with aggregates containing monotone and antimonotone aggregates only, no equivalences and no implications other than negations. For any sets \(X, Y\) and \(Z\) such that \(Y \subseteq Z\), if \(Y \models F^X\) then \(Z \models F^X\).

**Proof.** Let \(G\) be \(F\) with each monotone aggregate replaced by (19) and each antimonotone aggregate replaced by (20). It is easy to verify that \(G\) is a nested expression. Nested expressions have all negative occurrences of atoms in the scope of negation, so if \(Y \models G^X\) then \(Z \models G^X\) by Lemma (5.8). It remains to notice that \(F^X\) and \(G^X\) are satisfied by the same sets of atoms by Propositions 3.2 and 3.1. \(\square\)

**Proposition 3.5.** Consider theories with aggregates consisting of formulas of the form

\[
F \rightarrow a,
\]

where \(a\) is an atom or \(\bot\), and \(F\) contains monotone and antimonotone aggregates only, no equivalences and no implications other than negations. If, for every aggregate, computing \(op(W) \prec N\) requires polynomial time then the problem of the existence of a stable model of theories of this kind is an NP-complete problem.

**Proof.** NP-hardness follows from the fact that theories with aggregates are a generalization of traditional programs, for which the same problem is NP-complete. For inclusion in NP, it is sufficient to show that the time required to check if a set \(X\) of atoms is a stable model of \(\Gamma\) is polynomial. An algorithm that does this test is in Figure 5. It is easy to verify that it is a polynomial time algorithm. It remains to prove that it is correct. If \(X \not\models \Gamma\) then it is trivial. Now assume that \(X \models \Gamma\). It is sufficient to show that

(a) \(\Delta\) is classically equivalent to \(\Gamma^X\), and
(b) the last value of \(Y\) (we call it \(Z\)) is the unique minimal model of \(\Delta\).

Indeed, for part (a), we notice that, since \(X \models \Gamma\), \(\Gamma^X\) is

\[
\{F^X \rightarrow a^X : F \rightarrow a \in \Gamma \text{ and } X \models a\} \cup \{F^X \rightarrow a^X : F \rightarrow a \in \Gamma \text{ and } X \not\models a\}.
\]

The first set is \(\Delta\). The second set (which includes the case in which \(a = \bot\)) is a set of \(\bot \rightarrow \bot\). Indeed, each \(a^X = \bot\), and since \(X \models \Gamma\) \(X\) doesn’t satisfy any \(F\) and then \(F^X = \bot\).

For part (b) it is easy to verify that the while loop iterates as long as \(Y \not\models \Delta\), so that \(Z \models \Delta\). Now assume, in sake of contradiction, that there is a set \(Z'\) that satisfies \(\Delta\) and that is not a superset of \(Z\). Consider, in the execution of the
algorithm, the first atom \( a \notin Z' \) added to \( Y \), and that value of \( Y \subseteq Z' \) to which \( a \) has been added to. This means that \( \Delta \) contains a formula \( G \rightarrow a \) such that \( Y \models G \). Recall that \( G \) stands for a formula of the form \( F^X \), where \( F \) is a formula with aggregates with monotone and antimonotone aggregates only and without implications (other than negations) or equivalences. Consequently, by Lemma 5.14, \( Z' \models G \). On the other hand, \( a \notin Z' \), so \( Z' \not\models G \rightarrow a \), contradicting the hypothesis that \( Z' \) is a model of \( \Delta \).  

5.11 Proof of Proposition 3.6

**Lemma 5.15.** Let \( F \) and \( G \) two propositional formulas, and let \( F' \) and \( G' \) the result of replacing each occurrence of an atom \( a \) in \( F \) and \( G \) with a propositional formula \( H \). If \( F \) and \( G \) are strongly equivalent to each other then \( F' \) and \( G' \) are strongly equivalent to each other.

**Proof.** It follows from Proposition 2.7, in view of the following fact: if \( F \) and \( G \) are equivalent in the logic of here-and-there to each other then \( F' \) and \( G' \) are equivalent in the logic of here-and-there to each other.

**Lemma 5.16.** Let \( F \) and \( G \) be two propositional formulas that are AND-OR combinations of \( \top, \bot \) and atoms only. If \( F \) and \( G \) are classically equivalent to each other then they are strongly equivalent to each other also.

**Proof.** In view of Proposition 2.7, it is sufficient to show that, for every set \( X \) of atoms, \( F^X \) is classically equivalent to \( G^X \). By Lemma 5.1 we can distribute the reduct operator in \( F^X \) to its atoms. If follows that \( F^X \) is classically equivalent to \( F \) with all occurrences of atoms that don’t belong to \( X \) replaced by \( \bot \), and similarly for \( G^X \). The fact that \( F^X \) is classically equivalent to \( G^X \) now follows from the classical equivalence between \( F \) and \( G \).

Next Lemma immediately follows from our definition of satisfaction of aggregates (Section 3.1 of this paper), and the definition of \([L \leq S]\) and \([S \leq U]\) and Proposition 1 from [Ferraris and Lifschitz 2005b].

**Lemma 5.17.** For every weight constraints \( L \leq S \) and \( S \leq U \) and any set \( X \) of atoms,

(a) \( X \models [L \leq S] \) iff \( X \models \text{sum}(S) \geq L \), and

(b) \( X \models [S \leq U] \) iff \( X \models \text{sum}(S) \leq U \).

**Proposition 3.6.** In presence of nonnegative weights only, \([N \leq S]\) is strongly equivalent to \( \text{sum}(S) \geq N \), and \([S \leq N]\) is strongly equivalent to \( \text{sum}(S) \leq N \).

**Proof.** We start with (a), with the special case when rule elements \( F_1, \ldots, F_n \) of \( S \) are distinct atoms. Since the aggregate is monotone then, by Lemma 3.2, we just need to show that \([N \leq S]\) is strongly equivalent to (19). As classical equivalence holds between \([N \leq S]\) and \( \text{sum}(S) \geq N \) by Lemma 5.17, the same relationship holds between \([N \leq S]\) and (19). As both formulas are AND-OR combinations of atoms, the claim follows by Lemma 5.16. The most general case of (a) follows from the special case, by Lemma 5.15.
For part (b), we know, by Lemma 3.2, that antimonotone aggregate \( \text{sum}(S) \leq U \) (written as a formula (10)) is strongly equivalent to formula

\[
\bigwedge_{I \subseteq \{1, \ldots, n\}} \left( \sum_{i \in I} w_i > U \right) \land \left( \bigwedge_{i \in I} F_i \right).
\]

By applying DeMorgan’s law to this last formula (which preserves equivalence in the logic of here-and-there and then it is a strongly equivalent transformation by Proposition 2.7) we get \( S \leq U \).

5.12 Proof of Proposition 3.7

Given a PDB-aggregate of the form (10) and a set \( X \) of literals, by \( I_X \) we denote the set \( \{ i \in \{1, \ldots, n\} : X \models F_i \} \).

**Lemma 5.18.** For each PDB-aggregate of the form (10), a set \( X \) of atoms satisfies a formula of the form \( G(I_1, I_2) \) iff \( I_1 \subseteq I_X \subseteq I_2 \).

**Proof.**

\[
X \models G(I_1, I_2) \iff X = F_i \text{ for all } i \in I_1, \text{ and } X \not\models F_i \text{ for all } i \in \{1, \ldots, n\} \setminus I_2
\]

iff \( X \models F_i \text{ for all } i \in I_1, \text{ and for every } i \text{ such that } X \models F_i, i \in I_2 \)

iff \( I_1 \subseteq I_X \text{ and } I_X \subseteq I_2 \).

\[ \square \]

**Lemma 5.19.** For every PDB-aggregate \( A \), \( A_{tr} \) is classically equivalent to (13).

**Proof.** Consider a set \( X \) of atoms. By Lemma 5.18, \( X \models A_{tr} \) iff \( X \) satisfies one of the disjunctive terms \( G(I_1, I_2) \) of \( A_{tr} \) and then iff

\( A_{tr} \) contains a disjunctive term \( G(I_1, I_2) \) such that \( I_1 \subseteq I_X \subseteq I_2 \).

It is easy to verify that if this condition holds then one of such terms \( G(I_1, I_2) \) is \( G(I_X, I_X) \). Consequently,

\[
X \models A_{tr} \iff A_{tr} \text{ contains disjunctive term } G(I_X, I_X)
\]

iff \( \text{op}(W_{I_X}) < N \).

We have essentially found that \( X \models A_{tr} \iff X \models A \). The claim now follows by Proposition 3.1(a).

**Lemma 5.20.** For any PDB-aggregate \( A \), \( A_{tr} \) is strongly equivalent to

(a)

\[
\bigvee_{I \subseteq \{1, \ldots, n\}, \text{op}(W_I) < N} G(I, \{1, \ldots, n\})
\]

if \( A \) is monotone, and to

(b)

\[
\bigvee_{I \subseteq \{1, \ldots, n\}, \text{op}(W_I) < N} G(\emptyset, I)
\]

if it is antimonotone.
Proof. To prove (a), assume that \( A \) is monotone. Then, if \( A_{tr} \) contains a disjunctive term \( G(I_1, I_2) \) then it contains the disjunctive term \( G(I_1, \{1, \ldots, n\}) \) as well. Consider also that formula \( G(I_1, \{1, \ldots, n\}) \) entails \( G(I_1, I_2) \) in the logic of here-and-there. Then, by Proposition 2.7, we can drop all disjunctive terms of the form \( G(I_1, I_2) \) with \( I_2 \neq \{1, \ldots, n\} \), preserving strong equivalence. Formula \( A_{tr} \) becomes

\[
\bigvee_{I_1 \subseteq \{1, \ldots, n\}} G(I_1, \{1, \ldots, n\}).
\]

It remains to notice that, since \( A \) is monotone, if \( op(W_{I_1}) \prec N \) then \( op(W_I) \prec N \) for all \( I \) superset of \( I_1 \).

The proof for (b) is similar. □

**Proposition 3.7** For any monotone or antimonotone PDB-aggregates \( A \) of the form (10) where \( F_1, \ldots, F_n \) are atoms, \( A_{tr} \) is strongly equivalent to (13).

Proof. Let \( S = \{F_1 = w_1, \ldots, F_n = w_n\} \). Lemma 5.19 says that \( A_{tr} \) is classically equivalent to (13) for every formulas \( F_1, \ldots, F_n \) in \( S \). We can then prove the claim of this proposition using Lemma 5.16, by showing that both \( A_{tr} \) and (13) can be strongly equivalently rewritten as AND-OR combinations of

\[-F_1, \ldots, F_n, \top, \bot, \text{ if } A \text{ is monotone, and}\]

\[-\neg F_1, \ldots, \neg F_n, \top, \bot, \text{ if } A \text{ is antimonotone.}\]

About (13), this has already been shown in the proof of Proposition 3.6, while, about \( A_{tr} \), this is shown by Lemma 5.20. Indeed, each \( G(I, \{1, \ldots, n\}) \) is a (possibly empty) conjunction of terms of the form \( F_i \), and each \( G(\emptyset, I) \) is a (possibly empty) conjunction of terms of the form \( \neg F_i \), since each \( F_i \) is an atom. □

5.13 Proof of Proposition 3.8

We observe, first of all, that the definition of satisfaction of FLP-aggregates and FLP-programs in [Faber et al. 2004] is equivalent to ours. The definition of a reduct is different, however. Next lemma is easily provable by structural induction.

**Lemma 5.21.** For any nested expression \( F \) without negations and any two sets \( X \) and \( Y \) of atoms such that \( Y \subseteq X \), \( Y \models F^X \iff Y \models F \).

**Lemma 5.22.** For any FLP-aggregate \( A \) and any set \( X \) of atoms, if \( X \models A \) then

\[Y \models A^X \iff Y \models A.\]

Proof. Let \( A \) have the form (10). Since \( X \models A \), \( A^X \) has the form

\[op(\{F_1^X = w_1, \ldots, F_n^X = w_n\}) \prec N.\]

In case of FLP-aggregates, each \( F_i \) is a conjunction of atoms. Then, by Lemma 5.21, \( Y \models F_i^X \iff Y \models F_i \). The claim immediately follows from the definition of satisfaction of aggregates. □

**Proposition 3.8.** The stable models of a positive FLP-program under our semantics are identical to its stable models in the sense of [Faber et al. 2004].
Proof. It is easy to see that if $X \not\models \Pi$ then $X \not\models \Pi^X$ and $X \not\models \Pi^X$, so that $X$ is not a stable model under either semantics. Now assume that $X \models \Pi$. We will show that the two reducts are satisfied by the same subsets of $X$. It is sufficient to consider the case in which $\Pi$ contains only one rule

$$A_1 \land \cdots \land A_m \rightarrow a_1 \lor \cdots \lor a_n. \quad (49)$$

If $X \not\models A_1 \land \cdots \land A_m$ then $\Pi^X = \emptyset$, and $\Pi^X$ is the tautology

$$\bot \rightarrow (a_1 \lor \cdots \lor a_n)^X.$$

Otherwise, $\Pi^X$ is rule (49), and $\Pi^X$ is

$$A_1^X \land \cdots \land A_m^X \rightarrow (a_1 \lor \cdots \lor a_n)^X.$$

These two reducts are satisfied by the same subsets of $X$ by Lemmas 5.21 and 5.22. \qed

ACKNOWLEDGMENTS

We thank Vladimir Lifschitz and our anonymous reviewers for many useful comments on previous versions of this paper.

REFERENCES


Cabalar, P. and Ferraris, P. 2007. Propositional theories are strongly equivalent to logic programs.\textit{ Theory and Practice of Logic Programming 7.}


\textsuperscript{16}http://www.cs.utexas.edu/users/otto/papers.html .


*Ann. Pure Appl. Logic* 134, 1, 63–82.

Pearce, D. 1997. A new logical characterization of stable models and answer sets. In *Non-
Monotonic Extensions of Logic Programming (Lecture Notes in Artificial Intelligence 1216)*, 
J. Dix, L. Pereira, and T. Przymusinski, Eds. Springer-Verlag, 57–70.

Pearce, D. 1999. From here to there: Stable negation in logic programming. In *What Is Nega-

Pearce, D., Tompits, H., and Woltran, S. 2001. Encodings for equilibrium logic and logic 
programs with nested expressions. In *Proceedings of Portuguese Conference on Artificial In-
telligence (EPIA)*. 306–320.

Pelov, N., Denecker, M., and Bruynooghe, M. 2003. Translation of aggregate programs to 
normal logic programs. In *Proceedings Answer Set Programming*.

Peiov, N., Denecker, M., and Bruynooghe, M. 2007. Well-founded and stable semantics of 
logic programs with aggregates. *TPLP* 7, 3, 301–353.

Soininen, T. and Niemelä, I. 1998. Developing a declarative rule language for applications 
in product configuration. In *Proceedings of International Symposium on Practical Aspects of 

of the AAAI Spring Symposium on Answer Set Programming*.

Son, T. C., Pontelli, E., and Tu, P. H. 2007. Answer sets for logic programs with arbitrary 

Trajcevski, G., Baral, C., and Lobo, J. 2000. Formalizing (and reasoning about) the specifi-
cations of workflows. In *Proceedings of the Fifth IFCIS International conference on Cooperative 
Information Systems (CoopIS’2000)*.

*Theory and Practice of Logic Programming* 3(4,5), 609–622.

Received December 2008; revised January 2010; accepted June 2010