Künstliche Intelligenz manuscript No.
(will be inserted by the editor)

Answer Set Programming from a Logical Point of View

Pedro Cabalar · David Pearce · Agustín Valverde

Received: date / Accepted: date

Abstract In this paper we provide an introductory explanation of the underlying semantics of Answer Set Programming in terms of *Equilibrium Logic*. Rather than a thorough formal presentation of this formalism and its properties, we emphasize the intuitive meaning of its main logical definitions, explaining their effect on some example programs. We also overview some of the main extensions and relations to other logical approaches.

Keywords Answer Set Programming · Equilibrium Logic · Non-Classical Logics

1 Introduction

ASP is a rule-based paradigm sharing the same syntax as Prolog but with a different reading. Take a rule of the form:

`smoke :- fire.` (1)

ASP uses a *bottom-up* reading of (1): "smoke is produced by fire." That is, whenever `fire` belongs to our current set of beliefs or certain facts, `smoke` must also be included in that set too. On the contrary, Prolog's *top-down* reading could be informally stated as "to obtain smoke, we need fire." That is, the rule describes a procedure to get `smoke` as a goal which consists in pursuing `fire` as a new goal. Regardless of the application direction, it seems clear that rules have a conditional form with a right-hand condition (*body*) and a left-hand consequent (*head*) that in our example (1) respectively correspond to `fire` and `smoke`. Thus, a straightforward logical formalisation would be understanding (1) as the implication $\mathit{fire} \rightarrow \mathit{smoke}$ in classical propositional logic. This guarantees, for instance, that if we add *fire* as a program fact, we will get *smoke* as a conclusion (by application of *modus ponens*). So, the "operational" aspect of rule (1) can be captured by classical implication. However, classical logical semantics is not enough to cover the intuitive meaning of a program rule. If our program just contains rule (1), it is clear that *fire* is not satisfied, since *no rule can yield* that atom, and so, *smoke* is not obtained either. However, implication $\mathit{fire} \rightarrow \mathit{smoke}$, which amounts to the classically equivalent disjunction $\neg \mathit{fire} \vee \mathit{smoke}$, has three classical models: $\emptyset$ (both atoms false), $\{\mathit{smoke}\}$ and $\{\mathit{fire}, \mathit{smoke}\}$. Note that the two last models seem to consider situations in which *smoke* or *fire* could be arbitrarily *assumed as true*, even though the program provides *no way to prove them.* An important observation is that $\emptyset$ happens to be the smallest model (with respect to set inclusion). This model is interesting because, somehow, it reflects the principle of not adding arbitrary

Pedro Cabalar
University of Corunna, SPAIN
E-mail: cabalar@udc.es

David Pearce
Universidad Politécnica de Madrid, SPAIN
E-mail: david.pearce@upm.es

Agustín Valverde
University of Málaga, SPAIN
E-mail: a_valverde@ctima.uma.es

true atoms that we are not forced to believe, and it coincides with the expected meaning for a program just containing (1). The existence of a least classical model is, in fact, guaranteed for logic programs without negation (or disjunction), so-called *positive logic programs*, and so, it was adopted as the main semantics (van Emden and Kowalski, 1976) for logic programming until the introduction of negation. However, when negation came into play, classical logic was revealed to be insufficient again, even under the premise of minimal models selection. Suppose we have a program Π_1 consisting of the rules:

```
fill :- empty, not fire.                    (2)
empty.                                      (3)
```

where (2) means that we always *fill* our gas tank if it is *empty* and there is no evidence on *fire*, and (3) says that the tank is empty indeed. As before, *fire* cannot be proved (it is not head of any rule) and so, the condition of (2) is satisfied, producing *fill* as a result. The straightforward logical translation of (2) is $\mathit{empty} \wedge \neg \mathit{fire} \rightarrow \mathit{fill}$ that, in combination with fact (3), produces three models: $T_1 = \{\mathit{empty}, \mathit{fill}\}$, $T_2 = \{\mathit{empty}, \mathit{fire}\}$ and $T_3 = \{\mathit{empty}, \mathit{fire}, \mathit{fill}\}$. Unfortunately, there is no least classical model any more: both T_1 (the expected model) and T_2 are minimal with respect to set inclusion. After all, the previous implication is classically *equivalent* to $\mathit{empty} \rightarrow \mathit{fire} \vee \mathit{fill}$ which does not capture the directional behaviour of rule (2). The undesired minimal model T_2 is assuming *fire* to be true, although there is no way to prove that fact in the program. So, apparently, classical logic is too weak for capturing the meaning of logic programs in the sense that it provides the expected model(s), but also accepts other models (like T_2 and T_3) in which some atoms are abitrarily assumed to be true but not "justified by the program."

Suppose we had a way to classify true atoms distinguishing between those just being an *assumption* (classical model T) and those being also *justified* by program rules. In our intended models, the set of justified atoms should precisely coincide with the set of assumed ones in T. As an example, suppose our assumed atoms are $T_3 = \{\mathit{empty}, \mathit{fire}, \mathit{smoke}\}$. Any justification should include *empty* because of fact (3). However, rule (2) seems to be unapplicable, because we are currently assuming that *fire* is possibly true, $\mathit{fire} \in T_3$, and so '`not fire`' is not acceptable – there is some (weak) evidence about fire. As a result, atom *fill* is not necessarily justified and we can only derive $\{\mathit{empty}\}$, which is strictly smaller than our initial assumption T_3. Something similar happens for assumption $T_2 = \{\mathit{empty}, \mathit{fire}\}$. If we take classical model $T_1 = \{\mathit{empty}, \mathit{fill}\}$ instead as an initial assumption, then the body of rule (2) becomes applicable, since no evidence on *fire* can be found, that is, $\mathit{fire} \notin T_1$. As a result, the justified atoms are now $\{\mathit{empty}, \mathit{fill}\} = T_1$ and the classical model T_1 becomes the unique intended (stable) model of the program.

The method we have just used with the example can be seen as an informal description of the original definition of the stable models semantics (Gelfond and Lifschitz, 1988). This definition consisted of classical logic reinforced with an *extra-logical* program transformation (for interpreting negation) and then using application of rules to obtain the actually derived or justified information.

In the rest of the paper we show how it is possible to provide an equivalent definition that exclusively resorts to logical concepts but using a different underlying formalism, weaker than classical logic.

2 The logical point of view

2.1 Rule Satisfaction

At the basis of ASP is the computation of stable models or answer sets. Programs comprise *rules* that from now on we will regard as logical formulas. For instance a disjunctive rule may be written in the form

$$K_1 \vee \ldots \vee K_k \leftarrow L_1, \ldots L_m, \mathit{not}\ L_{m+1}, \ldots, \mathit{not}\ L_n \quad (4)$$

where the L_i and K_j are atomic formulas in predicate calculus. As we saw in the previous section, implication $\leftarrow$ is sometimes written `:-` when in textual form. In the logical semantics of ASP we will follow standard practice and assume that rules are variable-free or *ground* and that a program is replaced by its ground version where all rules have been uniformly instantiated. As a logical formula (4)) corresponds to

$$L_1 \wedge \ldots \wedge L_m \wedge \neg L_{m+1} \wedge \ldots \wedge \neg L_n \rightarrow K_1 \vee \ldots \vee K_k \quad (5)$$

If $k = 1$ the rule is called *normal.* If $k = 0$ the rule is a *constraint* and we regard it as the formula

$$L_1 \wedge \ldots \wedge L_m \wedge \neg L_{m+1} \wedge \ldots \wedge \neg L_n \rightarrow \bot \quad (6)$$

where $\bot$ is the symbol for logical falsity. As before the antecedent $L_1 \wedge \ldots \wedge L_m \wedge \neg L_{m+1} \wedge \ldots \wedge \neg L_n$ of the implication will be called the *body* of the rule, while the consequent $K_1 \vee \ldots \vee K_k$ is the rule *head*.

Following our introductory discussion we will approach the logical semantics of ASP by considering a second kind of truth that is in a sense weaker than certainty. This truth value corresponds to "not false but not provably true" and is a first approximation to the idea of "true by default". On this basis propositions can have three truth values, they may be certain or *true in the strong sense*, *false*, or not false but nevertheless *true in a weaker or less certain sense.*

What does it mean to say that a rule is *satisfied*? In the setting of classical (propositional) logic the rule (5) is satisfied as long as $K_1 \vee \ldots \vee K_k$ is true whenever the body $L_1 \wedge \ldots \wedge L_m \wedge \neg L_{m+1} \wedge \ldots \wedge \neg L_n$ is true. Regarding a classical model as a set of atoms T, this means that T trivially satisfies (5) if any one or more of $L_1, \ldots L_m, \ldots, \neg L_{m+1}, \ldots, \neg L_n$ is not true in T. But now in the presence of our third truth-value, satisfaction of a rule becomes more complex. Take:

$$\mathit{empty} \wedge \neg \mathit{fire} \rightarrow \mathit{fill} \tag{7}$$

corresponding to the logical notation of the simple rule (2). Suppose that *fill* were false, $\neg$*fire* were certain and *empty* were only true in the weaker sense. Then we would have to say that the rule is not satisfied despite the fact that its antecedent is not certain. To be satisfied we would expect *fill* to be at least weakly true, given that *empty* is. In other words, to see whether (7) is satisfied we have to check two conditions: that *fill* is certain whenever *empty* and $\neg$*fire* are certain, and also that *fill* is at least weakly true if *fill* is weakly true and $\neg$*fire* is also true.[1]

To account for this extra condition on rule satisfaction, we need an extended concept of model that reflects both kinds of truth. A convenient representation is to consider a pair of sets of ground atoms, $\langle H, T\rangle$, where H represents the certain atoms and T contains in addition to atoms in H also all those that are true only in the weaker sense[2]. In other words, T is our "initial assumption" while the subset H contains those atoms from T currently considered as justified. Let At be the collection of all atomic formulas in our given language.

[1] Notice that $\neg$*fire* is, read as "we have no evidence on fire," is either certain (when indeed *fire* is false) or false (when some evidence exists, ie , *fire* is certain or weakly true).

[2] These sets are informally known as "here" and "there". The reader familar with possible worlds semantics may think of them as the atoms verified at two worlds, 'here' and 'there'.

Then $H \subseteq T \subseteq At$ and all atoms in $At \backslash T$ are considered false in this model. A pair $\langle H, T\rangle$ as above is called an HT-*interpretation* and said to be *total* when $H = T$ (that is, when all assumptions are justified).

As we did for atoms, formulas can also be considered to be false, to be true by default or to be certain. We will use a satisfaction relation $\langle H, T\rangle \models \varphi$ to represent that $\langle H, T\rangle$ makes formula φ to be certain or justified. Sometimes, however, we may happen that this relation does not hold $\langle H, T\rangle \not\models \varphi$ while in classical logic satisfaction $T \models \varphi$ using the assumptions in T is true. Then, we may say that the formula is just weakly satisfied. Finally, when φ is not even classically satisfied by T, $T \not\models \varphi$, we can guarantee that the formula is false. Formally, the fact that an interpretation $\langle H, T\rangle$ *satisfies* a formula φ (or makes it certain), written $\langle H, T\rangle \models \varphi$, is recursively defined as follows:

– $\langle H, T\rangle \not\models \bot$
– $\langle H, T\rangle \models p$ iff $p \in H$
– $\langle H, T\rangle \models \varphi \wedge \psi$ iff $\langle H, T\rangle \models \varphi$ and $\langle H, T\rangle \models \psi$
– $\langle H, T\rangle \models \varphi \vee \psi$ iff $\langle H, T\rangle \models \varphi$ or $\langle H, T\rangle \models \psi$
– $\langle H, T\rangle \models \varphi \rightarrow \psi$ iff both (i) $T \models \varphi$ implies $T \models \psi$ and (ii) $\langle H, T\rangle \models \varphi$ implies $\langle H, T\rangle \models \psi$

By abuse of notation, we use '$\models$' both for classical and for HT-satisfaction: the ambiguity is resolved by the form of the left interpretation (a single set T for classical and a pair $\langle H, T\rangle$ for HT). We say that an interpretation $\langle H, T\rangle$ is a model of a theory (set of formulas) Γ iff $\langle H, T\rangle \models \varphi$ for all $\varphi \in \Gamma$. We say that a propositional theory Γ *entails* some formula φ, written $\Gamma \models \varphi$, if any model of Γ is also a model of φ. These definitions correspond to the so-called *logic of Here-and-There* (Heyting, 1930), stronger than intuitionistic logic but weaker than classical logic.

As we can see, everything is pretty standard excepting for the interpretation of implication, which imposes a stronger condition than in classical logic. In order to satisfy $\langle H, T\rangle \models \varphi \rightarrow \psi$, the standard condition would be (ii), that is, if the antecedent is certain, then the consequent must be certain too. In our case, however, we additionally require (i) which informally means that, if the antecedent is just true by default, then the consequent must also be so. Satisfaction of negation $\neg\varphi$ is not included above because it can be defined in terms of implication as the formula $\varphi \rightarrow \bot$. Using that abbreviation and after some analysis, it can be proved that $\langle H, T\rangle \models \neg\varphi$ amounts to $T \not\models \varphi$, that is, $\neg\varphi$ is certain when φ is false.

It is not difficult to see that, for total interpretations, $\langle T, T\rangle \models \varphi$ amounts to classical satisfaction $T \models \varphi$. An interesting property that can be proved from the definition of satisfaction is that $\langle H, T\rangle \models \varphi$ implies $T \models \varphi$ for any arbitrary formula φ. In other words, if an interpretation $\langle H, T\rangle$ makes φ certain, then it is also (classically) true for the set of assumptions T. This somehow generalises the condition $H \subseteq T$ (anything certain is also assumed true) to the case of arbitrary formulas. If we apply this property to our program example Π_1, this means that any of its models $\langle H, T\rangle \models \Pi$ must satisfy $T \models \Pi$ as well. As we saw, we have only three possibilities for the latter, T_1, T_2 and T_3. On the other hand, the program fact (3) fixes $\mathit{empty} \in H$. Now, take assumption $T_1 = \{\mathit{empty}, \mathit{fill}\}$. The only model we get is $\langle T_1, T_1\rangle$ because the other possible subset $H = \{\mathit{empty}\}$ of T_1 does not satisfy (2): *empty* is certain, *fire* is false, so we should get *fill*. Take T_2 instead. Apart from $\langle T_2, T_2\rangle$, in this case we also get a model $\langle H, T_2\rangle$ with $H = \{\mathit{empty}\}$. In such a case, *fire* is only assumed true, but not certain. As a result, the rule is satisfied because its condition $\neg\mathit{fire}$ is false (we have some evidence on *fire*) and so $\langle\{\mathit{empty}\}, T_2\rangle$ becomes a model. This is a clear evidence that our initial assumption adding *fire* is not necessarily certain when we check the program rules. In the case of $T_3 = \{\mathit{empty}, \mathit{fire}, \mathit{fill}\}$ we have a similar situation. Interpretations with $H = \{\mathit{empty}\}$, $H = \{\mathit{empty}, \mathit{fire}\}$ or $H = \{\mathit{empty}, \mathit{fill}\}$ are also models. Note that in all of them, the only atom that is always certain is *empty*, pointing out again that *fire* or *fill* are not necessarily certain (cannot be proved using the program rules).

We can now easily read off the truth conditions for a rule like (5) in an HT-model. But we have achieved more than this. Satisfaction is now inductively defined for all ground formulas and so we have a semantics that covers program rules that include arbitrary nestings of the logical operators. This, as we shall see, will be useful later on for understanding other kinds of ASP constructions.

2.2 Stable models are equilibrium models

From the logical point of view, it is now an easy task to define a stable model. The intuition is that we will be interested in cases where anything assumed true in set T eventually becomes necessarily certain, ie , $H = T$ is the only possibility for assumption T. As defined in (Pearce, 1997), given a ground, disjunctive program Π, a model $\langle H, T\rangle \models \Pi$ is an *equilibrium model* of Π, if (i) $H = T$; (ii) for any model $\langle H', T\rangle$ with $H' \subset H$, $\langle H', T\rangle \not\models \Pi$. (i) means that $\langle H, T\rangle$ (or simply $\langle T, T\rangle$) is a total model with no uncertainty, while (ii) is a minimality condition saying that Π has no 'lesser' model keeping T fixed as the set of non-false atoms. When $\langle T, T\rangle$ is an equilibrium model of program Π, we say that T alone is a *stable model* or *answer set* of Π. We can apply the same definitions to any arbitrary propositional theory.

Back to our example, the only stable model of Π_1 is the expected $T_1 = \{\mathit{empty}, \mathit{fill}\}$. This is because for the other two classical models $T_2 = \{\mathit{empty}, \mathit{fire}\}$ and $T_3 = \{\mathit{empty}, \mathit{fill}, \mathit{fire}\}$ we could see in the previous section that there were smaller sets H' that formed possible models of the program such as $\langle\{\mathit{empty}\}, T_2\rangle$ or $\langle\{\mathit{empty}\}, T_3\rangle$. In the case of T_1, however, the only obtained model is $\langle T_1, T_1\rangle$ and no smaller $H \subset T_1$ can be used to form a model.

We can also capture the stable model semantics in terms of theory or program *completions*. The intuitive idea is also based on checking derivability of arbitrary assumptions, but relies this time on a syntactic (fixpoint) definition. Given any propositional theory Γ, let $Cn(\Gamma)$ denote all its conclusions using the logic of HT. Formally, $Cn(\Gamma) \overset{\text{def}}{=} \{\varphi \mid \Gamma \models \varphi \text{ in HT}\}$. Now, assume we have some program or theory Π. We can think about each potential stable model of Π in terms of the set of formulas Γ that we could derive from it. Suppose we assume that some set of formulas Γ is a potential candidate for being (the conclusions of) a stable model. We cannot directly add Γ to Π, because this would transform any assumption $\varphi \in \Gamma$ into an immediately certain or derivable formula. However, we can add instead the negation $\neg\varphi$ of any $\varphi \notin \Gamma$ since there is no evidence of φ in our current assumption Γ. A set of formulas Γ is said to be a *completion* of a theory Π iff:

$$Cn(\Pi \cup \{\neg\varphi : \varphi \notin \Gamma\}) = \Gamma. \qquad (8)$$

In other words, if we collect the negation of all formulas not in the assumption Γ and we add them as hypotheses to Π, the information we can derive (using HT consequence) eventually coincides with the initial assumption Γ. Each completion is the set of formulas true in a stable model, and each stable model corresponds in this way to one completion. Obviously, there may be different assumptions Γ that satisfy the fixpoint condition (8) or none at all, in the same way there may be different stable models of a program, or none at all either. Condition (8) can be relaxed by just ranging φ on literals (that is, any atom p or its negation $\neg p$) not belonging to Γ, and we still get the same result. Moreover, if Π is a disjunctive logic program, then it suffices to let φ range over atoms (Pearce, 1999). These properties continue to hold if we replace Cn by weaker consequence relations, such as that of intuitionistic or other intermediate logics contained in HT. [3]

2.3 Program equivalence

A central logical concept in the analysis of programs is that of program *equivalence*. In a weak sense, two programs, Π_1 and Π_2 are equivalent if they have the same answer sets. However, since we are in a setting of nonmonotonic semantics, this kind of equivalence is not robust. It does not give us a logical replacement theorem allowing us to substitute one program for another in any context: adding new rules or new facts to such weakly equivalent programs may result in their becoming non-equivalent. To understand this subtle difference, consider a program Π_2 containing the facts *empty* and *fill*. The unique stable model of Π_2 is obviously $T_1 = \{\mathit{empty}, \mathit{fill}\}$ and thus coincides with the only stable model of Π_1. However, they do not seem to capture an interchangeable piece of knowledge. This becomes clear when we add new information like, for instance, if we are told that *fire* is detected around the gas station. Program $\Pi_1 \cup \{\mathit{fire}\}$ will have the unique stable model $\{\mathit{empty}, \mathit{fire}\}$ since rule (2) is not applicable any more, whereas program $\Pi_2 \cup \{\mathit{fire}\}$ will produce the unique stable model $\{\mathit{empty}, \mathit{fill}, \mathit{fire}\}$, since in that case, filling the tank is unconditional.

Research in ASP has explored stronger concepts of equivalence, where substitutability in specific contexts is supported. Two programs, Π_1 and Π_2 are said to be *strongly* equivalent, if, for any program, Π, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets; in other words they are interchangeable without semantic loss in any context. Strong equivalence has a simple, logical characterisation: programs are strongly equivalent if and only if they have the same HT-models (Lifschitz et al, 2001). Since the logic HT of here-and-there has a sound and complete proof theory, there is a calculus providing a simple logical test for strong equivalence. This can be applied in program analysis, for example to replace complex rules by equivalent but simpler ones.[4]

An important property, first shown by (Cabalar and Ferraris, 2007), is that any arbitrary propositional theory is strongly equivalent to a general form of logic program, whose rules are of the type:

$$L_1 \wedge \ldots \wedge L_m \wedge \neg L_{m+1} \wedge \ldots \wedge \neg L_n \rightarrow$$
$$\rightarrow K_1 \vee \ldots \vee K_k \vee \neg K_{k+1} \vee \ldots \vee \neg K_j$$

(atomic L, K). In other words a disjunctive program allowing negation in the rule heads. How to construct an equivalent general program is studied in (Cabalar et al, 2005). As a simple example, the formula $\mathit{empty} \vee \mathit{offer} \rightarrow (\neg \mathit{fire} \rightarrow \mathit{fill} \wedge \mathit{pay})$ is strongly equivalent to the logic program:

$$\begin{array}{rr} \mathit{empty} \wedge \neg \mathit{fire} \rightarrow \mathit{fill} & \mathit{empty} \wedge \neg \mathit{fire} \rightarrow \mathit{pay} \\ \mathit{offer} \wedge \neg \mathit{fire} \rightarrow \mathit{fill} & \mathit{offer} \wedge \neg \mathit{fire} \rightarrow \mathit{pay} \end{array}$$

2.4 Aggregates in ASP

Logical formalisations can be useful to analyse typical constructs in ASP that provide more flexibility than plain propositional programs, such as aggregate operators, which are applied to sets of values. As an example, suppose we want to specify that a person authoring more than n books is considered to be a writer. For simplicity, assume that our logic program only talks about some fixed person and that we write the rule

$$\mathtt{count}\{X : b(X)\} \geq n \rightarrow \mathit{writer} \tag{9}$$

so that $b(X)$ means that the author wrote some book X. Different semantics for ASP aggregates have been proposed in the literature, many of them relying on syntactic transformations. An interesting alternative proposed in (Ferraris, 2011) is simply translating an aggregate as a logical formula interpreted in HT. Going on with our example, suppose that $n = 2$ and that our program domain includes book constant names h (*Hamlet*), o (*Othello*) and m (*Macbeth*). Without entering into detail, Ferraris' translation of (9) produces the formula:

$$\begin{array}{l}(h \vee o \vee m) \wedge (h \rightarrow o \vee m) \\ \quad \wedge (o \rightarrow h \vee m) \wedge (m \rightarrow h \vee o) \rightarrow \mathit{writer}\end{array} \tag{10}$$

[3] See (Cabalar et al, 2017a) for more detailed information and references.

[4] Using HT-logic we can only test the equivalence of ground rules and programs. In practice it makes sense to apply a first-order, quantified version of HT that captures the strong equivalence of programs with variables. For details, see (Lifschitz et al, 2007; Mints, 2010).

The antecedent of this formula requires that we author some book ($h \vee o \vee m$) and that if we authored any of them, we also write, at least, one of the other two. As we can see, an interesting feature in this translation is that, in its general case, it makes use of nested implications, something not considered before in traditional logic programming, but perfectly possible once we have a general semantics for arbitrary theories provided by HT and Equilibrium Logic. In our particular example, formula (10) can be rewritten (under HT-equivalence) as the more readable logic program:

$$h \wedge o \rightarrow \mathit{writer}$$
$$h \wedge m \rightarrow \mathit{writer}$$
$$o \wedge m \rightarrow \mathit{writer}$$

One valuable feature of logical encodings is that they sometimes allow analysing different semantic alternatives using a same formal language. For instance, one of the most recent alternative semantics for aggregates, proposed by Gelfond and Zhang (2014), has been recently proved (Cabalar et al, 2017b) to correspond to a different translation into HT formulas. Under this translation, our example rule (9) corresponds, instead, to the rules:

$$h \wedge o \wedge \neg m \rightarrow \mathit{writer}$$
$$h \wedge m \wedge \neg o \rightarrow \mathit{writer}$$
$$o \wedge m \wedge \neg h \rightarrow \mathit{writer}$$
$$h \wedge m \wedge o \rightarrow \mathit{writer}$$

that are classically equivalent to Ferraris' translation, but not HT-equivalent in the general case. Once the two semantics are captured as logical formulas in HT, studying their relationship amounts to a logical analysis that has allowed, for instance, proving some correspondence results in (Cabalar et al, 2017b).

To illustrate a difference between the two semantics, suppose we had instead the rule

$$\texttt{count}\{X : b(X)\} \geq n \rightarrow b(a) \tag{11}$$

meaning that if we write at least n books we also write an autobiography a. Obviously, the autobiography is also a book, and so, the aggregate contains a kind of self-reference. Now, suppose that $n = 0$, that is, any number of books can be accepted, including no one. Ferraris' HT-translation for (11) corresponds to:

$$\top \rightarrow b(a) \tag{12}$$

as it considers the aggregate as tautological, and so, if we only have rule (11), we get that the only stable model is $\{b(a)\}$. On the other hand, Gelfond-Zhang's HT-translation would correspond to:

$$b(a) \vee \neg b(a) \rightarrow b(a)$$

where the antecedent *is not an HT-tautology.* In fact, this formula is strongly equivalent to the logic program:

$$b(a) \rightarrow b(a)$$
$$\neg b(a) \rightarrow b(a)$$

where the first rule is a tautology, but the second one forms a well-known negative loop that produces no stable model (if no other rules are added). As we can see, Gelfond-Zhang's semantics considers, in this way, that rule (11) is somehow paradoxical or ill-formed: for establishing the aggregate value, we depend on the conclusion $b(a)$ that in its turn depends on the aggregate again.

3 Extensions

3.1 Strong negation

One of the simplest extension of the basic language of ASP is the addition of a second negation operator to represent explicit falsity. The most appropriate name for this is *strong negation.* It was introduced into logic programming independently by (Pearce and Wagner, 1991, 1990) and (Gelfond and Lifschitz, 1990). In logical terms strong negation, symbolised by '$\sim$', validates these characteristic formulas:

N1. $\sim (\alpha \rightarrow \beta) \leftrightarrow \alpha \wedge \sim\beta$
N2. $\sim(\alpha \wedge \beta) \leftrightarrow \sim\alpha \vee \sim \beta$
N3. $\sim(\alpha \vee \beta) \leftrightarrow \sim\alpha \wedge \sim\beta$
N4. $\sim \sim\alpha \leftrightarrow \alpha$
N5. $\sim\neg\alpha \leftrightarrow \alpha$
N6. (for atomic α) $\sim\alpha \rightarrow \neg\alpha$

Here '$\alpha \leftrightarrow \beta$' abbreviates $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$ and we suppose that $\neg\alpha$ as before is equivalent to $\alpha \rightarrow \bot$.

Despite its simplicity, strong negation has often been misunderstood.[5] It was introduced into logic by David Nelson (Nelson, 1949). Later **N1–N6** were proposed by (Vorob'ev, 1952a,b) to provide an axiom system for strong negation. If added to intuitionistic logic or its extensions (ie any superintuitionistic logic) these axioms result in a conservative extension of that logic.

[5] It is slightly unfortunate that (Gelfond and Lifschitz, 1990) called it "classical" negation.

Answer set programs with strong negation comprise rules like (5) in the disjunctive case, where now L, K range over strong or *objective* literals, ie. atoms possibly prefixed by strong negation. The semantics of HT-models is essentially as before, except that a model $\langle H, T\rangle$ now comprises sets of objective literals, where $H \subseteq T$ and T does not contain any inconsistent pairs $(A, {\sim}A)$ of atoms. Additional clauses govern the satisfaction of formulas in strong negation, models while the equilibrium condition defining answer sets is the same as before (Pearce, 1997).

In the logical semantics of answer sets it is important to make clear the different behaviours of '$\neg$' and '$\sim$'. For instance while the law of double negation holds for strong but not intuitionistic negation:
$\vdash {\sim}{\sim}\alpha \leftrightarrow \alpha$
$\not\vdash \neg\neg\alpha \leftrightarrow \alpha$
the property of *contraposition* holds for weak but not for strong negation:
$\alpha \rightarrow \beta \ \vdash\ \neg\beta \rightarrow \neg\alpha$
$\alpha \rightarrow \beta \ \not\vdash\ {\sim}\beta \rightarrow {\sim}\alpha$
Notice that a program rule such as $\neg B \rightarrow A$ has $\neg A \rightarrow \neg\neg B$ as its contrapositive.

The fact that strong negation appears in program rules directly before atoms is not a genuine restriction. Any arbitrary formula in a logic with strong negation is equivalent to one where all occurrences of '$\sim$' are driven-in to stand before atoms. The Vorob'ev axioms show how to perform the transformation.

There is another feature of strong negation that makes it attractive from a computational point of view. Programs with strong negation can be reduced to ordinary programs in an extended language by replacing any strongly negated atom ${\sim}p$ by a new symbol, say p' and adding the formula $p' \rightarrow \neg p$. This reduction technique was first used by Gurevich (Gurevich, 1977) for first-order logic and later independently rediscovered by (Gelfond and Lifschitz, 1990) in their treatment of logic programs.

3.2 Hybrid systems

Answer set programming can be combined with other forms of knowledge representation and reasoning, and some such hybrid systems have been implemented and are in practical use. In some case, hybrid approaches lend themselves to treatment from a logical perspective by extending the logical framework that we have been discussing so far. An early and conceptually convincing approach to combining answer set semantics with other systems was proposed by Riccardo Rosati (Rosati, 2005, 2006), in particular in the form $\mathcal{DL} + log$. This work is directed at combining logic programs with ontologies formulated in description logics. However, Rosati's semantics can be applied to conjoin programs with other kinds of data and knowledge structures – including first-order theories – that receive a classical interpretation. A key feature is that in such a hybrid system some terms and relations may appear both within, say, an ontology or knowledge base, and also within a set of program rules. The meaning of such terms has therefore to be determined both from the role they play in the knowledge structure and from their interpretation within the logic program. From a logical point of view, the former is essentially classical and monotonic while the latter is nonclassical and nonmonotonic.

Rosati's semantics is somewhat complicated, and for details we refer to the papers just mentioned. However, in the framework of first-order equilibrium logic there is an equivalent and simpler characterisation. It suffices to take the equilibrium models of the conjunction of the two components of the hybrid system, say a first-order theory $\mathcal{T}$ and a logic program Π, but to add the excluded middle, $R(x) \vee \neg R(x)$, for those relations R that belong to the classical theory.[6] For details, see (de Bruijn et al, 2010).

Another approach to combining ontologies and rules under stable model semantics is that of description logic programs (dl-programs) proposed by (Eiter et al, 2004). This can also be captured in a logical framework, but with a more far-reaching extension of the basic language. In this case programs comprise so-called dl-rules that may include special dl-atoms in the rule body. These atoms refer to external relations belonging to a description logic program or ontology and are thus evaluated externally. Their stable model semantics can also be characterised within equilibrium logic, extended in a suitable way to accommodate the dl-atoms (Fink and Pearce, 2010).

[6] Notice that the meaning of such a relation is still co-determined by the two components $\mathcal{T}$ and Π because normally not all models of $\mathcal{T}$ will be able to be enriched into equilibrium models of $\mathcal{T} \cup \Pi$.

3.3 Temporal theories

Many applications of ASP involve temporal reasoning in dynamic domains and solve different kinds of problems such as temporal simulation, temporal explanation, planning or even online execution of reactive systems with applications in robotics. An immediate choice to cover these features from a logical point of view was extending Equilibrium Logic to cope with modal temporal operators from *Linear-Time Temporal Logic* (LTL) (Kamp, 1968) such as $\bigcirc\alpha$ ("α holds at next state"), $\Box\alpha$ ("α always holds from now on") or $\Diamond\alpha$ ("α holds at some point from now on"). This was proposed in (Cabalar and Pérez, 2007), receiving the name of *Temporal Equilibrium Logic* (TEL). The semantics of TEL is defined starting from a temporal extension of HT, we call THT. Given a finite propositional signature At, an LTL-*interpretation* $\mathbf{T}$ is an infinite sequence of sets of atoms, $T_0, T_1, \ldots$ with $T_i \subseteq At$ for all $i \geq 0$. Given two LTL-interpretations $\mathbf{H}, \mathbf{T}$ we define $\mathbf{H} \leq \mathbf{T}$ as: $H_i \subseteq T_i$ for all $i \geq 0$. A THT-*interpretation* $\mathbf{M}$ for At is a pair of LTL-interpretations $\langle \mathbf{H}, \mathbf{T} \rangle$ satisfying $\mathbf{H} \leq \mathbf{T}$. A THT-interpretation is said to be *total* when $\mathbf{H} = \mathbf{T}$. Given an interpretation $\mathbf{M} = \langle \mathbf{H}, \mathbf{T} \rangle$, we recursively define when $\mathbf{M}$ *satisfies* a temporal formula φ at some state $i \in \mathbb{N}$ as:

- $\mathbf{M}, i \models p$ iff $p \in H_i$ with p an atom
- $\wedge, \vee, \bot, \rightarrow$ as in the propositional case of HT, using $\mathbf{H}$ and $\mathbf{T}$ instead of H and T
- $\mathbf{M}, i \models \bigcirc\, \varphi$ iff $\mathbf{M}, i{+}1 \models \varphi$
- $\mathbf{M}, i \models \Box\varphi$ iff for all $k \geq i$ s.t. $\mathbf{M}, k \models \varphi$
- $\mathbf{M}, i \models \Diamond\varphi$ iff for some $k \geq i$ such that $\mathbf{M}, k \models \varphi$.

An interpretation $\mathbf{M}$ is a *temporal equilibrium model* of a theory Γ if it is a total model of Γ, that is, $\mathbf{M} = \langle \mathbf{T}, \mathbf{T} \rangle \models \Gamma$, and there is no $\mathbf{H} < \mathbf{T}$ such that $\langle \mathbf{H}, \mathbf{T} \rangle \models \Gamma$. An LTL-interpretation $\mathbf{T}$ is a *temporal stable model* (TS-model) of a theory Γ iff $\langle \mathbf{T}, \mathbf{T} \rangle$ is a temporal equilibrium model of Γ. For a recent survey on TEL containing more results and bibliography, see (Cabalar, 2015).

3.4 Other extensions

As is clear from the temporal extension of ASP, a positive feature of the logical perspective on answer set programming is that it suggests how new logical constructs and operators can be added to the basic language of programs. Aside from the temporal extension of the language, we can mention also the extension of first-order equilibrium logic for dealing with evaluable functions proposed in (Cabalar, 2011), which directly incorporates the study of partial functions in intuitionistic logic made by (Scott, 1979) to the case of HT and equilibrium models. Another example is *Epistemic Equilibrium Logic* due to (Fariñas del Cerro et al, 2015), which starts from modal extensions of intuitionistic logic to provide a fully logical, alternative semantics to so-called *epistemic specifications* originally proposed by Gelfond (1991). This extension allows one to include expressions like $\mathbf{K}p$ (resp. $\mathbf{M}p$) in rule bodies to stand for "atom p holds in any (resp. some) stable model of this program."

4 Other logical perspectives

Not surprisingly answer set programming can be viewed from other logical perspectives. For instance from the very early days of stable model semantics there was a strong interest in relating stable models to other non-monotonic formalisms, such as default and autoepistemic logics. It turns out that several of these alternative logical views derive directly from our here-and-there perspective in virtue of reduction or embedding techniques between logical systems, some of which are well-known and predate ASP by many years. Without going into too many technical details, let us briefly review two such methods: one involves a reduction to second-order logic, the other a translation or translations into modal logics.

4.1 Reductions to second-order logic

In this logical perspective we deal with QBFs or *quantified boolean formulas.* For a full description of QBFs and their the semantics, see eg (Kleine Büning et al, 1995). Suppose that our ASP language is without strong negation and that V is the set of propositional variables or atoms. Denote by V' the disjoint alphabet $V' = \{p' : p \in V\}$. For any formula φ with variables from V, let φ' be the result of replacing each variable $p \in V$ by p'. If $V = \{p_1, \ldots, p_n\}$ and $U = \{q_1, \ldots, q_n\}$ are indexed sets of atoms, then let $V \leq U$ abbreviate $\bigwedge_{i=1}^{n}(p_i \rightarrow q_i)$, and $V < U$ abbreviate $(V \leq U) \& \neg(U \leq V)$. For any formula φ with variables from V let φ^* be a translation defined recursively as follows.

1. if φ is an atom or $\bot$, then $\varphi^* = \varphi$;

2. if $\varphi = (\varphi_1 \circ \varphi_2)$, for $\circ \in \{\wedge, \vee\}$, then $\varphi^* = {\varphi_1}^* \circ {\varphi_2}^*$;
3. if $\varphi = (\varphi_1 \rightarrow \varphi_2)$, then $\varphi^* = ({\varphi_1}^* \rightarrow {\varphi_2}^*) \wedge (\varphi_1' \rightarrow \varphi_2')$.

Suppose that φ is a formula with atoms in V and that $H, T \subseteq V$ are interpretations. Then it can be shown (Pearce et al, 2001) that $\langle H, T\rangle$ is an HT-model of φ if and only if $H \cup T'$ is a (classical) model of $(V \leq V') \wedge \varphi^*$.

The primed formulas in φ^* play the role of formulas evaluated in the 'T' component of the HT-model while unprimed formulas correspond to those evaluated at 'H'. The property $V \leq V'$ expresses the requirement that truth persists from 'H' to 'T'.

Again suppose that φ is a formula with atoms in V. Then, $\langle T, T\rangle$ is an equilibrium model of φ if and only T' is a model of

$$\varphi' \wedge \neg\exists V\big((V < V') \wedge \varphi^*\big). \tag{13}$$

Formula (13) is a QBF. These formulas generalise ordinary propositional formulas by allowing quantification over propositional variables. Informally, a QBF of the form $\forall p\, \exists q\, \Phi$ states that for all truth assignments of p there is a truth assignment of q such that Φ is true. QBFs can also be used to express properties such as whether a theory has an equilibrium model or whether a formula is an equilibrium consequence of a given theory, ie true in all equilibrium or stable models.

The presence of efficient QBF-solvers means that these encodings can be used as a basis for implementing equilibrium logic. More importantly, the complexity classes associated with QBFs of different kinds are well-understood, and so QBF-reductions provide useful information about the complexity of the various reasoning tasks being encoded.

The table below summarises some complexity results obtained by analysing the quantifier order of the different QBF encodings. Each row associates a complexity class for a decision problem with respect to disjunctive logic programs in the left column and propositional theories in HT-logic, in the right column. In each case the decision problem is *complete* for the class in question. From top to bottom the decision problems are: existence of an HT-model, existence of an equilibrium model, whether a formula is an equilibrium consequence of a theory or program, and lastly the problems of checking (weak) equivalence and strong equivalence.[7]

	LPs	*Theories*
model existence	NP	NP
equil model existence	Σ_2^P	Σ_2^P
equil consequence	Π_2^P	Π_2^P
equivalence	Π_2^P	Π_2^P
strong equivalence	$coNP$	$coNP$

Notice that there is no increase of complexity when moving from (disjunctive) logic programs to general programs or arbitrary (propositional) theories.

The above reduction can also be applied in a straightforward manner to first-order theories, thereby obtaining for them a characterisation of stable or equilibrium models in second-order logic (Ferraris et al, 2007).

4.2 Modal embeddings

It is well-known that Heyting's intutionistic logic $\mathcal{H}$ can be embedded into the modal logic $S4$ using a translation first proposed by (Gödel, 1933) and later studied in more depth by (McKinsey and Tarski, 1948). There are different variants of the Gödel-McKinsey-Tarski translation. One version merely consists in prefixing each subformula of $\mathcal{H}$ by a necessity operator '$\Box$'; let us denote this translation by τ. Then we have

$$\vdash_{\mathcal{H}} \varphi \Leftrightarrow \vdash_{S4} \tau(\varphi), \tag{14}$$

(where $\vdash$ with subscripts denotes theoremhood).

From the mid-1970s logicians began to study a second translation, this time from modal logics to modal logics, showing how reflexive logics[8] could be embedded into non-reflexive ones. This is known as the *splitting* translation, which we may denote by the superscript operator '$^+$'. The effect is to replace each occurrence of $\Box$ by $\Box^+$ where $\Box^+\varphi$ abbreviates $\varphi \wedge \Box\varphi$, leaving other formulas unchanged.

Applying the splitting translation to $S4$ for example yields an embedding into a model logic known as $wK4$

[7] The main references are as follows. For the complexity of satisfiability in many-valued logics such as **HT**, see (Mundici, 1987). For the complexity of reasoning tasks associated with disjunctive logic programs, see (Eiter and Gottlob, 1995). For the strong equivalence of logic programs, see (Pearce et al, 2001) and also independently some results of (Lin, 2002) and (Turner, 2003). For the full details of the reduction method sketched here, see (Pearce et al, 2001, 2009).

[8] Ie. logics captured semantically by possible worlds frames with a reflexive accessibility relation

that has a natural topological interpretation (Esakia, 1976, 2004). While extensions of $S4$ have often been studied as epistemic logics of knowledge, extensions of $wK4$ may be considered as candidates for doxastic logics of belief. Among them is the well-known $KD45$.

The translations τ and $+$ continue to be applicable not just to the base logics $\mathcal{H}$ and $S4$, but to their extensions as well, yielding new embeddings into stronger modal logics. A modal companion of HT-logic (ie. a modal logic into which HT can be embedding via τ) is $SW5$[9] a weakening of the logic $S5$. Via the splitting translation, $SW5$ in turn can be embedded into $KD45$. It transpires that these embeddings can be lifted to the nonmonotonic case. Regarding equilibrium logic as the nonmonotonic extension of HT, let us add a star * to denote the nonmonotonic versions of modal logics. Then we have the following commuting diagram:

$$\begin{tikzcd}
EL \arrow[r, "\tau"] & SW5^* \arrow[r, "+"] & KD45^* \\
HT \arrow[u, dotted] \arrow[r, "\tau"] & SW5 \arrow[u, dotted] \arrow[r, "+"] & KD45 \arrow[u, dotted] \\
\mathcal{H} \arrow[u] \arrow[r, "\tau"] & S4 \arrow[u] \arrow[r, "+"] & wK4f \arrow[u]
\end{tikzcd}$$

Here the right arrows indicate embeddings while the solid upward arrows indicate logical extensions (strengthenings). The broken upward arrows denote nonmonotonic extensions. Note that $KD45^*$ is none other than autoepistemic logic. In the early days of stable model semantics several different formal embeddings into nonmonotonic modal logics were discovered by (Gelfond and Lifschitz, 1988; Przymusinski, 1991; Lifschitz and Schwarz, 1993; Marek and Truszczynski, 1993b; Chen, 1993) and others. These were established in a piecemeal and *ad hoc* manner. However, with hindsight we can see that they can be systematically derived in a straightforward way from the diagram above using τ and $^+$; details can be found in (Pearce and Uridia, 2008).

5 Some literature and concluding remarks

HT-logic was first described by (Heyting, 1930) in his formal analysis of intuitionistic logic. Although he used HT primarily as a technical device, interest in this logic quickly grew. It appeared again in (Gödel, 1932) as a 3-valued logic, and was analysed in depth in a work of Łukasiewicz presented in 1938 (Łukasiewicz, 1941). He provided the first axiomatisation of HT, as an extension of intuitionistic logic, and showed that disjunction is definable in the logic. Later the logic was studied in Russia by (Smetanich, 1960) and in Japan by (Umezawa, 1959; Hosoi, 1966). (Umezawa, 1959) provided a simpler axiom scheme, and completeness of this was shown by (Hosoi, 1966). Although the logic has various names, the term "here-and-there" became common, probably from the 1960s when its semantics became well-understood in the setting of possible worlds. Among quantified versions of HT we can mention (Ono, 1983; Pearce and Valverde, 2004; Mints, 2010).

Equilibrium logic was presented as a nonmonotonic extension of HT in (Pearce, 1997). It is described in much more detail in (Pearce, 2006). First-order versions can be found in (Pearce and Valverde, 2004, 2005). A variant designed to incorporate intensional functions into the ASP language can be found in (Cabalar et al, 2014). As we have seen in Section 2.4, the logical translation of aggregate operators involves conjunctions or disjunctions of expressions whose length depends on the size of the program domain. When this domain is infinite, propositional conjunction and disjunctions need to be extended to the infinitary case. The corresponding *Infinitary Equilibrium Logic* is studied in (Harrison et al, 2017).

[9] See (Marek and Truszczynski, 1993a).

References

Cabalar P (2011) Functional answer set programming. Theory and Practice of Logic Programming 11(2-3):203–233

Cabalar P (2015) Stable models for temporal theories. In: Proc. of the 13th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning LPNMR 2015, Lexington, KY, USA, September 27-30, pp 1–13

Cabalar P, Ferraris P (2007) Propositional theories are strongly equivalent to logic programs. Theory and Practice of Logic Programming 7(6):745–759

Cabalar P, Pérez G (2007) Temporal equilibrium logic: a first approach. In: Proc. of the 11th International Conference on Computer Aided Systems Theory, (EUROCAST'07). LNCS (4739), pp 241–248

Cabalar P, Pearce D, Valverde A (2005) Reducing propositional theories in equilibrium logic to logic programs. In: EPIA, Springer, Lecture Notes in Computer Science, vol 3808, pp 4–17

Cabalar P, Fariñas del Cerro L, Pearce D, Valverde A (2014) A free logic for stable models with partial intensional functions. In: JELIA, Springer, Lecture Notes in Computer Science, vol 8761, pp 340–354

Cabalar P, Fandinno J, del Cerro LF, Pearce D, Valverde A (2017a) On the properties of atom definability and well-supportedness in logic programming. Lecture Notes in Computer Science 10423:624–636

Cabalar P, Fandinno J, Schaub T, Schellhorn S (2017b) Gelfond-Zhang Aggregates as propositional formulas. In: Proc. of the 14th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning, LPNMR17, Espoo, Finland, July 3-6, pp 117–131

Chen J (1993) Minimal knowledge + negation as failure = only knowing (sometimes). In: LPNMR, MIT Press, pp 132–150

de Bruijn J, Pearce D, Polleres A, Valverde A (2010) A semantical framework for hybrid knowledge bases. Knowl Inf Syst 25(1):81–104

Eiter T, Gottlob G (1995) On the computational cost of disjunctive logic programming: Propositional case. Ann Math Artif Intell 15(3-4):289–323

Eiter T, Lukasiewicz T, Schindlauer R, Tompits H (2004) Combining answer set programming with description logics for the semantic web. In: KR, AAAI Press, pp 141–151

van Emden MH, Kowalski RA (1976) The semantics of predicate logic as a programming language. Journal of the ACM 23:733–742

Esakia L (1976) The modal logic of topological spaces. Preprint, Georgian Academy of Sciences pp 1–22

Esakia L (2004) Intuitionistic logic and modality via topology. Ann Pure & Applied Log 127:155–170

Fariñas del Cerro L, Herzig A, Su EI (2015) Epistemic equilibrium logic. In: IJCAI, AAAI Press, pp 2964–2970

Ferraris P (2011) Logic programs with propositional connectives and aggregates. ACM Trans Comput Log 12(4):25

Ferraris P, Lee J, Lifschitz V (2007) A new perspective on stable models. In: Proceedings of the 20th Intl. Joint Conference on Artificial Intelligence (IJCAI'07), Hyderabad, India, to appear

Fink M, Pearce D (2010) A logical semantics for description logic programs. In: JELIA, Springer, Lecture Notes in Computer Science, vol 6341, pp 156–168

Gelfond M (1991) Strong introspection. In: Proc. of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, Volume 1., pp 386–391

Gelfond M, Lifschitz V (1988) The stable models semantics for logic programming. In: Proc. of the 5th Intl. Conf. on Logic Programming, pp 1070–1080

Gelfond M, Lifschitz V (1990) Logic programs with classical negation. In: ICLP, MIT Press, pp 579–597

Gelfond M, Zhang Y (2014) Vicious circle principle and logic programs with aggregates. Theory and Practice of Logic Programming 14(4-5):587–601

Gödel K (1932) Zum intuitionistischen aussagenkalkül. Anzeiger der Akademie der Wissenschaften Wien, mathematisch, naturwissenschaftliche Klasse 69:65–66

Gödel K (1933) Eine interpretation de intuitionistischen aussagenkalküls. Ergebnisse Math Colloq 4:39–40

Gurevich Y (1977) Intuitionistic logic with strong negation. Studia Logica 36(1–2):49–59

Harrison A, Lifschitz V, Pearce D, Valverde A (2017) Infinitary equilibrium logic and strongly equivalent logic programs. Artif Intell 246:22–33

Heyting A (1930) Die formalen Regeln der intuitionistischen Logik. Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse pp 42–56

Hosoi T (1966) The axiomatization of the intermediate propositional systems S_n of Gödel. Journal of the Faculty of Science of the University of Tokyo 13:183–187

Kamp JA (1968) Tense logic and the theory of linear order. PhD thesis, University of California at Los Angeles

Kleine Büning H, Karpinski M, Flögel A (1995) Resolution for quantified boolean formulas. Inf Comput 117(1):12–18

Lifschitz V, Schwarz G (1993) Extended logic programs as autoepistemic theories. In: LPNMR, MIT Press, pp 101–114

Lifschitz V, Pearce D, Valverde A (2001) Strongly equivalent logic programs. ACM Trans Comput Log 2(4):526–541

Lifschitz V, Pearce D, Valverde A (2007) A characterization of strong equivalence for logic programs with variables. In: LPNMR, Springer, Lecture Notes in Computer Science, vol 4483, pp 188–200

Lin F (2002) Reducing strong equivalence of logic programs to entailment in classical propositional logic. In: Fensel D, Giunchiglia F, McGuinness DL, Williams MA (eds) Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002, Morgan Kaufmann, pp 170–176

Łukasiewicz J (1941) Die logik und das grundlagenproblem. Les Entreties de Zürich sur les Fondaments et la Méthode des Sciences Mathématiques 6-9, (1938) 12:87–100

Marek W, Truszczynski M (1993a) Nonmonotonic Reasoning: Context-Dependent Reasoning. Springer

Marek W, Truszczynski M (1993b) Reflective autoepistemic logic and logic programming. In: LPNMR, MIT Press, pp 115–131

McKinsey JJ, Tarski A (1948) Some theorems about the sentential calculi of lewis and heyting. J Symb Log 13:1–15

Mints G (2010) Cut-free formulations for a quantified logic of here and there. Ann Pure Appl Logic 162(3):237–242

Mundici D (1987) Satisfiability in many-valued sentential logic is np-complete. Theor Comput Sci 52:145–153

Nelson D (1949) Constructible falsity. Journal of Symbolic Logic 14(2):16–26

Ono H (1983) Model extension theorem and craigs interpolation theorem for intermediate predicate logics. Reports on Mathematical Logic 15:41–58

Pearce D (1997) A new logical characterisation of stable models and answer sets. In: Non monotonic extensions of logic programming. Proc. NMELP'96. (LNAI 1216), Springer-Verlag

Pearce D (1999) Stable inference as intuitionistic validity. J Log Program 38(1):79–91

Pearce D (2006) Equilibrium logic. Ann Math & Art Intell 47(1):3–41

Pearce D, Uridia L (2008) The gödel and the splitting translations. In: Brewka G, Marek V, Truszczynski M (eds) Nonmonotonic Reasoning. Essays Celebrating its 30th Anniversary, College Publications, pp 335–360

Pearce D, Valverde A (2004) Towards a first order equilibrium logic for nonmonotonic reasoning. In: Proc. of the 9th European Conf. on Logics in AI (JELIA'04), pp 147–160

Pearce D, Valverde A (2005) A first order nonmonotonic extension of constructive logic. Studia Logica 80(2-3):321–346

Pearce D, Wagner G (1990) Reasoning with negative information, i: Strong negation in logic programs. In: L.Haaparanta, M. Kusch I. Niiniluoto (Eds), Language, Knowledge and Intentionality, Acta Philosophica Fennica, vol 49, pp 430–453

Pearce D, Wagner G (1991) Logic programming with strong negation. In: ELP, Springer, Lecture Notes in Computer Science, vol 475, pp 311–326

Pearce D, Tompits H, Woltran S (2001) Encodings for equilibrium logic and logic programs with nested expressions. In: EPIA, Springer, LNCS, vol 2258, pp 306–320

Pearce D, Tompits H, Woltran S (2009) Characterising equilibrium logic and nested logic programs: Reductions and complexity' . TPLP 9(5):565–616

Przymusinski T (1991) Stable semantics for disjunctive programs. New Generation Computing 9:401–424

Rosati R (2005) Semantic and computational advantages of the safe integration of ontologies and rules. In: PPSWR, Springer, Lecture Notes in Computer Science, vol 3703, pp 50–64

Rosati R (2006) Dl+log: Tight integration of description logics and disjunctive datalog. In: KR, AAAI Press, pp 68–78

Scott D (1979) Identity and existence in intuitionistic logic. Lecture Notes in Mathematics 753:660–696

Smetanich YS (1960) On completeness of a propositional calculus with an additional operation of one variable (in russian). Trudy Moscovskogo matrmaticheskogo obshchestva 9:357–372

Turner H (2003) Strong equivalence made easy: nested expressions and weight constraints. TPLP 3(4-5):609–622

Umezawa T (1959) On intermediate many-valued logics. Journal of the Mathematical Society of Japan 11:116–128

Vorob'ev N (1952a) A constructive propositional calculus with strong negation (in russian). Doklady Akademii Nauk SSR 85:465–468

Vorob'ev N (1952b) The problem of deducibility in constructive propositional calculus with strong negation (in russian). Doklady Akademii Nauk SSR 85:689–692