A Tarskian Semantics for Answer Set Programming

Marc Denecker
Department of Computer Science
K.U. Leuven
3001 Heverlee, Belgium

Yuliya Lierler
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0633, USA

Miroslaw Truszczynski
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0633, USA

Joost Vennekens
Campus De Nayer | K.U. Leuven
Department of Computer Science
2860 Sint-Katelijne-Waver, Belgium

Abstract

Extended logic programming was introduced initially as an epistemic logic for default and autoepistemic reasoning. With time the reasons for interest in the language have shifted. In 1999 extended logic programming was proposed as the basis of the answer-set programming (ASP) paradigm for modeling and solving search problems and in that role the language is primarily used now. However, in the new context the original epistemic intuitions lose their explanatory relevance. How answer-set programs are connected with the specifications of problems they model and how they are built by following the universally accepted generate-define-test (GDT) methodology is much more easily explained in a classical Tarskian semantics, in which models correspond to possible worlds, rather than to belief states of an epistemic agent. That point seems to have been missed by the research community and ASP has never been cast in a Tarskian setting. In this paper, we fill that gap. The result is a radical departure from the traditional epistemic view on ASP, with important implications for its relation to classical logic.

Introduction

Around 1975, the view of logic programs as classical Horn theories broke down when the *negation-as-failure* (NAF) operator was introduced. The question of a suitable semantics stirred the logic programming community till late 1980s, when Gelfond and Lifschitz proposed to interpret NAF as an autoepistemic or default operator "I do not believe that". Inspired by autoepistemic logic of Moore (1985) and default logic of Reiter (1980), they defined the stable-model semantics for programs with negation (Gelfond 1987; Gelfond and Lifschitz 1988). Shortly thereafter, motivated by applications to commonsense reasoning, they generalized the language and the semantics in the formalism called *extended logic programming* (Gelfond and Lifschitz 1991).

About a decade later researchers realized that (extended) logic programs can be used to model combinatorial search problems so that stable models, or *answer sets* as they are now more commonly called, correspond to solutions of the problems (Marek and Truszczynski 1999; Niemelä 1999). This observation, combined with the growing availability of fast software tools for answer set computation, led to the declarative programming paradigm called *answer-set programming* (ASP).[1] Today, versions of extended logic programming language are used in most implementations of the ASP paradigm (vide an overview of ASP systems by Calimeri et al. (2011)), and combinatorial search is their main application. Nevertheless, this programming paradigm can be implemented using arbitrary logics with a model semantics, as noted already by Marek and Truszczynski (1999), and stressed in the language independent framework MX($\mathcal{L}$) for *model expansion* (that generalizes Herbrand model generation) (Mitchell and Ternovska 2005). Several solvers were built for dialects based on first order logic (FO), some early examples being psgrnd/aspps (East and Truszczynski 2001) and IDP (Mariën et al. 2005).

For the extended logic programming, or the ASP language as it is currently called, the shift to computational search problems prompted developments aimed at making the language, programming methodology and software tools better suited for that task. Most notable were language extensions such as constraints, choice rules, aggregate operators, and weight expressions (Simons, Niemelä, and Soininen 2002), and splitting results that led to modular forms of the ASP language (Oikarinen and Janhunen 2008; Ferraris et al. 2009). Furthermore, practical experience with the use of ASP solvers gave rise to what has become the *de facto* standard methodology of ASP: *Generate-Define-Test* (GDT) (Lifschitz 2002). In this methodology, a programmer conceives an ASP program as consisting of groups of rules to *generate* the search space, to *define* auxiliary concepts, and to *test* (impose) constraints.

The shift in focus also meant that the original epistemic intuitions behind the semantics of extended logic programming were no longer useful and often obscured rather than guided. For example, consider the constraint that a graph coloring should color each node:

$$\leftarrow Node(x), \texttt{not}\ Colored(x).$$

In the epistemic view, the rule reads "*there is no x such that it is believed that x is a node and it is not believed that x is colored.*" The informal reading is far from what the rule is meant to represent. In contrast, the syntax and the informal semantics of the first-order logic sentence:

$$\forall x(Node(x) \Rightarrow Colored(x))$$

[1]Marek et al. 2011 provide an account of the origins of ASP.

directly and literally corresponds to the intended meaning of the constraint.

The mismatch between the epistemic view and the use of programs to model search problems is related to the interpretation of an answer set. In the epistemic view, it is a representation of a state of belief of a rational agent. However, search problem statements are "objective" — they do not make any explicit or implicit references to agent's *beliefs* (vide the benchmark problems used in ASP programming competitions (Gebser et al. 2007; Denecker et al. 2009; Calimeri et al. 2011) and all major applications of ASP to date). Introducing agents and their belief states as conceptual devices to help design and interpret programs is an unnecessary overhead.

We claim that it is more natural to interpret an answer set in the Tarskian tradition, as a *possible world*. Indeed, when solving search problems we are concerned with finding *structures* that satisfy constraints. This matches the Tarskian view of a structure as a representation of a state of affairs in the presence of constraints. Moreover, ASP language extensions, such as choice rules, weight constraints, aggregates and modules, do not have any epistemic content. Explaining them from the epistemic view is hard and not intuitive.[2]

Our goal is to show that looking at ASP from the Tarskian perspective helps explain language extensions motivated by ASP needs, affects our view on the informal semantics of ASP formalisms, and offers a foundation for its programming methodology. To this end, guided by the Tarskian perspective, we design a formalism called ASP-FO, which naturally captures programs written in current dialects of ASP according to the GDT methodology. Its semantics, based on *standard* FO structures, in key aspects coincides with the stable semantics developed by Pelov et al. 2007. We then study the *informal semantics* for ASP-FO— a body of intuitions regarding the informal meaning of its syntactical elements that is coherent with the possible-world view on the models of ASP-FO theories and that explains their formal semantics. Our discussion offers a Tarskian view on ASP-FO and, by extension, on current ASP languages.

Our analysis amounts to turning ASP from a *subjective* epistemic logic into an *objective* one. As such it raises fundamental questions about ASP: its relation to FO and extensions of it, the role of the Closed World Assumption, the nature of negation as failure and strong negation. Although we introduce ASP-FO as a vehicle to study the GDT methodology and the possible-world interpretation of ASP, it is an expressive knowledge representation language in its own right for which efficient ASP tools exist. The language includes FO constraints and FO rule bodies and supports several recent features of ASP languages. In particular, its modular structure is reminiscent of ASP modules (Oikarinen and Janhunen 2008; Gelfond 2002), it provides ways to specify open or closed domain assumptions (Gelfond and Przymusinska 1993; Heymans, Nieuwenborgh, and Vermeir 2008; Ferraris et al. 2009), and interpreted and uninterpreted functions (Lin and Wang 2008; Alviano et al. 2011). Last but not least, ASP-FO is supported by several efficient ASP systems, for example, the IDP system that is one of the best ASP solvers (Wittocx, Mariën, and Denecker 2008; Calimeri et al. 2011).

[2]See, e.g., the Texas Action Group discussion on the matter (`http://www.cs.utexas.edu/users/vl/tag/choice_discussion`).

Generate-Define-Test methodology

GDT is an effective methodology to encode search problems in ASP. In GDT, a programmer conceives the problem as consisting of three parts: GENERATE, DEFINE and TEST (Lifschitz 2002). The role of GENERATE is to *generate the search space*. Nowadays this is often encoded by a set of choice rules:

$$\{A\} \leftarrow B_1, \ldots, B_n, \mathtt{not}\ C_1, \ldots, \mathtt{not}\ C_m, \tag{1}$$

where A, B_i and C_i are atoms. Such a rule states that atom A can be arbitrarily true or false, if the condition expressed by the rule's body holds. The DEFINE part is a set of definitions of some auxiliary predicates. Each definition is encoded by a group of rules

$$A \leftarrow B_1, \ldots, B_n, \mathtt{not}\ C_1, \ldots, \mathtt{not}\ C_m, \tag{2}$$

where A, B_i, C_j are atoms and A is the auxiliary predicate that is being defined. These rules describe how to derive the auxiliary predicates often from the generated predicates, typically in a deterministic way. Finally, the TEST part eliminates generated answer sets that do not satisfy desired constraints. They are represented by constraint rules:

$$\leftarrow B_1, \ldots, B_n, \mathtt{not}\ C_1, \ldots, \mathtt{not}\ C_m, \tag{3}$$

A set of these three types of rules will be called a *GDT program.*

For instance, the GDT-program (4) below encodes the Hamiltonian cycle problem. The example illustrates that an ASP program conceived in the GDT way typically shows a rich internal structure.

$$\begin{array}{ll}
\text{GENERATE} & \{In(x,y)\} \leftarrow Edge(x,y). \\
\hline
\text{DEFINE} & Node(V). \ \ldots \ Node(W). \\
\cline{2-2}
 & Edge(V,V'). \ \ldots \ Edge(W,W'). \\
\cline{2-2}
 & T(x,y) \leftarrow In(x,y). \\
 & T(x,y) \leftarrow T(x,z), T(z,y). \\
\hline
\text{TEST} & \leftarrow In(x,y), In(x,z), y \neq z. \\
\cline{2-2}
 & \leftarrow In(x,z), In(y,z), x \neq y. \\
\cline{2-2}
 & \leftarrow Node(x), Node(y), \mathtt{not}\ T(x,y).
\end{array} \tag{4}$$

Each of the three parts may consist of independent components. For instance, TEST in the example above consists of three independent constraints; DEFINE contains separate definitions for three predicates $Node$, $Edge$ and T. This internal structure exists in the mind of programmers, but is not explicit in ASP programs and often becomes apparent only when we investigate the dependencies between predicates. This motivates us to define a logic that does make the internal structure of a GDT-program explicit.

Concepts of Tarskian model semantics

A *vocabulary* Σ is a set of *predicate* and *function* symbols, each with a non-negative integer *arity*. Terms, formulas and sentences are defined as in FO.

An *interpretation* (or *structure*) $\mathfrak{A}$ of a vocabulary Σ is given by a non-empty set $dom(\mathfrak{A})$, the *domain* of $\mathfrak{A}$, and, for each symbol τ of Σ, a value $\tau^{\mathfrak{A}}$, the *interpretation* of τ. If τ is an n-ary function symbol, $\tau^{\mathfrak{A}}$ is an n-ary total function over $dom(\mathfrak{A})$. If τ is an n-ary predicate symbol, $\tau^{\mathfrak{A}}$ is an n-ary relation over $dom(\mathfrak{A})$. If $\mathfrak{A}$ is an interpretation of a vocabulary Σ, we call Σ the vocabulary of $\mathfrak{A}$ and write it as $\Sigma_{\mathfrak{A}}$. An interpretation of the empty vocabulary consists only of its domain.

If $\Sigma' \subseteq \Sigma_{\mathfrak{A}}$, we define the *projection* of $\mathfrak{A}$ on Σ', written $\mathfrak{A}|_{\Sigma'}$, to be the interpretation of Σ' with the same domain and the same interpretation of each symbol $\tau \in \Sigma'$ as $\mathfrak{A}$. We then also say that $\mathfrak{A}$ is an *extension* (also called *expansion*) of its projection $\mathfrak{A}|_{\Sigma'}$.

Let $\mathfrak{A}$ and $\mathfrak{A}'$ be interpretations of the same vocabulary Σ, having the same domain, and assigning the same values to every function symbol in Σ. We say that $\mathfrak{A}$ is a *subinterpretation* of $\mathfrak{A}'$, written $\mathfrak{A} \subseteq \mathfrak{A}'$, if, for every predicate symbol P of Σ, the relation $P^{\mathfrak{A}}$ interpreting this predicate symbol in $\mathfrak{A}$ is a subset of the corresponding relation $P^{\mathfrak{A}'}$.

A variable assignment θ for an interpretation $\mathfrak{A}$ assigns to each variable v an element $\theta(v)$ in $dom(\mathfrak{A})$. When x is a variable and d an element of $dom(\mathfrak{A})$, we write $\theta[x : d]$ for a variable assignment that assigns d to x but is otherwise the same as θ. The interpretation $t^{\mathfrak{A},\theta}$ of a term in an interpretation $\mathfrak{A}$ under variable assignment θ is defined through the standard induction. As usual, we assume that $\wedge, \forall, \Rightarrow$ are defined in terms of $\neg, \vee$ and $\exists$.

Definition 1 (Satisfiability relation $\mathfrak{A}, \theta \models \varphi$) *Let φ be an FOL formula and $\mathfrak{A}$ a structure over a vocabulary containing all function and relation symbols in φ. We define $\mathfrak{A}, \theta \models \varphi$ by induction on the structure of φ:*

- $\mathfrak{A}, \theta \models P(\bar{t})$ *if* $\bar{t}^{\,\mathfrak{A},\theta} \in P^{\mathfrak{A}}$;
- $\mathfrak{A}, \theta \models \psi \vee \phi$ *if* $\mathfrak{A}, \theta \models \psi$ *or* $\mathfrak{A}, \theta \models \phi$;
- $\mathfrak{A}, \theta \models \neg\psi$ *if* $\mathfrak{A}, \theta \not\models \psi$;
- $\mathfrak{A}, \theta \models \exists x\ \psi$ *if for some* $d \in dom(I)$, $\mathfrak{A}, \theta[x : d] \models \psi$.

When φ is a sentence (no free variables), then θ is irrelevant and we write $\mathfrak{A} \models \varphi$.

In a Tarskian model semantics, a structure represents a potential state of affairs. For a sentence φ and a structure $\mathfrak{A}$, $\mathfrak{A} \models \varphi$ formalizes that φ is **true** in the state of affairs as given by $\mathfrak{A}$. If all we know about the state of affairs is that φ is true in it, then a structure $\mathfrak{A}$ is a *possible* state of the world, or a *possible world*, if and only if $\mathfrak{A} \models \varphi$. [3]

Part of our work is to integrate different logics, ASP and FO in particular. When logics $\mathcal{L}_1, \ldots, \mathcal{L}_n$ have possible-world semantics that are based on the same formal notion of a structure, then there is a natural way to do so. A *theory* T of the *multi-logic* $\langle \mathcal{L}_1, \ldots, \mathcal{L}_n \rangle$ is a set of expressions, each from a logic $\mathcal{L}_i$ in the collection. A structure $\mathfrak{A}$ satisfies such theory T, denoted $\mathfrak{A} \models T$, if it satisfies each $\varphi \in T$. Thus, the meaning of a multi-logic theory is the *monotone conjunction* of the meaning of its expressions.

[3]Some associate the term *possible world* with Kripke semantics of modal logic. We here associate it more generally with the view of a model as a representation of a possible state of affairs, as in Tarski's view on FO model semantics and also in Kripke semantics.

The logic ASP-FO

We introduce a modular form of ASP to represent the different kind of modules in GDT programs. We define a G-module, D-module and T-module.

Definition 2 *A* choice rule *is an expression of the form:* $\forall \bar{x}\ (\{P(\bar{t})\} \leftarrow \varphi)$, *where φ is an FO formula, $P(\bar{t})$ is an atom and $\bar{x}$ includes all free variables appearing in the rule. A* G-module *is a set of choice rules with the same predicate in their head.*

Definition 3 *A* D-module *$\mathcal{D}$ is a pair $\langle Ext, \Pi \rangle$ where Ext is a set of predicates, called* defined *or* output predicates, *and Π is a set of rules of the form*

$$\forall \bar{x}\ (P(\bar{t}) \leftarrow \varphi), \tag{5}$$

where $P(\bar{t})$ is an atom such that $P \in Ext$, and φ is an FO formula with all its free variables amongst $\bar{x}$.

For a D-module $\mathcal{D}$, we denote the set of its defined predicate symbols by *Ext*($\mathcal{D}$). We write *Par*($\mathcal{D}$) for the set of all symbols in Π except the defined predicates. We call *Par*($\mathcal{D}$) the set of *parameter* or *input symbols*. For a set of rules Π, we denote by $heads(\Pi)$, the set of all predicate symbols appearing in the head of a rule $r \in \Pi$. In the following we identify a D-module $\langle heads(\Pi), \Pi \rangle$ with Π.

Definition 4 *A* T-module *is an FO sentence.*

Definition 5 *An* ASP-FO-theory *is a set of* G*-modules,* D*-modules and* T*-modules.*

There is an obvious syntactical match between these language constructs and those used in ASP to express GENERATE, DEFINE and TEST modules. For instance, an ASP constraint (3) corresponds to the T-module– FO sentence:

$$\forall \bar{x}(\neg(B_1 \wedge \cdots \wedge B_n \wedge \neg C_1 \wedge \cdots \wedge \neg C_m)),$$

where $\bar{x}$ is the set of variables occurring in (3). We identify normal rules (2) with the universal closure of

$$A \leftarrow B_1 \wedge \cdots \wedge B_n \wedge \neg C_1 \wedge \cdots \wedge \neg C_m.$$

Note that an ASP-FO theory preserves the internal structure of the GENERATE, DEFINE and TEST parts. For instance, we may write the Hamiltonian cycle theory as:

GENERATE	$\{\forall x \forall y(\{In(x,y)\} \leftarrow Edge(x,y))\}$
DEFINE	$\{Vertex(V) \leftarrow \mathbf{t}, \ldots, Vertex(W) \leftarrow \mathbf{t}\}$
	$\{Edge(V,V') \leftarrow \mathbf{t}, \ldots, Edge(W,W') \leftarrow \mathbf{t}\}$
	$\left\{ \begin{array}{l} \forall x \forall y(T(x,y) \leftarrow In(x,y)) \\ \forall x \forall y(T(x,y) \leftarrow In(x,z) \wedge In(z,y)) \end{array} \right\}$
TEST	$\forall x \forall y \forall z \neg(In(x,y) \wedge In(x,z) \wedge y \neq z)$
	$\forall x \forall y \forall z \neg(In(x,z) \wedge In(y,z) \wedge x \neq y)$
	$\forall x \forall y(Vertex(x) \wedge Vertex(y) \Rightarrow T(x,y))$

(6)

In defining the formal semantics of ASP-FO, we aim to ensure that three conditions are satisfied. First of all, the structures are to be viewed as possible-worlds so that they represent possible states of affairs, not states of belief. We do not restrict to Herbrand interpretations. Second, to respect the modular structure of an ASP-FO theory, its semantics should be modular, that is, defined in terms of the semantics of its modules. We use the simple multi-logic semantics of the previous section: a structure $\mathfrak{A}$ is a model of a ASP-FO theory T iff it is a model of each of its modules. In other words, an ASP-FO theory can be understood as a monotone conjunction of its modules. Third, as ASP-FO is to reflect the GDT methodology, ASP-FO theories resulting from GDT programs must have the same meaning.

The definition of satisfaction of a T-module, i.e., an FO sentence, is standard (Definition 1). It follows that ASP-FO is a conservative extension of FO.

The semantics for a D-module is a generalization of the stable semantics to arbitrary structures and to FO rule bodies. For reasons explained in the next section, we use the semantics that was introduced by Pelov et al. (2007) and, in the way we follow here, by Vennekens et al. (2007).

We recall that a Herbrand interpretation $\mathcal{M}$ is a stable model of a normal program Π if it is the least Herbrand model of the Gelfond-Lifschitz reduct $\Pi^{\mathcal{M}}$. In that reduct, negative literals in the bodies of rules are interpreted by $\mathcal{M}$: a literal `not` A evaluates to false if $\mathcal{M} \models A$, and to true if $\mathcal{M} \not\models A$. Positive body literals are not interpreted by $\mathcal{M}$, but by interpretations arising during the fixpoint construction of the least model. There is a way to simulate this for arbitrary formulas using the following satisfaction relation.

Definition 6 (Satisfaction by pairs of interpretations) *Let φ be an FO formula, $\mathfrak{A}$ and $\mathfrak{B}$ interpretations of all symbols in φ having the same domain and assigning the same values to all function symbols, and let θ be a variable assignment. We define the relation $(\mathfrak{A}, \mathfrak{B}), \theta \models \varphi$ by induction on the structure of φ:*

- $(\mathfrak{A}, \mathfrak{B}), \theta \models P(\bar{t})$ *if* $\mathfrak{A}, \theta \models P(\bar{t})$,
- $(\mathfrak{A}, \mathfrak{B}), \theta \models \neg\varphi$ *if* $(\mathfrak{B}, \mathfrak{A}), \theta \not\models \varphi$,
- $(\mathfrak{A}, \mathfrak{B}), \theta \models \varphi \vee \psi$ *if* $(\mathfrak{A}, \mathfrak{B}), \theta \models \varphi$ *or* $(\mathfrak{A}, \mathfrak{B}), \theta \models \psi$
- $(\mathfrak{A}, \mathfrak{B}), \theta \models \exists x \psi$ *if for some* $d \in dom(\mathfrak{A})$, $(\mathfrak{A}, \mathfrak{B}), \theta[x : d] \models \psi$.

This truth assignment interprets positive occurrences of atoms in $\mathfrak{A}$, and negative occurrences in $\mathfrak{B}$. Indeed, (positive) atoms are interpreted in $\mathfrak{A}$, but every occurrence of $\neg$ switches the role of $\mathfrak{A}$ and $\mathfrak{B}$. We note that if $\mathfrak{A}, \mathfrak{B}$ are identical, then this satisfaction relation coincides with the one of Definition 1. That is, $(\mathfrak{A}, \mathfrak{A}), \theta \models \varphi$ iff $\mathfrak{A}, \theta \models \varphi$.

For two structures $\mathfrak{A}$ and $\mathfrak{A}'$ that have the same domain and interpret disjoint vocabularies, $\mathfrak{A} \circ \mathfrak{A}'$ denotes the structure that interprets the union of the vocabularies of $\mathfrak{A}$ and $\mathfrak{A}'$, has the same domain as $\mathfrak{A}$ and $\mathfrak{A}'$, and coincides with $\mathfrak{A}$ and $\mathfrak{A}'$ on their respective vocabularies.

Definition 7 (Parameterized stable-model semantics) *For a* D*-module $\mathcal{D}$, an interpretation $\mathcal{M}$ of Ext($\mathcal{D}$) is a* stable model *of $\mathcal{D}$ relative to an interpretation $\mathfrak{A}_p$ of Par($\mathcal{D}$) if $\mathcal{M}$ is the least*[4] *of all interpretations $\mathfrak{A}$ of Ext($\mathcal{D}$) that have the same domain as $\mathfrak{A}_p$, interpret function symbols in the same way as $\mathfrak{A}_p$ and for each rule $\forall \bar{x}\ (P(\bar{t}) \leftarrow \varphi)$ of $\mathcal{D}$ and each variable assignment θ, if $(\mathfrak{A}_p \circ \mathfrak{A}, \mathfrak{A}_p \circ \mathcal{M}), \theta \models \varphi$ then $\mathfrak{A}, \theta \models P(\bar{t})$.*

[4]The term "least" is understood with respect to the notion of subinterpretation defined earlier. One can show that such a least interpretation always exists.

This parameterized stable-model semantics generalizes the original one and extends it in three ways: it is parameterized, that is, it builds stable models on top of a given interpretation of the parameter symbols; it handles FO bodies; and it works for arbitrary (non-Herbrand, in general) interpretations.

Example 1 Let us consider a D-module $\mathcal{D}$:

$$\mathcal{D} = \left\{ \begin{array}{l} \forall x(p(x) \leftarrow \neg q(x)) \\ \forall x(q(x) \leftarrow \forall y\ r(y, x)). \end{array} \right\}$$

This is a non-recursive module with a parameter r and defined symbols p and q. It defines p as the complement of q, and q as the set of x that have incoming edges in the "graph" r from each domain element. The module determines for each interpretation $\mathfrak{A}_p$ of the parameter r a set of stable models relative to $\mathfrak{A}_p$, which are interpretations of the defined symbols p and q. Due to absence of (negative) recursion, this set is a singleton, i.e., each $\mathfrak{A}_p$ determines a unique stable model.

Let us consider an interpretation $\mathfrak{A}_p$ with domain $Dom = \{1, 2\}$ and $r^{\mathfrak{A}_p} = \{(1, 1), (2, 1), (2, 2)\}$. Here, 1 is the only node that has an incoming edges from both 1 and 2. Hence we expect that $\mathfrak{A}$ with $p^{\mathfrak{A}} = \{2\}, q^{\mathfrak{A}} = \{1\}$ is the unique stable model relative to $\mathfrak{A}_p$. Let us verify that $\mathfrak{A}$ is indeed a stable model. It should hold that $\mathfrak{A}$ is the least $\mathfrak{A}'$ such that for each variable assignment θ, if $(\mathfrak{A}' \circ \mathfrak{A}_p, \mathfrak{A} \circ \mathfrak{A}_p), \theta \models \neg q(x)$ then $\mathfrak{A}, \theta \models p(x)$ and if $(\mathfrak{A}' \circ \mathfrak{A}_p, \mathfrak{A} \circ \mathfrak{A}_p), \theta \models \forall y\ r(y, x)$ then $\mathfrak{A}, \theta \models q(x)$.

First, we check that the property holds for $\mathfrak{A}' = \mathfrak{A}$. The pair $(\mathfrak{A} \circ \mathfrak{A}_p, \mathfrak{A} \circ \mathfrak{A}_p)$ consists of identical interpretations, and the condition boils down to checking that $\mathfrak{A} \circ \mathfrak{A}_p$ satisfies the two rules in the standard FO sense. It does. For example, for $\theta(x) = 1$, it holds that $\mathfrak{A} \circ \mathfrak{A}_p, \theta \models \forall y\ r(y, x)$ and also that $\mathfrak{A} \models q(x)$.

Second, we verify that $\mathfrak{A}$ has no strict subinterpretation $\mathfrak{A}'$ satisfying the property. Let is consider a strictly smaller $\mathfrak{A}'$. Then either $q^{\mathfrak{A}'} \subsetneq q^{\mathfrak{A}}$ or $q^{\mathfrak{A}'} = q^{\mathfrak{A}}$ and $p^{\mathfrak{A}'} \subsetneq p^{\mathfrak{A}}$. In both cases, at least one rule is violated. In the first case, $q^{\mathfrak{A}'}$ is empty, and the rule defining q is violated for $\theta(x) = 1$. Indeed, $\mathfrak{A}', \theta \not\models q(x)$ while it holds that $(\mathfrak{A}' \circ \mathfrak{A}_p, \mathfrak{A} \circ \mathfrak{A}_p), \theta \models \forall y\ r(y, x)$. To see that, the formula $\forall y\ r(y, x)$ is positive and is evaluated in $\mathfrak{A}' \circ \mathfrak{A}_p$, hence in $\mathfrak{A}_p$. In the second case, $p^{\mathfrak{A}'}$ is empty and the rule defining p is violated for $\theta(x) = 2$. Indeed, we have $\mathfrak{A}', \theta \not\models p(x)$ while $(\mathfrak{A}' \circ \mathfrak{A}_p, \mathfrak{A} \circ \mathfrak{A}_p), \theta \models \neg q(x)$ (for the latter, we note that since q occurs negatively, $\neg q(x)$ is evaluated in $\mathfrak{A}, \theta$). $\square$

The semantics of parameterized stable models turns a D-module $\mathcal{D}$ into a *non-deterministic* function from interpretations of *Par*($\mathcal{D}$) to interpretations of *Ext*($\mathcal{D}$). A structure $\mathfrak{A}$ *satisfies* $\mathcal{D}$ if it agrees with the functions defined by $\mathcal{D}$:

its interpretation of $\mathit{Ext}(\mathcal{D})$ is one of possible images of its interpretation of $\mathit{Par}(\mathcal{D})$. A formal definition follows.

Definition 8 *A structure $\mathfrak{A}$ is a* model *of a* D*-module $\mathcal{D}$ (notation $\mathfrak{A} \models \mathcal{D}$) if $\mathfrak{A}|_{\mathit{Ext}(\mathcal{D})}$ is a stable model of $\mathcal{D}$ relative to $\mathfrak{A}|_{\mathit{Par}(\mathcal{D})}$.*

Building on the example above, the interpretation $\mathfrak{A} \circ \mathfrak{A}_p$ is a model of the D-module $\mathcal{D}$ discussed there.

We now turn our attention to G-modules. We note that the point of a choice rule is to "open up" certain atoms $P(\bar{d})$ – to allow them to be true without forcing them to be true.

Definition 9 *A structure $\mathcal{M}$ is a model of a* G*-module $\mathcal{G}$ if for each variable assignment θ such that $\mathcal{M}, \theta \models P(\bar{x})$ there is a choice rule $\forall \bar{y}\ (\{P(\bar{t})\} \leftarrow \varphi)$ in $\mathcal{G}$ such that $\bar{t}^{\mathcal{M},\theta} = \bar{x}^{\theta}$ and $\mathcal{M}, \theta \models \varphi$.*

A G-module can be translated to an equivalent singleton G-module, using a process similar to *predicate completion*. First, we note that any choice rule $\forall \bar{x}\ (\{P(\bar{t})\} \leftarrow \varphi)$ can be rewritten as $\forall \bar{y}\ (\{P(\bar{y})\} \leftarrow \exists \bar{x}(\bar{y} = \bar{t} \land \varphi))$. Next, any finite set of choice rules $\forall \bar{x}\ (\{P(\bar{x})\} \leftarrow \varphi_i)$ can be combined into a single choice rule $\forall \bar{x}\ (\{P(\bar{y})\} \leftarrow \varphi_1 \lor \cdots \lor \varphi_n))$. It is straightforward to show that these transformations are equivalence-preserving.

Together with this result, the following theorem implies that each (finite) G-module is equivalent to an FO sentence.

Theorem 1 *An interpretation $\mathcal{M}$ satisfies a singleton* G*-module $\{\forall \bar{x}\ (\{P(\bar{x})\} \leftarrow \varphi)\}$ if and only if $\mathcal{M}$ satisfies $\forall \bar{x}\ (P(\bar{x}) \Rightarrow \varphi)$.*

For instance, the singleton G-module of the GENERATE part of (4) corresponds to the following FO sentence:

$$\forall x \forall y (\mathit{In}(x, y) \Rightarrow \mathit{Edge}(x, y)). \tag{7}$$

This theorem shows that G-modules are redundant in ASP-FO, since they can be simulated by T-modules.

ASP-FO is an open domain logic with uninterpreted function symbols. Logic programming and ASP often restrict the semantics to Herbrand interpretations only.

Definition 10 *The* Herbrand *module over a set σ of function symbols is the expression $\mathcal{H}(\sigma)$. We say that $\mathcal{M} \models \mathcal{H}(\sigma)$ if $dom(\mathcal{M})$ is the set of variable-free terms that can be built from σ and for each such term t, $t^{\mathcal{M}} = t$.*

From a knowledge representation perspective, a Herbrand module is useful in applications with complete knowledge of the domain. By adding $\mathcal{H}(\sigma)$ for the set σ of all function symbols of Σ to an ASP-FO theory, we limit its semantics to Herbrand models of σ. By adding $\mathcal{H}(\sigma)$ for a strict subset σ of function symbols, the remaining function symbols behave as uninterpreted symbols and take arbitrary interpretation in the Herbrand universe consisting of the terms of σ. A Herbrand module $\mathcal{H}(\sigma)$ in ASP-FO can be seen as a compact way of expressing the combination of the FO unique name axioms $\mathit{UNA}(\sigma)$, and the domain closure axiom $\mathit{DCA}(\sigma)$. The latter can be expressed in ASP-FO by means of D- and T-modules[5] in exactly the same way as in the logic FO(ID) (Denecker 2000). As such, Herbrand modules are redundant in ASP-FO but we keep them for notational simplicity (cf. Theorem 3 below).

[5]It is a well-known consequence of the compactness theorem for FO that $DCA(\sigma)$ cannot be represented in FO if σ contains a non-constant function symbol.

Proposition 2 *If $\mathfrak{M}$ is an ASP-FO module containing precisely the non-logical symbols Σ and $\mathfrak{A}, \mathfrak{B}$ are two structures such that $\mathfrak{A}|_{\Sigma} = \mathfrak{B}|_{\Sigma}$, then $\mathfrak{A} \models \mathfrak{M}$ if and only if $\mathfrak{B} \models \mathfrak{M}$.*

The essence of this result is that a module, like an FO sentence but unlike an ASP program, does not impose constraints on symbols that do not appear in it. Thus, in ASP-FO there is no default Closed World Assumption.

Our limitation to the above sorts of modules in the logic ASP-FO is somewhat arbitrary. It is straightforward to extend it with additional types of modules as long as they each have a possible world semantics for which Proposition 2 holds.

Relationship with FO and ASP. ASP-FO is not only a conservative extension of FO but also of the basic ASP language of normal programs. Note that a set of normal rules can be seen as a D-module defining all predicates.

Theorem 3 *For a normal program Π over vocabulary Σ, a structure $\mathfrak{A}$ is a stable model of Π if and only if $\mathfrak{A}$ is a model of the ASP-FO theory $\{(\Sigma_P, \Pi), \mathcal{H}(\Sigma_F)\}$, where Σ_P, Σ_F is the set of all predicate and function symbols of Σ, respectively.*

This theorem allows us to represent an entire normal logic program as a single D-module (and an auxiliary Herbrand module). However, as stated before, what we would like to show is the equivalence of GDT-programs in ASP and the corresponding ASP-FO theories.

Let us now consider a GDT-program Π consisting of a set of choice rules of form (1), normal rules of form (2) and constraints of form (3). We define the *(positive) predicate dependency graph* of Π as the directed graph with all predicate symbols of Π as its vertices and with an edge from P to Q whenever P appears in the head of a rule and Q occurs positively in the body of that rule (i.e., in the scope of an even number of negations).

Without loss of generality we assume that each predicate of Π appears in the head of at least one of its rules. By $heads(\Pi)$ we denote the set of all predicate symbols appearing in the heads of the rules of the form (1) or (2) in Π. A partition $\Pi_0, \ldots, \Pi_n$ of Π is a *splitting*[6] of Π if:

– for each i, Π_i is either a singleton containing a constraint, the set of all choice rules for some predicate P, or a normal logic program;

– $heads(\Pi_i) \cap heads(\Pi_j) = \emptyset$ for $i \neq j$;

– for any strongly connected component S of the predicate dependency graph of Π, $S \subseteq heads(\Pi_i)$ for some i;

– for any predicate symbol P occurring in the head of some choice rule in Π there is no edge from P to P in the predicate dependency graph of Π.

[6]The conditions on splitting follow the requirements stated in the Symmetric Splitting Theorem in (Ferraris et al. 2009).

We can identify each Π_i in a splitting with an ASP-FO module in the obvious way: a Π_i that consists of a constraint corresponds to a T-module, a Π_i consisting of choice rules corresponds to a G-module, and a Π_i consisting of normal rules corresponds to a D-module.

Theorem 4 *For a GDT-program Π, if $\Pi_0, \ldots, \Pi_n$ is a splitting of Π, then an interpretation $\mathcal{M}$ is answer set of Π if and only if $\mathcal{M}$ is a model of $\{\mathfrak{M}_0, \ldots, \mathfrak{M}_n, \mathcal{H}(\Sigma)\}$, where each $\mathfrak{M}_i$ is the ASP-FO module corresponding to Π_i.*

For instance, the horizontal lines within GENERATE, DEFINE, and TEST parts of the Hamiltonian cycle program (4) identify a partition that satisfies the conditions of a splitting. Theorem 4 states that the answer sets of (4) coincide with models of the ASP-FO theory (6).

The practice of ASP demonstrates that the vast majority of GDT programs admit a splitting. Theorem 4 shows that ASP-FO (i) embeds this fragment of ASP in a direct way, and (ii) interprets those ASP programs as the *monotone conjunction* of their components.

One way in which ASP programs occasionally violate the conditions for admitting a splitting is by containing both choice rules and normal rules for the same predicate P. Such cases can be eliminated in an equivalence preserving way: we introduce an auxiliary predicate P', substitute in Π every choice rule $\{P(\bar{t})\} \leftarrow \varphi$ by $\{P'(\bar{t})\} \leftarrow \varphi$, and add the rule $P(\bar{x}) \leftarrow P'(\bar{x})$.

Theorem 4 fails to take into account three common extensions of the ASP language: aggregates (or weight expressions), disjunction in the head, and strong negation. Allowing aggregates in T-modules is straightforward, and they can also be allowed in the bodies of rules of D-modules, using the semantics developed by Pelov et al. (2007). As for disjunction in the heads of rules, Definition 8 can be adjusted to support it by requiring that a stable model $\mathcal{M}$ be a *minimal* rather than the *least* interpretation satisfying the conditions given there. Finally, strong negation can be "translated away" as in ASP by means of additional predicates. Strong negation is discussed in the next section.

Relation to FO(ID). A theory in FO(ID) is a set of FO sentences and *inductive definitions*.[7] These definitions are syntactically identical to D-modules of ASP-FO, but are interpreted under a two-valued parameterized variant of the well-founded semantics, rather than the parameterized stable-model semantics used in ASP-FO.

Definition 11 *A Σ-interpretation $\mathfrak{A}$ is a model of an FO(ID) definition Δ (notation $\mathfrak{A} \models \Delta$) if $\mathfrak{A}|_{Ext(\mathcal{D})}$ is the well-founded model of Δ relative to $\mathfrak{A}|_{Par(\mathcal{D})}$, as defined in (Denecker and Vennekens 2007).*

Since the well-founded model is always unique, but not necessarily two-valued, a definition Δ expresses a *deterministic*, but *partial* function from *Par*(Δ)-interpretations to *Ext*(Δ)-interpretations; it is only defined when the well-founded model in $\mathfrak{A}|_{Par(\Delta)}$ is two-valued. A structure $\mathfrak{A}$ satisfies $\mathcal{D}$ if $\mathfrak{A}|_{Ext(\mathcal{D})}$ is the map of $\mathfrak{A}|_{Par(\mathcal{D})}$.

[7]Some versions of FO(ID) allow also boolean combinations of FO formulas and definitions (Denecker and Ternovska 2008).

Well-founded and stable-model semantics are related (Van Gelder 1993; Fitting 2002; Denecker, Marek, and Truszczynski 2000). This relationship generalizes to D-modules.

Proposition 5 ((Pelov, Denecker, and Bruynooghe 2007)) *If the well-founded model of $\mathcal{D}$ relative to $\mathfrak{A}_{Par(\Delta)}$ is two-valued, it is also the unique stable model of Δ relative to $\mathfrak{A}_{Par(\Delta)}$.*

Denecker and Ternovska 2008 introduced the notion of a *total definition*. An FO(ID) definition is *total* if it has only two-valued well-founded models and hence expresses a total, deterministic function from *Par*$(\mathcal{D})$-interpretations to *Ext*$(\mathcal{D})$-interpretations. They also identified conditions that guarantee that definitions are total. These include the condition of no negative occurrences of defined symbols in the bodies of rules, which defines the class of *positive* definitions; a more general condition of stratification at the predicate level; and an even more general condition of local stratification. Thus, many definitions occurring in practice are total. For total definitions (D-modules), the logics ASP-FO and FO(ID) coincide.

Looking back at the ASP-FO theory for the Hamiltonian circuit program, we see that all three of its D-modules are positive. Hence, it is equivalent to the FO(ID) theory of the same syntactic form.

Relation to the equilibrium logic first-order ASP. Equilibrium logic (Pearce 1997) is a formalism based on the logic of *here-and-there*, a strengthening of the intuitionistic logic that still, however, fails to satisfy the law of excluded middle (Heyting 1930). The equilibrium logic provided a way to generalize the semantics of answer sets of programs to arbitrary propositional theories (Pearce 1997) and to the full FO case (Pearce and Valverde 2008; Ferraris, Lee, and Lifschitz 2011). There is a formal connection between ASP-FO D-modules and first order ASP programs based on equilibrium logic. Restricting the latter to formulas representing rules of the form (5), one can prove that the semantics coincide if the bodies of rules (5) have no nested occurrences of negation (Truszczynski 2012). However, the two generalizations of ASP differ if nested occurrences are allowed. For instance, the D-module $\{P \leftarrow \neg\neg P\}$ has only $\emptyset$ as a model, while in the equilibrium semantics also $\{P\}$ is a model.

More importantly though, they differ at the conceptual level. The logic ASP-FO directly extends FO, and its semantics interprets all classical connectives (that is, all connectives apart from the rule operator used in D-modules) in the classical way. It has a clear modular structure and uses classical monotone conjunction to combine the meaning of modules. The equilibrium logic version of first-order ASP is based on the quantified logic HT that in many respects differs from classical FO and, arguably, lacks its direct connection to everyday linguistic patterns.

Multiple D-modules for the same concepts. An ASP-FO theory may have multiple D-modules defining the same predicate, as in the example below.

$$\{\ \forall x\ (Human(x) \leftarrow Male(x) \vee Female(x))\ \}$$
$$\{\ \forall x\ (Human(x) \leftarrow Adult(x) \vee Child(x))\ \}$$

This theory states that the class of humans is the union of the classes of males and females, and that it is also the union of the classes of adults and children. Taking the union of these two singleton modules produces a single, weaker module, which neither entails that humans are male or female, nor that they are adults or children. Such distinctions cannot be made in a direct way within the non-modular language of ASP.

Informal semantics of ASP-FO and the Generate-Define-Test methodology

A formal semantics is a mathematical definition. Therefore, it does not explain by itself what expressions in a logic mean in the real world. The logic's *informal semantics* provide a general (even if informal) account of what logic expressions state about the world by giving a system of coherent interpretations of the logic's syntactic and semantic objects. A logic's formal semantics strongly constrains its informal semantics by establishing a mathematical relation between connectives and semantical objects. Nevertheless, it does not entirely fix it. At the very least, we have to be clear how the semantical objects relate to the world that is being modelled (Denecker 2004). For instance, the formal semantics of ASP defines that `not` P holds in an answer set $\mathcal{M}$ iff $P \notin \mathcal{M}$. Traditionally, an answer set $\mathcal{M}$ is interpreted as the set of believed literals in a belief state of the rational agent. Thus, `not` P, which formally holds in case of absence of P in the answer set, informally means that *the agent does not believe* P, which is a standard view on negation as failure. In the same way, the constraint from the introduction

$$\leftarrow vertex(x), \texttt{not}\ colored(x).$$

formally eliminates answer sets that contain $vertex(c)$ and not $colored(c)$, for some c, and hence, it's informal semantics is that *there is no x such that the agent believes that x is a vertex and does not believe that x is colored.* In this paper, we interpret an answer set as a Tarskian representation of a possible state of the world. Since now $\mathcal{M}$ does not reflect a state of belief, we need to revise our intuitions regarding the logic connectives and rules.

It is a common adage in knowledge representation that humans are only able to comprehend a large theory if its meaning is composed from the meaning of its components through a simple and natural semantic composition operator. The most basic composition operator is simple conjunction. It is the use of this operator that causes FO to be monotonic. The meaning of an ASP-FO theory is constructed from the meaning of its individual modules by precisely the same form of conjunction. Thus, ASP-FO's nonmonotonicity is derived from its modules, not from the general composition law. This is in perfect agreement with our intuition of modules as imposing constraints on possible worlds, independently from each other. Whatever analysis remains to be done has then to be concerned with individual modules.

Informal semantics of T-modules/FO sentences. FO sentences express propositions about an objective world, not about beliefs, intentions, or other propositional attitudes. In Tarskian model semantics, a structure $\mathfrak{A}$ serves as a mathematical abstraction of an objective world. The recursive rules of the definition of truth of a sentence in $\mathfrak{A}$ (Definition 1) specify the formal semantics of FO simply by translating each formal connective into an informal one: $\wedge$ into the natural language "and", $\vee$ into "or", etc. Iterated application of these rules translates an FO sentence into a natural language sentence that accurately captures its meaning.

The existence of this informal semantics does not mean that each FO sentence has a self-evident meaning. Sentences with three or more nestings of quantifiers are hard to understand. The material implication $\psi \Rightarrow \varphi$ also may cause difficulties. Nevertheless, for a core fragment of FO, sentences have an accurate and reliable informal semantics. For example, given the informal meaning of the symbols $Node$ and T in the Hamiltonian circuit example, the informal semantics of

$$\forall x \forall y (Node(x) \wedge Node(y) \Rightarrow T(x, y))$$

is the proposition that each node can be reached from every other one. This accurately reflects the intended proposition, more than the epistemic reading of the corresponding ASP constraint:

$$\leftarrow Node(x), Node(y), \texttt{not}\ T(x, y).$$

which would be: "*it is not the case that for some x and y, it is believed that x and y are vertices and it is not believed that x can reach y*".

Informal semantics of choice rules. Choice rules in ASP are often explained in a computational way, as generators of the search space. Here we propose a declarative interpretation. The set of ASP choice rules for predicate P

$$\{P(\bar{t}_1)\} \leftarrow \varphi_1. \ \ \ldots \ \ \{P(\bar{t}_n)\} \leftarrow \varphi_n.$$

constitutes a G-module in ASP-FO which can be further translated in

$$\forall x (P(\bar{x}) \ \Rightarrow \ (\bar{x} = \bar{t}_1 \wedge \varphi_1) \vee \cdots \vee (\bar{x} = \bar{t}_n \wedge \varphi_n))$$

In the Tarskian possible-world perspective, this sentence says that P is universally false with exceptions explicitly listed in the consequent of the implication. In other words, a G-module expresses the *local closed world assumption* (LCWA) on P, together with an exception mechanism to relax this LCWA and reinstall the open world assumption (OWA) on certain parts of the domain. For instance, $\forall x \forall y (In(x, y) \Rightarrow Edge(x, y))$, an ASP-FO image of the ASP choice rule $\{In(x, y)\} \leftarrow Edge(x, y)$, states that $In(x, y)$ is false except when $Edge(x, y)$ is true, in which case $In(x, y)$ might be either true or false.

This analysis of ASP choice rule modules as FO sentences shows that logical connectives in choice rule bodies, *including* negation, have their standard FO meaning. However, the meaning of a choice module as a whole is not composed from the meaning of its individual rules by monotone conjunction. Instead, adding a rule to a module corresponds to adding a disjunct to its FO axiom. Hence, the underlying composition operator of this sort of module is actually anti-monotonic: the module becomes *weaker* with each rule added. This agrees with the role of a choice rule as expressing an *exception* to the LCWA imposed by the module. The more exceptions there are, the weaker this LCWA.

Informal semantics of D-modules. In the GDT methodology, D-modules serve to *define* a set of auxiliary predicates (Lifschitz 2002) and do so using a *rule-based, potentially recursive* syntax. Even though current ASP practice tacitly assumes that the stable-model semantics is a correct semantics for such modules, this is actually far from trivial. As far as we know, this issue has not yet been addressed in the literature. Our results allow us to present the following argument to fill this gap.

Informal rule-based definitions (such as Definition 1) abound in mathematics. They express a precise, objective form of informal knowledge. A formal rule-based definition construct should match with the informal one. The three most common forms of definitions in formal sciences are non-inductive definitions, monotone inductive definitions (e.g., transitive closure) and definitions by induction over a well-founded order (e.g., the definition of $\models$ in FO, cf. Definition 1). Denecker 1998; 2000 was first to argue that rules under the well-founded semantics provide a uniform and correct formalization of these. Later, Denecker et al. 2008 and Denecker and Ternovska 2008 extended the original arguments. A full discussion of the arguments is beyond the scope of this paper but the essence is that an informal inductive definition describes how to construct the defined relation by iterated application of rules and that the well-founded semantics correctly "simulates" this construction for the three aforementioned forms of definitions.

Not every formal rule set can be understood as a "good" informal inductive definition (i.e., one that a formal scientist would accept). In particular, a "good" definition should define for each object whether it is an element of the defined set or not. In formal terms, this means that a "good" formal rule set should have a total, i.e., 2-valued, well-founded model. Accordingly, such definitions are called *total* (Denecker 2000; Denecker and Ternovska 2008). By Proposition 5, parameterized stable and well-founded semantics coincide for total definitions. Thus, the above arguments apply immediately also to total D-modules. In other words, for such modules, the stable model semantics defined here is an equally correct formalization of these forms of definitions. To the best of our knowledge, this is the first detailed explanation of why the stable model semantics is correct in this case.

It does not apply to all of ASP, though. First, the analysis by Denecker and co-authors consistently interprets structures as possible worlds; therefore, our argument does not apply to the epistemic interpretation of stable models. Second, when we go beyond total D-modules, the correspondence to FO(ID) breaks down. In FO(ID), such rule sets are unsatisfiable, whereas in ASP-FO, they may have 0, 1 or more models. How such rule sets can be interpreted is an open question, but in practice there seems little need for non-total D-modules. Indeed, D-modules are non-total only in case of cycles over negation. In early applications of ASP, such cycles over negation were used to encode the generate and test parts of the search problem. However, more recently, these roles have been taken over by choice rules and constraints. Consequently, cycles over negation in D-modules have become very rare. In fact, in the current practice of ASP, D-modules almost always seem to be either positive or to contain only locally stratified negation. (but see below for an exception).

Comparison with the epistemic view. As observed earlier in (Denecker 2004), it is most interesting that the same mathematical principle can play a very different role depending on whether we take an epistemic or a possible world view on ASP. Under the stable model semantics, no atom belongs to an answer set unless it is derived by some rule (in an appropriate cycle-free manner). Under the epistemic view of an answer set, the informal explanation is that a *rational* agent should only *believe* an atom (or literal) if he has a justification for doing so. In the Tarskian setting, this explanation does not work, simply because the presence of an atom in an answer set does not reflect that it is *believed* but rather that it is *true* in the possible world. Thus, what the stable semantics expresses in the Tarskian view is that atoms cannot be true unless there is a reason for them to be so, which is a form of Closed World Assumption (CWA). In particular, it is a *global* CWA on *all* predicates. Of course, this is a strong assumption that often needs to be relaxed and this is where choice rules naturally step in. In epistemic ASP, on the other hand, no implicit CWA is imposed; if CWA is desired it must be stated explicitly, e.g., by rules $-P(\bar{x}) \leftarrow \texttt{not}\ P(\bar{x})$ involving strong negation (Gelfond and Lifschitz 1991). Since there is, therefore, no implicit global CWA to "open,", the role of choice rules is difficult to explain in this context. A remarkable conclusion is that the mathematical principle to formalize rationality in the epistemic view of stable models actually expresses a form of CWA in the possible world view of stable models.

The form of CWA implemented by the parameterized stable-model semantics in ASP-FO differs from other instances of CWA. It is *local*, i.e., applied only to the defined predicates *Ext*($\mathcal{D}$), and it is also *parameterized*, in the sense that it is applied *given* the parameter $\mathfrak{A}_P$. For instance, the D-module $(\{P\}, \{P \leftarrow Q\})$ imposes CWA on P, it does not entail P and yet, in contrast to other forms of CWA, it neither entails $\neg P$. This is due to the parameter Q, which causes the ASP-FO semantics to admit two models: if the parameter Q is true, then P can be derived, so $\{Q, P\}$ is a model; if the parameter Q is false, then P cannot be derived and, by the CWA, must be false, so $\emptyset$ is also a model. Strikingly, this particular form of CWA, which deviates from standard forms of CWA, coincides for the important fragment of total D-modules with the precise and well-known mathematical principle of definition by induction. Whether the form of CWA underlying D-modules has natural KR applications beyond total definitions is an intriguing question. Such applications might be found in ASP programs that utilize cycles over negation for purposes other than to express choices or constraints, e.g., to express causal rules as in (Lifschitz and Turner 1999).

On the nature of negation and rule operator. Taking a possible world view also forces us to modify our interpretation of negation as failure. The embedding of ASP constraints and choice rules in FO shows that ASP's unary rule operator $\leftarrow$ for constraints as well as negation as failure `not` in such rules are the same as classical negation. As for negation in D-modules, we started this section by noting that in a Tarskian view, negation cannot be epistemic, since there is no epistemic agent around. To understand its meaning, let us look at what negation means in informal definitions, for instance, in the condition of the following (informal) rule from our (informal) Definition 1: $\mathfrak{A}, \theta \models \neg\psi$ if $\mathfrak{A}, \theta \not\models \psi$. The definition is by structural induction, hence this rule should not be applied before rules deriving subformulas of $\neg\psi$. Once this condition is met, the rule derives $I, \theta \models \neg\varphi$ when *it is not the case that* $\mathfrak{A}, \theta \models \psi$. This is standard objective negation as formalized by classical negation in FO.

The difference between a rule $\forall x\ (P(\bar{t}) \leftarrow \varphi)$ in an FO(ID) definition or a D-module and a material implication $\forall x\ (P(\bar{t}) \Leftarrow \varphi)$ therefore does not lie in the interpretation of the connectives of φ. Instead, it lies in the rule operator $\leftarrow$, which differs from material implication $\Leftarrow$. Previous studies of inductive definitions called this operator also the *production operator*, reflecting its role of producing new elements of the defined relation. As discussed in (Denecker and Vennekens 2007), part of its meaning is the restriction that such elements should be produced in accordance with the well-founded order over which the induction is happening. This makes a rule indeed quite different from a material implication.

Strong negation. In the epistemic view, strong negation $-P$ is used to express explicit negative information. How should it be interpreted in the Tarskian view? It cannot be interpreted as classical objective negation since, as we argued above, ASP's negation as failure `not` occupies that role. A simple solution is suggested by considering the standard elimination procedure for strong negation. It operates by introducing, for each predicate P/n a new predicate P^-/n, substituting $P^-(\bar{t})$ for each occurrence of strong negation literal $-P(\bar{t})$ in the ASP program, and adding the explicit constraint:

$$\leftarrow P(\bar{x}), P^-(\bar{x}).$$

Interpreted in the Tarskian view, this translation gives an accurate picture of what strong negation means: it is a predicate constructor connective that, applied to a predicate symbol, yields another predicate symbol of the same arity, loosely connected to the first one by a semantic condition: that P and $-P$ are disjoint relations. A developer using strong negation $-P$ is free to chose his interpretation and to axiomatize $-P$ as long as this condition is satisfied. Sometimes P^- is axiomatized to be the complement of P, in other words, the classical negation of P. For instance, this is the case when $-P$ is defined by the ASP rule:

$$-P(\bar{x}) \leftarrow \texttt{not}\ P(\bar{x}).$$

Note that, in the epistemic view, this rule implements a closed world assumption for P. As a concluding example, let us apply the above ideas to the following ASP solution for the graph coloring problem:

$$\begin{array}{l}
\{\mathit{Color}(x,c)\} \leftarrow \mathit{Node}(x), \mathit{Color}(c).\\
\leftarrow \mathit{Color}(x,c1), \mathit{Color}(x,c2), c1 \neq c2.\\
\leftarrow \mathit{Node}(x), -\mathit{Colored}(x).\\
\mathit{Colored}(x) \leftarrow \mathit{Color}(x,c), \mathit{Node}(x), \mathit{Color}(c).\\
-\mathit{Colored}(x) \leftarrow \mathit{Node}(x),\ \texttt{not}\ \mathit{Colored}(x).\\
\ldots \textit{\% definitions for Node and Color.}
\end{array}$$

In the Tarskian interpretation, *Colored* is defined to be the set of nodes that have a color. Here, the predicate $-\mathit{Colored}$ is not defined as the complement of *Colored* but as the subset of nodes in the complement. The ASP-FO translation of the program above follows:

$$\begin{array}{l}
\forall x \forall c(\mathit{Color}(x,c) \Rightarrow \mathit{Node}(x) \land \mathit{Color}(c))\\
\neg\exists x \exists c1 \exists c2(\mathit{Color}(x,c1) \land \mathit{Color}(x,c2) \land c1 \neq c2)\\
\neg\exists x(\mathit{Node}(x) \land \mathit{Colored}^-(x))\\
\left\{\ \forall x \forall c(\mathit{Colored}(x) \leftarrow \mathit{Color}(x,c) \land \mathit{Node}(x) \land \mathit{Color}(c))\ \right\}\\
\left\{\ \forall x(\mathit{Colored}^-(x) \leftarrow \mathit{Node}(x) \land \neg\mathit{Colored}(x))\ \right\}\\
\ldots \textit{\% definitions for Node and Color.}
\end{array}$$

We note that the definition of $\mathit{Colored}^-$ entails that this predicate is disjoint from *Colored*.

Discussion

Interpreting the answer-set semantics as a Tarskian possible-world semantics is a major mental leap which radically affects our interpretation of the ASP formalism, its composition laws, and the meaning of its connectives. While many ASP researchers may have already made this leap in their day-to-day programming under the Generate-Define-Test methodology, this paper offers the first detailed discussion of its consequences. To conduct our analysis, we presented the formalism ASP-FO, whose modular structure is geared specifically towards the GDT paradigm. This logic formally differs from FO(ID) only by the use of the parameterized stable instead of well-founded semantics for D-modules, semantics that differ only in case of loops over negation. By studying our possible-world perspective on ASP-FO, we obtained an informal semantics for the GDT fragment of ASP, which combines modules by means of the standard conjunction, and captures the roles of different modules in GDT-based programs. In particular, the close connection between ASP-FO and FO(ID) allowed us to provide, to the best of our knowledge, the first argument for the correctness of the stable model semantics as a formalization of the concept of an (inductive) definition.

We proposed ASP-FO as a theoretical mechanism to study GDT and ASP from a Tarskian perspective. However, ASP-FO is also a viable language for knowledge representation and ASP problem solving. Similarly to FO(ID), ASP-FO includes FO constraints, FO rule bodies, modules and non-interpreted functions. It is an open domain logic and its models can be infinite. In general, the satisfiability problem is undecidable (and not just co-semidecidable) — the result can be obtained by adapting the corresponding result concerning the logic FO(ID) (Denecker and Ternovska 2008). In many search problems, however, a finite domain is given. That opens a way to practical problem solving. One can apply finite Herbrand model generation or *model expansion* (Mitchell and Ternovska 2005). At present, all ASP tools use languages based either on extended logic programming or classical logic and as such support various fragments of ASP-FO: answer set solvers (Calimeri et al. 2011), and FO-based systems psgrnd/aspps (East and Truszczynski 2001), the Enfragmo system (Aavani et al. 2012), and the IDP system (Wittocx, Mariën, and Denecker 2008). Most of these systems, including the latter three, also support versions of inductive definitions, aggregates, arithmetic, as well as other language extensions. The IDP system — one of the most efficient ASP systems (Calimeri et al. 2011) — is the first and, at present, the only system that supports full FO(ID) as well as ASP-FO.

Finally, let us put the goals of this paper in a broader historical perspective. First, both logic programming and nonmonotonic reasoning were anti-theses to classical logic (FO), motivated by respectively computational and representational issues with the latter. The work on ASP-FO and earlier on FO(ID) effectively presents a synthesis of these paradigms with FO. Second, the view of logic programs as definitions was already present in Clark's view, albeit implicitly, and his completion semantics is not fully adequate to formalize this idea. Later, Gelfond and Lifschitz proposed to interpret logic programs as epistemic theories. The view on D-modules presented in this paper is a proposal to "backtrack" to Clark's original view.

Acknowledgments

The authors are grateful to Maurice Bruynooghe, Michael Gelfond, Vladimir Lifschitz, Eugenia Ternovska, the past and present members of the KRR group of the KUL, and anonymous reviewers for discussions and comments. The first and fourth author were supported by Research Foundation-Flanders (FWO-Vlaanderen) and by GOA 2003/08 "Inductive Knowledge Bases". The second author was supported by a CRA/NSF 2010 Computing Innovation Fellowship. The third author was supported by the NSF grant IIS-0913459.

References

Aavani, A.; Wu, X.; Tasharrofi, S.; Ternovska, E.; and Mitchell, D. G. 2012. Enfragmo: A system for modelling and solving search problems with logic. In *Proceedings of LPAR 2012*, LNCS, volume 7180, 15–22. Berlin: Springer.

Alviano, M.; Calimeri, F.; Faber, W.; Ianni, G.; and Leone, N. 2011. Function symbols in ASP: Overview and perspectives. In Brewka, G.; Marek, V.; and Truszczynski, M., eds., *Nonmonotonic Reasoning – Essays Celebrating Its 30th Anniversary*. London: College Publications.

Calimeri, F.; Ianni, G.; Ricca, F.; Alviano, M.; Bria, A.; Catalano, G.; Cozza, S.; Faber, W.; Febbraro, O.; Leone, N.; Manna, M.; Martello, A.; Panetta, C.; Perri, S.; Reale, K.; Santoro, M. C.; Sirianni, M.; Terracina, G.; and Veltri, P. 2011. The third answer set programming system competition: Preliminary report of the system competition track. In *Proceedings of LPNMR 2011*, LNCS, volume 6645, 388–403. Berlin: Springer.

Denecker, M.; Bruynooghe, M.; and Marek, V.W. Logic programming revisited: Logic programs as inductive definitions. 2001. *ACM Transactions on Computational Logic* 2(4), 623–654.

Denecker, M., and Ternovska, E. 2008. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic* 9(2).

Denecker, M., and Vennekens, J. 2007. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In *Proceedings of LPNMR 2007*, LNCS, volume 4483, 84–96. Berlin: Springer.

Denecker, M.; Vennekens, J.; Bond, S.; Gebser, M.; and Truszczynski, M. 2009. The second answer set programming system competition. In *Proceedings of LPNMR 2009*, LNCS, volume 5753, 637–654. Berlin: Springer.

Denecker, M.; Marek, V.; and Truszczynski, M. 2000. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In Minker, J., ed., *Logic-based Artificial Intelligence*. Boston: Kluwer. 127–144.

Denecker, M. 1998. The well-founded semantics is the principle of inductive definition. In *Proceedings of JELIA 1998*, LNCS, volume 1489, 1–16. Berlin: Springer.

Denecker, M. 2000. Extending classical logic with inductive definitions. In *Proceedings of CL 2000*, LNCS, volume 1861, 703–717. Berlin: Springer.

Denecker, M. 2004. What's in a model? Epistemological analysis of logic programming. In *Proceedings of KR 2004*, 106–113. Palo Alto, CA, AAAI Press.

East, D.; Truszczynki, M. 2001. Propositional Satisfiability in Answer-Set Programming In *Proceedings of KI 2001*, LNCS, volume 2174, 138–153. Berlin: Springer.

Ferraris, P.; Lee, J.; Lifschitz, V.; and Palla, R. 2009. Symmetric splitting in the general theory of stable models. In *Proceedings of IJCAI 2009*, 797–803.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2011. Stable models and circumscription. *Artificial Intelligence* 175:236–263.

Fitting, M. 2002. Fixpoint semantics for logic programming a survey. *Theoretical Computer Science* 278(1-2):25–51.

Gebser, M.; Liu, L.; Namasivayam, G.; Neumann, A.; Schaub, T.; and Truszczynski, M. 2007. The first answer set programming system competition. In *Proceedings of LPNMR 2007*, LNCS, volume 4483, 3–17. Berlin: Springer.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proceedings of ILPCS*, 1070–1080. Cambridge, MA: MIT Press.

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

Gelfond, M., and Przymusinska, H. 1993. Reasoning on open domains. In *Proceedings of LPNMR 1993*, 397–413. Cambridge, MA: MIT Press.

Gelfond, M. 1987. On stratified autoepistemic theories. In *Proceedings of AAAI 1987*, 207–211. Menlo Park, CA: Morgan Kaufman.

Gelfond, M. 2002. Representing knowledge in A-Prolog. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, LNCS, volume 2408, 413–451. Berlin: Springer.

Heymans, S.; Nieuwenborgh, D. V.; and Vermeir, D. 2008. Open answer set programming with guarded programs. *ACM Transactions on Computational Logic* 9(4).

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

Lifschitz, V., and Turner, H. 1999. Representing transition systems by logic programs. In *Proceedings of LPNMR 1999*, LNCS, volume 1730, 92–106. Berlin: Springer.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138:39–54.

Lin, F., and Wang, Y. 2008. Answer set programming with functions. In *Proceedings of KR 2008*, 454–465. AAAI Press.

Marek, V., and Truszczynski, M. 1999. Stable models and an alternative logic programming paradigm. In Apt, K.; Marek, V.; Truszczynski, M.; and Warren, D., eds., *The Logic Programming Paradigm: a 25-Year Perspective*. Berlin: Springer. 375–398.

Marek, V.; Niemelä, I.; and Truszczynski, M. 2011. Origins of answer-set programming – some background and two personal accounts. In Brewka, G.; Marek, V.; and Truszczynski, M., eds., *Nonmonotonic Reasoning – Essays Celebrating Its 30th Anniversary*. London: College Publications.

Mariën, M.; Mitra, R.; Denecker, M.; and Bruynooghe, M. 2005. Satisfiability Checking for PC(ID), In *Proceedings of LPAR 2005*, LNCS, volume 3835, 565–579. Berlin: Springer.

Mitchell, D., and Ternovska, E. 2005. A framework for representing and solving NP search problems. In *Proceedings of AAAI 2005*, 430–435. AAAI Press.

Moore, R. C. 1985. Semantical considerations on nonmonotonic logic. *Artificial Intelligence* 25(1):75–94.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25:241–273.

Oikarinen, E., and Janhunen, T. 2008. Achieving compositionality of the stable model semantics for Smodels programs. *Theory and Practice of Logic Programming* 5–6:717–761.

Pearce, D. 1997. A New Logical Characterisation of Stable Models and Answer Sets. In *Proceedings of NMELP 1996*, LNCS, volume 1216, 57–70. Berlin: Springer.

Pearce, D., and Valverde, A. 2008. Quantified equilibrium logic and foundations for answer set programs. In *Proceedings of ICLP 2008*, LNCS, volume 5366, 546–560. Berlin: Springer.

Pelov, N.; Denecker, M.; and Bruynooghe, M. 2007. Well-founded and stable semantics of logic programs with aggregates. *Theory and Practice of Logic Programming* 7(3):301–353.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence* 138:181–234.

Truszczynski, M. 2012. Connecting first-order ASP and the logic FO(ID) through reducts. a manuscript.

Van Gelder, A. 1993. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences* 47(1):185–221.

Vennekens, J.; Wittocx, J.; Mariën, M.; and Denecker, M. 2007. Predicate introduction for logics with a fixpoint semantics. part i: Logic programming. *Fundamenta Informaticae* 79(1-2):187–208.

Wittocx, J.; Mariën, M.; and Denecker, M. 2008. The IDP system: a model expansion system for an extension of classical logic. In Denecker, M., ed., *Logic and Search, Computation of Structures from Declarative Descriptions, LaSh 2008*, 153–165.