A Tarskian Informal Semantics for Answer Set Programming*

Marc Denecker[1], Yulia Lierler[2], Miroslaw Truszczynski[2], and Joost Vennekens[3]

1 **Department of Computer Science, K.U. Leuven**
3001 Heverlee, Belgium
marc.denecker@cs.kuleuven.be
2 **Department of Computer Science, University of Kentucky**
Lexington, KY 40506-0633, USA
yulia|mirek@cs.uky.edu
3 **Campus De Nayer | Lessius Mechelen | K.U. Leuven**
2860 Sint-Katelijne-Waver, Belgium
joost.vennekens@cs.kuleuven.be

Abstract

In their seminal papers on stable model semantics, Gelfond and Lifschitz introduced ASP by casting programs as epistemic theories, in which rules represent statements about the knowledge of a rational agent. To the best of our knowledge, theirs is still the only published systematic account of the intuitive meaning of rules and programs under the stable semantics. In current ASP practice, however, we find numerous applications in which rational agents no longer seem to play any role. Therefore, we propose here an alternative explanation of the intuitive meaning of ASP programs, in which they are not viewed as statements about an agent's beliefs, but as objective statements about the world. We argue that this view is more natural for a large part of current ASP practice, in particular the so-called Generate-Define-Test programs.

1998 ACM Subject Classification D.1.6 Logic Programming; I.2.4 Knowledge Representation Formalisms and Methods

Keywords and phrases Answer set programming, informal semantics, generate-define-test

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

The key postulate of declarative programing is that programs *reflect* the way information about a domain of discourse is described in natural language. The syntax must align with linguistic patterns we use and the formal semantics must capture the way we understand them. To put it differently, a declarative programming formalism (logic) must have an *informal semantics*, an intuitive and precise link between the formal syntax and semantics of the logic, and informal intended meanings of natural language expressions describing the "real world". This informal semantics explains what programs mean or, in other words, provides programs with an informal but precise natural language reading.

Having an informal semantics is important to a declarative programming formalism. It facilitates effective coding by offering intuitions to guide the programmer, and provides a basis for the programming methodology. It promotes understanding of code and helps in teaching how to program. It

* The first and the fourth author are 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.

28th International Conference on Logic Programming.
Editors: Agostino Dovier, Vítor Santos Costa; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

suggests extensions of the logic to facilitate expressing new types of knowledge, and it helps explain the relationship to other logics. The postulate is essential anywhere the emphasis on declarativeness is paramount, in particular, in addition to declarative programming, also in knowledge representation and database query languages and, more generally, in all contexts where we need to think how information about a domain of discourse has been or is to be encoded in a logic.

First-order (FO) logic has a clear informal semantics aligned with the classical *Tarskian* formal semantics of the logic. Indeed, the constructors of the FO language directly correspond to natural language connectives and expressions "for all" and "there is". The formal Tarskian semantics of these syntactic constructors is given by *interpretations* (*structures*), which are abstract mathematical representations of informally understood "possible objective state of affairs," and it literally reflects the informal understanding of the natural language connectives and quantifying expressions. It is that informal semantics that makes FO sentences legible and their intended meaning clear, and is largely responsible for the widespread use of FO logic in declarative programming, knowledge representation and database query languages.

Our main goal in this work is to analyze the role of informal semantics in the development of answer set programming (ASP) and its effective use. The key step is to clarify what informal semantics we have in mind. According to the intuitions Gelfond and Lifschitz exploited when introducing the stable-model (answer-set) semantics [10, 11], a program is a formal representation of a set of *epistemic* propositions believed by a rational introspective agent, and stable models of the program represent that agent's *belief sets*. This *epistemic* informal semantics linked ASP to autoepistemic logic by Moore [16] and default logic by Reiter [20], and supported applications in nonmonotonic reasoning. However, the epistemic perspective does not seem to be relevant to the way ASP is predominantly used now, as a formalism for modeling search problems [14, 17]. We argue that for such use of ASP a *Tarskian* informal semantics, not unlike the one for the FO logic, fits the bill better.

Interestingly, while the Tarskian informal semantics of ASP seems to have been implicitly followed by a vast majority of ASP users, it has never been explicitly identified or analyzed. We do so in this paper. We describe that informal semantics and show how it explains the way ASP developed and how it is intimately related to the currently dominating form of ASP, the *generate-define-test* (GDT) ASP [12]. We point out how the Tarskian informal semantics connects GDT ASP with the logic FO(ID). We argue that taking the Tarskian informal semantics seriously strongly suggests that the language of GDT ASP can be streamlined while in the same time generalizing the current one.

We present the Tarskian informal semantics for ASP in the context of a more general formalism, which we introduce first. We call it *first-order answer set programming* of ASP-FO for short. ASP-FO can be viewed as a modular first-order generalization of GDT ASP with unrestricted interpretations as models, with open and closed domains, non-Herbrand functions, and FO constraints and rule bodies. It is closely connected to the logic FO(ID) [3, 6] and has formal connections to the equilibrium logic [18]. ASP-FO generalizes GDT ASP and so the informal semantics we develop for ASP-FO applies to GDT ASP, too.

2 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 [12]. The role of GENERATE is to *generate the search space*. Nowadays this is often encoded by a set of choice rules:

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

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. This condition may refer to other generated predicates,

or to defined predicates. The DEFINE part is a set of definitions of some auxiliary predicates. Each definition is encoded by a group of rules

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

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 from the generated predicates or from other defined 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:

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

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.

GENERATE	$\{In(x,y)\} \leftarrow Edge(x,y).$
DEFINE	$Node(V).\ \ldots\ Node(W).$
	$Edge(V,V').\ \ldots\ Edge(W,W').$
	$T(x,y) \leftarrow In(x,y).$
	$T(x,y) \leftarrow T(x,z), T(z,y).$
TEST	$\leftarrow In(x,y), In(x,z), y \neq z.$
	$\leftarrow In(x,z), In(y,z), x \neq y.$
	$\leftarrow Node(x), Node(y), \texttt{not}\ T(x,y).$

(4)

Each of the three parts may again 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 which does make the internal structure of a GDT-program explicit.

3 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* 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 $\mathfrak{A} \models \varphi$.

4 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 $\langle \mathit{Ext}, \Pi \rangle$ 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 with $P \in \mathit{Ext}$, 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 $\mathit{Ext}(\mathcal{D})$. We write $\mathit{Par}(\mathcal{D})$ for the set of all other symbols in Π. We call $\mathit{Par}(\mathcal{D})$ 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), and 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:

$$\begin{array}{ll}
\text{GENERATE} & \{\forall x \forall y(\{In(x,y)\} \leftarrow Edge(x,y))\} \\
\hline
\text{DEFINE} & \{Vertex(V) \leftarrow \mathbf{t}, \ldots, Vertex(W) \leftarrow \mathbf{t}\} \\
& \{Edge(V,V') \leftarrow \mathbf{t}, \ldots, Edge(W,W') \leftarrow \mathbf{t}\} \\
& \left\{ \begin{array}{l} \forall x \forall y(T(x,y) \leftarrow In(x,y)) \\ \forall x \forall y(T(x,y) \leftarrow In(x,z) \wedge In(z,y)) \end{array} \right\} \\
\hline
\text{TEST} & \forall x \forall y \forall z \neg(In(x,y) \wedge In(x,z) \wedge y \neq z) \\
& \forall x \forall y \forall z \neg(In(x,z) \wedge In(y,z) \wedge x \neq y) \\
& \forall x \forall y(Vertex(x) \wedge Vertex(y) \Rightarrow T(x,y))
\end{array} \tag{6}$$

In defining the formal semantics of ASP-FO, we aim to ensure that three conditions are satisfied. First, the structures are to be viewed as possible worlds, i.e., they should 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 therefore simply define that 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. [19] and, in the way we follow here, by Vennekens et al. [22]. It uses a pair of interpretations to simulate the construction of the Gelfond-Lifschitz reduct.

▶ **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 $d \in dom(I)$, $(\mathfrak{A}, \mathfrak{B}), \theta[x : d] \models \psi$.

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 $\mathfrak{B}$.

For two structures $\mathfrak{A}$ and $\mathfrak{B}$ that have the same domain and interpret disjoint vocabularies, $\mathfrak{A} \circ \mathfrak{B}$ denotes the structure that interprets the union of the vocabularies of $\mathfrak{A}$ and $\mathfrak{B}$, has the same domain as $\mathfrak{A}$ and $\mathfrak{B}$, and coincides with $\mathfrak{A}$ and $\mathfrak{B}$ 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[1] 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 $\mathfrak{A}, \theta \models P(\bar{t})$.

[1] 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 in three ways: it is parameterized, i.e., it builds stable models on top of a given interpretation of the parameter symbols; it handles FO bodies; and it works for arbitrary (not only Herbrand) interpretations.

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

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 $\forall \bar{y}\ (\{P(\bar{y})\} \leftarrow \exists \bar{x}(\bar{y} = \bar{t} \wedge \varphi))$. Next, any finite set of choice rules $\forall \bar{x}\ (\{P(\bar{x})\} \leftarrow \varphi_i)$ can be combined into a single choice rule $\forall \bar{x}\ (\{P(\bar{y})\} \leftarrow \varphi_1 \vee \cdots \vee \varphi_n))$. 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. Thus, G-modules are redundant in ASP-FO, since they can be simulated by T-modules.

▶ **Theorem 10.** *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: $\forall x \forall y(In(x, y) \Rightarrow Edge(x, y))$.

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

▶ **Definition 11.** 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, $t^{\mathcal{M}} = t$.

Herbrand modules are 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. Herbrand modules can be expressed by means of D- and T-modules (as in the logic FO(ID) [3]). Thus, they are redundant.

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 12.** *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*[2] 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.

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 13.** *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 13 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 13 shows that ASP-FO (i) extends this fragment of ASP in a direct way, and (ii) interprets those ASP programs as the *monotone conjunction* of their components.

Theorem 13 fails to take into account three common extensions of the ASP language: aggregates (or weight expressions), disjunction in the head, and strong negation. Each of these limitations can be lifted (we do not discuss the details due to space restrictions).

Relation to FO(ID). A theory in FO(ID) is a set of FO sentences and *inductive definitions*.[3] 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 14.** 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 [7].

Denecker and Ternovska [6] 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. Syntactic conditions such as no negative occurrences of defined symbols in the bodies of rules (defining the class of *positive* definitions), predicate stratification and local stratification all guarantee that a definition is total. Thus, many definitions occurring in practice are total. For total definitions (D-modules), the well-founded and stable models coincide [19]. Consequently, the logics ASP-FO and FO(ID) restricted to total definitions (D-modules) coincide, too.

[2] The conditions on splitting follow the requirements stated in the Symmetric Splitting Theorem in [9].
[3] Some versions of FO(ID) allow also boolean combinations of FO formulas and definitions [6].

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. There is a formal connection between ASP-FO D-modules and extensions of ASP to the first-order setting based on equilibrium logic [18] and the operator SM [8]. We consider the latter two under the restriction to formulas representing rules of the form (5). In this case, the semantics coincide if the bodies of rules (5) have no nested occurrences of negation [21]. 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 these generalizations also $\{P\}$ is a model. More importantly though, they differ at the conceptual level. The logic ASP-FO directly extends FO. The equilibrium logic version of first-order ASP is based on the quantified logic HT that differs substantially from FO and, arguably, lacks its direct connection to everyday linguistic patterns.

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

A formal semantics is just a mathematical definition and therefore, by itself, it does not yet explain how expressions in a logic relate to the real world. For this, also an *informal semantics* is needed, i.e., an intuitive interpretation for the logic's syntactic and semantic objects. In this paper, we interpret an answer set as a Tarskian representation of a possible state of the world. The aim of this section is to investigate in detail how the connectives of ASP should be understood in this new perspective.

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 meanings 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. 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.

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

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

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, $\forall x \forall y(In(x,y) \Rightarrow Edge(x,y))$, the 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 and do so using a *rule-based, potentially recursive* syntax [12]. 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 [2, 3] was first to argue that rules under the well-founded semantics provide a uniform and correct formalization of these. Later, Denecker et al. [5] and Denecker and Ternovska [6] 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* [3, 6]. Since parameterized stable and well-founded semantics coincide for total definitions, the above arguments apply immediately also to total D-modules. Therefore, the work by Denecker and his coauthors also provides a detailed explanation of why total D-modules under the stable model semantics correctly formalize the natural language concept of an inductive definition. To the best of our knowledge, such an explanation has not yet appeared in print before.

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. 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 $\sim P(\bar{x}) \leftarrow \texttt{not}\ P(\bar{x})$ involving strong negation [11]. 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. A more detailed discussion on the importance of the informal semantics of the "models" of a logic program can be found in [4].

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 but it does not entail $\neg P$. 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 [13].

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. Indeed, let us look at what negation means in informal definitions, for instance, in 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 [7], 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.

6 Discussion

Interpreting the answer-set semantics as a Tarskian possible-world semantics is a major mental leap which 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. 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.

We proposed ASP-FO as a theoretical mechanism to study GDT and ASP from a Tarskian perspective. However, ASP-FO is also a viable ASP logic for which several efficient ASP tools exists. Similarly to FO(ID), ASP-FO 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) [6]. 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* [15] and the corresponding tools [1]. Also, the IDP system [23] implements both the FO(ID) and ASP-FO semantics.

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.

References

1 A. Aavani, X. Wu, S. Tasharrofi, E. Ternovska, and D. G. Mitchell. Enfragmo: A system for modelling and solving search problems with logic. In N. Bjørner and A. Voronkov, editors, *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2012*, volume 7180 of *LNCS*, pages 15–22, Berlin, 2012. Springer.

2 M. Denecker. The well-founded semantics is the principle of inductive definition. In J. Dix, L. Fariñas del Cerro, and U. Furbach, editors, *Proceedings of the European Workshop on Logics in Artificial Intelligence, JELIA 1998*, volume 1489 of *LNCS*, pages 1–16, Berlin, 1998. Springer.

3 M. Denecker. Extending classical logic with inductive definitions. In J.W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L.M. Pereira, Y. Sagiv, and P.J. Stuckey, editors, *Proceedings of First International Conference on Computational Logic, CL 2000*, volume 1861 of *LNCS*, pages 703–717, Berlin, 2000. Springer.

4 M. Denecker. What's in a model? Epistemological analysis of logic programming. In D. Dubois, C.A. Welty, and M.-A. Williams, editors, *Proceedings of the 9th International Conference on Prin-*

ciples of Knowledge Representation and Reasoning, pages 106–113, Palo Alto, CA, 2004. AAAI Press.

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

6 M. Denecker and E. Ternovska. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic*, 9(2):1–50, 2008.

7 M. Denecker and J. Vennekens. Well-founded semantics and the algebraic theory of non-monotone inductive definitions. In C. Baral, G. Brewka, and J.S. Schlipf, editors, *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2007*, volume 4483 of *LNCS*, pages 84–96, Berlin, 2007. Springer.

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

9 P. Ferraris, J. Lee, V. Lifschitz, and R. Palla. Symmetric splitting in the general theory of stable models. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI-2009*, pages 797–803, Palo Alto, CA, 2009. AAAI Press.

10 M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080, Cambridge, MA, 1988. MIT Press.

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

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

13 V. Lifschitz and H. Turner. Representing transition systems by logic programs. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 1730 of *LNCS*, pages 92–106, Berlin, 1999. Springer.

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

15 D.G. Mitchell and E. Ternovska. A framework for representing and solving NP search problems. In *Proceedings of the 20th National Conference on Artificial Intelligence, AAAI 2005*, pages 430–435, Palo Alto, CA, 2005. AAAI Press.

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

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

18 D. Pearce and A. Valverde. Quantified equilibrium logic and foundations for answer set programs. In *Proceedings of the 24th International Conference on Logic Programming, ICLP 2008*, volume 5366 of *LNCS*, pages 546–560, Berlin, 2008. Springer.

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

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

21 M. Truszczynski. Connecting first-order asp and the logic FO(ID) through reducts, In E. Erdem, Y. Lierler, Y. Lierler, and D. Pearce, editors, *Correct Reasoning, Essays of Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *LNCS*, Berlin, 2012. Springer.

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

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