2022

Historical Review of Variants of Informal Semantics for Logic Programs under Answer Set Semantics: GL'88, GL'91, GK'14, D-V'12

Yuliya Lierler

TPLP: Page 1–13. © The Author(s), 2021. Published by Cambridge University Press 2021
doi:10.1017/xxxxx

Historical Review of Variants of Informal Semantics for Logic Programs under Answer Set Semantics: GL'88, GL'91, GK'14, D-V'12

YULIYA LIERLER
University of Nebraska Omaha

Abstract

This note presents a historical survey of informal semantics that are associated with logic programming under answer set semantics. We review these in uniform terms and align them with two paradigms: Answer Set Programming and ASP-Prolog— two prominent Knowledge Representation and Reasoning Paradigms in Artificial Intelligence.

1 Introduction

The transcript of the talk by Donald E. Knuth titled *Let's Not Dumb Down the History of Computer Science* published by ACM (2021) includes the statement:

> ... it would really be desirable if there were hundreds of papers on history written by computer scientists about computer science.

This quote was inspirational for this technical note devoted to a historical survey of informal semantics that are associated with logic programming under answer set semantics (in the sequel we drop *under answer set semantics* when referring to logic programming and logic programs).

We focus on four seminal publications and align informal semantics discussed there using the same style of presentation and propositional programs. We trust that within such settings key ideas and tangible differences between the distinct views come to the surface best. The earliest publication of the four dates to 1988, the latest dates to 2014. It would seem that the subject of informal semantics is only peripheral scoring at such a low count of major references. Rather, the word *informal* makes this subject rare in the discussions of logic programming. Nevertheless, 2014 reference is an introductory chapter titled *Informal Semantics* of the textbook on *Knowledge Representation, Reasoning, and Design of Intelligent Agents* by Gelfond and Kahl. The prominent position of this chapter points at the importance of the subject, *especially when we consider to pass on the knowledge and practice of logic programming to broad audience*.

As the presentation unfolds, a story of two views on logic programs will emerge: one via the prism of answer set programming (ASP) and another via the prism of ASP-Prolog. We reserve the term — ASP — to constraints programming paradigm, where one while coding specifications of a considered problem also realizes the form of the solutions to this problem as answer sets of the program. The term — ASP-Prolog — is used to denote a knowledge representation language geared to model and capture domain knowledge with the underlying intelligent agent in mind. One may utilize ASP-Prolog as a programming language, but may also simply use it for describing specifications without thinking about a computational task or solving this task.

This presentation of the four surveyed publications almost follows their timeline starting with the earliest work. In many places we present the original quotes from the discussed sources to avoid misrepresentation of the originals.

2 Formal and informal semantics of basic programs by GL'88

We start by recalling the formal and informal semantics of basic logic programs as they were introduced by Gelfond and Lifschitz (1988).

A *basic rule* is an expression of the form

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

where A, B_i, and C_j are propositional atoms. The atom A is the *head* of the rule and expression $B_1, \ldots, B_n, \texttt{not}\ C_1, \ldots, \texttt{not}\ C_m$ is its *body*. A *basic (logic) program* is a finite set of such rules. In the sequel we introduce rules of somewhat different syntactic structure, yet we agree to call the left hand side of the rule operator, denoted by $\leftarrow$, *head* and the right hand side *body*.

For a rule r of the form (1) and a set X of atoms, the reduct r^X is defined whenever there is no atom C_j for $j \in \{1, \ldots, m\}$ such that $C_j \in X$. If the reduct r^X is defined, then it is the rule

$$A \leftarrow B_1, \ldots, B_n. \tag{2}$$

The reduct P^X of the program P consists of the rules r^X for all $r \in P$, for which the reduct is defined. A set X of atoms *satisfies* rule (2) if A belongs to X or there exists $i \in \{1, \ldots, n\}$ such that $B_i \notin X$. We say that a set X of atoms is a *model* of a program consisting of rules of the form (2), when X satisfies all rules of this program. A set X is a *stable model/answer set* of P, denoted $X \models_{st} P$, if it is a $\subseteq$-least model of P^X.

Quotes by Gelfond and Lifschitz (1988) on Intuitive Meaning of Basic Programs (verbatim modulo names for programs and sets of atoms):

Quote 1: The intuitive meaning of stable sets can be described in the same way as the intuition behind "stable extensions" in autoepistemic logic: they are "possible sets of beliefs that a rational agent might hold" (Moore 1985) given P as his premises. If X is the set of (ground) atoms that I consider true, then any rule that has a subgoal *not* C with $C \in X$ is, from my point of view, useless; furthermore, any subgoal *not* C with $C \notin X$ is, from my point of view, trivial. Then I can simplify the premises P and replace them by P^X. If X happens to be precisely the set of atoms that logically follow from the simplified set of premises P^X, then I am "rational".

Later, Gelfond and Lifschitz (1991) say about a basic program the following

Quote 2: A "well-behaved" program has exactly one stable model, and the answer that such a program is supposed to return for a ground query A is *yes* or *no*, depending on whether A belong to the stable model or not. (The existence of several stable models indicates that the program has several possible interpretations).

The historical roots of stable model semantics for logic programs as a formal tool for model theoretic declarative semantics of Prolog are apparent in these quotes. The expectation is to consider a well-behaved program with a single stable model. Yet, the authors acknowledge the possibility of programs with several stable models that *indicates that the program has several possible interpretations* or several *possible sets of beliefs*. In 1988 the distinction between *possible interpretations* and *possible sets of beliefs* was not apparent and the terms where used synonymously. Yet, the former lent itself to ASP. The later inspired ASP-Prolog. We now review these frameworks.

Answer Set Programming Marek and Truszczynski (1999) and Niemelä (1999) open a new era for stable model semantics by proposing the use of logic programs as constraint programming paradigm for modeling combinatorial search problems. This marks the birth of *ASP*. Here is what the abstract of the paper by Marek and Truszczynski says:

> We demonstrate that inherent features of stable model semantics naturally lead to a logic programming system that offers an interesting alternative to more traditional logic programming... The proposed approach is based on the *interpretation of program clauses as constraints*. In this setting programs do not describe a single intended model, but a family of stable models. These stable models encode solutions to the constraint satisfaction problem described by the program. ... We argue that the resulting logic programming system is well-attuned to problems in the class NP, has a well-defined domain of applications, and an emerging methodology of programming.

Lifschitz (2002) coins a term *generate-define-test* for this emerging methodology that splits program rules into three groups:

- the *generate* group is responsible for defining a large class of "potential solutions";
- the *test* group is responsible for stating conditions to weed out potential solutions that do not satisfy problem's specifications; and
- the *define* group is responsible for defining concepts that are essential in stating the conditions of *generate* and *test*.

In this work, "the idea of ASP is to represent a given computational problem by a program whose answer sets correspond to solutions, and then use an answer set solver to find a solution". The paper illustrates the use of ASP to solve a sample planing problem. Yet, there are no references to how one would intuitively read, for example, an occurrence of atom *A* or expression *not A* in rules. To the best of our knowledge Denecker et al. (2012) in addition to earlier reviewed quotes in this section are the major two accounts for reconciling the use of ASP (not ASP-Prolog) in practice and intuitive readings of answer sets and rules of programs. The previous to the last section reviews an account by Denecker et al.

ASP-Prolog Interpreting answer sets as possible sets of beliefs implies the presence of an intelligent agent behind a program. This champions the view of a logic program as a knowledge representation and reasoning formalism for the design of intelligent agents. This is largely a view advocated in the textbook by Gelfond and Kahl (2014). Brief discussions by Gelfond and Lifschitz (1991) and Section 2.2.1 of the mentioned textbook are major two accounts that speak of the use of ASP-Prolog in practice and intuitive readings of answer sets and rules of programs. The following two sections of this note are devote to these accounts.

"Formalizing" Quotes 1 and 2 We now attempt to make the claims of the first quote precise with the allowance that programs with multiple answer sets that correspond to possible sets of beliefs/possible interpretations are *the first class citizens*. In other words, each answer set represents a set of beliefs of a rational agent (or *I*). In the sequel we drop the reference to "or I" and use "an agent" in the discourse. This agent may have multiple sets of beliefs — each answer set corresponds to a belief set. We denote this informal semantics as $\mathcal{G}_{\mathscr{I}}$, where $\mathscr{I}$ stands for *intended interpretations* of program's propositional atoms. It is typical in the informal semantics for classical logic expressions that each atom *A* has an intended interpretation, $\mathscr{I}(A)$, which is represented linguistically as a noun phrase about the application domain. The informal semantics $\mathcal{G}_{\mathscr{I}}$ consists of three components

- the interpretation of structures — here, answer sets — denoted by $\mathcal{G}^{\mathbb{S}}_{\mathscr{I}}$, and

Table 1. *The Gelfond-Lifschitz (1988) informal semantics of answer sets – sets of atoms.*

A set X of atoms	A state $\mathcal{G}^{\mathbb{S}}_{\mathscr{I}}(X)$ of affairs
$A \in X$ for atom A	$\mathscr{I}(A)$ is true in state $\mathcal{G}^{\mathbb{S}}_{\mathscr{I}}(X)$ of affairs
$A \notin X$ for atom A	$\mathscr{I}(A)$ is false in state $\mathcal{G}^{\mathbb{S}}_{\mathscr{I}}(X)$ of affairs

- the interpretation of syntactical expressions in a program, denoted by $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}$,
- the interpretation of semantic relations such as satisfaction, denoted by $\mathcal{G}^{\models}_{\mathscr{I}}$.

The first component determines a function from an answer set/a set of beliefs encoded by a set X of atoms to belief states of an agent that have abstraction X (note, we cannot distinguish between these belief states using abstraction X.) The second component determines the informal reading of syntactical expressions in a program. The third component determines the informal reading of satisfaction relation.

In the view of informal semantics $\mathcal{G}_{\mathscr{I}}$, an answer set serves a role of an abstraction of *belief states* of some agent. Yet, we can also identify an answer set X with a *possible state of affairs*: indeed, *An agent in some belief state – represented by set X of atoms – considers the set of all atoms in X to be the case (believes in them), whereas any atom A that does not belong to X is believed to be false by the agent, i.e., is not the case.* Thus, we may explain the meaning of a program in terms of what atoms an agent with its knowledge of the application domain encoded as the program believes as true and what atoms an agent believes as false. Generally, an agent in some belief state considers certain states of affairs as possible and the others as impossible. For basic programs, set X of atoms defines a *unique* state of affairs that agent regards as possible in a belief state that X represents. Thus, we may identify any belief state captured by X with this state of affairs. We denote this state of affairs under an intended interpretation $\mathscr{I}$ as $\mathcal{G}^{\mathbb{S}}_{\mathscr{I}}(X)$. Table 1 summarizes this abstraction function.

Example 1
Consider a set of beliefs encoded as a set

$$X = \{student(mary), male(john)\} \tag{3}$$

of atoms under the obvious intended interpretation $\mathscr{I}$ for the propositional atoms in X. This X represents a state of affairs in which the agent considers that both statements *Mary is a student* and *John is a male* are true. At the same time the agent considers any other statements, including *John is a student* and *Mary is a male*, false. The $\mathcal{G}^{\mathbb{S}}_{\mathscr{I}}(X)$ component of informal semantics of basic programs provides us with this understanding of our sample X.

Table 2 shows the Gelfond-Lifschitz (1988) informal semantics $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}$ of syntactic elements of programs. As it is clear from this table, under $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}$, extended logic programs have both classical and non-classical connectives. On the one hand, the comma operator is classical conjunction and the rule operator $\leftarrow$ is classical implication. On the other hand, the implicit composition operator (constructing a program out of individual rules) is non-classical, because it performs a closure operation (resulting in the implementation of closed-world assumption): the agent knows

Table 2. *The Gelfond-Lifschitz (1988) informal semantics for basic logic program.*

	Φ	$\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}(\Phi)$
1.	propositional atom A	$\mathscr{I}(A)$
2.	expression of the form `not` C	it is not the case that $\mathscr{I}(C)$
3.	expression of the form Φ_1, Φ_2	$\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}(\Phi_1)$ and $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}(\Phi_2)$
4.	rule $Head \leftarrow Body$	if $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}(Body)$ then $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}(Head)$ *(in the sense of material implication)*
5.	program $P = \{r_1, \ldots, r_n\}$	All the agent knows is: $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}(r_1)$ and $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}(r_2)$ and $\ldots$ $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}(r_n)$

Table 3. *The Gelfond-Lifschitz (1988) informal semantics for the satisfaction relation.*

$\models_{st}$	$\mathcal{G}^{\models_{st}}_{\mathscr{I}}$
$X \models_{st} P$	Given $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}(P)$, X could be a state of affairs inferred from this knowledge so that any proposition in X is the case whereas any proposition not in X is not the case

only what is explicitly stated. Table 3 presents the final component $\mathcal{G}^{\models_{st}}_{\mathscr{I}}$ of the $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}$ informal semantics.

3 Formal and informal semantics of extended programs by GL'91

Here, we recall the formal and informal semantics of extended logic programs by Gelfond and Lifschitz (1991). An alternative view of the informal semantics for extended logic programs is provided in (Gelfond and Kahl 2014, Section 2.2.1) reviewed next.

A *literal* is either an atom A or an expression $\neg A$, where A is an atom. An *extended rule* is an expression of the form (1), where A, B_i, and C_j are propositional literals. An *extended program* is a finite set of extended rules. Gelfond and Lifschitz (1991) also considered *disjunctive* rules of the form D_1 *or* $\ldots$ *or* $D_l \leftarrow B_1, \ldots, B_n, \texttt{not}\, C_1, \ldots, \texttt{not}\, C_m$, where D_i, B_i, and C_j are propositional literals. Yet, the discussion of such rules is outside of this note.

A *consistent* set of propositional literals is a set that does not contain both A and its *complement* $\neg A$ for any atom A. A *believed literal set* X is a consistent set of propositional literals. A believed literal set X *satisfies* an extended rule r of the form (1) if A belongs to X or there exists an $i \in \{1, \ldots, n\}$ such that $B_i \notin X$ or a $j \in \{1, \ldots, m\}$ such that $C_j \in X$. A believed literal set is a *model* of a program P if it satisfies all rules in P. For a rule r of the form (1) and a believed literal set X, the reduct r^X is defined whenever there is no literal C_j for $j \in \{1, \ldots, m\}$ such that $C_j \in X$. If the reduct r^X is defined, then it is the extended rule of the form (2). The reduct P^X of

the program P consists of the rules r^X for all $r \in P$, for which the reduct is defined. A believed literal set X is an *answer set* of P, denoted $X \models_{st} P$, if it is a $\subseteq$-least model of P^X.

Quotes by Gelfond and Lifschitz (1991) on Intuitive Meaning of Extended Programs

Quote 3: For an extended program, we will define when a set X of ground *literals* qualifies as its *answer set*. ... A "well-behaved" extended program has exactly one answer set, and this set is consistent. The answer that the program is supposed to return to a ground query A is *yes*, *no*, or *unknown*, depending on whether the answer set contains A, $\neg A$, or neither. The answer *no* corresponds to the presence of explicit negative information in the program.

Consider, for instance, the extended program Π_1 consisting of just one rule:

$$\neg q \leftarrow not\ p.$$

Intuitively, this rule means: "q is false, if there is no evidence that p is true." We will see that the only answer set of this program is $\{\neg q\}$. The answers that the program should give to the queries p and q are, respectively *unknown* and *false*.

As another example, compare two programs that don't contain *not*:

$$\neg p \leftarrow, \quad p \leftarrow \neg q \qquad \text{and} \qquad \neg p \leftarrow, \quad q \leftarrow \neg p$$

... Thus our semantics is not "contrapositive" with respect to $\leftarrow$ and $\neg$; it assigns different meanings to the rules $p \leftarrow \neg q$ and $q \leftarrow \neg p$. The reason is that it interprets expressions like these as *inference rules*, rather than conditionals.

This quote parallels Quote 2 about basic programs: the notion of well-behaved program resurfaces. In comparison to basic programs, extended programs provide us with a new possibility to answer queries against a program — namely, *unknown*. The following quote parallels Quote 1 about basic programs:

The answer sets of Π are, intuitively, possible sets of beliefs that a rational agent may hold on the basis of information expressed by the rules of Π. If X is the set of (ground) literals that the agent believes to be true, then any rule that has a subgoal *not* L with $L \in X$ will be of no use to him, and he will view any subgoal *not* L with $L \notin X$ as trivial. Thus he will be able to replace the set of rules Π by the simplified set of rules Π^X. If the answer set of Π^X coincides with X, then the choice of X as the set of beliefs is "rational".

The following quote states precise relationship between basic and extended programs:

Quote 4: the semantics of extended programs, applied to basic programs, turns into the stable model semantics. But there is one essential difference: The absence of an atom A in a stable model of a general program represents the fact that A is false; the absence of A and $\neg A$ in an answer set of an extended program is taken to mean that nothing is known about A.

In the section on *Representing Knowledge Using Classical Negation*, Gelfond and Lifschitz (1991) say

The difference between *not* p and $\neg p$ in a logic program is essential whenever we cannot assume that the available positive information about p is complete, i.e., when the "closed world assumption" [Reiter, 1978] is not applicable to p. The closed world assumption for a predicate p can be expressed in the language of extended programs by the rule

$$\neg p \leftarrow \texttt{not}\ p$$

When this rule is included in the program, *not* p and $\neg p$ can be used interchangeably in the bodies of other rules. Otherwise, we use *not* p to express that p is not known to be true, and $\neg p$ to express that p is false.

To summarize, Gelfond and Lifschitz describe an informal semantics for extended programs based on *epistemic* notions of default and autoepistemic reasoning. We now present the infor-

Table 4. *The Gelfond-Lifschitz (1991) informal semantics of answer sets – sets of literals.*

A believed literal set X	A belief state $B \in \mathcal{GL}^{\mathbb{S}}_{\mathscr{I}}(X)$ that has abstraction X
$A \in X$ for atom A	B has the belief that $\mathscr{I}(A)$ is true; i.e., $\mathscr{I}(A)$ is true in all states of affairs possible in B
$\neg A \in X$ for atom A	B has the belief that $\mathscr{I}(A)$ is false; i.e., $\mathscr{I}(A)$ is false in all states of affairs possible in B
$A \notin X$ for atom A	B does not have the belief that $\mathscr{I}(A)$ is true; i.e., $\mathscr{I}(A)$ is false in some state of affairs possible in B
$\neg A \notin X$ for atom A	B does not have the belief that $\mathscr{I}(A)$ is false; i.e., $\mathscr{I}(A)$ is true in some state of affairs possible in B

mal semantics $\mathcal{GL}_{\mathscr{I}}$ for extended programs just as we presented $\mathcal{G}_{\mathscr{I}}$ for basic programs. This presentation at times (in particular, Example 2) follows the lines by Denecker et al. (2019).

We begin by discussing a crucial difference between $\mathcal{G}_{\mathscr{I}}$ and $\mathcal{GL}_{\mathscr{I}}$. Informal semantics $\mathcal{GL}_{\mathscr{I}}$ views a believed literal set X as an abstraction of a belief state of some agent; $\mathcal{G}_{\mathscr{I}}$ views a set X of atoms as an abstraction of a state of affairs. The change from "sets of atoms" to "sets of literals" is crucial. Recall how an agent in some belief state considers certain states of affairs as possible and the others as impossible. Within $\mathcal{G}_{\mathscr{I}}$, set X of atoms ends up to represent a unique possible state of affairs associated with a belief state so that we may identify these two concepts. Yet, believed literal set X is the set of all literals L that the agent believes in, that is, those that are true in all states of affairs that agent regards as possible. Importantly, it is not the case that a literal L that does not belong to X is believed to be false by the agent. Rather, it is *not believed* by the agent or as stated in Quote 4 *nothing is known about L* to the agent. Denecker et al. (2019) takes the following interpretation of a statement *literal L is not believed by an agent/nothing is known about L*: literal L is false in some states of affairs the agent holds possible, and L must be true in at least one of the agent's possible states of affairs (unless the agent believes the complement of L). This note adopts such an interpretation. We denote the class of informal belief states that are abstracted to a given formal believed literal set X under an intended interpretation $\mathscr{I}$ as $\mathcal{GL}^{\mathbb{S}}_{\mathscr{I}}(X)$. Table 4 summarizes this abstraction function.

Example 2

We may view this example as a continuation of Example 1. Here we consider what would seem the same belief set but change the perspective on it from the point of view of informal semantics of basic programs to that of extended programs. Consider believed literal set (3) under the obvious intended interpretation $\mathscr{I}$ for the elements in X. This X is the abstraction of any belief state in which the agent believes that *Mary is a student* and *John is a male*, and nothing is known about such statements as *John is a student* or *Mary is a male*. One such belief state is the state B_0 in which the agent considers the following states of affairs as possible:

1. John is the only male in the domain of discourse; Mary is the only student.
2. John and Mary are both male students.
3. John and Mary are both male; Mary is the only student.

Table 5. *The Gelfond-Lifschitz (1991) informal semantics for some expressions in extended programs.*

Φ	$\mathcal{GL}^{\mathbb{L}}_{\mathscr{I}}(\Phi)$
propositional literal $\neg A$	it is not the case that $\mathscr{I}(A)$
expression of the form `not` C	the agent does not know that $\mathcal{GL}^{\mathbb{L}}_{\mathscr{I}}(C)$

Table 6. *The Gelfond-Lifschitz (1991) informal semantics for the satisfaction relation.*

$\models_{st}$	$\mathcal{GL}^{\models_{st}}_{\mathscr{I}}$
$X \models_{st} P$	Given $\mathcal{GL}^{\mathbb{L}}_{\mathscr{I}}(P)$, X could be the set of literals the agent believes

4. John is the only male; John and Mary are both students

Another belief state corresponding to X is the state B_1 in which the agent considers the states of affairs 2-4 of B_0 as possible. Indeed, for each of these belief states, it holds that Mary is a student and John is a male in all possible states of affairs of that belief state. Thus, each of the literals in X is believed in each of the belief states B_0 and B_1. On the other hand, John is a student precisely in the state of affairs 2 and 4; Mary is a male in the states of affairs 2 and 3. Hence, literals $\neg student(john)$ and $\neg male(mary)$ are not believed in either of the two belief states B_0 and B_1.

The component $\mathcal{GL}^{\mathbb{L}}_{\mathscr{I}}$ captures the informal readings of the connectives of the Gelfond-Lifschitz (1991) informal semantics of extended programs. We summarize it by (i) the entries in rows 1, 3-5 of Table 2, where we replace $\mathcal{G}^{\mathbb{L}}_{\mathscr{I}}$ by $\mathcal{GL}^{\mathbb{L}}_{\mathscr{I}}$, and (ii) the entries in Table 5. The definition of $\mathcal{GL}^{\mathbb{L}}_{\mathscr{I}}$ suggests that of the two negation operators, symbol $\neg$ is classical negation, whereas `not` is a non-classical negation. It is commonly called *default negation*. The component $\mathcal{GL}^{\models_{st}}_{\mathscr{I}}$ explains what it means for a believed literal set X to be an answer set/stable model of an extended program. Table 6 presents its definition.

Provided account of informal semantics of extended logic programs echos the interpretation of an extended program as a possible "set of beliefs" and can be seen as an informal semantics for the syntactic constructs that are fundamental in ASP-Prolog.

4 Informal semantics of extended logic programs by GK'14

Gelfond and Kahl (2014) consider a language of extended logic programs with addition of (i) disjunctive rules and (ii) rules called *constraints* that have the form

$$\leftarrow B_1, \ldots, B_n, \texttt{not}\ C_1, \ldots, \texttt{not}\ C_m, \tag{4}$$

where B_i, and C_j are propositional literals (empty head can be identified with $\bot$). It is due to note that constraints are the first class citizens in the use of ASP since a while. In particular, they are the kinds of rules that populate the *test* group of *generate-define-test* programs mentioned earlier. We come back to this point in the next section. To generalize the concept of an answer set to extended programs with constraints it is sufficient to provide a definition of rule satisfaction when the head of the rule is empty: A believed literal set X *satisfies* a rule of the form (1), where A is empty if there exists an $i \in \{1,\ldots,n\}$ such that $B_i \notin X$ or a $j \in \{1,\ldots,m\}$ such that $C_j \in X$. As before, we omit presenting definitions for programs with disjunctive rules.

Quote by Gelfond and Kahl (2014) on Intuitive Meaning of Extended Programs with Constraints

Informally, program Π can be viewed as a specification for answer sets —- sets of beliefs that could be held by a rational reasoner associated with Π. Answer sets are represented by collections of ground literals. In forming such sets the reasoner must be guided by the following informal principles:

1. Satisfy the rules of Π. In other words, believe in the head of a rule if you believe in its body.
2. Do not believe in contradictions.
3. Adhere to the "Rationality Principle" that says, "Believe nothing you are not forced to believe."

Let's look at some examples. ...

Example 2.2.1.

$p(b) \leftarrow q(a).$ "Believe $p(b)$ if you believe q(a)"
$q(a).$ "Believe $q(a)$"

Note that the second rule is a fact. Its body is empty. Clearly any set of literals satisfies an empty collection, and hence, according to our first principle, we must believe $q(a)$. The same principle applied to the first rule forces us to believe $p(b)$. The resulting set $S1 = \{q(a), p(b)\}$ is consistent and satisfies the rules of the program. Moreover, we had to believe in each of its elements. Therefore, it is an answer set of our program. Now consider set $S2 = \{q(a), p(b), q(b)\}$. It is consistent, satisfies the rules of the program, but contains the literal $q(b)$, which we were not forced to believe in by our rules. Therefore, $S2$ is not an answer set of the program. ...

Example 2.2.2. *(Classical Negation)*

$\neg p(b) \leftarrow \neg q(a).$ "Believe that $p(b)$ is false if you believe $q(a)$ is false"
$\neg q(a).$ "Believe that $q(a)$ is false"

There is no difference in reasoning about negative literals. In this case, the only answer set of the program is $\{\neg p(b), \neg q(a)\}$. ...

Example 2.2.4. *(Constraints)*

$p(a)$ *or* $p(b).$ "Believe $p(a)$ or believe $p(b)$"
$\leftarrow p(a).$ "It is impossible to believe $p(a)$"

The first rule forces us to believe $p(a)$ or to believe $p(b)$. The second rule is a constraint that prohibits the reasoner's belief in $p(a)$. Therefore, the first possibility is eliminated, which leaves $\{p(b)\}$ as the only answer set of the program. In this example you can see that the constraint limits the sets of beliefs an agent can have, but does not serve to derive any new information. Later we show that this is always the case. ...

Example 2.2.5. *(Default Negation)* Sometimes agents can make conclusions based on the absence of information. For example, an agent might assume that with the absence of evidence to the contrary, a class has not been canceled. ... Such reasoning is captured by default negation. Here are two examples.

$p(a) \leftarrow$ `not` $q(a).$ "If $q(a)$ does not belong to your set of beliefs then $p(a)$ must"

No rule of the program has $q(a)$ in its head, and hence, nothing forces the reasoner, which uses the program as its knowledge base, to believe $q(a)$. So, by the rationality principle, he does not. To satisfy the only rule of the program, the reasoner must believe $p(a)$; thus, $\{p(a)\}$ is the only answer set of the program. ...

Table 7. *The Gelfond-Kahl (2014) informal semantics for extended programs with constraints*

Φ	$\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(\Phi)$
propositional atom A	believe $\mathscr{I}(A)$
propositional literal $\neg A$	believe that $\mathscr{I}(A)$ is false
expression of the form `not` C	the agent is not made to $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(C)$
expression of the form Φ_1, Φ_2	$\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(\Phi_1)$ and $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(\Phi_2)$
constraint $\leftarrow Body$	it is impossible to $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(Body)$
rule $Head \leftarrow Body$	if $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(Body)$ then $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(Head)$ *(in the sense of material implication)*
program $P = \{r_1, \ldots, r_n\}$	All the agent believes is: $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(r_1)$ and $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(r_2)$ and $\ldots$ $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}(r_n)$

We now state the informal semantics hinted by the quoted examples in unifying terms of this paper. We denote it by $\mathcal{GK}_{\mathscr{I}}$ and detail its three components $\mathcal{GK}^{\mathbb{S}}_{\mathscr{I}}$, $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}$, and $\mathcal{GK}^{\models}_{\mathscr{I}}$. To begin with $\mathcal{GK}^{\mathbb{S}}_{\mathscr{I}}$ coincide with $\mathcal{GL}^{\mathbb{S}}_{\mathscr{I}}$.

Tables 7 presents $\mathcal{GK}^{\mathbb{L}}_{\mathscr{I}}$. In this presentation we take the liberty to identify an expression *(proposition) p does not belong to your set of beliefs* used in the examples of the quote listed last with expression *the agent is not made to believe (proposition) p*. We summarize $\mathcal{GK}^{\models}_{\mathscr{I}}$ by the entries in Table 6, where we replace $\mathcal{GL}^{\models}_{\mathscr{I}}$ by $\mathcal{GK}^{\models}_{\mathscr{I}}$.

5 Informal semantics of GDT theories by D-V'12

As discussed earlier Lifschitz (2002) coined a term *generate-define-test* for the commonly used methodology when applying ASP towards solving difficult combinatorial search problems. Under this methodology a program typically consists of three parts: the *generate*, *define*, and *test* groups of rules.

The role of *generate* is to generate the search space. In modern dialects of ASP *choice rules* of the form

$$\{A\} \leftarrow B_1, \ldots, B_n, \texttt{not}\ C_1, \ldots, \texttt{not}\ C_m, \tag{5}$$

are typically used within this part of the program. Symbols A, B_i, and C_j in (5) are propositional atoms. The *define* part consists of basic rules (1). This part defines concepts required to state necessary conditions in the *generate* and *test* parts of the program. The *test* part is usually modeled by constraints of the form (4), where B_i, and C_j are propositional atoms.

Denecker et al. (2012) defined the logic ASP-FO, where they took the *generate*, *define*, and *test* parts to be the first class citizens of the formalism. In particular, the ASP-FO language consists of three kinds of expressions G-modules, D-modules, and T-modules. The authors then present formal and informal semantics of the formalism that can be used in practicing ASP.

Here we simplify the language ASP-FO by focusing on its propositional counterpart. We call this language GDT. Focusing on propositional case of ASP-FO helps us in highlighting the key contribution by Denecker et al. (2012) — the development of *objective* informal semantics for logic programs used within ASP or *generate-define-test* approach.

A *G-module* is a set of choice rules with the same atom in the head; this atom is called *open*. A *D-module* is a basic logic program whose atoms appearing in the heads of the rules are called *defined* or *output*. A *T-module* is a constraint. A *GDT theory* is a set of G-modules, D-modules, and T-modules so that no G-modules or D-modules coincide on open or defined atoms. To define the semantics for GDT theory we introduce several auxiliary concepts including that of an *input answer set* Lierler and Truszczyński (2011) and *G-completion*. For a basic program P, we call a set X of atoms an *input answer set* of P if X is an answer set of a program $P \cup (X \setminus Heads(P))$, where $Heads(P)$ denote the set of atoms that occur in the heads of the rules in P.

Rules occurring in modules of GDT theory are such that their bodies have the form

$$B_1, \ldots, B_n, \texttt{not}\ C_1, \ldots, \texttt{not}\ C_m. \tag{6}$$

Given *Body* of the form (6) by $Body^{cl}$ we denoted a classical formula of the form

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

For a G-module G of the form $\{\{A\} \leftarrow Body_1, \ldots, \{A\} \leftarrow Body_n\}$ by *G-completion*, $Gcomp(G)$ we denote the classical formula $A \rightarrow Body_1^{cl} \vee \cdots Body_n^{cl}$.

For a GDT theory P composed of G-modules $G_1, \ldots, G_i$, D-modules $D_1, \ldots, D_j$, T-modules $\leftarrow Body_1, \ldots, \leftarrow Body_k$, we say that set X of atoms is an *answer set* of P, denoted $X \models_{st} P$, if

- X satisfies formulas $Gcomp(G_1)$, …, $Gcomp(G_i)$ (we associate a set X of atoms with an interpretation of classical logic that maps propositional atoms in X to truth value *true* and propositional atoms outside of X to truth value *false*; we then understand the concept of satisfaction in usual terms of classical logic.);
- X is an input answer set of D-modules $D_1 \ldots D_j$; and
- X satisfies formulas $Body_1^{cl} \rightarrow \bot, \ldots, Body_k^{cl} \rightarrow \bot$.

We refer the reader to Denecker et al. (2019) to the discussion of Splitting Theorem results that often allows us to identify ASP logic programs with GDT theories.

We now provide the informal semantics for GDT theory P by Denecker et al. (2012; 2019). We denote it by $\mathcal{DV}_{\mathscr{J}}$ and detail its three components $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}$, $\mathcal{DV}^{\mathbb{S}}_{\mathscr{J}}$ and $\mathcal{DV}^{\models}_{\mathscr{J}}$. To begin with $\mathcal{DV}^{\mathbb{S}}_{\mathscr{J}}$ coincide with $\mathcal{G}^{\mathbb{S}}_{\mathscr{J}}$. We summarize $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}$ by (i) the entries in rows 1-3 of Table 2, where we replace $\mathcal{G}^{\mathbb{L}}_{\mathscr{J}}$ by $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}$, and (ii) the entries in Table 8. Table 9 presents $\mathcal{DV}^{\models}_{\mathscr{J}}$. Note how an entry in the right column of Table 9 gives us clues on how to simplify the parallel entry in the right column of Table 3. We can rewrite it as follows: *For basic program P, property* $\mathcal{G}^{\mathbb{L}}_{\mathscr{J}}(P)$ *holds in the state* $\mathcal{G}^{\mathbb{S}}_{\mathscr{J}}(X)$ *of affairs*.

Provided account of informal semantics of GDT theories echos the interpretation of answer set of a basic program as a possible "interpretation" and can be seen as an informal semantics for the syntactic constructs that are fundamental in ASP practice nowadays.

6 Conclusions and Acknowledgments

In this note we reviewed four papers and their accounts on informal semantics of logic programs under answer set semantics. We put these accounts into a uniform perspective by focusing on three components of each of the considered informal semantics, namely, (i) the interpretation of

Table 8. *The Denecker et al. (2012) informal semantics for some expressions in GDT theories.*

Φ	$\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(\Phi)$
T-theory/constraint $\leftarrow Body$	it is impossible that $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(Body)$
G-module G of the form $\{\{A\} \leftarrow Body_1, \ldots, \{A\} \leftarrow Body_n\}$	if $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(A)$ then $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(Body_1)$ or … $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(Body_n)$ *(in the sense of material implication)*
rule $Head \leftarrow Body$ in a D-module	if $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(Body)$ then $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(Head)$ *(in the sense of definitional implication)*
D-module $\{r_1, \ldots, r_n\}$ with defined atom A	All that is known about A is: $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(r_1)$ and $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(r_2)$ and … $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(r_n)$
GDT theory $P = \{M_1, \ldots, M_n\}$	$\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(M_1)$ and … $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(M_n)$

Table 9. *The Denecker et al. (2012)informal semantics for the satisfaction relation.*

$\models_{st}$	$\mathcal{DV}^{\models_{st}}_{\mathscr{J}}$
$X \models_{st}$ GDT theory P	Property $\mathcal{DV}^{\mathbb{L}}_{\mathscr{J}}(P)$ holds in the state $\mathcal{DV}^{\mathbb{S}}_{\mathscr{J}}(X)$ of affairs.

answer sets; (ii) the interpretation of syntactic expressions; and (iii) the interpretation of semantic relation satisfaction. We also discussed the relations of the presented informal semantics to two programming paradigms that emerged in the field of logic programming after the inception of the concept of a stable model: ASP and ASP-Prolog.

We would like to thank Michael Gelfond, Marc Denecker, Jorge Fandinno, Vladimir Lifschitz, Miroslaw Truszczynski, Joost Vennekens for fruitful discussions on the topic of this note. Marc Denecker brought my attention to the subject of informal semantics and his enthusiasm on the questions pertaining this subject was contagious.

The author was partially supported by NSF 1707371.

References

DENECKER, M., LIERLER, Y., TRUSZCZYŃSKI, M., AND VENNEKENS, J. A Tarskian informal semantics for answer set programming. In *Technical Communications of the 28th International Conference on Logic Programming (ICLP)* 2012, pp. 277–289.

DENECKER, M., LIERLER, Y., TRUSZCZYNSKI, M., AND VENNEKENS, J. 2019. The informal semantics of answer set programming: A tarskian perspective. *CoRR*, *abs/1901.09125*.

GELFOND, M. AND KAHL, Y. 2014. *Knowledge representation, reasoning, and the design of intelligent agents*. Cambridge University Press.

GELFOND, M. AND LIFSCHITZ, V. The stable model semantics for logic programming. In KOWALSKI, R.

AND BOWEN, K., editors, *Proceedings of International Logic Programming Conference and Symposium* 1988, pp. 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.

KNUTH, D. E. AND SHUSTEK, L. 2021. Let's not dumb down the history of computer science. *Commun. ACM*, *64*, 2, 33–35.

LIERLER, Y. AND TRUSZCZYŃSKI, M. 2011. Transition systems for model generators — a unifying approach. *Theory and Practice of Logic Programming, 27th Int'l. Conference on logic Programming (ICLP) Special Issue*, *11*, 4-5, 629–646.

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

MAREK, V. AND TRUSZCZYNSKI, M. Stable models and an alternative logic programming paradigm. In APT, K., MAREK, V., TRUSZCZYNSKI, M., AND WARREN, D., editors, *The Logic Programming Paradigm: a 25-Year Perspective* 1999, pp. 375–398. Springer, Berlin.

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.