A Classification Theory of Semantics of Normal Logic Programs: Strong and Weak Properties

Ekaterina Timoshenko, Irina Zelenskaja, Mikhail Roshchin

January 23, 2004

Abstract

This paper gives the presentation of the research in the field of classifying the semantics of logic programs with negation-as-failure. We present some most important semantics from the point of approach of J. Dix that consists of the idea that all semantics induce in a natural name the non-monotonic inference relation \( \triangleright \). By Dix a certain semantics can be uniquely characterized by two types of its properties: strong and weak. The most important strong properties are Cumulativity and Rationality. From this point of view we describe some LP-Semantics (COMP, COMP\(_3\)) and some NMR-semantics (stratified semantics, WFS and its extensions, STABLE). Dix proposed nine weak principles which inspired the class so-called Well-behaved semantics.
Contents

1 Introduction 4
  1.1 Historical remarks .................................. 4
  1.2 Dix’s approach .................................. 4

2 Logic Programming Background 5

3 Semantics and its sceptical version 7
  3.1 Properties of consequence relations ................. 7
  3.2 SEM and SEM_scept ................................. 12

4 LP-semantics versus NMR-semantics 15
  4.1 Program Completion ................................. 15
  4.2 NMR-Intuition compared with LP-Intuition .......... 19
  4.3 Supported Herbrand model for stratified program .... 20

5 Introduction 24

6 Well-Founded Semantics 24
  6.1 Two definitions of Well-founded Semantics ........... 25

7 Extensions of Well-Founded-Semantics 29
  7.1 Extended well-founded semantics .................... 29
  7.2 Generalized well-founded semantics ................. 30
  7.3 WFS+ ............................................. 31
  7.4 WFS' and REG-SEM ................................ 32
  7.5 O-SEM ........................................... 34
  7.6 Well-Founded-by-Case-Semantics .................... 37
  7.7 Extended-Well-Founded-Semantics ................. 41
  7.8 Conclusion ..................................... 43

8 Introduction 48
  8.1 STABLE semantics ................................. 48
  8.2 Well-behaved semantics ............................ 48

9 STABLE 50

10 Well-behaved Semantics 53
  10.1 Relevance and Reduction .......................... 54
  10.2 PPE and Modularity .............................. 55
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.3 Closure Conditions</td>
<td>56</td>
</tr>
<tr>
<td>10.4 $M_P$-extension and Transformation</td>
<td>57</td>
</tr>
<tr>
<td>10.5 Deducible Rules</td>
<td>58</td>
</tr>
<tr>
<td>10.6 Main definition and results</td>
<td>59</td>
</tr>
<tr>
<td>10.7 Representation conjectures</td>
<td>62</td>
</tr>
<tr>
<td><strong>11 Conclusion</strong></td>
<td>63</td>
</tr>
</tbody>
</table>
Chapter I
written by Ekaterina Timoshenko

1 Introduction

This section briefly presents a history of the problem to give semantics to the
logic program with negation-as-failure and also one of the main approaches
to solve this problem considered by Dix [JD95a].

1.1 Historical remarks

A semantics for logic program have been considered from two different di-
rections that were developing in parallel.

The first attempt to give semantics to negation-as-failure became from
logic programming (so-called Logic-Programming-semantics). It was intro-
duced by Clark [Cla78] and called program completion. But the main
problem here is that the completion of a program may in certain cases be in-
consistent. To circumvent this problem, Fitting [Fit85] considered 3-valued
Herbrand models of the program completion. Later, Kunen [Kun87] identi-
fied a weaker version of Fitting’s semantics which is recursively enumerable.
However, the last two approaches also do not overcome all the objections
that have been raised regarding the completion.

The second attempt was inspired by non-monotonic reasoning (NMR-
semantics). This line of research started from the stratified semantics [Min93]
for the corresponding class of logic programs and is still of the interest for
the logic programming community. The most important NMR-semantics
are the stable semantics [[] and well-founded semantics [[] with its extensions.

Nowadays, there are very close relationships between these two lines of
research.

Actually, there is no optimal (or best suited) semantics and the particular
choice depends on the application area.

1.2 Dix’s approach

One way for classifying semantics for logic programming is by studying which
properties they satisfy.

The Dix’s approach is based on the work kraus, Lehmann, Magidor [KLM90]
and Makinson [Mak94] in general non-monotonic reasoning and consist of
the idea that all semantics induce non-monotonic inference relation ”$\models$".
By Dix a certain semantics can be uniquely characterized by two types of its properties:

- **Strong properties** that are properties of non-monotonic inference relation "|~.
- Weak properties that reflect the specific idea of *negation-as-failure* in logic programming.

The weak properties are more specific to logic program and should be satisfied by every reasonable semantics.

## 2 Logic Programming Background

To play with semantics of normal logic programs we need some background notions from logic programming.

**Definition 2.1.** A (normal) logic program is finite, or countably infinite, set of rules, (implicitly universally quantified) clauses of the form

\[ A \leftarrow B_1,\ldots,B_m,\neg C_1,\ldots,\neg C_n, \]

where \( A, B_i \) and \( C_j \), for \( i = 1,\ldots,m \) and \( j = 1,\ldots,n \) are literals over some given first order language.

\( A \) is called the head of the clause, the right part of the clause called the body could be empty, in which case it evaluates to true and is called a fact (ground clause).

A clause is called definite (or positive) if \( n = 1 \).

All the literals may contain free variables. Therefore every rule in program \( P \) can have infinite number of possible instantiations. To define the ground instantiation we need the following:

**Definition 2.2.** A Herbrand universe of the program \( P \) is the set of variable-free terms of the language \( L_P \) given by the symbols in \( P \). A Herbrand base \( B_p \) is the corresponding set of ground atoms.

A Herbrand model \( A \) is a model whose universe is the Herbrand universe of the program \( P \).

A Herbrand model \( A \) of an arbitrary theory \( T \) (not necessarily a logic program) is called minimal, if there is no other model \( A' \) of \( T \) such that for all atoms \( a \) of the Herbrand base \( B_T \):

\[ A' \models a \implies A \models a. \]
Definition 2.3. The ground instantiation of a logic program is formed by substituting elements of the Herbrand universe by variables of the logic program in all possible ways. An instantiation rule is one in the Herbrand instantiation.

Definition 2.4. A partial interpretation is a set of propositional letters and negated propositional letters. A partial interpretation \( I \) is two-valued if for each proposition letter \( a \) either \( a \) or \( \neg a \) is in \( I \), but not both.

Thus, over its Herbrand universe, each logic program is equivalent to a propositional logic program that does not contain any functional symbols but only propositional variables and their negations (set of ground instantiations \( P_{\text{inst}} \)).

Definition 2.5. For a logic program \( P \) the dependency graph \( G_P \) is a finite direct graph whose vertices are the predicates symbols from \( P \). There is a positive (resp. negative) edge from \( R \) to \( R' \) iff there is a clause in \( P \) with \( R \) in its head and \( R' \) occurring positively (resp. negatively) in its body.

We also say

- \( R \) depends on \( R' \) if there is a path in \( G_P \) from \( R \) to \( R' \) (by definition, \( R \) depends on itself),
- \( R \) depends positively on \( R' \) if there is a path in \( G_P \) from \( R \) to \( R' \) containing only positive edges (by definition, \( R \) depends positively on itself),
- \( R \) depends negatively on \( R' \) if there is a path in \( G_P \) from \( R \) to \( R' \) containing at least one negative edge.

A generalization of \( G_P \) is the infinite instantiated dependency graph \( G^\text{inst}_P \) whose vertices are the elements of \( B_P \). The edges are defined analogously: instead of \( P \) one takes \( P_{\text{inst}} \).

Finally, let \( Th(\Phi) \) denotes the classical deductive closure of the set of formulae \( \Phi \) (the set of all sentences that can be proved from \( \Phi \), \( Fml \) denotes the set of all formulae and \( \text{MIN-MOD} (T) \) denotes the class of all two-valued minimal Herbrand models of an arbitrary theory \( T \)).

Besides two-valued logic a three-valued logic is also used to provide semantics for programs. The idea is that a query can yield three outcomes: it may succeed, it may fail and it may also diverge.
Definition 2.6. A three-valued logic is a logical structure where three truth values \( \text{t} \) "true", \( \text{f} \) "false", \( \text{u} \) "undefined" and the Kleene connectives \( \lor \), \( \land \), \( \neg \), \( \leftarrow \) are possible.

\( \leftarrow \) is the weak implication, where \( \text{u} \leftarrow \text{u} \) is considered to be true.

There are three most important (partial) lattices of truth values and respectively three different ordering of them:

<table>
<thead>
<tr>
<th>Lattice</th>
<th>Ordering on truth</th>
<th>Partial Lattice</th>
<th>Ordering on knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{f} \leq \text{t} )</td>
<td>( \leq )</td>
<td>( \text{f} \leq \text{u} \leq \text{t} )</td>
<td>( \text{t} \geq \text{k} )</td>
</tr>
</tbody>
</table>

Definition 2.7. A three-valued Herbrand interpretation \( I \) is a pair \( \langle T; F \rangle \), consisting of the sets of atoms \( T \) (the truth atoms) and \( F \) (the false atoms).

Let \( \text{True}(I) \) denotes the atoms that are true in \( I \) and \( \text{False}(I) \) respectively the atoms that are false in \( I \). Then we can represent \( I \) by just enumerating its ground literals:

\[
I = \text{True}(I) \cup \{ \neg x : x \in \text{False}(I) \}.
\]

An interpretation \( I \) on atoms can be uniquely extended to an interpretation \( \hat{I} \) of all sentences and the same with relations \( \leq \) and \( \leq_k \). Thus, to define a mapping from the set of three-valued interpretation into itself, it suffices to consider mappings from \( 3^{BP}_B \) into itself. To indicate our interest in the \( \leq_k \)-ordering we will write \( \mapsto 3^{BP}_B \rightarrow 3^{BP}_k \).

Thus any semantics based on a two-valued theory \( \Sigma \) can be seen as a three-valued interpretation \( I = \langle T; F \rangle \) with \( T = \{ A : A \text{ is a ground atom with } \Sigma \models A \} \) and \( F = \{ A : A \text{ is a ground atom with } \Sigma \models \neg A \} \). But in general \( I \) may no longer be a three-valued model of \( \Sigma \).

3 Semantics and its sceptical version

This section introduces and describes the properties of an entailment relation \( \models \) and then represents them in Dix’s specialized setting. Also there adaptation of \( \models \)-formalism to logic programs, i.e. to sceptical entailment relation \( \models_P \), and formal definition of semantics SEM.

3.1 Properties of consequence relations

Every certain semantics satisfy certain abstract properties which were considered by Kraus, Lehmann and Magidor [KLM90] and [LM92] for an entailment relation \( \models \) between single formulae.
Definition 3.1. If $\alpha$ and $\beta$ are formulas then the pair $\alpha \vdash \beta$ (read "from $\alpha$ sensibly conclude $\beta$") is called a conditional assertion. Certain well-behaved sets of conditional assertions will be deemed as consequence relations.

So, if $\vdash$ is a consequence relation than $\alpha \vdash \beta$ indicates that the pair $\langle \alpha, \beta \rangle$ is in the consequence relation $\vdash$ and $\alpha \nvdash \beta$ indicates that it is not in the relation.

Consequence relations are expected to satisfy the following properties:

**Reflexivity** $\alpha \vdash \alpha$

Reflexivity seems to be satisfied universally by any kind of reasoning based on some notion of consequence.

**Left Logical Equivalence**

\[
\frac{\models \alpha \leftrightarrow \beta, \alpha \vdash \gamma}{\beta \vdash \gamma}
\]

Left Logical Equivalence expresses the requirement that logically equivalent formulas have exactly the same consequences. The consequences of a formula should depend on its meaning not on its form.

**Right Weakening**

\[
\frac{\models \alpha \rightarrow \beta, \gamma \vdash \alpha}{\gamma \vdash \beta}
\]

Right Weakening obviously implies that one may replace logically equivalent formulas by one another on the right of the $\vdash$ symbol.

**Cut**

\[
\frac{\alpha \land \beta \vdash \gamma, \alpha \vdash \beta}{\alpha \vdash \gamma}
\]

It expresses the fact that one may first add an hypothesis to the facts he knows to be true and prove the plausibility of his conclusion from this enlarged set of facts and then deduce (plausibly) this added hypothesis from the facts.

**And**

\[
\frac{\alpha \vdash \beta, \alpha \vdash \gamma}{\alpha \vdash \beta \land \gamma}
\]

**Or**

\[
\frac{\alpha \vdash \gamma, \beta \vdash \gamma}{\alpha \lor \beta \vdash \gamma}
\]

**Cautious Monotonicity**

\[
\frac{\alpha \vdash \beta, \alpha \vdash \gamma}{\alpha \land \beta \vdash \gamma}
\]
Cautious Monotonicity expresses the fact that learning a new fact, the truth of which could have been plausibly conclude should not invalidate previous conclusions. If \( \alpha \) is a reason enough to believe \( \beta \) and also to believe \( \gamma \), then \( \alpha \) and \( \beta \) should be also enough to make us believe \( \gamma \), since \( \alpha \) was enough anyway and, on this basis, \( \beta \) was expected.

From a pragmatic point of view Cautious Monotonicity is very important since we typically learn new facts and we would like to minimize the updating we have to make for our beliefs. Cautious Monotonicity and Cut together tells us, that if the new facts learned we expected to be true, nothing changes in our beliefs. This will help to minimize the updating. From a semantic point of view, we want to argue the case for Cautious Monotonicity on the following example.

**Example 3.2.** Suppose we tell you:

- It will be raining tonight, and
- Normally, Fireball should win the race tomorrow.

Wouldn’t you coincide that we think that Even if it is rains tonight, normally, Fireball should win the race tomorrow?

**Lemma 3.3.** The rules Cut and Cautious Monotonicity may be expressed together by the following principle:

if \( \alpha \models \beta \) then the plausible consequence of \( \alpha \) and \( \alpha \land \beta \) coincide.

**Definition 3.4.** A consequence relation \( \models \) is said to be cumulative iff it contains all instances of the Reflexivity axiom and closed under the inference rules of Left Logical Equivalence, Right Weakening, Cut and Cautious Monotonicity.

\[
\text{Rationality} \quad \frac{\alpha \not\models \beta, \alpha \models \gamma}{\alpha \land \beta \not\models \gamma}
\]

\[
\text{Negation Rationality} \quad \frac{\alpha \land \gamma \not\models \beta, \alpha \land \neg \gamma \not\models \beta}{\alpha \not\models \beta}
\]

Negation Rationality can be equivalently expressed by "if \( \beta \) cannot be induced by the \( \alpha \land \gamma \) and \( \beta \) cannot be induced from \( \alpha \land \neg \gamma \) then \( \beta \) cannot be induced from \( \alpha \)".

\[
\text{Disjunctive Rationality} \quad \frac{\alpha \not\models \gamma, \beta \not\models \gamma}{\alpha \lor \beta \not\models \gamma}
\]
In [KLM90] was proposed a general model-theory for "\( \sim \)". Authors suggested models that may be described as a set of worlds equipped with a preference relation.

**Definition 3.5.** A set of conditional assertions that satisfies the Left Logical Equivalence, Right Weakening, Reflexivity, And, Or and Cautious Monotonicity properties is called a preferential consequence relation.

**Definition 3.6.** A preferential model \( W \) is a triple \( \langle S, \models, \prec \rangle \), where \( S \) is an arbitrary set, the elements of which will be called states, \( \models \) is a satisfaction relation between elements of set \( S \) and propositions of the language \( L \) under consideration, i.e. \( \models \subseteq S \times L \) and \( \prec \) is a preference relation of the model between elements of \( S \), i.e. \( \prec \subseteq S \times S \).

The main idea of preferential models is that the agent has, in his mind, a partial ordering on possible states of the world. State \( s \) is less than state \( t \) if, in the agent’s mind, \( s \) is preferred to or more natural than \( t \). The agent is willing to conclude \( \beta \) from \( \alpha \) if all most natural states that satisfy \( \alpha \) also satisfy \( \beta \).

For example, axiom system \( C \) is consist of Right Weakening, Reflexivity, And, Left Logical Equivalence, Cautious Monotonicity and Cut; \( P \) is consist of the rules of \( C \) and the rule Or and \( R \) is consist of \( P \) and the rule Rationality.

According to Makinson [Mak94] for a given set of formulas \( Fml \) logic may be modelled by a mapping

\[
C : 2^{Fml} \rightarrow 2^{Fml}
\]

and for every subset \( \Phi \) of \( Fml \) \( C(\Phi) \) is understood to be the set of all consequences of the set \( \Phi \) of assumptions.

Let \( \Phi \) and \( \Psi \) be any arbitrary subsets of formulae. Then the first property we consider is very natural and called

**Supraclassicality:** \( Th(\Phi) \subseteq C(\Phi) \).

Supraclassicality says that the conclusions that can be drawn non-monotonically from a set of premises \( \Phi \) includes the classical consequences of \( \Phi \).

A powerful interaction condition that has a natural counterpart within preferential semantic is

**Distributivity:** \( C(\Phi) \cap C(\Psi) \subseteq C(Th(\Phi) \cap Th(\Psi)) \).

A more intuitive and weaker version of Distributivity is if \( \Phi = Th(\Phi) \) and \( \Psi = Th(\Psi) \) then \( C(\Phi) \cap C(\Psi) \subseteq C(\Phi \cap \Psi) \).
Cumulativity: $\Phi \subseteq \Psi \subseteq C(\Phi)$ implies $C(\Phi) = C(\Psi)$.

The relation between "$\sim$" and $C$ is given by

$$\Psi \subseteq C(\Phi) \iff \Phi \sim \Psi.$$  

For the restriction of $C$ to finite sets $C_{fin}$ the following lemma holds.

Lemma 3.7. ($C$ versus $\sim$)

1. If And holds: Cumulativity (for $C_{fin}$) $\simeq$ Cautious Monotony and Cut.

2. If Cut holds: Supraclassicality (for $C_{fin}$) $\simeq$ Reflexivity and Right Weakening.

3. Distributivity (for $C_{fin}$) $\simeq$ Or and Left Logical Equivalence.

Proof. Lets prove a) (both b) and c) can be proved analogously).

Cumulativity splits into two implications:

$$\Phi \subseteq \Psi \subseteq C(\Phi) \implies C(\Phi) \subseteq C(\Psi) \quad (1)$$

$$\Phi \subseteq \Psi \subseteq C(\Phi) \implies C(\Psi) \subseteq C(\Phi) \quad (2)$$

Using connection between $C$ and $\sim$, these two equations translate to

$$If \Phi \subseteq \Psi, \Phi \sim \Psi then: \forall \text{formulae } \gamma (\Phi \sim \gamma \implies \Psi \sim \gamma), \quad (3)$$

$$If \Phi \subseteq \Psi, \Phi \sim \Psi then: \forall \text{formulae } \gamma (\Psi \sim \gamma \implies \Phi \sim \gamma). \quad (4)$$

Let $C_{fin}$ satisfy Cumulativity. To prove Cautious Monotony and Cut for $\sim$ we simply set $\Phi := \{\alpha\}$ and $\Psi := \{\alpha, \beta\}$.

To prove the implications (1) and (2) for $C_{fin}$ using that $\sim$ satisfy Cut and Cautious Monotonicity we can represent $\Phi$ and $\Psi$ as conjunctions of their elements (this is justified because AND holds) and we are done.

To illustrate Cumulativity and Rationality lets consider the Closed World Assumption CWA and Circumscription CIRC.

CWA intuitively means that any information that is not mentioned in a database is taken to be false. More precisely, if a positive instance of a predicate is not contained in the database, its negation is assumed to hold.
Definition 3.8. A Closed World Assumption of a given theory \( T \) is a closure operation

\[
CWA(T) = Th(T \cup \{ \neg \phi_{pos} : \phi_{pos} \text{ is a positive formula with } T \not\models \phi_{pos} \})
\]

Definition 3.9. Circumscription is semantically defined as a set of of all two-valued minimal Herbrand models of an arbitrary theory \( T \):

\[
CIRC(T) = \text{MIN-MOD}(T).
\]

We write \( \text{circ}(T) \) for the corresponding set of formulae \( \{ \phi : CIRC(T) \models \phi \} \).

Both CWA and CIRC induce in an obvious way \( \sim \)-relations:

\[
T \sim_{CWA} \phi \text{ iff } \phi \in CWA(T) \text{ and } T \sim_{CIRC} \phi \text{ iff } \phi \in \text{circ}(T).
\]

CWA was one of the first non-monotonic closure operation but in special situation it can be too strong, e.g. \( CWA(\{ \phi \vee \psi \}) \) is inconsistent (if neither \( \phi, \psi \) nor their negations are tautologies and \( \phi, \psi \) are independent from each other). This shows that \( CWA(T) \) does not satisfy Cautious Monotony: from \( CWA(\{ \phi \vee \psi \}) \) we can derive \( \phi \) and \( \neg \phi \) (because the theory is inconsistent), but if we add \( \phi \) then \( \neg \phi \) is no longer derivable, i.e. \( CWA(\{ \phi \vee \psi, \phi \}) \neq \text{Fml} \).

In the same time CIRC avoids these CWA’s inconsistencies for universal theories \( T \).

The following lemma shows that CIRC is cumulative as is CWA for consistent theories:

Lemma 3.10. (Properties of CWA and CIRC)

1. CWA satisfies Cut.
2. If \( CWA(\Phi) \) is consistent then \( \Phi \subseteq \Psi \subseteq CWA(\Phi) \) implies \( CWA(\Phi) \subseteq CWA(\Psi) \).
3. CIRC is cumulative but not rational.

3.2 SEM and SEM\text{\_scept}

Dix[JD95a] defined how a semantics for a logic program can be viewed as an inference relation (consequence relation \( \sim \)):

- Proof-theoretically, semantics \( \text{SEM}_P(U) \) of a program \( P \) together with a set \( U \) of atoms can be defined as a set of literals that are derivable from \( P \) and \( U \) for a particular derivation mechanism (such as SLDNF-resolution).
• Model-theoretically, all existing semantics can be defined as subsets of the set of all 3-valued models of $P \mathit{MOD}_{3-val} (P)$.

**Definition 3.11.** A semantics $\text{SEM}$ is a mapping from the class of all programs into the power set of all 3-valued Herbrand structures. $\text{SEM}$ assigns to every program $P$ a set of 3-valued Herbrand models of $P$:

$$\text{SEM}_P \subseteq \mathit{MOD}^{\mathit{Herb}_L}_3 (P).$$

Let $P \cup U$ denotes a logic program, where $U$ is a distinguish set of atoms (no nontrivial program clauses are contained in $U$) and $\text{SEM}_P(U)$ denote $\text{SEM}_P \cup U$. $P$ still may contain clauses with empty bodies.

The following definition already indicates a fundamental difference to the general "$|$"-framework: "$|$" is not defined between arbitrary program clauses.

**Definition 3.12.** (Sceptical entailment relation $|$)

Let $P$ be a program and $U$ a set of atoms. Any semantics $\text{SEM}$ induces a sceptical entailment relation $\text{SEM}^\text{scept}$ as follows:

$$\text{SEM}^\text{scept}(U) := \bigcap_{M \in \text{SEM}_P(U)} \{ L : L \text{ is a positive or negative literal with } M \models L \}.$$ 

Comparing with the "$|$" framework of Kraus, Lehman and Magidor, Dix equivalently defined a $|$-relation between sets of atoms $U$ (positive literals) on the left hand side, and sets of arbitrary literals $X$ on the right hand side:

$$u_1 \land \ldots \land u_n \models x_1 \land \ldots \land x_m \iff \{ x_1, \ldots, x_m \} \subseteq \text{SEM}^\text{scept}_P (\{ u_1, \ldots, u_n \}).$$

If we are interesting in deriving ground literals from a logic program $P$, we use the notation

$$\text{GCWA}(P) = \{ L : L \text{ is a ground literal with } \text{MIN-MOD}(T) \models L \}$$

for the corresponding sceptical semantics. The induced sceptical relation $\models_{\text{GCWA}}$ is defined for a fixed program $P$ and is a relation between sets of atoms and sets of literals:

$$\{ u_1 \land \ldots \land u_n \} \models_{\text{GCWA}} (x_1 \land \ldots \land x_m) \iff \{ x_1, \ldots, x_m \} \subseteq \text{GCWA}(P \cup \{ u_1, \ldots, u_n \}).$$
The greatest difference of \( \sim_P \)-setting to the general \( \sim \)-framework is that Dix do not have "\( \sim \)" (or \( C \)) as a relation on the whole set of formulae. A second difference is that Dix interpret the rules as *infinistic*: definition of entailment is a relation between *finite* sets of atoms and literals

\[
\sim_P \subseteq 2^{\text{Atoms}} \times 2^{\text{Literals}}.
\]

**Remark 3.13.** *(Extending \( \sim_P \) to a relation between sets of literals)*

\( \sim_P \)-entailment can be easily extended in a uniform way to a relation between arbitrary consistent sets of literals by setting

\[
\text{SEM}_{P_\text{scept}}(X) := \text{SEM}_{P_\text{scept}}(U),
\]

where \( X = U \cup \neg V \) (\( U, V \) sets of atoms) and \( P^* \) is the program obtained from \( P \) by just deleting all clauses "\( a \leftarrow \text{body} \)" with \( \neg a \in X \). Underlying language of \( P^* \) is \( \mathcal{L}_P \): atoms in \( P \) should not get lost by cancelling some clauses.

Thus, by generalizing \( \sim_P \subseteq 2^{\text{Literals}} \times 2^{\text{Literals}} \) Dix got stronger rules. The rules of Right Weakening, Reflexivity, And, Left Logic Equivalence are completely trivial and always satisfied. Therefore the only interesting rules Cumulativity (Cautious Monotony and Cut) and a strengthened form of Cautious Monotony - the Rationality.

**Cumulativity:** If \( U_1 \subseteq U_2 \subseteq \text{SEM}_{P_\text{scept}}(U) \), then \( \text{SEM}_{P_\text{scept}}(U_1) = \text{SEM}_{P_\text{scept}}(U_2) \).

**Rationality:** If \( U_1 \subseteq U_2 \subseteq \{ A : \text{SEM}_{P_\text{scept}}(U_1) \models \neg A \} = \emptyset \), then \( \text{SEM}_{P_\text{scept}}(U_1) \subseteq \text{SEM}_{P_\text{scept}}(U_2) \).

Rationality is in any sceptical semantics a stronger form of Cautious Monotonicity because "\( \alpha \sim \beta \)" implies "\( \alpha \sim \beta \)".

This can be illustrated that e.g. even CIRC satisfies Cautious Monotony at least when all considered models are finite Rationality is not fulfilled:

**Example 3.14.** *(CIRC is nor rational)*

\[ P_{CIRC} : p \leftarrow \neg b \]
\[ b \leftarrow c \]
\[ c \leftarrow p, \neg a \]

\[ \text{MIN-MOD}(P_{CIRC}) = \{ \{ p, a \}, \{ b \} \} \ i.e. \]
"\( \text{GCWA}(P_{CIRC}) \models \neg p \)", "\( \text{GCWA}(P_{CIRC}) \models \sim \neg c \)" and
\[ \text{MIN-MOD}(P_{CIRC} \cup \{ p \}) = \{ \{ p, a \}, \{ p, c, b \} \}
\]
therefore "\( \text{GCWA}(P_{CIRC} \cup \{ p \}) \models \sim \neg c \)".

**Remark 3.15.** *(Finite versus Infinite)*

It turns out that for abstract properties Cautious Monotony, Cut or Rationality there is no difference between formulating it for infinite, finite or even
one element sets $U_2$. This is obvious by the structure of the rule: if Cautious Monotony (resp. Cut) holds for one element set $U_2$ then it is also holds for finite sets $U_2$. The step from finite to infinite is not so obvious but all semantics justify it. For Rationality even the step from one element to finite sets is not so obvious. But again there are no counterexamples.

In the following we will observe all semantics for the satisfiability to the Cumulativity and Rationality.

**Definition 3.16.** A semantics is called cumulative if it satisfies Cautious Monotony and Cut.

**Definition 3.17.** A semantics is called rational if for every atom $a$ and program $P$ the following holds:

$$\neg a \notin SEM(P) \implies SEM(P) \subseteq (P \cup \{a \leftarrow\}).$$

4 LP-semantics versus NMR-semantics

In this section the two different points of view of the logic programs, the Logic Programming and Non-Monotonic Reasoning, are discussed. LP-semantics represents by program completion, NMR-semantics by stratified semantics.

4.1 Program Completion

The LP-viewpoint of logic programs is closely related to the procedural SLDNF-resolution (SLD-resolution plus negation-as-failure). According to the SLDNF-resolution, when the selected literal is positive then the usual SLD-like procedure is used to obtain a new resolvent, and when the selected literal, say $\neg A$, is negative then the following rule is used:

$$\neg A \text{ succeeds iff } A \text{ finitely fails},$$

$$\neg A \text{ finitely fails iff } A \text{ succeeds.}$$

That is, if $\neg A$ succeeds then it is deleted from the query, and, if it is finitely fails, the query fails.

But we can not establish with SLDNF-resolution soundness and completeness results for the classes of logic programs. Clark [?] proposed to solve this problem by strengthening a program $P$ to its completion $comp(P)$ and compare the SLDNF-resolution with $comp(P)$. Intuitively, in the completion the implication are replaced by equivalences. The idea is to collect
all rules having the same head predicate into a single rule whose body is a disjunction of conjunctions then replace the symbol \( \leftarrow \) by \( \leftrightarrow \). This states in effect that the predicate is completely defined by the given rules.

**Definition 4.1.** For a propositional program \( P \) and for each propositional letter \( p \) occurring in \( P \) the completion of \( P \) contains the formula

\[
p \leftrightarrow \bigvee\{q_1 \land \ldots \land q_k : \leftarrow q_1 \land \ldots \land q_k \text{ is a rule of } P\}
\]

The completion contains no other formulas.

**Definition 4.2.** The two-valued program completion semantics \( \text{COMP} \) of a program \( P \) is the set of all literals that are true in all two-valued models of the program completion.

Clark’s completion is not cumulative: \( \text{Cut} \) holds but \( \text{Cautious Monotony} \) fails that can be demonstrated by following example.

**Example 4.3. (COMP is not cautious monotonic)**
\[
P_{\text{comp}} = \{a \leftarrow \neg p, a \leftarrow b, p \leftarrow a, p \leftarrow b, b \leftarrow b\}.
\]
\[
\text{comp}(P_{\text{comp}}) = \{a \leftrightarrow \neg p \lor b, p \leftrightarrow a \lor b, b \leftrightarrow b\}.
\]
\[
\text{comp}(P_{\text{comp}}) = \text{Th}(\{a \leftarrow (\neg p \lor b), p \leftarrow (a \lor b), b \leftarrow b\}) = \text{Th}(\{a, b, p\}).
\]

But \( \text{comp}(P_{\text{comp}} \cup \{p\}) \) is the conjunction of \( a \leftrightarrow (\neg p \lor b) \) and \( p \leftrightarrow (a \lor b \lor \text{true}) \), so it is exactly \( \text{Th}(\{p, a \leftrightarrow b\}) \).

**Example 4.4.** Given logic program \( P = \{p \leftarrow \neg p, a, b \leftarrow \neg c\} \).
The completion of \( P \) is \( \{p \leftarrow \neg p, a \leftarrow \text{true}, b \leftarrow \neg c, c \leftarrow \text{false}\} \) which is inconsistent. Then \( \text{COMP} = \{p, \neg p, a, \neg a, b, \neg b, c, \neg c\} \).

To solve problem of inconsistency Fitting [Fit85] and Kunen [Kun87] suggested a 3-valued interpretation, there the third truth value is ”undefined”, corresponding to neither a propositional letter nor its negation being in the partial interpretation.

**Definition 4.5.** The semantics \( \text{COMP}_3 \) of a program \( P \) is the set of all literals that are true in all 3-valued partial models of the completion of \( P \).

For the Example 4.4 \( \text{COMP}_3 = \{a, b, \neg c\} \).
Fitting introduced a 3-valued analogue of immediate consequence operator the following \( \leq_k \)-monotone operator which acts on 3-valued Herbrand interpretations of a given program:
\( \Phi_P : 3^B_k \mapsto 3^B_k ; I \mapsto \Phi_P(I). \)

\[
\Phi_P(I)(A) = \begin{cases} 
t, & \text{if there is a clause } A \leftarrow L_1, \ldots, L_n \text{ in } P_{inst} \text{ with:} \\
 & \forall i \leq n \text{ we have:} \\
 & *I(L_i) = t. 
\end{cases} 
\begin{cases} 
f, & \text{if for all clauses } A \leftarrow L_1, \ldots, L_n \in P_{inst} \text{ we have:} \\
 & \exists i \leq n \text{ with:} \\
 & *I(L_i) = f. 
\end{cases} 
\begin{cases} 
u, & \text{otherwise.} 
\end{cases}
\]

The following lemma summarizes the relevant properties of the \( \Phi_P \) operator.

**Lemma 4.6.**

- If \( I \) consistent, then \( \Phi_P(I) \) is consistent,
- \( \Phi_P \) is monotonic,
- In general, \( \Phi_P \) is not continuous.

In analogy with the the 2-valued semantics, the \( \models \) relation states for the formula that is true in all 3-valued models of a theory. Then the following Fixpoint Lemma holds:

**Lemma 4.7. (Fixpoint)**

For every 3-valued interpretation \( I, I \models_3 \text{comp}(P) \) iff \( \Phi_P(I) = I. \)

Consequently, by Lemma 4.6 we get:

**Corollary 4.8.** The \( \leq_k \)-least fixpoint of \( \Phi_P \) is a consistent 3-valued model of \( \text{comp}(P) \).

Finally

**Theorem 4.9.** For any logic program \( P \) \( \text{COMP}_3(P) \) is itself a (consistent) 3-valued model least Herbrand model.

**Proof.** (Sketch)

It is straightforward to show that if partial interpretation \( I \) is consistent, so is \( \Phi_P(I) \). It can be shown by induction on the stages of the construction of \( \Phi_P \) that the least fixpoint is consistent. Finally, to show that a 3-valued interpretation is a fixpoint of \( \Phi_P \) iff it is a 3-valued model of \( P \) using Corollary 4.8.

\[ \Box \]
Thus, using the following example, one can show how the 3-valued logic approach offers a solution to the problem of possible inconsistency of completion w.r.t. 2-valued logic.

**Example 4.10.** Given logic program $P = \{ p \leftarrow \neg p \}$.

In classical logic $P$ is equivalent to $\{ p \}$.

$\text{comp}(P) = \{ p \leftarrow p \}$ which is inconsistent. So $\text{COMP}(P) = \{ p, \neg p \}$.

But $\text{COMP}_3(P) = (\emptyset, \emptyset)$ in which every ground atom is undefined, and consequently in which $p \leftarrow p$ is true.

A natural question is for which programs the 3-valued and 2-valued semantics of $\text{comp}(P)$ coincide. An answer was provided by Kunen [Kun87].

Kunen defined a variant of $\text{COMP}_3(P)$ that is recursive enumerable: it corresponds to truth in all models, not only Herbrand models, of $\text{comp}(P)$ and differs from Fitting’s in two important ways:

1. The iteration is always stops at $\omega$;

2. The Herbrand universe is defined with respect to a language with an infinite set of function symbols, which properly includes those that occur in the program.

To prove that 3-valued versions of COMP are not only cumulative but also rational we will need the following lemma.

**Lemma 4.11.**

Let $U_1 \subseteq U_2$ be (possibly infinite) sets of ground atoms.

1. For $U_2 \cap \text{False}(\text{lfp}(\Phi_{P \cup U_1})) = \emptyset$, we have:

   $I \leq_k I'$ implies $\Phi_{P \cup U_1}(I) \leq_k \Phi_{P \cup U_2}(I').$

2. For $U_2 \subseteq \text{True}(\text{lfp}(\Phi_{P \cup U_1}))$ we have:

   $I \leq_k \text{lfp}(\Phi_{P \cup U_1})$ implies $\Phi_{P \cup U_2}(I) \leq_k \text{lfp}(\Phi_{P \cup U_1})$.

**Theorem 4.12. (Cumulativity and Rationality of $\text{COMP}_3$)**

1. The Fitting-semantics $\text{SEM}_{\text{Fitting}}$, given by the $\leq_k$-least 3-valued Herbrand model (or, equivalently, by all Herbrand models) of $\text{comp}_3$, is cumulative and rational.
The Kunen-semantics $SEM_{Kunen}$, given by all 3-valued models, is cumulative and rational.

**Proof.**

1) This is just the application of the Lemma 4.11. To prove *Rationality* we use 1) of the lemma. Let the assumption of *Rationality* be fulfilled, i.e.

$$U_1 \subseteq U_2, U_2 \cup \{A : lfp(\Phi_{P \cup U_1}) \models \neg A\} = \emptyset.$$ 

This means that the assumptions are satisfied. We can therefore apply and iterate the lemma using $\emptyset \leq k \emptyset$ and get

$$lfp(\Phi_{P \cup U_1}) \leq k lfp(\Phi_{P \cup U_2}).$$

To prove the *Cut* we use 2) of the lemma 4.11. Let $U_1 \subseteq U_2 \subseteq lfp(\Phi_{P \cup U_1})$. The assumption of 2) are therefore satisfied: we can apply and iterate the lemma using $\emptyset \leq k lfp(\Phi_{P \cup U_1})$ and get

$$lfp(\Phi_{P \cup U_2}) \leq k lfp(\Phi_{P \cup U_1}).$$

2) Lemma 4.11 remains true if one replays $lfp(\Phi_{P \cup U_1})$ by $\Phi_{P \cup U_1}^{\omega}$. Considering truth with respect to ground literals, we have: $\Phi_P^{\omega} = \bigcup_{n \in \mathbb{N}} \Phi_P \uparrow n$. The underlying language does not matter.

### 4.2 NMR-Intuition compared with LP-Intuition

The weakness of COMP from the NMR-view (due to the different treatment of loops) is best shown by the following:

**Example 4.13. (COMP vs. NMR)**

\begin{align*}
P_{NMR} : & \quad p \leftarrow p \\
& \quad q \leftarrow \neg p \\
comp(P_{NMR}) : & \quad p \leftrightarrow p \\
& \quad q \leftrightarrow \neg p
\end{align*}

\begin{align*}
P'_{NMR} : & \quad p \leftarrow p \\
& \quad q \leftarrow \neg p \\
comp(P'_{NMR}) : & \quad p \leftrightarrow p \\
& \quad q \leftrightarrow \neg p
\end{align*}

\begin{align*}
? - q : & \quad No(COMP). \\
& \quad Yes(NMR).
\end{align*}

\begin{align*}
? - p : & \quad Yes(COMP). \\
& \quad No(NMR).
\end{align*}

For both programs, the answers of the completion semantics do not match our NMR-intuition!
In the case of $P_{NMR}$ we expect $q$ to be derivable, since we expect $\neg p$ to be derivable (the only possibility to derive $p$ is the rule $p \leftarrow p$ which, obviously, will never succeed). But $q \notin \text{Th}(\{q \leftrightarrow \neg p\}) = \text{comp}(P_{NMR})!$

In the case of $P'_{NMR}$ we expect $p$ not to be derivable (for the same reason), the only possibility to derive $p$ is the rule $p \leftarrow p$. But $p \in \text{Fml} = \text{Th}(\{r \leftrightarrow \neg r\}) = \text{comp}(P'_{NMR})!$

Note that the answers of the completion-semantics agree with the mechanism of SLDNF: $p \leftarrow p$ represents a loop. The completion of $P$ is inconsistent: this led Fitting to consider the 3-valued version of $\text{comp}(P)$ introduced in last section. This approach avoids the inconsistency (the query $\neg p$ is not answered ”yes”) but it still does not answer ”no” as we would like to have.

The last example motivates the need for a semantics that improves COMP and shows the difficulties in obtaining one: loops should be detected. In general, this problem is undecidable because the halting problem reduces to it. In addition, these two extensions of SLDNF do not agree on which loops should be detected. The following idea was investigated by Bol [Bol91]:

$$\text{NMR-semantics} = \text{SLDNF} + \text{Loop-check}.$$  

### 4.3 Supported Herbrand model for stratified program

Next approach for defining the semantics of logic programs that produced a single model is the so-called Stratified semantics. Stratified semantics could be applied for an interesting class of programs: the idea is to rule out all programs having cycle (not only a direct negative links) with a negative edge in their dependency graph.

The need in this approach arises from the problem in applying the immediate consequence operator $T_P$ to an arbitrary program: at the stage $i$ an atom $A$ may still be false (so that it gives rise to derive new positives atoms via a clause of the form $A_{\text{new}} \leftarrow \neg A$) but on a later stage $j > i$ the atom $A$ may become true, so that all previously drawn inferences have to be rejected. This is exactly caused by the non-monotonicity of $T_P$!

That means that for the immediate consequence operator $T_P$ defined as follows
\[ T_P : 2^{BP} \longrightarrow 2^{BP}; I \mapsto T_P(I) \]

\[ T_P(I)(A) = \begin{cases} 
  t, & \text{if there is a clause } A \leftarrow L_1, \ldots, L_n \text{ in } P_{\text{inst}} \text{ with:} \\
  & \forall i \leq n \text{ we have:} \\
  & *I(L_i) = t. \\
  f, & \text{otherwise.} 
\end{cases} \]

the properties "\( T_P \) is monotone and continuous" and "there exist a least Herbrand model \( M_P = T_P \uparrow^\omega = lfp(T_P) \)" get lost.

This problem is avoided for the class of stratified programs.

**Definition 4.14.** A program \( P \) is called stratified if it is decomposable as \( P = P_1 \cup \ldots \cup P_n \) such that for \( i = 1, \ldots, n \) the following holds:

1. Each clause of \( P \) occurs in exactly one \( P_i \);
2. All defining clauses for a relation symbol \( R \) occur in the same \( P_i \);
3. If a relation symbol \( R \) occur positively in a clause in \( P_i \) then all clauses containing \( R \) in their heads are contained in \( \bigcup_{j \leq i} P_j \);
4. If a relation symbol \( R \) occur negatively in a clause in \( P_i \) then all clauses containing \( R \) in their heads are contained in \( \bigcup_{j < i} P_j \).

We say that "\( P \) is stratified via \( P_1 \cup \ldots \cup P_n \)" and call every \( P_i \) "stratum of \( P \)."

Now it is possible to define the ordinal power of \( T_P \) (it has to be distinguish from \( T_P \uparrow^\omega \)):

\[ T_P^\omega : 2^{BP} \longrightarrow 2^{BP}; I \mapsto T_P^\omega(I) \]

\[ T_P^0(I) = I, \]
\[ T_P^{n+1}(I) = T_P(T_P^n(I)) \cup T_P^n, \]
\[ \ldots \]
\[ T_P^\omega(I) = \bigcup_{n=0}^\infty T_P^n(I). \]

This can be used to select a unique Herbrand model among all minimal Herbrand models of \( \text{comp}(P) \) for any stratified program \( P \):

**Definition 4.15.**

For \( P = P_1 \cup \ldots \cup P_n \) a stratification of \( P \), we define the supported Herbrand model \( M_P^{supp} \) as follows:

\[ M_1 = T_{P_1}^\omega(\emptyset) \]
\[ M_2 = T_{P_2}(M_1) \]
\[ \ldots \]
\[ M_n = T_{P_n}(M_{n-1}) \]
\[ M_P^{supp} = M_n \]
To prove the *Cumulativity* of $M_P^{\sup}$ we need the following lemma.

**Lemma 4.16.**

Let $U_1 \subseteq U_2$ be (possibly infinite) sets of ground atoms.

1. If $P \cup U_1 = P_1 \cup \ldots \cup P_n$ is a stratification of $P \cup U_1$, then $P \cup U_2$ is also stratified. A stratification of $P \cup U_2 = P_1' \cup \ldots \cup P_n'$, where $P_i'$ is obtained from $P_i$ by adding those atoms of $U_2$, whose relation symbols already occur in the head of a clause in $P_i$.

   In addition, we have:
   
   $I \subseteq I'$ implies $T_{P_i}^{\omega}(I) \subseteq T_{P_i'}^{\omega}(I)$.

2. If $M_{P \cup U_1}^{\sup} \models U_2$ we have:

   $$M_{i+1}^{'} = T_{P_{i+1}}^{\omega}(M_i) \subseteq T_{P_{i+1}}^{\omega}(M_i) = M_{i+1}.$$ 

**Corollary 4.17.** *Cumulativity and Rationality of $M_P^{\sup}$* The stratified semantics $M_P^{\sup}$ is cumulative. As a complete semantics, it is also rational.

**Proof.** Using 1) of Lemma 4.16 we get the Cautious Monotony, while 2) proves Cut. Because of the completeness (for any atom $a$ either $M_P^{\sup} \models a$ or $M_P^{\sup} \models \neg a$) we have: "$a \not\models \neg b$" implies "$a \not\models b$" and Rationality is reduced to Cautious Monotony. \qed

The main advantage of the supported semantics is that there is a general agreement that this two-valued Herbrand model is the only possible semantics (from the NMR-viewpoint) for stratified programs.

The stratified semantics fails for programs in which some variables are defined (directly or indirectly) in terms of their own negations, because these variables are never ranked. For such programs we need an extra intermediate neutral truth value for certain of the negatively recursively defined variables. This approach yields the well-founded construction that would be shown in the next section.
References


5 Introduction

Every semantics is characterized by a set of particular features and properties and through this properties it can be uniquely determined. Kraus, Lehmann, Magidor and Makinson introduced the set of certain rules such that rule of Reflexivity, Right Weak etc (see Chapter I). The most important rules from this set are **Cumulativity** and **Rationality** that play a fundamental role in none-monotonic reasoning.

Dix in his paper [JD95a] make a new approach: he claims if any semantics satisfies **Cumulativity**, **Cut** and **Rationality** from the above set of rules then it obeys strong properties i.e has regular behaviour and can be determined by these.

In the next two section we will consider the Well-Founded-Semantics and its extensions from this point of view. Will we also see that although **WFS** is cumulative and rational not all its extensions has the same properties.

6 Well-Founded Semantics

In the previous section was discussed one of the classical semantics, namely supported semantics. The main drawback of this semantics is that, it is defined only for stratified program. For example the program $P$: $v(a, b)$, $v(b, c)$, $e(c)$, $e(x) ← v(x, y)$, $¬e(y)$ is not stratifiable, because of $e(a) ← v(a, a)$, $¬e(a)$ and so the supported semantics can not be defined for $P$. That is why it was needed to look for the extensions of the supported semantics which could give to such program a meaning. Two such classical extensions were constructed: the Well-Founded-Semantics and the Stable Semantics. In this section we will talk about one of these- Well-Founded-Semantics.

There are many definitions of **WFS**, but we consider only two. The first one, which will be discussed, is the Allen Van Gelders [vGRS91] definition. The key idea of his formulation was the concept of an "unfounded set". After that we give another definition of **WFA** which has been introduced by Przymusinski [TP89]. He has used monotone operator $Φ^J_P$ to define **WFS**, and with the help of this operator we show that **WFS** obeys the properties of **Rationality** and **Cumulativity**.
6.1 Two definitions of Well-founded Semantics

Definition 6.1. Let $P$ be a logic program, $B_p$ is Herbrand base, $I$ partial interpretation 2.1. We say that $A \subseteq B_p$ is an unfounded set of $P$ w.r.t. $I$ if each atom $p \in A$ satisfies the following condition:

For each rule $p \leftarrow b_1, \ldots, b_m, notc_1, \ldots, notc_n$ of $P$ whose head is $p$, at least one of the following holds:

- for some $b_i, \neg b_i \in I$ or for some $notc_j, c_j \in I$;
- for some $b_i, b_i \in A$.

Definition 6.2. The greatest unfounded set of $P$ w.r.t. $I$, denoted by $U_p(I)$, is the union of all sets that are unfounded w.r.t. $I$.

Definition 6.3. Define operators $T_p: B_p \cup \neg B_p \rightarrow B_p$, $U_p: B_p \cup \neg B_p \rightarrow B_p$ and $W_p: B_p \cup \neg B_p \rightarrow B_p \cup \neg B_p$ as follows, where $\neg B_p$ denotes the set that contains all the negated atoms of $B_p$:

- $p \in T_p(I)$ if, for some atoms $b_1, \ldots, b_m, c_1, \ldots, c_n$, $p \leftarrow b_1, \ldots, b_m, notc_1, \ldots, notc_n \in P$ and $I \models b_1 \land \ldots \land b_m \land \neg c_1 \land \ldots \land \neg c_n$;
- $U_p(I)$ is the greatest unfounded set of $P$ w.r.t. $I$;
- $W_p(I) = T_p(I) \cup \neg U_p(I)$.

Definition 6.4. Operator $T$ is called monotonic if $T(I) \subseteq T(J)$, whenever $I \subseteq J$.

Lemma 6.5. Operator $T_p$, $U_p$, $W_p$ are monotone.

Proof. Immediate from 6.3

One has to be mentioned here: operator $T_p$ treats both the positive and negative subgoals symmetrically. The presence or absence of $p$ is not important for the truth of the subgoal $notp$. We will require the presence of $\neg p$.

Definition 6.6. Let $\alpha$ range over all countable ordinals. The sets $I_\alpha$ and $I^\infty$, whose elements are literals in the Herbrand base of a program $P$, are defined recursively:

1. For limit $\alpha$, $I_\alpha = \bigcup_{\beta < \alpha} I_\beta$

Note that 0 is a limit ordinal, and $I_0 = \emptyset$
2. For successor ordinal
\[ \alpha = \gamma + 1 \]
\[ I_{\gamma+1} = W_p(I_\gamma) \]

3. Finally, define
\[ I^\infty = \bigcup_\alpha I_\alpha \]

Tarski in his work showed that \( I^\infty \) is least fix point of operator \( W_p \).
Since the Herbrand base is countable, we have for some countable ordinal \( \alpha \)
\( I^\infty = I_\alpha \)

Definition 6.7. **Well-Founded Semantics** of a program \( P \) is the least
fixpoint of \( W_p \), or the limit \( I^\infty \).

Example 6.8. Let \( P \) be a program consisting of the following clauses
\[
\begin{align*}
s & \leftarrow q \\
q & \leftarrow \neg p \\
p & \leftarrow p \\
r & \leftarrow \neg r \\
U_P(\emptyset) & = \{p\}, \\
T_P(\emptyset) & = \emptyset, \\
W_P \uparrow^1 & = \{\neg p\}, \\
U_P(W_P \uparrow^1) & = U_P(\emptyset), \\
T_P(W_P \uparrow^1) & = \{q\}, \\
W_P \uparrow^2 & = \{\neg p, q\}, \\
U_P(W_P \uparrow^2) & = U_P(W_P \uparrow^1), \\
T_P(W_P \uparrow^2) & = \{q, s\}, \\
W_P \uparrow^3 & = \{\neg p, q, s\}, \\
T_P(W_P \uparrow^3) & = T_P(W_P \uparrow^2), \\
U_P(W_P \uparrow^3) & = U_P(W_P \uparrow^2), \\
W_P \uparrow^4 & = W_P \uparrow^3, \\
WFS(P) & = W_P \uparrow^4 = \{\neg p, q, s\}.
\end{align*}
\]

Before we show that \( WFS \) possesses the strong properties, introduce the
Przymusinski’s operator \( \Phi^J_P \) [TP89] and give the another definition Well-
Founded-Semantics through this operator.

Theorem 6.9. The operator \( \Phi^J_P : \mathcal{B}^P \rightarrow \mathcal{B}^P ; I \rightarrow \Phi^J_P(I) \) is \( \leq_t \)-monotone
and has \( \leq_t \) – least fixpoint:
\[ \text{lfp}(\Phi^J_P) = \Phi^J_P \uparrow^\omega, \text{ where } \Phi^J_P \uparrow^0 = \{\emptyset, B_P\}. \]
Definition 6.10. The well-founded model $WFS(P)$ is 3-valued one can be defined as follows:

$$
\Phi_J^P(I)(A) = \begin{cases} 
  t & \text{if there is a clause } A \leftarrow L_1, \ldots, L_n \text{ in } P_{inst} \text{ with,} \\
  \forall i \leq n \text{ we have:} \\
  \quad - J(L_i) = t, \text{ or} \\
  \quad - L_i \text{ is positive, } J(L_i) \neq f \text{ and } I(L_i) = t. \\
  f & \text{if for all clauses } A \leftarrow L_1, \ldots, L_n \text{ in } P_{inst} \text{ with,} \\
  \exists i \leq n \text{ we have:} \\
  \quad - J(L_i) = f, \text{ or} \\
  \quad - L_i \text{ is positive, } J(L_i) \neq t \text{ and } I(L_i) = f. \\
  u & \text{otherwise.} 
\end{cases}
$$

$J$ represents the safe knowledge, i.e. $True(J)$ and $False(J)$ never decrease. $I$ stands for current knowledge.

Note that $\Phi_J^P$ is also $\leq_k$ - monotone. Let us define the new operator $\Omega_P$:

$$
\Omega_P: 3^{3^B}_k \rightarrow 3^{3^B}_k; J \rightarrow \Phi_J^P \uparrow^\omega
$$

Definition 6.11. The least fixpoint of $\Omega_P$ is well-founded model:

$$
WFS(P) = lfp(\Omega_P)
$$

All necessary notions are given and we can go to the next point, namely the strong properties of the WFS. We show that Well-founded-Semantics obeys two strong properties Rationality and Cumulativity. In order to prove rationality we first prove the following lemma.

Lemma 6.12. Let $U_1 \subseteq U_2$ possibly infinite sets of ground atoms.

1. For $U_2 \cap False(WFS(P \cup U_1)) = \emptyset$, we have:
   - $J \leq_k J' \text{ implies } \Phi_{P \cup U_1}^J \uparrow^\omega \leq_k \Phi_{P \cup U_2}^J \uparrow^\omega$

2. For $WFS(P \cup U_1) \models U_2$, we have:
   - $J \leq_k WFS(P \cup U_1) \text{ implies } \Phi_{P \cup U_2}^J \uparrow^\omega \leq_k WFS(P \cup U_1)$

Proof. It is easy to see that the following is true for ground atoms $A$:

- $A \in True(\Phi_{P \cup U}^J \uparrow^\omega) \text{ iff } \exists \text{ a finite } U_0 \subseteq U: A \in True(\Phi_{P \cup U_0}^J \uparrow^\omega)$
- $A \in False(\Phi_{P \cup U}^J \uparrow^\omega) \text{ iff } \forall \text{ finite } U_0 \subseteq U: A \in False(\Phi_{P \cup U_0}^J \uparrow^\omega)$
So we can assume, w. l. o. g. that $U_1, U_2$ are both finite.

1. Let $J \leq_k J'$. It suffices to prove the 1 and 2 below

   (a) For all $n \in \mathbb{N}$: $True(\Phi^J_{P \cup U_1} \uparrow^n) \subseteq (True(\Phi^J_{P \cup U_2} \uparrow^n))$ This follows immediately from the definition of $\Phi^J$ and assumption that $J \leq_k J'$.

   (b) $False(\Phi^J_{P \cup U_1} \uparrow^n) \subseteq False(\Phi^J_{P \cup U_2} \uparrow^n).$

   The assumption $U_2 \cup False(WFS(P \cup U_1)) = \emptyset$ and the finiteness of $U_2$ imply the existence of a $n_0 \in \mathbb{N}$ such that:

   $U_2 \cap False(\Phi^J_{P \cup U_1} \uparrow^{n_0})) = \emptyset.$

   We obtain $False(\Phi^J_{P \cup U_1} \uparrow^{n_0}) \subseteq False(\Phi^J_{P \cup U_2} \uparrow^0).$ Inspecting the definitions of $\Phi^J_P$, $\Phi^J_{P \cup U_1}$ can be applied on the right-hand side and $\Phi^J_{P \cup U_2}$ can be applied on the left-hand side without changing the subset-inclusion:

   for all $i \in \mathbb{N}$: $False(\Phi^J_{P \cup U_1} \uparrow^{n_0+i}) \subseteq False(\Phi^J_{P \cup U_2} \uparrow^i)$.

2. From definition of $\Phi^J_{P \cup U_1}$ immediately follows that $\Phi^J_{P \cup U_1} = \Phi^J_{P \cup U_2}$ for all $J'$ such that $U_2 \subseteq True(J')$. Substituting $J'$ by $WFS(P \cup U_1)$ obtain

   $\Phi^WFS(P \cup U_1) \uparrow^\omega = \Phi^WFS(P \cup U_1) \uparrow^\omega$

   $\Phi^J_{P \cup U_2} \uparrow^\omega \leq_k \Phi^WFS(P \cup U_1) \uparrow^\omega$ $\Phi^WFS(P \cup U_2) \uparrow^\omega = \Phi^WFS(P \cup U_1) \uparrow^\omega = WFS(P \cup U_1)$

Theorem 6.13. $WFS$ is cumulative and rational

Proof. The proof is just the application of previous lemma. To prove Rationality we will use 1. Let the assumptions of Rationality be fulfilled, i.e. $U_1 \leq U_2$, $U_2 \cap \{A \mid WFS(P \cup U_1) \models \neg A\} = \emptyset$

This means that the assumptions of 1. are satisfied. We can apply and iterate the lemma using $\leq_k \emptyset$ and get $WFS(P \cup U_1) \leq_k WFS(P \cup U_2)$ or $lfp(\Phi^J_{P \cup U_1} \uparrow^\omega) \leq_k lfp(\Phi^J_{P \cup U_2} \uparrow^\omega)$

To prove the Cut we use 2. of the above lemma. Let $U_1 \leq U_2$ then $WFS(P \cup U_1 \models U_1)$ The assumptions of 2. are satisfied. We can apply the lemma again using $\leq_k lfp(\Phi^J_{P \cup U_1} \uparrow^\omega)$ and get $lfp(\Phi^J_{P \cup U_2} \uparrow^\omega) \leq_k lfp(\Phi^J_{P \cup U_1} \uparrow^\omega)$

□
Generally speaking the well-founded semantics was considered as an attempt to give a reasonable meaning as much program as possible even if they are not-stratified or when only a partial model exist. But there are some situations when it is not able to find the meaning of a program (we will see some such examples in the next section).

In the next section we will speak about some interesting extensions of WFS which have been constructed to manage with the problems, where WFS failed, and theirs properties.

7 Extensions of Well-Founded-Semantics

7.1 Extended well-founded semantics

The simplest way to get an extension of WFS is first evaluate Well-Founded-Semantics and then add all literals that are true in all minimal models.

**Definition 7.1.** Let $J$ is a three-valued interpretation and $J-\text{MIN/MOD}(P)$ denote the class of all minimal two-valued Herbrand models of $P$ that are consistent with $J$. Furthermore, let

- $T(J) := \text{True}(J-\text{MIN/MOD}(P))$
- $F(J) := \text{False}(J-\text{MIN/MOD}(P))$

There, for a set $S$ of Herbrand models, $\text{True}(S)$ resp. $\text{False}(S)$ stands for the set of all ground atoms $A$, which are true, resp. false in all models in $S$.

We define Extended well-founded semantics as: $\text{EWFS}(P) := \text{WFS}(P) + \{T(\text{WFS}(P)), F(\text{WFS}(P))\}$

Although $\text{EWFS}$ is stronger than $\text{WFS}$ ($\text{WFS} \leq_k \text{EWFS}$) it does not satisfied Cut. It can be shown by following counterexample:

**Example 7.2.** Consider the following program $P$:

- $a \leftarrow \neg a$
- $\beta \leftarrow \neg x, a$
- $y \leftarrow \neg \beta$
- $z \leftarrow \neg y$

$\text{WFS}$ of the program is $\{-x\}, \text{EFWS} = \{-x\} + (T(\neg x), F(\neg x))$. Looking for such atoms in our program $P$ get that $\text{EFWS}(P) = \{a, \beta, \neg x\}$. Concerning the meaning of Cut we have to consider $\{\text{EWFS} \cup a\}$, $\{\text{EWFS} \cup \beta\}$ and $\{\text{EWFS} \cup \neg x\}$. But already by finding $\{\text{EWFS} \cup \beta\}$ we can see that
Cut does not holds any more: \( \{EWF S \cup \beta\} = \{a, \beta, \neg x \neg y, z\} \) Thus by adding a derivable atom ‘\( \beta \)’, we are able to derive more atoms than without: Cut is not satisfied.

We could iterate above construction assure that Cut is satisfied. This give us a new semantics defined by Baral, Lobo and Minker [BLM90] which we discus in the next section.

### 7.2 Generalized well-founded semantics

The idea of getting new semantics which would be satisfied Cut is the following: to add all literals true in all minimal models and extend WFS by using the operator \( \Omega_P \) defined in the section 6.1. The Generalized well-founded semantics is then defined as follows:

**Definition 7.3.** Let \( J \) is a three-valued interpretation and \( J-MIN/MOD(P) \) denote the class of all minimal two-valued Herbrand models of \( P \) that are consistent with \( J \). Furthermore, let

- \( T(J) = \text{True}(J-MIN/MOD(P)) \)  
- \( F(J) = \text{False}(J-MIN/MOD(P)) \)

There, for a set \( S \) of Herbrand models, \( \text{True}(S) \) resp. \( \text{False}(S) \) stands for the set of all ground atoms \( A \), which are true, resp. false in all models in \( S \).

\[
GWFS(P) := \text{lfp}(\Omega^G_P).
\]

where the operator \( \Omega^G_P \) is defined as follows:

\[
\Omega^G_P \cdot \sigma_{SP}^i \rightarrow \sigma_{SP}^i; J \rightarrow \Phi^J_P \uparrow^o +\{T(J), F(J)\}.
\]

Generalized-Well-Founded-Semantics was constructed as one of the extensions of WFS satisfying Cut. In spite of this very useful achievement, this semantics has another serious drawback.

**Example 7.4.** (Strange behavior of GWFS) Consider two programs \( P_{GWFS} \):

- \( p \leftarrow \neg b \)
- \( b \leftarrow c \)
- \( c \leftarrow p, \neg a \)
- \( a \leftarrow \neg b \)

and \( P_{GWFS_C} \):

- \( p \leftarrow \neg b \)
- \( b \leftarrow p, \neg a \)
- \( a \leftarrow \neg b \)
Let \( J(P_{GFS}) = \{ p, a, \neg b \} \) then
\[ \text{MIN/MOD}(P_{GFS}) = \{ p, a \}; \{ \neg b \} \]
\[ GWFS(P_{GFS}) = \{ \neg c, p, a, \neg b \} \]
\[ J(P_{GFS_C}) = \{ p, \neg a \}, GWFS(P_{GFS}) = \{ b, \neg p, \neg a \}. \]

The explanation of such strange behavior of GWFS (equivalent programs are assigned the different semantics) is that too much literals are added. There is also another possibility to add only positive literals true in all minimal models and iterate this until no new literals can be derived. This yields a semantics \( WFS^+ \) that extends \( WFS \) but still very close to it. This semantics will be discussed in the next section.

### 7.3 WFS+

Before we give the definition of \( WFS^+ \) let us consider the following problem. Given a semantics \( Sem \) (see Chapter I) that does not satisfy \textbf{Cut}. The question is how \( SEM \) can be extended to a semantics \( SEM^+ \) satisfying \textbf{Cut}? The answer gives us the following theorem:

**Theorem 7.5.** Let \( SEM \) be a rational semantics and define an increasing sequence of sets of atoms \( M_\alpha \) by \( M_0 := \emptyset \), \( M_{\alpha+1} := \{ a : a \text{ an atom with } a \in SEM(P \cup M_\alpha) \} \) for successor ordinals and \( M_\alpha := \bigcup_{\alpha < \lambda} \). For \( \alpha < \beta \) we have:

\[ SEM(P \cup M_\alpha) \leq_k SEM(P \cup M_\beta) \]

There exists therefore a \( \gamma \) such that \( M_\gamma = M_{\gamma+1} \). Define

\[ SEM^+(P) := SEM(P \cup M_\gamma) \]

By constructing the \( WFS^+ \) the idea was to get an extension of Well-Founded-Semantics that satisfies the property of Supraclassicality:

**Supraclassicality:** if \( P \models a \) then \( a \in WFS^+ \) (for all atoms \( a \))

**Definition 7.6.** Let \( P \) be a normal program, and define

\[ SEM(P) := WFS(P) \cup \{ a : a \text{ an atom with: } P \models a \} \]

Let us formally prove that \( SEM(P) \) defined above is rational. Let \( U \) be a set of atoms and \( U \cap FALSE(SEM(P)) = \emptyset \). In order to prove \textbf{Rationality} we have to show that \( SEM(P) \leq_k SEM(P \cup U) \). Definition of \( SEM(P) \) implies \( U \cap FALSE(WFS(P) \cup a) = \emptyset \) or, \( U \cap FALSE(WFS(P)) = \emptyset \). It is already proved that \( WFS \) is rational therefore

31
\[ \text{WFS}(P) \leq_k \text{WFS}(P \cup U) \]

Using the above observation we immediately get \( \text{SEM}(P) = \text{WFS}(P) + \{a: P \models a\} \leq_k \text{WFS}(P \cup U) + \{a: P \cup U \models a\} = \text{SEM}(P \cup U) \)

We need one more observation to see that all assumptions of the lemma really apply.

\( P \models a \) implies \( \neg a \not\in \text{WFS}(P) \)

The reason is that although \( \text{WFS}(P) \) is in general a three-valued model of \( P \), it can always be extended to a two-valued one: just declare all atoms \( a \) with neither \( a \in \text{WFS}(P) \) nor \( \neg a \in \text{WFS}(P) \) to be true. The resulting two-valued model is a model of \( P \), since in the head of the rules only positive atoms occur. It is easy to see that all assumptions of theorem 4.6 are satisfied and we get a cumulative and rational semantic \( \text{WFS}^+ = \text{SEM}^+ \).

**Theorem 7.7.** \( \text{WFS}^+ \) is the weakest extensions of \( \text{WFS} \) that satisfies Cut and Supraclassicality. In additional \( \text{WFS}^+ \) satisfies:

- Generalized Supraclassicality. If \( P \cup \text{WFS}^+(P) \models a \) then \( a \in \text{WFS}^+(P) \).

### 7.4 WFS’ and REG-SEM

We can obtain a more special variant of \( \text{WFS}^+ \) if we consider not all minimal models extending \( \text{WFS} \), but only models \( A \) which satisfy the certain conditions. For example \( \text{WFS}^+ \) derives \( a \) from the program \( a \leftarrow \neg a \). We formulate a condition that excludes models assigning \( a \) to be \text{true}(such literals are not derivable, they are assigned \text{u}) in programs containing \( a \leftarrow \neg a \):

**C:** If all rule bodies for a false in \( A \) then \( A \models \neg a \)

Before we give a definition of another extension, some notions have to be introduced

**Definition 7.8.** A three-valued Herbrand Model \( M \) is justified for a program \( P \), if every \( a \in \text{True}(M) \) is justified. The notion of justifiableness of an atom is defined inductively: \( a \) is justified if:

- there is a clause \( a \leftarrow \neg c_1, \ldots, \neg c_m \) in the \( P_{\text{inst}} \) such that \( \{c_1, \ldots, c_m\} \subseteq \text{False}(M) \), or

- there is a clause \( a \leftarrow b_1, \ldots, b_n \neg c_1, \ldots, \neg c_n \) in the \( P_{\text{inst}} \) such that \( b_1, \ldots, b_n \) are already justified and \( \{c_1, \ldots, c_n\} \subseteq \text{False}(M) \).

We define

\[
\text{JUST} - \text{MOD}(P) := \{M: M \in \text{MOD}_{3-val}^{\text{Herbo}_P} \text{ and } M \text{is justified by } P\}
\]

\[
\text{REG} - \text{MOD}(P) := \{M: M \text{ is } \leq_k \text{ maximal in } \text{JUST} - \text{MOD}(P)\}
\]

32
Definition 7.9.

\[ \text{REG} - \text{SEM}(P) = \bigcap_{M \in \text{REG-MOD}(P)} \{ L : L \text{ is pos. or neg. literals} \mid M \models L \} \]

Example 7.10. (\text{REG-SEM} is not cumulative) \( P_1 \):

\[
\begin{align*}
    a & \leftarrow \neg b \\
    c & \leftarrow \neg b \\
    b & \leftarrow c, \neg a
\end{align*}
\]

Generally speaking there is no certain algorithm for finding \text{REG-SEM}. One of the possible variants which was used in this and next examples is just going through all possible models checking where they are justified or not.

The regular model of the program \( P_1 \) is \( \{ a, \neg b, c \} \). Therefore \( a \) and \( c \) the only derivable symbols. If we add \( c \) to \( P_1 \) we get another regular model \( \{ b, \neg a, c \} \). Thus \( a \) does not longer hold.

The another example shows us that \text{REG-SEM} and \text{WFS}+ are incomparable.

Example 7.11. \( P_2 \):

\[
\begin{align*}
    a & \leftarrow \neg b \\
    c & \leftarrow \neg b \\
    b & \leftarrow c, \neg a \\
    p & \leftarrow \neg p
\end{align*}
\]

\text{WFS}+ for the program \( P_2 \) is \( \{ p \} \). But \text{REG-SEM} only derives \( a \) and \( c \). So this two semantics can not be compared.

Lemma 7.12. If \( A \) is a justified models for \( P \) then there is a regular model \( A' \) of \( P \) with \( A \leq_k A' \).

\[ \text{Proof.} \] The proof immediately follows from the def 2.4.1 \[ \square \]

We use this lemma to prove that \textbf{Cut} holds for the \text{REG-SEM}

Lemma 7.13. Let \( U_1 \subseteq U_2 \) be (infinite) sets of ground atoms. Let \text{REG-SEM}(P \cup U_1 \models U_2). Then every regular models of \( P \cup U_2 \) is also a regular model of \( P \cup U_1 \).

\[ \text{Proof.} \] From the assumption \text{REG-SEM}(P \cup U_1 \models U_2) and definition of \text{REG-SEM} follows that every regular model \( M \) of \( P \cup U_1 \) is also the model of \( U_2 \) and hence a justifiable model for \( P \cup U_2 \). Using the previous lemma we can extend \( M \) to a regular model \( M' \) for \( P \cup U_2 \). Since \( M' \) is justifiable for \( P \cup U_1 \) we have \( M' = M' \). \[ \square \]
We are now interested in any semantics which could be comparable with the \( \text{REG} - \text{SEM} \). Let us define a restricted version of \( WFS^+ \) by requiring that the added atoms are both contained in all classical models and in all regular model.

**Definition 7.14.** Let \( P \) be a normal program. Define \( \text{SEM}(P) := WFS(P) \cup \{ a: a \text{ an atom with: } P \models a \text{ and } \text{REG} - \text{MOD} \models a \} \).

It is easy to see that the assumption of theorem 4.6 are satisfied and we get a rational semantics \( WFS' = \text{SEM}^+ \).

**Theorem 7.15.** \( WFS' \) is the rational extension of \( WFS \) satisfies \textbf{Cut} and \textbf{C}, i.e. a cumulative and rational extension of \( WFS \).

From the definitions of \( WFS' \) and \( WFS^+ \) it is easy to see that \( WFS \leq_k WFS' \leq_k WFS^+ \). Furthermore from the definition \( WFS' \) it is obvious that \( \text{REG} - \text{SEM} \) is a \( \leq_k \) extension of \( WFS' \).

### 7.5 O-SEM

The next extension of the Well-Founded-Semantics has been constructed by adding suitable negative literals. Such extension was proposed by Pereira [LMP95] and got a name O-Semantics. The idea was to find among all sustainable \( A \)-Models of a program \( P \), one that determines the meaning of \( P \) then the Closed World Assumptions is enforced.

**Definition 7.16.** An Assumption Model of a program \( P \), or \( A \)-Model, is a pair \( (A; M) \) where \( A \subseteq \neg B_P \) and \( M = WFS(P + A) \).

**Definition 7.17.** An \( A \)-Model \( (A; M) \) is consistent iff \( A \cup M \) is an interpretation, i.e. there exists no assumption \( \neg c \in A \) such that \( c \in M \).

**Definition 7.18.** A consistent \( A \)-Model \( (A; M) \) is defeated by a consistent \( (A'; M') \) iff \( \exists \neg a \in A \text{ such that } a \in M' \).

**Definition 7.19.** An \( A \)-Model is sustainable iff it is consistent and not defeated by any consistent \( A \)-Model.

**Theorem 7.20.** The set off all sustainable \( A \)-Models of a program is nonempty. On the basis of the set union and set intersection among their \( A \) sets, the \( A \)-Models ordered by \( \leq \) form a lower semilattice.

**Definition 7.21.** A Candidate Structure \( CS \) of a program \( P \) is any sub-semilattice of the lower semilattice of all sustainable \( A \)-Models of \( P \).
Definition 7.22. Let \( \{(A_1;M_1), \ldots, (A_n;M_n)\} \) be the set of all maximal A-Models, in Candidate Structure CS. Let \( J = (A_j;M_j) \) be the join of all such A-Models, in the complete lattice of all A-Models. An A-Models \( (A_j;M_j) \) is untenable w. r. t. CS iff it is the maximal in CS and there exists \(-a \in A_i\) such that \( a \in M_j\).

Definition 7.23. (Retained CS) The Retained Candidate Structure \( R(CS) \) of a Candidate Structure CS is defined recursively in the following way, where \( J \) is the join of the elements of CS in the complete lattice of all A-Models:

- \( J \cup CS \) if there are no untenable A-Models in CS
- Otherwise, let Unt be the set of all untenable A-Models w. r. t. CS. Then \( R(CS) = R(CS - \text{Unt}) \).

Definition 7.24. The O-Semantics \( (O - SEM) \) of a program P is defined by the Retained Candidate Structure of the semilattice of all sustainable A-Models of P.

Theorem 7.25. The O - SEM of a program P is always defined by a complete lattice of sustainable A-models.

Proof. Since every candidate structure is a semilattice of sustainable A-models, it is enough to prove that the join \( J = (A_j;M_j) \) is a sustainable A-Model.

If we assume that \( J \) is inconsistent then at least one maximal A-model in CS is untenable. Accordingly since in the final CS there are by definition no untenable A-Models, \( J \) is consistent. By the definition of candidate structure J cannot be defeated by any other consisted A-Model D because, in this case, at least one other element of CS would also be defeated by D and we come to contradiction.

Example 7.26. Consider the program P:
\[
\begin{align*}
c & \leftarrow \neg a, \neg c \\
c & \leftarrow \neg b, \neg c \\
a & \leftarrow \neg b, \neg c \\
a & \leftarrow d \\
b & \leftarrow d \\
c & \leftarrow d \\
d & \leftarrow \neg d
\end{align*}
\]
To find $O$-Sem we have to find the Retained Candidate Structure $R(CS)$. To see which of two constructing rules is applicable we need to know all $A$-Models and join of all models.

$S = (((\neg a); \emptyset), ((\neg b); \emptyset), ((\neg c); \emptyset), ((\neg a, \neg b); \emptyset), (\emptyset; \emptyset))$

These all models are sustainable. The semilattice of all sustainable $A$-Models Candidate Structure ia an figure 1.

The join of its maximal $A$-models is $((\neg a, \neg b, \neg c); \{a, c\})$ is untenable since it contains $\neg a, \neg b$ in the assumptions and $\{a, c\}$ is a consequence of the join. So $R(CS) = R(CS')$ and shown in figure2.

Again the join of of all elements is $(\neg a, \neg b); \{\}$, there are no untenable in $CS'$. So the $O$-Model is $(\neg a, \neg b)$. The $O$-Sem of a program is shown in figure 3.

In the previous example we show how the $O-Sem$ of the program can be found. But we are of course interested in the properties of this semantics. It was proved that the $O-Sem$ is cumulative but not rational. We show
that $O-SEM$ is not rational in the next example.

**Example 7.27.** Given a program $P$

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

Firstly find the $O-SEM$ of the given program. The set of $A$-Models is equal to $(((\neg p); \{q\}), (\neg q; \{}), (\{}; \{}))$. But the second $A$-model is defeated by the first one. Thus we have only two sustainable models: $(((\neg p); \{q\}), (\{}; \{}))$. The join $J$ is equal the set of sustainable models, and $R(CS) = (\neg p; \{q\}), (\{}; \{}))$.

Consider another program $P_1 = P \cup \{a\}$. The $O$-SEM of this program is equal to $((\neg q); \{a, p\}), (\{}; \{}))$. One can see that $\neg p$ is derivable from program $P$ but from program $P_1$ only $p$ is derivable (and therefore $\neg p$ is not derivable as Rationality requires).

**Theorem 7.28.** The $O$-Semantics is **cumulative** but not rational.

### 7.6 Well-Founded-by-Case-Semantics

Logic programming without negation does not expressive enough for many application. Actually we need to express negation and incomplete information. That is why we are interested to be able to define the semantics for the program with negation. Somebody can say that we already have Well-Founded-Semantics which can treat with negation. It seems to be so,
but not always, sometimes the \textit{WFS} fails to capture the meaning of some very simple programs. Give just an example of two programs where \textit{WFS} can not be defined.

\textbf{Example 7.29.} Consider the following programs $P_1$
\begin{align*} 
  a & \leftarrow \neg b \\
  b & \leftarrow \neg a \\
  c & \leftarrow \neg a, \neg b \\
  \text{and } P_2 \\
  a & \leftarrow \neg b \\
  b & \leftarrow \neg a \\
  d & \leftarrow a \\
  d & \leftarrow b
\end{align*}

By intuition of \textit{WFS}: positive literals can only be derived from known negative literals. In program $P_1$ we have incomplete information, and thus $\neg b$ and $\neg a$ are assigned the undefined values to the both of literals. We have that conjunction $\neg a \land \neg b$ is always false, and according to the negation as failure $c$ should be false. But well-founded models fails assigning the false to $c$.

The same situation with the program $P_2$, here all free atoms are undefined. So we saw that \textit{WFS} failed moreover all its extensions described above are not able to define the meaning of $P_1$ and $P_2$.

Recently several attempts have been made to find extensions of the Well-Founded-Semantics which are able to define semantics of programs like $P_1$ and $P_2$. In this section we introduced the John S. Schlipf’s Well-Founded-by-Case-Semantics and in the next section discuss another extension called Extended-Well-Founded-Semantics which was obtained by modifying Shlips’s construction.

Before we give the definition of Well-founded-by-case-semantics ($WFS_C$) we need to define one more notion.

\textbf{Definition 7.30.} Let $P$ be a propositional program. For each propositional letter $p$ in $P$, the \textbf{stable-by-case completion} of $P$:
\begin{align*} 
  p & \leftrightarrow \bigvee \{\neg q_1 \land \ldots, \land \neg q_n: p \leftarrow \neg q_1 \land \ldots, \land \neg q_n \text{ is a derived rule of } P \}.
\end{align*}

\textbf{Example 7.31.} Let $P$:
\begin{align*} 
  a & \leftarrow \neg b \\
  b & \leftarrow \neg c \\
  c & \leftarrow \neg a
\end{align*}

Then the stable-by-case-completion is $\{a \leftarrow \neg b, b \leftarrow \neg c, c \leftarrow \neg a\}$. 

38
Now we give the definition of Well-Founded-by-Case-Semantics.

**Definition 7.32.** The Well-Founded-by-Case-Semantics \((WFS_C)\) of a program \(P\) is the set of all literals true in all \(3\)-valued models of the stable-by-case-completion of \(P\).

**Example 7.33.** Consider the following program \(P\):
\[
\begin{align*}
  b & \leftarrow \neg a \\
  p & \leftarrow \neg p \\
  p & \leftarrow a
\end{align*}
\]

The stable-by-case-completion \(\{p \leftrightarrow \text{true} \lor a, a \leftrightarrow \neg b, b \leftrightarrow \neg a\}\). By definition the \(WFS_C\) is the set of all literals true in all models of \(P\). It is easy to see there is only one such literal \(p\). Thus \(WFS_C = \{p\}\).

**Theorem 7.34.** For any logic program \(P\) \(WFS(P) \leq_k WFS_C(P)\).

**Proof.** For \(M\) the set of literals
\[
WFS(P \cup M) = \{p; \text{there is a rule } p \leftarrow \neg c_1 \land \ldots \land \neg c_k \text{ where } \neg c_1, \ldots, \neg c_k \in M\} \cup \{\neg p; \text{for every rule } p \leftarrow \neg c_1 \land \ldots \land \neg c_k, \text{some } c_i \in M\}
\]
Define \(M_\gamma = \bigcup_{\mu<\gamma} M_\mu\) and denote the \(WFS(P \cup M_\gamma) = J_\gamma\). The first \(J_\gamma = M_\gamma\). Now we have to prove that \(J_\gamma \subseteq WFS_C(P)\). Assume that \(M_\gamma \not\subseteq WFS_C(P)\). Show that every positive literal in \(M_\mu\) is in \(WFS_C(P)\).

At once we can formulate the next observation.

**Theorem 7.35.** There is a program \(P\) and \(s \in P\) such that \(\neg s\) holds in every minimal model of \(P\) but \(\neg s\) is not inferred by the Well-Founded-by-Case-Semantics.

39
Proof. Let \( P \):
\[
\begin{align*}
  a & \leftarrow \neg b \\
  b & \leftarrow \neg c \\
  c & \leftarrow \neg a \\
  s & \leftarrow a \land b \land c
\end{align*}
\]
The stable-by-case completion of \( P \) is \{a \leftarrow \neg b, b \leftarrow \neg c, c \leftarrow \neg a, s \leftarrow \neg b \land \neg c \land \neg a\}.

The Well-founded-by-Case-Semantics can not say anything about \( a, b, c \) so it can not infer \( \neg s \).

The next thing we do in this section is to prove that, although two very differently defined \( WFS_C \) and \( WFS^+ \) are identical.

**Theorem 7.36.** For all programs \( P \): \( WFS^+(P) = WFS_C(P) \)

Since the proving this theorem is actually based of the another theorem, let us prove the additional theorem.

**Theorem 7.37.** Let \( P \) be a logic program and let \( a \in P \). Then \( a \) holds in all models of \( P \) iff \( a \leftarrow \) is derivable from \( P \).

**Proof.** Assume that if \( a \leftarrow \) is not derivable rule of \( P \) and construct a such model of \( P \) where \( a \) is false. Enumerate the propositional letters of \( P \) as \( a_0, \ldots, a_n \). For each ordinal number \( i \), we will set \( a_i \) to be either \( a_i \) or \( \neg a_i \) below. Then the final interpretation will be \( I = (a_0, a_1, \ldots, a_n, \ldots) \).

Let \( a_0 = \neg a_0 \). By inductive hypothesis, \( a_0 \leftarrow a_0 \land a_1 \land \ldots \land a_i \) is not derivable form \( P \). If we add \( a_{i+1} \) to our rule and if both
\[
\begin{align*}
  a_0 & \leftarrow a_0 \land a_1 \land \ldots \land a_i \land a_{i+1} \\
  a_0 & \leftarrow a_0 \land a_1 \land \ldots \land a_i \land \neg a_i
\end{align*}
\]
were derived rules of \( P \) then \( a_0 \leftarrow a_0 \land a_1 \land \ldots \land a_i \land a_{i+1} \) would be derived also. Define
\[
\begin{align*}
  \bullet & \quad a_{i+1} = \neg a_{i+1} \text{ if } a_0 \leftarrow a_0 \land a_1 \land \ldots \land a_i \land a_{i+1} \text{ is derived rule of } P \\
  \bullet & \quad a_{i+1} \text{ otherwise.}
\end{align*}
\]
We constructed our rules so that there is no a derived rule \( a_0 \leftarrow a_0 \land a_1 \land \ldots \land a_n \) of \( P \). Show that if there is a derived rule \( a_i \leftarrow a_0 \land a_1 \land \ldots \land a_n \) of \( P \) then \( a_i = a_i \). Since the \( a_i = \neg a_i \)
\[
\begin{align*}
  a_0 & \leftarrow a_0 \land a_1 \land \ldots \land a_i \land a_{i+1} \text{ is derived rule of } P. \text{ The composition of } \\
  a_i & \leftarrow a_0 \land a_1 \land \ldots \land a_n \text{ and } a_0 \leftarrow a_0 \land a_1 \land \ldots \land a_i \land a_{i+1} \text{ gives us a derived rule } a_0 \leftarrow a_0 \land a_1 \land \ldots \land a_i \land a_k \text{ where } k \text{ is maximum of } i \text{ and } n. \text{ Since } I \text{ satisfies all the derived rules of } P, \text{ and hence all the rules of } P, \text{ I is a model of } P. \)
Now we come back to the theorem 7.36

**Proof.** Using the previous theorem it is easy to see that $WFS_C(P \cup M_\gamma) = WFS_C(P)$. $WFS(P \cup M_\gamma) \leq_k WFS_C(P)$ this follows from 7.34

$$WFS(P \cup M_\gamma) \leq_k WFS_C(P) = WFS_C(P)$$

Another direction can be proved as follows. Suppose an atom $a \in WFS_C(P)$. This means that $a$ is true in all three-valued models of the stable-by-case completion of $P$. This can only happen if some of the $q_{ij}$ in

$$a \leftarrow [(-q_{1,1} \land \ldots \land -q_{1,n_1}) \lor \ldots \lor (-q_{n,n_1} \land \ldots \land -q_{n,n_n})]$$

evaluate to false in such way that the whole conjunction becomes true. We can replace all $q_i$ to $q'_i$ such that $q'_i \leftarrow$ is a derived rule and $P \models q'_i$ from the theorem 7.37 and simulating the derivation within $WFSq_i^+$ obtain the desirable result.

And at the end we check whether $WFS_C$ successfully assigns the reasonable meaning the both program from the example 7.29 $WFS_C$ of $P_2$ is equal $d$. The stable-by-case completion of $P_1$ is $\langle \{a \leftrightarrow \neg b\}, \{b \leftrightarrow \neg a\}, \{c \leftrightarrow \neg a \land \neg b\} \rangle$. We have again that conjunction $\neg a \land \neg b$ false, and false value should be assigns to $c$, the semantics failed.

### 7.7 Extended-Well-Founded-Semantics

As we saw the $WFS_C$ failed by assigning the false value to $c$ from the example 7.29 that is rather essentially. Hu and Yuan [HY91] proposed the extension of $WFS$ by capturing all atom which are true and false in the well-founded model, as well as all other atoms whose value can be determined independently but are missing in the Well-Founded-Semantics.

First we define the oriented consequences of a program which fully describe the orientation of the program.

**Definition 7.38.** Let $P$ be a program, then the oriented consequences of $P$ are clauses defined inductively as follows:

- Any instantiated rule (see Chapter I) of $P$ is a oriented consequence of $P$;
- If $a \leftarrow \neg a, c_1, \ldots, c_n$ is an oriented consequence of $P$, so is $a \leftarrow c_1, \ldots, c_n$;
- If both $a \leftarrow b, c_1, \ldots, c_n$ and $a \leftarrow \neg b, l_1, \ldots, l_m$ are oriented consequences so is $a \leftarrow c_1, \ldots, c_n, l_1, \ldots, l_m$.  

41
• If both \( a \leftarrow b, c_1, \ldots, c_n \) and \( b \leftarrow l_1, \ldots, l_m \) are oriented consequences so is \( a \leftarrow c_1, \ldots, c_n, l_1, \ldots, l_m \)

The set of all oriented consequence called derived rules, i.e the set of all ground clauses which can be derived from the \( P \). John Schlipf showed [JS91] that ground atom \( a \) is a logical consequence iff it is an oriented consequences of the program.

There are many redundant oriented consequences which need not to be consider. As example consider the oriented sequence , which has some positive literals in the body. Let \( \alpha: a \leftarrow b, c_1, \ldots, c_n \) be the oriented consequence of some program \( P \), \( a \) and \( b \) - ground atoms. If we assume, \( \alpha \) is the only oriented consequence of the program that it can use it to derive \( A \). Since \( b \) is the atom in the body, it has to satisfied \( b \leftarrow l_1, \ldots, l_m \) then using above definition we see that \( a \leftarrow c_1, \ldots, c_n, l_1, \ldots, l_m \) is also oriented clause, which contradicts with our assumption. This example demonstrates that if an atom can be derived from an oriented clause with some positive atoms in the body, then it can be always derived from oriented clause without positive literals. Actually we need only consider those oriented sequences that have no positive literals in the body.

**Definition 7.39.** A reduced oriented consequence is an oriented consequence \( a \leftarrow c_1, \ldots, c_n \) of a program \( P \) such that

1. For any oriented consequence \( a \leftarrow l_1, \ldots, l_m \), \( \{l_1, \ldots, l_m\} \not\subset \{c_1, \ldots, c_n\} \) and,
2. All \( c_i \) are negative literals.

Then the Extended-Well-Founded-Semantics (\( WFS_E \)) can be defined as follows: a ground atom \( a \) can be assumed true if there exists an oriented consequence with the head \( a \) such that its body has to bee true, a ground atom \( a \) assumed false only if for all oriented consequences with \( a \) as a head, their bodies have already been shows false. More precisely.

**Definition 7.40.** Let \( P \) be a program. The extended-well-founded-model of \( P \) is an interpretation \( I_m = (T_m, F_m) \) such that \( T_m \) is the set of all true-justified atoms, and \( F_m \) - false-justified atoms respectively. The Extended-Well-Founded Semantics is specified by the extended-well-founded-model for a given program.

Now consider again the program \( P_1 \) from the example 7.29
\[
a \leftarrow \neg b
\]
\[ b \leftarrow \neg a \]
\[ c \leftarrow \neg a \neg b \]

Extended-Well-Founded-Semantics of this program is equal \( WFS_E(P1) = (\{\emptyset\}, \{c\}) \).

This example shows the difference between the Well-Founded-by-Case-Semantics and Extended-Well-Founded-Semantics.

### 7.8 Conclusion

We considered all extensions of \( WFS \) only from one point of view, i.e. we concentrated on strong properties, induced by Kraus, Lehmann and Magidor.

But as we saw not all extensions of \( WFS \) satisfy these properties. For example \( \text{O-Sem} \) does not obey the property of \textbf{Rationality} and \( \text{REG - SEM} \) is not \textbf{cumulative}, and the Generalized-Well-Founded-Semantics has very strange behaviour, is assigns for two equivalent programs \( GWFS \) assigns two different semantics. It becomes obvious that any semantics expect strong properties, has another, that determine it’s strange behaviour. These properties were called \textit{weak properties} and will be discussed in detail (see Chapter III). Only taking both the \textit{strong} and \textit{weak} properties together we can get the uniquely determined by these properties semantics.
References


References


8 Introduction

8.1 STABLE semantics

In the first part of this chapter, there is a description of the stable models semantics. This model-theoretic declarative semantics for normal programs generalizes the previous referred semantics for restricted classes of programs, in the sense that for such classes the results are the same and, moreover, for some non-restricted programs a meaning is still assigned.

The basic idea behind the stable model semantics came from the field of non-monotonic reasoning formalism. There, literals of the form not A are viewed as default literals that may or may not be assumed or, alternatively, as epistemic literals, expressing that A is not believed. If program has a total well-founded model, that model is the unique stable model. There are also programs which do not have total well-founded models, but do have unique stable models. Gelfond and Lifschitz define a stable model to be one that reproduces itself in a certain three stage transformation, which is called the stability transformation. If a program has only one stable model, that is called its unique stable model. Stable models refer to 2-valued logic.

The stable model semantics is very often identical to well-founded semantics. And at first it appeared that the only difference was that the well-founded semantics defined a partial model when there were multiple stable models. However, it turns out that there also are programs with a unique stable model and only a partial well-founded model.

The main difference between WFS and STABLE is that in the definition of the first one, more and more atoms are declared to be true (or false): once a decision has been drawn, it will never be rejected. In the definition of STABLE however, a guess is made and then a particular model is constructed and used to justify the guess or to reject it.

8.2 Well-behaved semantics

Negation in logic programming differs a lot from classic logic. Many semantics have been suggested in the last 15 years, each trying to capture certain intuitions about the role of the negation operator. However two radically different semantics emerged and are used extensively: the well-founded semantics WFS and the stable semantics STABLE. While WFS is
polynomial time computable and always consistent, STABLE is located on the second level of the polynomial hierarchy and often inconsistent. Most of other semantics were defined by very technical and complicated fixpoint construction and they show irregular behavior.

A line of research by J. Dix begun in [Dix95a] and [Dix95b] introduced the notion of a well-behaved semantics and aims at a classification of well-behaved semantics according to other, clearly formulated declarative properties.
9 STABLE

The idea of the stable semantics is that in an intended (two-valued) model any atom should have a definition reason to be true or false. Let us introduce the Gelfond-Lifschitz transformation: for a program \( P \) and a model \( N \subseteq B_P \) we define

\[
P^N := \{ \text{rule}^N : \text{rule} \in P \}
\]

where

\[
(A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m)^N := \begin{cases} 
A \leftarrow B_1, \ldots, B_n, & \text{if } \forall (j) : C_j \notin N, \\
t, & \text{otherwise.}
\end{cases}
\]

Note that \( P^N \) is always a definite program. We can therefore compute its least Herbrand model \( M_{PN} \) and check whether it coincides with the model \( N \).

**Definition 9.1.** \( N \) is called a stable model of \( P \), if \( M_{PN} = N \).

Let us consider the program \( P \) consisting of the clause

\[
\text{wins}(x) \leftarrow \text{move\_from\_to}(x,y), \neg \text{wins}(y)
\]

together with facts

\[
\text{move\_from\_to}(a,b), \text{move\_from\_to}(b,a), \text{move\_from\_to}(b,c), \text{move\_from\_to}(c,d),
\]

There are two stable models: both contain \( \{ \neg \text{wins}(d), \text{wins}(c) \} \). In the first we also have \( \{ \text{wins}(b), \neg \text{wins}(a) \} \) while the second contains \( \{ \neg \text{wins}(b), \text{wins}(a) \} \). The sceptical semantics therefore derives \( \neg \text{wins}(d) \) and \( \text{wins}(c) \) which fits with our intuitions. Unfortunately, stable models do not always exist:

**Example 9.2.** (Non-) existence of stable models

\[
P_{\neg \exists \text{stab}}: \quad a \leftarrow \neg b \\
P_{\exists \text{stab}}: \quad a \leftarrow b
\]

\[
P_{\neg \exists \text{stab}} \text{ has no stable models. } P_{\exists \text{stab}} \text{ has the unique stable model } \{p, b\}.
\]

**Example 9.3.** Adding irrelevant clauses

\[
P_{\text{stratified}}: \quad a \leftarrow \neg b \\
P_{\neg \exists \text{stable}_\text{model}}: \quad a \leftarrow \neg b
\]

\[
P_{\text{stratified}} \text{ and } P_{\neg \exists \text{stable}_\text{model}} \text{ both have models.}
\]
The unique stable model of $P_{stratified}$ coincides with the supported model: therefore, $a$ is derivable. If the clause $p \leftarrow \neg p$ is added, $a$ is no longer derivable because no stable model exists.

This example shows that the truth-value of an atom $a$ also depends on atoms that are totally unrelated with $a$. It seems that this problem could be easily solved by redefining STABLE in the case where no stable models exist. The easiest way is to take the well-founded model $WFS(P)$ and to define:

$$STABLE^*(P) := \begin{cases} WFS(P), & \text{if no stable model exists,} \\ STABLE(P), & \text{otherwise.} \end{cases}$$

But that is not a decision. This solves only the problem with the two particular programs above. More complex programs, showing that even $STABLE^*$ suffers from the same shortcomings, can be easily constructed.

The following examples show that STABLE is not cumulative (e.g., the Cut holds but Cautious Monotony fails).

Example 9.4. **STABLE is not cautious monotonic**

$$P_{-cum}: \quad a \leftarrow \neg b \quad P_{Makinson}: \quad a \leftarrow \neg b$$

$$b \leftarrow \neg a \quad b \leftarrow \neg a, p$$

$$p \leftarrow \neg p \quad p \leftarrow a$$

The first three clauses of $P_{-cum}$ have two minimal models $M_1 = \{a, \neg b, p\}$ and $M_2 = \{\neg a, b, p\}$, neither of them being stable. The fourth clause stabilizes $M_1$, i.e. the whole program has $M_1$ as its only stable model. Therefore, $STABLE(P_{-cum})$ implies $p$ and $a$. But adding $p$ to $P_{-cum}$ we get two stable models: $M_1$ and $M_2$. $a$ does no longer follow from $P_{-cum} \cup \{p\}$. The same applies to $P_{Makinson}$.

**Theorem 9.5. Schlipf: restricted version of cumulativity**

If $a \in WFS(P)$ then $STABLE(P) = STABLE(P \cup \{a\})$

**Definition 9.6. (STABLE)’**

$$STABLE'(P) := \begin{cases} STABLE(P \cup \{a : P \models a\}) & \text{if stable model exists,} \\ WFS^+(P) & \text{otherwise.} \end{cases}$$

**Lemma 9.7. (STABLE’ \leq_k STABLE)**

$STABLE'$ is a supraorthodox semantics below $STABLE$ satisfying Cut.

51
Proof. We only have to prove that $\text{STABLE}'$ satisfies $\text{Cut}$. Let $\beta \in \text{STABLE}'(P)$ and $\gamma \in \text{STABLE}'(P \cup \{\beta\})$. We have to show that $\gamma \in \text{STABLE}'(P)$. This is trivially true if $\text{STABLE}'(P) = WFS^+(P)$. Let us therefore assume that there exists at least one stable model of $P \cup \{a : P \models a\}$. Our assumption about $\beta$ (i.e. every stable model of $P \cup \{a : P \models a\}$ satisfies $\beta$) and the fact that $\text{Cut}$ holds for $\text{STABLE}$ gives:

$$\gamma \in \text{STABLE}(P \cup \{\beta\} \cup \{a : P \models a\}) \text{ implies } \gamma \in \text{STABLE}(P \cup \{a : P \models a\}).$$

Our assumption about $\gamma$ is $\gamma \in \text{STABLE}(P \cup \{\beta\} \cup \{a : P \cup \{\beta\} \models a\})$. But we also have

$$\{a : P \cup \{\beta\} \models a\} \subseteq \text{STABLE}(P \cup \{a : P \models a\})$$

so we can use $\text{Cut}$ again and get $\gamma \in \text{STABLE}(P \cup \{\beta\} \cup \{a : P \models a\})$

Note that here two-valued stable models were used, so that all classical consequences of $P \cup \{\beta\}$ hold in all stable models. For three-valued stable models this doesn’t hold. \hfill \Box

$\text{STABLE}'$ is weaker than $\text{STABLE}$, because $\text{STABLE}$ is not cumulative and some inconsistencies are removed (e.g. for the program "$p \leftarrow \neg p"$ $\text{STABLE}$ is inconsistent while $\text{STABLE}'$ is not. The introduction of new atoms in general decreases the number of stable models). Note that while $\text{STABLE}_C$ might still be inconsistent for some programs $P$, our $\text{STABLE}'$ is always consistent (by construction). Schlipf’s $\text{STABLE}_C$ is still weaker than $\text{STABLE}'$. 

52
10 Well-behaved Semantics

Negation in logic programming differs a lot from classic logic. Many semantics have been suggested in the last 15 years, each trying to capture certain intuitions about the role of the negation operator. However two radically different semantics emerged and are used extensively: the well-founded semantics WFS and the stable semantics STABLE. While WFS is polynomial time computable and always consistent, STABLE is located on the second level of the polynomial hierarchy and often inconsistent. Most of the other semantics were defined by very technical and complicated fixpoint constructions and they show irregular behavior.

A line of research begun by Dix introduces the notion of a well-behaved semantics and aims at a classification of well-behaved semantics according to other, clearly formulated declarative properties. There are nine weak principles that should hold for any semantic whatsoever. First principles are Reduction and Relevance. Reduction formalizes that explicit facts (resp. explicit missing predicates) should be considered true (resp. false). Relevance is induced from the strange behavior of STABLE. The author also presents here variants of PPE and Modularity: properties induced from irregular behavior of GWFS and STN (semantics for disjunctive programs). Then, he discusses the Cut and some instances of extended Cut that can be view as closure conditions: they are induced from irregular behavior of WFS_E and WFS_S. The next two principles are M
-P-extension and Transformation, abstracted from the supported semantics and the underlying NMR-intuition. Then, the author illustrates the relationship between some of these conditions. Finally, Dix formulates the notion of a well-behaved semantics and two representation-conjectures.

All known semantics are defined by declaring some of the (three-valued) models of \( P \) as the intended models. It turns out that the supported model, the stable models (two-valued) and the well-founded model satisfy the following conditions:

\[ C_1: \text{If } a \leftarrow \text{rhs} \in P \text{ and } A \models \text{rhs}, \text{ then } A \models a. \]

\[ C_2: \text{If } A \models a \text{ then there is } a \leftarrow \text{rhs} \in P \text{ with } A \models \text{rhs}. \]

\[ C_3: \text{If all rule bodies for } a \text{ are false in } A \text{ then } A \models \neg a. \]

\[ C_4: \text{If } A \models \neg a \text{ then all rule bodies for } a \text{ are false in } A. \]

Note that we are always interested in the associated sceptical semantics \( \text{SEM}^\text{scept} \) which is given by the \( \cap_k \)-intersection of all intended models. The author wants to list reasonable properties of \( \text{SEM}^\text{scept} \) in order to describe particular semantics as uniquely determined by these properties. Any sceptical semantics \( \text{SEM}^\text{scept} \) such that \( \text{SEM} \) is based on models satisfying \( C_1-C_4, \)

53
also satisfies $C_1$ and $C_3$ but not necessarily $C_2$ and $C_4$. It is only considered $C_1$ as a property that should be always satisfied. $WFS^+$ (and also GWFS) do not satisfy $C_3$ because $WFS^+(\{a \leftarrow \neg a\}) = \{a\}$. $WFS^+$ however satisfies $C_3$.

10.1 Relevance and Reduction

There are two notions, that are needed for following two conditions:

dependencies of $(X) := \{A : X dependson A\}$.

$rel_{ul}(P,X)$ is the set of relevant rules of $P$ with respect to $X$, i.e. the set of rules that contain an $A \in dependencies_of(X)$ in their heads.

Given any semantics $SEM$ and a program $P$, it is perfectly reasonable that the truth-value of a literal $L$, with respect to $SEM(P)$, only depends on the subprogram formed from the relevant rules of $P$ with respect to $L$.

**Definition 10.1. Relevance**

Relevance states that for all literals $L$: $SEM(P)(L) = SEM(rel_{ul}(P,L))(L)$.

The set of relevant rules of a program $P$ with respect to a literal $L$ contains all rules, that could ever contribute to L’s derivation (resp. nonderivability). In general, $L$ depends on a large set of atoms: $dependencies_of(L) := \{A : L dependson A\}$. But rules that do not contain these atoms in their heads will never contribute to their derivation. Therefore, these rules should not affect the meaning of $L$ in $P$.

STABLE does not satisfy this principle. This is not only due to the nonexistence of stable models after adding a clause "$c \leftarrow \neg c$” to a program, it is a sense intrinsic in the definition of stable models: it could have a consistent stable semantics, but not satisfies Relevance.

Although the modified definition STABLE$^*$ does not satisfy Relevance.

The following semantics STABLE$^\#$:

$STABLE^\#(P)(L) := \begin{cases} STABLE(rel_{ul}(P,L))(L) & \text{if there exist stable models,} \\ WFS(rel_{ul}(P,L))(L) & \text{otherwise.} \end{cases}$

satisfies Relevance but, does not satisfy Cut.

There is another natural property satisfied for all semantics. Suppose we add a set $N$ of atoms to a program $P$. Then:

1. We can allow $N$ to contain negative literals. We define $P$ to be equal to $P \cup \{-A\}$ just in case $A$ does not appear in any head of $P$. 

54
2. We can allow to replace any positive occurrence of an atom $A$ by the associated defining rules.

The formal statement for possibility 1. is:

**Definition 10.2. Reduction**

Let a consistent set of literals $M \subseteq B_P \cup \neg B_P$ be given. The principle of Reduction states that $SEM(P \cup M) = SEM(P^M) \cup M$.

It is a very weak property. It says that whenever we have an explicit fact "$a \leftarrow"$ then we can replace every occurrence of $a$ by "$t$" (resp. by "$f$"). For example, program "$P \cup \{\neg x\}$" is just an abbreviation for the program obtained from $P$ by deleting all clauses with $x$ in their heads and keeping the language of $P$ ($L_{P \cup \{\neg x\}} := L_P$). Therefore Reduction assures that atoms that do not occur in the heads of a program are always assigned "$f$".

This property should definitely be among the minimal requirements for semantics of logic programs: it is nothing else than a very weak form of the negation-as-failure-idea.

The formal statement for possibility 2. is the Principle of Partial Evaluation introduced in the next section.

### 10.2 PPE and Modularity

The principle of partial evaluation (PPE) states that any semantics should assign the same meaning to a program $P$ and a partial evaluation of it.

**Definition 10.3. PPE, weak version**

Let $P$ be an instantiated program and let an atom $c$ occur only positively in $P$. Let $c \leftarrow \text{rhs}_1, \ldots, c \leftarrow \text{rhs}_n$ be all the rules of $P$ with $c$ in their heads. Assume further that $\text{rhs}_1, \ldots, \text{rhs}_n$ do not contain $n$.

$P_c$ denotes the program obtained from $P$ by deleting all rules with $c$ in their heads and replacing each rule "$\text{head} \leftarrow c, \text{body}"$ containing $c$ by the rules:

\[
\begin{align*}
\text{head} & \leftarrow \text{rhs}_1, \text{body} \\
& \vdots \\\n\text{head} & \leftarrow \text{rhs}_n, \text{body}
\end{align*}
\]

The weak PPE states that

\[SEM(P_c) = SEM(P) \setminus \{c, \neg c\}.\]

PPE can significantly decrease the complexity of computing semantics, since it allows us to reduce the program by making the underlying Herbrand base smaller: in general, less assignments of elements of $B_P$ have to be taken into account.
Definition 10.4. **PPE**

Let $P$ be an instantiated program and let an atom $c$ occur only positively in $P$. Let $c ← \text{rhs}_1, \ldots, c ← \text{rhs}_n$ be all the rules of $P$ with $c$ in their heads. Any program clause of the form "head $← c$, body" can be replaced by the rules

$$
\text{head} ← \text{rhs}_1, \text{body} \\
\vdots \\
\text{head} ← \text{rhs}_n, \text{body}
$$

(Note that the rules $c ← \text{rhs}_1, \ldots, c ← \text{rhs}_n$ are not removed in contrast to the weak PPE). The program obtained in this way is $P'$. The PPE is: $SEM(P') = SEM(P)$

Definition 10.5. **GPPE (strong version of PPE)**

GPPE is obtained from PPE by allowing $c$ to occur arbitrarily in $P$.

The principle **Modularity** has some similarities with PPE. It enables us to compute a semantics by modularizing it into certain "subprograms" (formed of the relevant rules). The semantics of these modules can be computed first and the semantics of the whole program can be determined by reducing this program with literals that were already determined.

Definition 10.6. **Weak Modularity**

Let $P$ be instantiated, $P = P_1 \cup P_2$, $B_{P_1} \cap B_{P_2} = \text{rel}_rul(P,A)$. Then $SEM(P) = SEM(P_1^{SEM(P_2)} \cup P_2)$.

Semantics of normal programs are satisfy Weak Modularity. It is useful to strengthen this condition by weakening the assumptions "$B_{P_1} \cap B_{P_2} = \{A\}$ and $P_2 = \text{rel}_rul(P,A)$":

Definition 10.7. **Modularity**

Let $P$ be instantiated, $P = P_1 \cup P_2$ and for every $A \in B_{P_2}$: $\text{rel}_rul(P,A) \subseteq P_2$. The principle of Modularity is: $SEM(P) = SEM(P_1^{SEM(P_2)} \cup P_2)$.

10.3 Closure Conditions

Definition 10.8. **Closure**

The principle of Closure is: $SEM(P^{SEM(P)}) = 0$.

Definition 10.9. **Weak Model-property**

If $a ← ¬b \in P$, and $¬b \in SEM(P)$, then $a \in SEM(P)$.  

56
This property ensures with Reduction, PPE and Cut that any semantics SEM(P) is a model of P (C₁ property). As follows, the extended Cut is implied by the Cut if GPPE holds. Therefore the Closure and the weak Model-Property become obsolete in the presence of Cut and GPPE.

10.4 $M_P$-extension and Transformation

Equivalent programs should have the same semantics. But this notion of equivalence depends on the underlying semantics: therefore there is a need to model a very weak form of equivalence, true for all possible semantics. Consider two programs $P$ and $Q$. If $Q$ is just a renaming of the constants in $P$, then $P$ and $Q$ should obviously be equivalent, i.e. renaming of SEM(Q) should be equal SEM(P):

**Definition 10.10. Isomorphy**

Let $P$ and $Q$ be isomorphic, i.e. there is an isomorphism $I$ from $B_{L_P}$ onto $B_{L_Q}$ s.t. $I(P)=Q$. The principle of Isomorphy is: $I(SEM(P))=SEM(Q)$.

Together with the properties Relevance and PPE Isomorphy also applies to more complex programs that need not to be isomorphic but consist of isomorphic subprograms and thus can also be proved to be equivalent (but there are still other intuitively correct equivalences possible between different programs).

**Definition 10.11. Normalform**

Let $P$ be a program. We define the following transformations:

- $T_1$: If head ← body and head ← body’ are two clauses from $P$ with body ⊆ body’, then the clause head ← body’ may be removed.
- $T_2$: Clauses of the form $x ← x, a_1, \ldots, a_n$, where $a_i$ are atoms, may be removed.

In this way with any program $P$ a normalform $normalform(P)$ is associated that is unique up to the order of the clauses and the literals in the clauses. The underlying language of $normalform(P)$ is $L_P$ and not $L_{normalform(P)}$.

**Definition 10.12. Equivalence**

The principle of Equivalence states that $SEM(P)=SEM(normalform(P))$.

For finite propositional programs, it is possible to use Equivalence and PPE to construct the model $M_P$.  

57
For infinite propositional programs there is still need for the explicit requirement that for definite programs any semantics should coincide with the least Herbrand model:

**Definition 10.13.** $M_P$-extension

If $P$ is a definite program, then $SEM(P) = \{ l : \text{l literal with } M_P \models l \}$.

**Definition 10.14.** Transformation

Let $P$ be a program, $P^+$ the program obtained from $P$ by cancelling all negative literals, and $P^-$ be the program obtained from $P$ by cancelling all program-clauses containing a negative literal. A semantics $SEM$ satisfies the principle of Transformation if the following holds:

- $\neg x \in SEM(P^+) \Rightarrow \neg x \in SEM(P)$,
- $x \in SEM(P^-) \Rightarrow x \in SEM(P)$

It follows that Transformation follows for finite programs from GPPE, Reduction and Relevance. Transformation is powerful enough to simulate WFS.

### 10.5 Deducible Rules

**Lemma 10.15.** $C_1$ is derivable, i.e. $SEM(P) \models P$.

A semantics satisfying Reduction, weak PP, CUT and the Weak Model-Property also satisfies the following:

If $a \leftarrow l_1, \ldots, l_n \in P$ and $l_1, \ldots, l_n \in SEM(P)$ then $a \in SEM(P)$.

**Proof.** Using weak PPE and the Weak Model-Property we can assume that all $l_i$ are positive. But then we apply Reduction and get $SEM(P \cup \{ l_1, \ldots, l_n \}) = SEM(P^{l_1, \ldots, l_n}) \cup \{ l_1, \ldots, l_n \}$, i.e. (applying Reduction to the fact $a \leftarrow$ again) $a \subseteq SEM(P \cup \{ l_1, \ldots, l_n \})$. Using Cut, the last set is contained in $SEM(P)$ and we are done.

**Lemma 10.16.** Deducible variant of Modularity

Let $P$ be instantiated, $P = P_1 \cup P_2$, $B_{P_1} \cap B_{P_2} = \{ A \}$ and $P_2 = \text{rel.rul}(P, A)$. Using Modularity, Relevance and Reduction we have:

- if $A \in SEM(P_2)$ then $SEM(P) = SEM(P_1 \cup \{ A \}) \cup SEM(P_2)$,
- if $\neg A \in SEM(P_2)$ then $SEM(P) = SEM(P_1 \cup \{ \neg A \}) \cup SEM(P_2)$.

If $SEM(P_2)$ is complete, i.e. $\forall x \in B_{P_2}: x \in SEM(P_2)$ or $\neg x \in SEM(P_2)$, then we have:
Lemma 10.17. Further variant of Modularity
Let $P$ be instantiated, $P = P_1 \cup P_2$, $B_{P_1} \cap B_{P_2} = \{A\}$ and $P_2 = rel_{rul}(P, A)$ and $SEM(P_2)$ complete.

Using Modularity, Relevance and Reduction we have:

$$SEM(P) = SEM(P_1^{SEM(P_1)}) \cup SEM(P_2)$$

Modularity is a restricted form of Cumulativity if we assume some of our principles introduced above:

Lemma 10.18. Extended Cumulativity implies Modularity
From Reduction, Relevance and Extended Cumulativity follows Modularity.

Lemma 10.19. Weak PPE is deducible
If Relevance holds then GPPE implies PPE and PPE implies weak PPE.

Lemma 10.20. GPPE implies a strong normalform
Let SEM satisfy Equivalence and GPPE. Then any finite propositional program $P$ can be equivalently transformed into a program $P^*$ where only negative literals occur in the bodies of rules. $P^*$ is called the strong normalform of $P$.

Lemma 10.21. GPPE implies extended versions of Cut and Cumulativity
Let SEM satisfy Reduction, Equivalence, Relevance, weak Model-Property and GPPE.

- If SEM satisfies Cut then it also satisfies Extended Cut for all programs in strong normalform.
- If SEM satisfies Cumulativity then it also satisfies Extended Cumulativity for all programs in strong normalform.

Lemma 10.22. Simulating Transformation for finite programs
For finite programs GPPE, Reduction, and Equivalence imply Transformation for programs in strong normalform.

Lemma 10.23. Simulating $M_P$-extension for finite propositional programs
For finite programs $M_P$-extension follows from weak PPE, Reduction and Equivalence.

10.6 Main definition and results
In order to define what is reasonable semantics, let us observe the following definitions.
Definition 10.24. Well-behaved semantics

A well-behaved semantics $\text{SEM}$ is a mapping

$$\{P : P \text{ a program}\} \leftarrow 2^{L \text{it}}$$

such that the following conditions are satisfied: Cut, Closure, weak Model-Property, Isomorphy, $Mp$-extension, Transformation, Relevance, Reduction, PPE and Modularity.

If we only consider finite propositional programs and we also assume GPPE then we are left with only: Reduction, Equivalence, Cut, Relevance, Isomorphy, weak Model-Property and Modularity. In addition, it is sufficient to state the last five conditions only for programs in strong normalform.
To derive other properties that are implied by our conditions, we need the next idea: fixing a semantics on a reasonable class of programs (e.g. axiomatically requiring $SEM(P_{\text{splitting}}) = 0$, $SEM(\{a \leftarrow \neg b, b \leftarrow \neg c, c \leftarrow \neg a\}) = 0$ and further axioms) already determines these semantics using our conditions. It is, however, possible that we still need more conditions (for different axiomatizations).

Using the principles that were introduced already, it is possible to prove:

**Theorem 10.25.** Well-behaved semantics are extensions of $M_{P}^{\text{supp}}$

Any well-behaved semantics $SEM$ coincides for stratified programs with $M_{P}^{\text{supp}}$.

**Proof.** We use the following two steps (in the indicated order) to reduce a program $P$ (and determining the truth value of any atom of $L_{P}$) until the empty program is obtained:

1. Iterating Reduction we can assume that $P$ (after Reduction) has the following two properties:
   - no clause of $P$ has an empty body, and
   - all atoms occurring in $P$ already occur in some head.

2. Let $M_{P}^{\text{pos}}$ be the set of all atoms that do not depend negatively on another atom. We can split $P$ into $P = P_{1} \cup P_{2}$,

where $P_{2}$ consists of all clauses that are solely built from atoms contained in $M_{P}^{\text{pos}}$.

We now get (using $M_{P}$-extension)
\[ SEM(P_2) = \{ \neg x : x \in M_{pos} \} \]

which is in accordance with \(M_P^{supp}\); note that \(P\) can be stratified in such a way that \(P_2\) is just in the first stratum. By Modularity \(P\) is reduced to the smaller

\[ P' = P_1^{SEM(P_2)}. \]

It is possible to iterate our procedure (first using 1. then 2.) and, because \(P\) is stratified (no atom depends negatively on itself) the procedure stops after a finite number of steps (at most after \(n\) steps where \(n\) is the number of strata) with the empty program. All atoms from \(P\) are evaluated according to \(M_P^{supp}\).

Here there were used only Reduction, Modularity and \(M_P\)-extension in our proof.

For finite propositional programs \(M_P\)-extension can be replaced by \(PPE\) and Equivalence (Reduction is already used).

Using Lemma 4.22 and 4.23 we have the following:

**Corollary 10.26.** The Case of finite propositional programs
For finite propositional programs the last theorem also holds for semantics satisfying only Reduction, Modularity, weak \(PPE\) and Equivalence.

**Corollary 10.27.**
Any well-behaved semantics \(SEM\) coincides for locally-stratified programs with Przymusinski’s unique perfect model.

**Theorem 10.28.** \(WFS\) is the weakest well-behaved extension of \(M_P^{supp}\)
Any well-behaved semantics is a \(\leq_k\)-extension of \(WFS\).

**Corollary 10.29.** The Case of finite propositional programs
For finite propositional programs the last theorem also holds for semantics satisfying only Reduction, \(GPPE\), Equivalence and \(Cut\). Relevance is not needed and the theorem therefore also applies to \(STABLE\).

**Lemma 10.30.** Well-behavedness of some semantics
The semantics \(WFS, WFS^+, O\text{-}SEM\) and \(REG\text{-}SEM\) are well-behaved.

It is known that the stable semantics is not cumulative (and that this does not depend on the nonexistence of stable models). Strong property of Cumulativity does not follow from our weak reasonable properties (\(REG\text{-}SEM\) is an additional candidate):
Lemma 10.31. Well-behaved does not imply Cumulativity
STABLE$^\text{cd}$ is a well-behaved semantics that is not cumulative.

Well-behaved semantics having GCWH-property cannot satisfy the Extended Cut or even $C_3$:

Lemma 10.32. Well-behaved and GCWH-property
Let $SEM$ be well-behaved.

- GCWH is incompatible with Extended Cut.
- GCWH is incompatible with $C_3$.

10.7 Representation conjectures
We claim that our reasonable conditions together with the property of Rationality exactly describe the semantics WFS, WFS$^+$ ($= WFS^C$) and WFS$'$. In particular, we have the following conjectures:

Conjecture 10.33. (Characterization of WFS, WFS$^+$ and WFS$'$) There are no well-behaved and rational semantics other than WFS, WFS$^+$ and WFS$'$.

Conjecture 10.34. (WFS$^+$) $WFS^+$ is the only well-behaved and rational semantics satisfying Supraclassicality.

Conjecture 10.35. (Characterization of WFS and WFS$'$) There are no well-behaved and rational semantics satisfying $C_3$ other than WFS and WFS$'$. 
11 Conclusion

As we observed above, weak and strong properties can be used to uniquely characterize certain semantics. As we saw, certain semantics recently defined showed some strange behavior. This inspired Dix to formulate a list of weak principles, where all semantics should be checked against (PPE, Modularity, Relevance, Cut etc.). Therefore, the author presented an axiomatic framework for reasoning about and defining semantics for logic programs. So it is possible to ask for all instances satisfying certain properties and to state and prove interesting representation theorems.
References


