The Well-founded Semantics Is the Principle of Inductive Definition

Marc Denecker
Department of Computer Science, K.U.Leuven,
Celestijnenlaan 200A, B-3001 Heverlee, Belgium.
Phone: +32 16 327555 — Fax: +32 16 327996
email: marcd@cs.kuleuven.ac.be

Abstract

Existing formalisations of inductive definitions in constructive mathematics are reviewed and strong correspondences with LP under least model and perfect model semantics become apparent. I point to fundamental restrictions of these existing formalisations and argue that the well-founded semantics overcomes these problems and hence, provides a superior formalisation of the principle of inductive definition. I point to fundamental differences between this knowledge theoretic interpretation of LP and the more commonly known interpretations of LP as default theories or autoepistemic theories.

1 Introduction

With the completion semantics [5], Clark aimed at formalising the meaning of a logic program as a set of definitions. To that aim, he maps a logic program to a set of First Order Logic (FOL) equivalences. Motivated by the research in Nonmonotonic Reasoning, logic programming is currently often seen as a default logic or autoepistemic logic. In [10], Gelfond proposes a semantics for stratified logic programs based on an autoepistemic interpretation of the formalism. In [11], Gelfond and Lifschitz motivate the stable semantics for logic programs from the perspective of logic programs as default and autoepistemic theories.

To compare these readings, consider the program P_0 with unique rule:

$$dead \leftarrow not\ alive$$

P_0 is propositional and hierarchical; all common semantics of LP (completion / perfect [3, 19] / stable [11] / wfs [25]) *agree*; for the above example, the unique model is $\{dead\}$.

In the interpretation of this program as a autoepistemic theory, P_0 corresponds to the autoepistemic theory (AEL):

$$AEL(P_0) = \{dead \leftarrow \neg K alive\}$$

which reads as: *one is dead* <u>*if*</u> <u>*it is not believed that*</u> *one is alive*. On the other hand, under completion semantics the meaning of this program is given by the FOL theory:

$$comp(P_0) = \{\neg alive \ , \ dead \leftrightarrow \neg alive\}$$

These readings show important differences. The completion reading of P_0 states that *alive* is false while the autoepistemic reading of P_0 gives no information about *alive*; hence *alive* is not known. The completion reading maps implication to equivalence and negation to classical objective negation, while the autoepistemic reading map negation to a modal operator ($\neg K$) and preserves the implication.

That despite this intuitive difference, the stable model -based on the default/autoepistemic reading- is the model of the completion is because a stable model is a *belief set*: the set of atoms which are believed while the model of the completion, as a model of a FOL theory, represents a possible state of the world. Actually, the meaning of the autoepistemic reading and completion can be compared only in model theoretic terms by comparing the possible world model of the autoepistemic theory with the set of models of the completion. The possible world model of the AEL theory $\{dead \leftarrow \neg K alive\}$ is $\{\{dead\}, \{alive, dead\}\}$. This set of models reflects indeed the intuitive meaning of $AEL(P_0)$: nothing is known on *alive* and (therefore) *dead* is always true. Adding the clause *alive* to P_0 would yield the possible world model $\{\{alive\}, \{alive, dead\}\}$: now *alive* is always true and nothing is known on *dead*. The unique model of the completion is $\{alive\}$ which entails that *dead* is always false.

This observation motivates a closer investigation of the relation between logic programming and inductive definitions. An inductive definition is a form of constructive knowledge. Constructive information *defines* a relation (or a collection of relations) through a constructive process of iterating a recursive recipe. This recipe defines new instances of the relation in terms of the presence (and sometimes the absence) of other tuples of the relation. A broad class of human knowledges in many areas of human expertise, ranging from common sense knowledge situations to mathematics, is of constructive nature. In [23], this is illustrated with a study of the ramification problem. We argue that causality information is an example of constructive information. Causes, effects and forces propagate in a dynamic system through a constructive process; hence, the semantics of causality rules is defined by an inductive definition.

In the context of mathematics, constructive information appears by excellence in inductive definitions. For example, as suggested by the name, the *transitive closure* of a binary relation is naturally perceived as the relation obtained trough the construction process of *closing* the relation under the transitivity rule. Not a coincidence, inductive definitions have been studied in constructive mathematics and intuitionistic logic, in particular in the sub-areas of Inductive and Definition logics, Iterated Inductive Definition logics and Fixpoint logics. The main goal of this paper is to review some of this work and to show how inductive definitions are formalised in these areas; this immediately reveals strong relationships with least model and perfect model semantics of logic programming (section 2). I point to fundamental knowledge theoretic problems in these formalisms (section 3) and argue that the logic program formalism under well-founded semantics provides a superior formalisation (section 4). Section 5 considers some implications.

2 Inductive Definitions in mathematics

One can distinguish between positive inductive definitions and definitions by induction on a well-founded set. A prototypical example of a definition by (positive) induction is the one of the transitive closure T_R of a graph R. T_R is defined inductively as follows. T_R contains an arc from x to y if

- R contains an arc from x to y;
- R contains an arc from x to z, and T_R contains an arc from z to y.

It could be formally represented by the rules:

$$\mathcal{D}_{trans} = \left\{ \begin{array}{l} tr(X,Y) \leftarrow graph(X,Y) \\ tr(X,Y) \leftarrow graph(X,Z), tr(Z,Y) \end{array} \right.$$

The intended interpretation of this definition is that the transitive closure is the *least* graph satisfying the implications rather than any graph satisfying the above implications. Alternatively, the transitive closure can be obtained in a constructive way by applying these implications in a bottom up way until saturation. It is commonly known that inductive definitions such as the one of transitive closure cannot be expressed in FOL, and a fortiori, not in the completion semantics[1].

Typical for the above sort of inductive definition is that the induction is positive: i.e. the defined concept depends positively on itself, and hence a unique least relation exists. In definitions by (possibly transfinite) induction on a well-founded poset, this is not necessarily the case. In definitions of this kind, a concept is defined for a domain element in terms of strictly smaller elements. An example is the definition of the ordinal powers of a monotonic operator. A simple first order example of a definition is the definition of even numbers in the well-founded poset $\mathbb{N}, \leq$. One could define that a natural number n is even by induction on $\leq$:

- $n = 0$ is even;
- if n is not even then $n+1$ is even; otherwise $n+1$ is not even.

A formal representation of the definition in the form of implications is:

$$\mathcal{D}^1_{even} = \left\{ \begin{array}{l} even(0) \\ even(s(X)) \leftarrow \neg even(X) \end{array} \right.$$

Now the defined predicate *even* occurs negatively in the body of the rule. Verify that in the natural numbers, this theory has infinitely many minimal models[2].

Its semantics can be described by a constructive process and is also expressed well by the Clark completed definition of the above implications:

$$\forall X.even(X) \leftrightarrow X = 0 \vee \exists Y.X = s(Y) \wedge \neg even(X)$$

The above two types of inductive definitions[3] require a different sort of semantics. This raises the question whether a uniform principle of inductive definition can be proposed which is correct for all inductive definitions and hence generalises and integrates

[1]A simple counterexample: verify that the unintended interpretation with domain $\{a, b\}$ and $I(graph) = \{(a,a)\}$ and $I(tr) = \{(a,a),(a,b)\}$ satisfies the completion of the implications.

[2]E.g. $\{even(0), even(2), ..\}$ but also $\{even(0), even(1), even(3), even(5), ..\}$.

[3]A more complex example showing the elements of transfinite induction in a richer context is the concept of *depth* of an element in a well-founded poset $P, \leq$. Define the depth of an element x of P by transfinite induction as the least ordinal which is a strict upper-bound of the depths of elements $y \in P$ such that $y < x$. Formally, let $F[X, D]$ mean that D is a larger ordinal than the depths of all elements

completion and minimisation. The first attempt to formalise such a principle was in the context of Iterated Inductive Definitions.

The study of inductive definitions in mathematics has started with Post [17], Spector [21] and Kreisel [14]. Important work in this area includes [8, 15, 16, 2, 4]. An offspring of this research is fixpoint logic, currently used in databases [1]. Below is an overview of ideas proposed in the area of Inductive, Iterated Inductive Definitions (IID) and fixpoint logics. The overview is an attempt to give a faithful and comprehensive presentation of the essential ideas in these areas, while we have taken the freedom to reformulate syntax or semantics in order to increase uniformity and comprehensibility.

2.1 Positive Inductive Definitions

Positive Inductive Definitions have been formalised in various ways. In the style of [8], an inductive definition on a given interpretation M is represented as a formula:

$$p(\overline{X}) \leftarrow F[\overline{X}, p]$$

where $F[\overline{X}, p]$ is a First Order Logic (FOL) formula with only positive occurrences of the defined symbol p but arbitrary occurrences of symbols interpreted in M. In fixpoint logic, the relation p would be denoted $\Gamma_{\Psi} F[\overline{X}, \Psi]$ (here p is replaced by a predicate variable Ψ).

[2] studies inductive definitions in a *abstract* representation with an obvious correspondence with definite logic programs. A definition on a domain D of propositional symbols is represented as a possibly infinite set $\mathcal{D}$ of rules $p \leftarrow B$ with $p \in D, B \subseteq D$[4].

[2] gives an overview of three equivalent mathematical principles for describing the semantics of a (Positive) inductive definition. They are equivalent with the way the least model semantics of definite logic programs can be defined [24].

- The model can be defined as the least model of the implications. E.g., in [8], this minimal model semantics is expressed through a circumscription-like axiom (expressing that p must be the least predicate rather than a minimal one).
- The model can be expressed constructively as the least fixpoint of a T_P-like operator associated with the definition. In the presentation of [2], inductive definitions are dually defined as monotonic T_P-like operators. This is the common way in fixpoint Logic (hence the name).

$Y < X$:

$$F[X, D] \equiv [\forall Y, D_Y.(Y < X \wedge depth(Y, D_Y) \rightarrow D_Y < D)]$$

Then, depth is represented by the rule:

$$\mathcal{D}_{depth} = \{\ depth(X, D_X) \leftarrow F[X, D_X] \wedge [\forall D.F[X, D] \rightarrow D_X \leq D]$$

Construction or Clark completion gives the semantics of this definition. The defined predicate *depth* occurs negatively in the body of the rule, and as a consequence, multiple unintended minimal models exist, such as $\{depth(0,0), depth(0,1), depth(1,2), .., depth(n, n+1), ..\}$.

[4]Definitions represented in the other style can be represented in this abstract way. Given the mathematical structure M and formula $F[\overline{X}, p]$, define the domain D as the set of atoms $p(\overline{x})$ with $\overline{x} \in M^n$. Define $\mathcal{D}$ as the set of rules $p(\overline{x}) \leftarrow B$ for each $\overline{x}$ and each set B of p-atoms such that $M \models F[\overline{x}, B]$; meaning that F is true for $\overline{x}$ in M when p is interpreted as the set B.

- The model can be expressed also as the interpretation in which each atom has a *proof tree*. Also this formalisation has been used in LP in [7]. Because it is less commonly used in LP, I present it here for a slightly extended version of the formalism of [2].

  Let be given a domain D including a subset $D_o \subseteq D$ with symbols $\mathbf{t}, \mathbf{f} \in D_o$, an interpretation M interpreting D_o ($M(\mathbf{t}) = \mathbf{t}, M(\mathbf{f}) = \mathbf{f}$), and a definition $\mathcal{D}$, a set of rules $p \leftarrow B$ with head $p \in Defined(\mathcal{D})$ and body B consisting of atoms of $Defined(\mathcal{D})$ and literals[5] of D_o. Often D_o is denoted by $Open(\mathcal{D})$. We assume that each symbol $p \in Defined(\mathcal{D})$ has at least one rule $p \leftarrow B \in \mathcal{D}$ (it may be the rule $p \leftarrow \{\mathbf{f}\}$). Also the body B of a rule is never empty (B may be the singleton $\{\mathbf{t}\}$).

  A $\mathcal{D}$-proof-tree $\mathcal{T}$ of $p \in D$ is a tree of literals of D with p as root such that:

  - all leaves of $\mathcal{T}$ are interpreted literals;
  - for each non-leaf node p with set of immediate descendants B: $p \leftarrow B \in \mathcal{D}$; hence p is a defined symbol in $\mathcal{D}$;
  - $\mathcal{T}$ is loop-free; i.e. contains no infinite branches.

  The model $\overline{M}^{\mathcal{D}}$ of $\mathcal{D}$ given M can be characterised as the set of atoms $p \in D$ which occur in the root of a proof-tree $\mathcal{T}$ such that all leaves are true literals in M. Note that interpreted literals have proof-trees consisting of one node; as consequence, $\overline{M}^{\mathcal{D}}$ extends M.

2.2 Iterated Inductive Definitions

The logics of Iterated Inductive Definitions are or can be seen as attempts to formalise the mathematical principle of definition by (transfinite) induction on a well-founded order. Iterated Inductive definitions were first introduced in [14] and later studied in [8] and [15]. [2] formulates the intuition of Iterated Inductive Definitions in the following way. Given a mathematical structure M fixing the interpretation of the interpreted predicates and function symbols, a positive inductive definition $\mathcal{D}$ prescribes the interpretation of the defined predicate(s). Once the interpretation of the defined symbols p is fixed, M can be extended with these interpretations, yielding a new interpretation $\overline{M}^{\mathcal{D}}$. On top of this structure, again new predicates may be defined in the similar way as before. The definition of this new predicates may depend negatively on the defined predicates p as these are interpreted in $\overline{M}^{\mathcal{D}}$. This principle can be iterated in an arbitrary, even transfinite sequence of positive inductive definitions.

In [2], the abstract definition logic defined there is not explicitly extended with this idea, but given the above intuition, the extension with negation is straightforward. Given a domain D and mathematical structure M, an Iterated Inductive Definition (IID) would be be a possibly transfinite sequence $\mathcal{D} = (\mathcal{D}_\alpha)_{\alpha < \alpha_{\mathcal{D}}}$ of positive inductive definitions such that:

- each defined symbol p is defined in a unique $\mathcal{D}_{\alpha_p}$; we call α_p the stratum of p;

[5]Allowing interpreted symbols to appear negatively simplifies the leap to inductive definitions with recursion over negation.

- for each rule $p \leftarrow B \in \mathcal{D}$, for each defined atom $q \in B$, $\alpha_q \leq \alpha_p$; for each defined atom q such that $\neg q \in B$, $\alpha_q < \alpha_p$.

The model $\overline{M}^{\mathcal{D}}$ of a definition can be obtained by transfinitely *iterating* the principle of positive inductive definition over the sequence $(\mathcal{D}_\alpha)_{\alpha<\alpha_\mathcal{D}}$.

There is an obvious correspondence between Iterated Inductive Definitions (IID's) and stratified logic programs under perfect model semantics [3, 18, 19]. Already in 84, [13] defines a semantics for stratified logic programs based on the Iterated Inductive Definition (IID) logic defined in [15]. To my knowledge, this was really the first time that the perfect model semantics for stratified logic programs was defined. Apparently this work stayed largely unnoticed, perhaps because, like the semantics in [15], it is based on sequent calculus, which to some extend increases the mathematical complexity and obscures the simple intuitions underlying this semantics.

Though the intuition of IID's as formulated in [2] is straightforward, it is not easy to see how this idea is implemented in IID logics such as those of [8], [4] and also in [15]. The reason for this seems as follows. The goal of this research was to investigate theoretical expressivity of transfinite forms of IID's. As explained in [4], a definability study makes only sense in a finitely represented logic, while transfinite IID's in the abstract setting above are per definition infinite objects. [8] investigates IID's encoded in a single FOL formula of the form $F[N, X, P, Q]$ and expresses its semantics through a second order formula.

It is interesting -if only from historical perspective- to see how transfinite definitions must be encoded finitely and how a perfect model-like semantics can be expressed in such a formalism. We illustrate this -tedious- encoding with a definition $\mathcal{D}^2_{even}$, constructed for the occasion:

$$\mathcal{D}^2_{even} = \begin{cases} (0) & even(0) \leftarrow \mathbf{t} \\ (n+1) & even(n+1)) \leftarrow \neg even(n) \\ (n) & even(n)) \leftarrow \neg even(n) \\ (\infty) & sw \leftarrow even(n), even(n+1) \\ (\infty+1) & ok \leftarrow \neg sw \end{cases}$$

The symbol sw (which abbreviates *something_wrong*) represents that two subsequent numbers are even, and ok is its negation. This definition can be stratified (the strata of the defined predicates are given). The model obtained after $\infty + 2$ iterations is $\{ok, even(2n) | n \in \mathbb{N}\}$.

The first step in encoding such an abstract IID $(\mathcal{D}_\alpha)$ yields a possibly infinite disjunction $F[N, X, P, Q]$. For any rule $p \leftarrow \{.., q, .., r, .., \neg s, ..\}$
with $\alpha_q = \alpha_p$, $\alpha_r < \alpha_p$, $\alpha_s < \alpha_p$, add one disjunct:
$N = \alpha_p \wedge X = p \wedge .. \wedge P(q) \wedge .. \wedge Q(\alpha_r, r) \wedge .. \wedge \neg Q(\alpha_s, s) \wedge ..$
In this infinitary formula $F[N, X, P, Q]$, N ranges over the ordinals $\alpha < \alpha_\mathcal{D}$, X over atoms; P over sets of atoms, Q over sets of tuples of ordinals and atoms. The formula corresponding to $\mathcal{D}^2_{even}$ is the following infinitary disjunction with disjuncts for each $0 \leq n$:

$$\begin{cases} N = 0 \wedge X = even(0) \vee ... \\ N = n+1 \wedge X = even(n+1) \wedge \neg Q(n, even(n)) \vee ... \\ N = n \wedge X = even(n) \wedge P(even(n)) \vee ... \\ N = \infty \wedge X = sw \wedge Q(n, even(n)) \wedge Q(n+1, even(n+1)) \vee ... \\ N = \infty + 1 \wedge X = ok \wedge \neg Q(\infty, p) \end{cases}$$

There is only one step more to go to reduce this formula to an equivalent finite IID-form. But first, we show how to express the semantics of the IID in an (infinitary second order) axiom. An IID-form of the above form is used to define a unique binary meta-predicate *holds*: $holds(\alpha, p)$ means that the stratum of p is α and that p is defined true. The two axioms below express essentially that at each stratum α, the set $\{p|holds(\alpha,p)$ is true$\}$ (denoted below as $holds(\alpha, .)$) satisfies the principle of positive inductive definition. The first axiom expresses that for each ordinal α and given *holds* for lower strata, $holds(\alpha, .)$ satisfies the implications in $\mathcal{D}_\alpha$. Below, $F[P(\tau)/holds(N,\tau)]$ denotes the formulas obtained by replacing each expression $P(\tau)$ for arbitrary term τ by $holds(N,\tau)$.

$$\forall N, X[holds(N,X) \leftarrow F[P(\tau)/holds(N,\tau), Q(\tau_1,\tau_2)/holds(\tau_1,\tau_2)]$$

Verify that if one assigns the values α_p to N and p to X, then this complex formula reduces to:

$$\forall N, X.holds(\alpha_p, p) \leftarrow \begin{cases} .. \\ .. \wedge holds(\alpha_q, q) \wedge .. \wedge \neg holds(\alpha_r, r) \wedge .. \quad \vee \\ .. \end{cases}$$

with a disjunct for each $p \leftarrow \{.., q, .., \neg r, ..\} \in \mathcal{D}$[6].

The second axiom expresses that for each ordinal α, $holds(\alpha, .)$ is the least set satisfying the implication of $\mathcal{D}_\alpha$. It is a second order axiom, using a set variable Ψ which ranges over sets of atoms:

$$\forall N.\forall \Psi.\{\forall X.\Psi(X) \leftarrow F[P(\tau)/\Psi(\tau), Q(\tau_1,\tau_2)/holds(\tau_1,\tau_2)]\} \rightarrow \forall X.[holds(N,X) \rightarrow \Psi(X)]$$

Finally, the infinitary IID-form F should be further encoded by a finite formula. This involves:

- encoding ordinals by a (primitive recursive) well-ordering on natural numbers. E.g. the total order $2 \preceq 3 \preceq .. \preceq 0 \preceq 1$ is a well-ordering encoding the ordinals $0, 1, .., \infty, \infty + 1$.
- encoding atoms by natural numbers: an obvious proposal here is to encode each atom by the natural number encoding the stratum of the atom; i.e. $even(n)$ by $n+2$, *sw* by 0 and *ok* by 1.
- encoding tuples of natural numbers by natural numbers. Details of this are tedious and irrelevant for this paper; we omit them.

In this encoding, an infinite number of disjuncts can be represented in a finite formula using quantification in the natural numbers. The different sets of disjuncts are encoded as follows:

[6] In [8], atoms $Q(\alpha_q, q)$ are not replaced by atoms $holds(\alpha_q, q)$ atoms but by open formulas $holds(\alpha_q, q) \wedge \alpha_q < N$. This open formula represents the restriction of *holds* to strata $< N$ (atoms q at higher strata are false in $holds(\alpha_q, q) \wedge \alpha_q < N$). The resulting, more complex axioms can be seen to be equivalent with our axioms for IID-forms obtained from a stratified abstract IID $\mathcal{D}$. The reason for this choice seems to be that the stratification condition, which can be defined nicely for abstract IID's, cannot easily be formulated directly for IID-forms. The more complex axioms determine a unique *holds* predicate even if $F[N, X, P, Q]$ encodes a nonstratifiable or incorrectly stratified definition (but the semantics may be unnatural then); in that case, our simpler axioms do not determine a unique *holds*-predicate due to mutual dependencies between predicates defined at lower and at higher level.

$$
\begin{array}{l}
N = 0 \wedge X = even(0) \\
\qquad \longrightarrow \quad N = 2 \wedge X = 2 \\
N = n+1 \wedge X = even(n+1) \wedge \neg Q(n, even(n)) \\
\qquad \longrightarrow \quad \exists M. N = M+1 \wedge 2 \leq M \wedge X = N \wedge \neg Q(M, M) \\
N = n \wedge X = even(n) \wedge P(even(n)) \\
\qquad \longrightarrow \quad 2 \leq N \wedge X = N \wedge P(N) \\
N = \infty \wedge X = sw \wedge Q(n, even(n)) \wedge Q(n+1, even(n+1)) \\
\qquad \longrightarrow \quad N = 0 \wedge X = 0 \wedge \exists M. 2 \leq M \wedge Q(M, M) \wedge Q(M+1, M+1) \\
N = \infty + 1 \wedge X = ok \wedge \neg Q(\infty, p) \\
\qquad \longrightarrow \quad N = 1 \wedge X = 1 \wedge \neg Q(0, 0)
\end{array}
$$

2.3 Inflationary Fixed-point Logic

[2] proposes another extension of positive inductive definitions with negation. With an arbitrary formula $F[\overline{X}, P]$ with negative occurrences of P allowed, the resulting T_P-like operator $\mathcal{T}_{F[\overline{X},P]}$ is not monotonic and may not have a least fixpoint. However, the operator $\mathcal{T}^i_{F[\overline{X},P]}(I) = I \cup \mathcal{T}_{F[\overline{X},P]}(I)$ is increasing (though not monotonic) and therefore a fixpoint can be constructed. This idea has been used in fixpoint logic with *inflationary semantics* [1].

Inflationary fixpoint logic is known to be expressive; however, it is not a natural formalisation of inductive definitions over a well-founded set, and therefore, this extension is not relevant in the context of this paper. For example, $\mathcal{D}^1_{even}$ constructing a formula $F_{even}[X, even]$ for $\mathcal{D}^1_{even}$ in the same way as for positive inductive definitions, we obtain: $X = 0 \vee \exists Y. X = s(Y) \wedge \neg even(Y)$.
After one application of the inflationary fixpoint operator, the unintended fixpoint $\{even(n) | n \in \mathbb{N}\}$.

3 A critique on Iterated Inductive Definitions

The stratified IID formalisms provide a correct treatment of inductive definitions with negation. The IID-forms as defined in e.g. [8] was not intended for use for Knowledge Representation and is absolutely unsuitable for such purpose. But any stratified formalism for inductive definitions with negation will pose certain fundamental problems.
(1) A stratification of a definition does not provide any information about the defined relations. This can be seen from the fact that choosing another stratification for a definition has no impact on its semantics; moreover, there exists ways to construct the semantics of an IID without recurring to a predefined syntactical stratification. It is undesirable that in IID's, a stratification must be chosen and this choice is explicitly reflected in the representation of the definition.

(II) The stratification of an Iterated Inductive Definition is based on a syntactical criterion. As a consequence, a rule set formulated for one alphabet may be stratifiable whereas the corresponding rule set in a *linguistic variant* of the alphabet may be non-stratifiable. The following variant of the definition $\mathcal{D}^1_{even}$ illustrates this. Assume that we use the alphabet: $\{even(n), successor(n, m) | n, m \in \mathbb{N}\}$ with a predicate representation of the concept of successor. In this alphabet, the natural representation of the inductive definition of *even* is the set with for each $n, m \in \mathbb{N}$ the following rules:

$$\mathcal{D}^4_{even} = \begin{cases} successor(n+1, n) \\ even(0) \\ even(n) \leftarrow successor(n,m), \neg even(m) \end{cases}$$

This variant definition cannot be stratified due to the presence of rules $even(m) \leftarrow successor(m,m), \neg even(m)$. It would be desirable that a formalisation would be less dependent of intuitively innocent linguistic variance.

(III) As a formalisation of inductive definitions on well-founded posets, the requirement of stratified IID's of an explicit stratification is problematic in general. A definition of a concept (like evenness or depth) for x in terms of all $y < x$ is mathematically well-constructed; yet a stratification for such a definition may be in general unknown. As an example, consider the inductive definition of *depth* of an element in a well-founded order (footnote 4, section 2). The need of an explicit stratification is unnecessary and unnatural.

3.1 WFS: An improved Principle of Inductive Definition

In this section, we argue that the mathematics of (a variant of) the well-founded semantics of logic programming [25] provides an improved formalisation of the principle of inductive definition.

Just like the perfect model, the model $\overline{M}^{\mathcal{D}}$ of a stratified Iterated Inductive Definition $\mathcal{D}$ is obtained by iterating the positive induction principle and constructing a sequence $(M_\alpha)_{\alpha<\alpha_{\mathcal{D}}}$ of interpretations of increasing sub-domains which starts with M and gives gradually better approximations of the model $\overline{M}^{\mathcal{D}}$. Each M_α defines the truth value of all symbols of the sub-alphabet Σ_α and leaves atoms defined at later levels undefined. The role of the stratification in this process is to *delay* the use of some part of the definition until enough information is available to safely apply the positive induction principle on that part of the definition.

The same ideas can be *implemented* in a different way, without relying on an explicit syntactical partitioning of the definition. Instead of using 2-valued interpretations of sub-alphabets, partial interpretations can be used. Here, a partial interpretation is a partial function from the set of atoms D to $\{\mathbf{t}, \mathbf{f}\}$. Equivalently, we use the classical formalisation as a total function from the set of atoms D to $\{\mathbf{t}, \mathbf{u}, \mathbf{f}\}$[7]. The positive induction principle can be conservatively extended for definitions with negation. For a definition $\mathcal{D}$, we define the Positive Induction Operator $\mathcal{PI}_{\mathcal{D}}$ which takes as input a partial interpretation I representing well-defined truth values for a subset of atoms, and derives an extended partial interpretation defining the truth values of other atoms that can be derived by positive induction. Definition of truth values of atoms for which not enough information is available is delayed. The model of a definition is obtained then by a fixpoint construction.

From a knowledge theoretic point of view, the key problem in the above enterprise is the definition of the principle of positive induction in the context of definitions with nega-

[7]This formalisation is mathematically equivalent with the previous one, is more common and leads to more elegant mathematics. Note that in this view, $\mathbf{u}$ plays a similar role as null-values in databases.

tion. A formalisation based on proof-trees shows most clearly the structural similarities between positive induction for PID's and for inductive definitions with negation.

We formalise the above ideas for a formalism which is the natural extension of the abstract definitions of [2] with negation; at the same time, it is an infinitary version of the propositional LP-formalism. Given is a domain D of propositional symbols. Now, a definition $\mathcal{D}$ consists of rules with negative literals in the bodies and defines symbols of $Defined(\mathcal{D}) = D \setminus D_o$. Given is an interpretation M of the non-defined symbols, called the interpreted symbols.

The definition of a $\mathcal{D}$-proof-tree $\mathcal{T}$ as defined in section 2.1 does not need to be altered: it is a tree of literals of D such that:

- no leaves contain defined atoms $p \in Defined(\mathcal{D})$;
- each non-leaf contains a defined atom p and has a set of direct descendants B such that $p \leftarrow B \in \mathcal{D}$;
- no infinite branches.

Hence, leaves contain interpreted literals and negations of defined atoms. Note that interpreted atoms have proof-trees consisting of one root node.

Definition 3.1 *The* Positive Induction Operator $\mathcal{PI}_\mathcal{D}$ *maps partial interpretations I to I' such that $\forall p \in D$:*

- *$I'(p) = \mathbf{t}$ if p has a proof-tree with all leaves true in I.*
- *$I'(p) = \mathbf{f}$ if each proof-tree of p has a false leaf in I;*
- *$I'(p) = \mathbf{u}$ otherwise, i.e. no proof-tree has only true leaves but there exists at least one without false leaves.*

The Positive Induction Operator is a monotonic operator w.r.t. the precision order $\leq_p$, the point-wise extension of $\mathbf{u} \leq_p, \mathbf{u} \leq_p \mathbf{t}$. Monotonic operators w.r.t. $\leq_p$ have a least fixpoint [9]. Hence, each interpretation M of the non-defined symbols can be extended to a fixpoint $\mathcal{PI}_\mathcal{D}\!\uparrow (M)$.

Definition 3.2 *$\mathcal{PI}_\mathcal{D}\!\uparrow (M)$ is the model $\overline{M}^\mathcal{D}$ of $\mathcal{D}$.*

The structural resemblance between positive induction in PID's and in $\mathcal{PI}_\mathcal{D}$ is apparent. There are some important properties. The first relates this semantics to WFS semantics of logic programs.

Proposition 3.1 *$\mathcal{PI}_\mathcal{D}$ and the 3-valued stable model operator [20] are identical. The well-founded model of $\mathcal{D}$ is the model $\overline{M}^\mathcal{D}$ of $\mathcal{D}$.*

Second, this semantics provides a conservative extension of the IID-style semantics, as the WFS is known to generalise least model semantics and perfect model semantics of stratified logic programs.

Third, certain definitions may have partial models (e.g. $\{p \leftarrow \neg p\}$. In the definition interpretation, $\overline{M}^\mathcal{D}(p) = \mathbf{u}$ means that the truth value of p is not defined and that $\mathcal{D}$ does not allow to constructively define the truth value of p. Hence, undefined atoms point to ambiguities in the definition. There seem to be two sensible treatments of ambiguous definitions. They could be considered as *inconsistent*, in a similar sense as in classical logic: ambiguous definitions have no models. The result is a 2-valued logic. A more permissive treatment is to allow definitions with partial models. The result would be a sort of *paraconsistent* definition logic, i.e. a logic in which definitions with local inconsistencies do not not entail every formula.

4 Implications of ID-view on LP

The reading of logic programs as autoepistemic or default theories on the one hand, and as definitions on the other hand, give essentially different perspectives on the meaning of logic programs, on the nature of the negation symbol and the implication symbol in LP. The different perspectives lead to different Knowledge Representation methodologies and lead to different language extensions for representing uncertainty.

Representing uncertainty is a well-known problem of logic programming. In the two views, the reason for this is different. In the AEL-view, the problem with uncertainty is that often closure axioms $\neg p \leftarrow \neg K(p)$ are implicitly or explicitly assumed. Without such closure axioms, a logic program represents uncertainty (see section 1) but then the problem is that no objective negative information can be represented. This led Gelfond and Lifschitz in [12] to re-introduce a form of classical negation in Extended Logic Programming.

In the definition view, the problem is because in LP all predicates must have definition. Hence, the natural idea is to extend the logic with open predicates which have arbitrary interpretation. In [6], this idea was elaborated in an extension of LP, called OLP-FOL. I argued there that OLP-FOL provides a knowledge theoretic interpretation of Abductive Logic Programming as a definition logic. A problem of this work is that it is based on completion semantics; completion is not a good formalisation of induction.

The knowledge theoretic interpretation of LP as inductive definitions gives also insight on the relationship with a class of logics outside the area of NMR: definition logics. This class includes fixpoint logics and terminological l and concept languages. In [22], Van Belleghem et al. showed a strong correspondence between OLP-FOL and terminological languages.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addusin-Wesley Publishing Company, 1995.

[2] P. Aczel. An Introduction to Inductive Definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland Publishing Company, 1977.

[3] K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, 1988.

[4] W. Buchholz, S. Feferman, and W. Pohlers W. Sieg. *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies.* Springer-Verlag, Lecture Notes in Mathematics 897, 1981.

[5] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.

[6] M. Denecker. A Terminological Interpretation of (Abductive) Logic Programming. In V.W. Marek, A. Nerode, and M. Truszczynski, editors, *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 15–29. Springer, Lecture notes in Artificial Intelligence 928, 1995.

[7] M. Denecker and D. De Schreye. Justification semantics: a unifying framework for the semantics of logic programs. Technical Report 157, Department of Computer Science, K.U.Leuven, 1992.

[8] S. Feferman. Formal theories for transfinite iterations of generalised inductive definitions and some subsystems of analysis. In A. Kino, J. Myhill, and R.E. Vesley, editors, *Intuitionism and Proof theory*, pages 303–326. North Holland, 1970.

[9] M. Fitting. A Kripke-Kleene Semantics for Logic Programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[10] M. Gelfond. On Stratified Autoepistemic Theories. In *Proc. of AAAI87*, pages 207–211. Morgan Kaufman, 1987.

[11] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the International Joint Conference and Symposium on Logic Programming*, pages 1070–1080. IEEE, 1988.

[12] M. Gelfond and V. Lifschitz. Logic Programs with Classical Negation. In D.H.D. Warren and P. Szeredi, editors, *Proc. of the 7th International Conference on Logic Programming 90*, page 579. MIT press, 1990.

[13] M. Hagiya and T. Sakurai. Foundation of Logic Programming Based on Inductive Definition. *New Generation Computing*, 2:59–77, 1984.

[14] G. Kreisel. Generalized inductive definitions. Technical report, Stanford University, 1963.

[15] P. Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J.e. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 179–216, 1971.

[16] Y. N. Moschovakis. *Elementary Induction on Abstract Structures.* North-Holland Publishing Company, Amsterdam- New York, 1974.

[17] E. Post. Formal reduction of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–215, 1943.

[18] H. Przymusinska and T.C. Przymusinski. Weakly perfect model semantics for logic programs. In R.A. Kowalski and K.A. Bowen, editors, *Proc. of the fifth international conference and symposium on logic programming*, pages 1106–1120. the MIT press, 1988.

[19] T.C. Przymusinski. On the semantics of Stratified Databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufman, 1988.

[20] T.C. Przymusinski. Well founded semantics coincides with three valued Stable Models. *Fundamenta Informaticae*, 13:445–463, 1990.

[21] C. Spector. Inductively defined sets of natural numbers. In *Infinitistic Methods (Proc. 1959 Symposium on Foundation of Mathematis in Warsaw)*, pages 97–102. Pergamon Press, Oxford, 1961.

[22] K. Van Belleghem, M. Denecker, and D. De Schreye. A strong correspondence between concept languages and open logic programming. Draft, K.U.Leuven, 1995.

[23] K. Van Belleghem, M. Denecker, and D. Theseider Dupré. Dependencies and ramifications in an event-based language. In *Proc. of the Ninth Dutch Artificial Intelligence Conference, 1997*, 1997.

[24] M. van Emden and R.A Kowalski. The semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 4(4):733–742, 1976.

[25] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.