

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 327544 — Fax: +32 16 327996
email: marcd@cs.kuleuven.ac.be

Abstract. Existing formalisations of (transfinite) 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 (wfs) overcomes these problems and hence, provides a superior formalisation of the principle of inductive definition. The contribution of this study for LP is that it (re-)introduces the knowledge theoretic interpretation of LP as a logic for representing definitional knowledge. I point to fundamental differences between this knowledge theoretic interpretation of LP and the more commonly known interpretations of LP as default theories or auto-epistemic theories. The relevance is that differences in knowledge theoretic interpretation have strong impact on knowledge representation methodology and on extensions of the LP formalism, for example for representing uncertainty.

Keywords: Inductive Definitions, Logic Programming.

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 auto-epistemic logic. In [11], Gelfond proposes a semantics for stratified logic programs based on an auto-epistemic interpretation of the formalism. In [12], Gelfond and Lifschitz motivate the stable semantics for logic programs from the perspective of logic programs as default and auto-epistemic 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, 21] / stable [12] / wfs [28]) agree; for the above example, the unique model is $\{dead\}$.

In the interpretation of this program as an auto-epistemic theory, P_0 corresponds to the auto-epistemic theory (AEL):

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

which reads as: *one is dead if it is not believed that 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 auto-epistemic 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 auto-epistemic reading map negation to a modal operator ($\neg K$) and preserves the implication.

How to explain that, despite this intuitive difference, the stable model -which formalises the default/auto-epistemic reading- corresponds to the model of the completion? The reason is that models in stable semantics and in classical logic play a different role. 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. Because models in both semantics play a different role, a simple comparison between them does not reveal the different meanings of both semantics.

Actually, a clear and correct model theoretic comparison of the meaning of the auto-epistemic reading and of the completion is possible if done on the basis of the *possible world model* of the auto-epistemic theory and of the set of models of the completion. Both are sets of models; in both sets the role of models is identical: they represent possible states of the world. Such a comparison confirms the intuitive differences between the two readings. 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 $AEL(P_0)$: *alive* can be true or false, hence nothing is known on *alive*; (therefore) *dead* is always true. Note that the belief set, i.e. the stable model, is the intersection of these possible states. In contrast, the set of models of the completion is the singleton $\{\{dead\}\}$. Interpreted as a possible world model, it represents that *dead* is known to be true, *alive* known to be 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. One example is Reiter's formalisation of situation calculus [23]; in this approach, a situation calculus can be understood as an inductive definition on the well-founded poset of situations. Another example is in [26], where we argue that causality information in the context of the ramification problem is a form of constructive information. Causes, effects and forces propagate in a dynamic system through a constructive process; consequently, the semantics of causality rules is defined by an inductive definition which, by its constructive nature, mirrors the physical process of the effect propagation.

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 through 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:

$$\mathcal{D}_{trans} = \begin{cases} tr(X, Y) \leftarrow graph(X, Y) \\ tr(X, Y) \leftarrow graph(X, Z), tr(Z, Y) \end{cases}$$

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 semantics¹.

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 such a definition is the definition of even numbers in the well-founded poset \mathbf{IN}, \leq . One defines 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:

$$\mathcal{D}_{even}^1 = \begin{cases} even(0) \\ even(s(X)) \leftarrow \neg even(X) \end{cases}$$

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 models².

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

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

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 $y < x$.

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. $\{even(0), even(2), \dots\}$ but also $\{even(0), even(1), even(3), even(5), \dots\}$.

Formally, let $F[X, D]$ mean that D is a larger ordinal than the depths of all elements $Y < X$: $F[X, D] \equiv$

$$\forall Y, D_Y. (Y < X \wedge \text{depth}(Y, D_Y) \rightarrow D_Y < D)$$

Then, depth is represented by the singleton definition D_{depth} :

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

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 may exist³.

One application of this definition is the definition of depth of a tree. Here the well-founded poset is the set of trees (with values from a given domain D) without infinite branches in a domain; the partial order is the subtree relation. For finitely branching trees, the depth is always a natural number; for infinitely branching trees, the depth may be an infinite ordinal. E.g. the tree with branches $(0, 1, 2), (0, 2, 3, 4), (0, 3, 4, 5, 6), \dots, (0, n, \dots, 2n), \dots$ is a tree with depth ∞ .

The above two types of inductive definitions 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 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 [19], Spector [24] and Kreisel [15]. Important work in this area includes [9, 16, 18, 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 I 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 [9], 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).

³ E.g. in the context of \mathbb{N}, \leq , an unintended minimal model is $\{\text{depth}(0, 0), \text{depth}(0, 1), \text{depth}(1, 2), \dots, \text{depth}(n, n + 1), \dots\}$.

[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 [27].

- The model can be defined as the least model of the implications. E.g., in [9], 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).
- 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 symbol domain D , including a subset $D_o \subseteq D$ which includes the truth values t, f , an interpretation M interpreting the symbols of D_o such that $M(t) = t, M(f) = f$. The symbols of D_o are called the open or interpreted symbols. Also given is a definition \mathcal{D} which is a set of rules $p \leftarrow B$ with head $p \in D \setminus D_o$ and body B consisting of atoms of $D \setminus D_o$ and positive or negative literals of D_o ⁵. The set $Defined(\mathcal{D}) = D \setminus D_o$ is called the set of defined symbols, the set of open symbols D_o is often denoted $Open(\mathcal{D})$. 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 $p \leftarrow \{f\}$). Also the body B of a rule is never empty (B may be the singleton $\{t\}$).

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

- all leaves of T are positive or negative open literals; all non-leaves contain defined atoms;
- for each non-leaf node p with set of immediate descendants $B: p \leftarrow B \in \mathcal{D}$;
- T is loop-free; i.e. contains no infinite branches.

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

⁵ Allowing positive or negative open literals is an extension to the formalism of [2]. It does not introduce any complexity because the interpretation of these literals is given. This extension will facilitate the leap to inductive definitions with recursion over negation.

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 a 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 [15] and later studied in [9] and [16]. [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 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}_{α_p} ; we call α_p the stratum of p ;
- 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, 20, 21]. Already in 84, [14] defines a semantics for stratified logic programs based on the Iterated Inductive Definition (IID) logic defined in [16]. 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 [16], it is based on sequent calculus, which to some extent 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 [9], [4] and also in [16]. 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. [9] investigates IID's encoded in an IID-form, a single FOL formula of the form $F[N, X, P]$, and expresses its semantics in a circumscription-like second order formula. The problem is that this encoding is extremely tedious and this blurs the simple intuitions behind this work and the similarities with the perfect model semantics.

Nevertheless, it is interesting -if only from historical perspective- to see how transfinite definitions can be encoded finitely as an IID-form and how a perfect model-like semantics can be expressed in such a notation. Consider the following definition \mathcal{D}_{even}^2 , constructed for the sole purpose of illustrating the encoding:

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

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 $\{ok, even(2n) | n \in \mathbb{N}\}$.

To encode such an abstract IID, a binary meta-predicate h (of *holds*) is used: $h(\alpha, p)$ means that the stratum of p is α and that p is defined true. The first step in encoding such an abstract IID (\mathcal{D}_α) yields a possibly infinite disjunction $F[N, X, P]$. 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:

$$N = \alpha_p \wedge X = p \wedge .. \wedge P(q) \wedge h(\alpha_r, r) \wedge \neg h(\alpha_s, s) \wedge ..$$

This disjunct is obtained as a conjunction of $N = \alpha_p \wedge X = p$, corresponding to the head p , a conjunct $P(q)$ for any atom q of the same stratum as the head, and a literal $h(\alpha_r, r)$ and $\neg h(\alpha_s, s)$ for the other literals $r, \neg s \in B$ defined in lower strata⁶.

The result is an infinitary formula $F[N, X, P]$. Here, N ranges over the ordinals $\alpha < \alpha_{\mathcal{D}}$, X over atoms and P over sets of atoms. The formula corresponding

⁶ In [9], literals $h(\alpha_q, q)$ are replaced by open formulas $h(\alpha_q, q) \wedge \alpha_q < N$. This open formula represents the restriction of h to strata $< N$ (atoms q at higher strata are false in $h(\alpha_q, q) \wedge \alpha_q < N$). 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 h predicate even if $F[N, X, P]$ 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 h -predicate due to mutual dependencies between predicates defined at lower and at higher level.

to \mathcal{D}_{even}^2 is the following infinitary disjunction with disjuncts for each $0 \leq n$:

$$\left\{ \begin{array}{l} N = 0 \wedge X = even(0) \vee \\ N = n + 1 \wedge X = even(n + 1) \wedge \neg h(n, even(n)) \vee \dots \\ N = n \wedge X = even(n) \wedge P(even(n)) \vee \dots \\ N = \infty \wedge X = sw \wedge h(n, even(n)) \wedge \\ \quad h(n + 1, even(n + 1)) \quad \vee \dots \\ N = \infty + 1 \wedge X = ok \wedge \neg h(\infty, p) \end{array} \right.$$

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. Two axioms express essentially that at each stratum α , the set $h(\alpha, \cdot) \equiv \{p | h(\alpha, p) \text{ is true}\}$ satisfies the definition \mathcal{D}_α . These axioms express the principle of positive inductive definition: that this set must satisfy the implications of \mathcal{D}_α and that it must be contained in each set satisfying the implications. Below, $F[P(\tau)/h(N, \tau)]$ denotes the formulas obtained by replacing each expression $P(\tau)$ for arbitrary term τ by $h(N, \tau)$.

The first axiom expresses that for each ordinal α and given h for lower strata, $h(\alpha, \cdot)$ satisfies the implications in \mathcal{D}_α :

$$\forall N, X. \{h(N, X) \leftarrow F[P(\tau)/h(N, \tau)]\}$$

One can verify that if one assigns the values α_p to N and p to X and eliminates false disjuncts, then this complex formula reduces to:

$$h(\alpha_p, p) \leftarrow \left\{ \begin{array}{l} \dots \\ \dots \wedge h(\alpha_q, q) \wedge \neg h(\alpha_r, r) \wedge \dots \\ \dots \end{array} \right. \vee$$

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

The second axiom expresses that for each ordinal α , $h(\alpha, \cdot)$ is contained in each set Ψ which satisfies the implications of \mathcal{D}_α . It is a second order axiom, using a set variable Ψ which ranges over sets of atoms and it is a variant of a circumscription axiom:

$$\forall N. \forall \Psi. [\forall X. \Psi(X) \leftarrow F[P(\tau)/\Psi(\tau)]] \rightarrow [\forall X. h(N, X) \rightarrow \Psi(X)]$$

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 $2 \preceq 3 \preceq \dots \preceq 0 \preceq 1$ is a well-ordering encoding the ordinals $0, 1, \dots, \infty, \infty + 1$.
- 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:

$$\begin{aligned}
& \{N = 0 \wedge X = \text{even}(0)\} \\
& \quad \longrightarrow N = 2 \wedge X = 2 \\
& \{N = n + 1 \wedge X = \text{even}(n + 1) \wedge \neg h(n, \text{even}(n)) \mid n \in \mathbb{N}\} \\
& \quad \longrightarrow \exists M. N = M + 1 \wedge 2 \leq M \wedge X = N \wedge \neg h(M, M) \\
& \{N = n \wedge X = \text{even}(n) \wedge P(\text{even}(n)) \mid n \in \mathbb{N}\} \\
& \quad \longrightarrow 2 \leq N \wedge X = N \wedge P(N) \\
& \{N = \infty \wedge X = \text{sw} \wedge h(n, \text{even}(n)) \wedge h(n + 1, \text{even}(n + 1)) \mid n \in \mathbb{N}\} \\
& \quad \longrightarrow N = 0 \wedge X = 0 \wedge \exists M. [2 \leq M \wedge h(M, M) \wedge h(M + 1, M + 1)] \\
& \{N = \infty + 1 \wedge X = \text{ok} \wedge \neg h(\infty, p)\} \\
& \quad \longrightarrow N = 1 \wedge X = 1 \wedge \neg h(0, 0)
\end{aligned}$$

The resulting finite IID-form is:

$$\left\{ \begin{array}{l} N = 2 \wedge X = 2 \vee \\ \exists M. [N = M + 1 \wedge 2 \leq M \wedge X = N \wedge \neg h(M, M)] \vee \\ 2 \leq N \wedge X = N \wedge P(N) \vee \\ N = 0 \wedge X = 0 \wedge \exists M. [2 \leq M \wedge h(M, M) \wedge h(M + 1, M + 1)] \vee \\ N = 1 \wedge X = 1 \wedge \neg h(0, 0) \end{array} \right.$$

2.3 Inflationary Fixed-point Logic

[2] proposes another extension of positive inductive definitions with negation. With an arbitrary formula $F[\bar{X}, P]$ with negative occurrences of P allowed, the resulting T_P -like operator $T_{F[\bar{X}, P]}$ is not monotonic and may not have a least fixpoint. However, the operator $T_{F[\bar{X}, P]}^i(I) = I \cup T_{F[\bar{X}, P]}(I)$ 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, if we construct a formula $F_{\text{even}}[X, \text{even}]$ for $\mathcal{D}_{\text{even}}^1$ in the same way as for positive inductive definitions, we obtain: $X = 0 \vee \exists Y. X = s(Y) \wedge \neg \text{even}(Y)$. After one application of the inflationary fixpoint operator, the unintended fixpoint $\{\text{even}(n) \mid n \in \mathbb{N}\}$ is obtained.

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. [9] 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}_{even}^1 illustrates this. Assume that we use the alphabet: $\{even(n), successor(n, m) | n, m \in \mathbf{IN}\}$ 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 $n, m \in \mathbf{IN}$ the following rules:

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

This variant definition cannot be stratified due to the presence of rules $even(m) \leftarrow successor(m, m), \neg even(m)$. A good formalisation should not be as 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 or the depth of a tree. The need of an explicit stratification is unnecessary and unnatural.

3.1 WFS: An improved Principle of Inductive Definition

In this section, I argue that the mathematics of (a variant of) the well-founded semantics of logic programming [28] 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 $\{t, f\}$. Equivalently, we use the classical formalisation as a total function from the set of atoms D to $\{t, u, f\}$ 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 negation. 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. In the new, more general setting, a definition \mathcal{D} consists of rules in which positive and negative open or defined literals may appear in the (nonempty) body. As before, the set of defined symbols that appear in the head of a rule is denoted $Defined(\mathcal{D})$; the set of open or interpreted symbols is denoted as $Open(\mathcal{D})$. Also given is an interpretation M of the open symbols $Open(\mathcal{D})$.

The definition of a \mathcal{D} -proof-tree T as defined in section 2.1 hardly needs to be altered: it is a tree of literals of D such that:

- leaves contain open literals or negative defined literals; non-leaves contain defined atoms $p \in Defined(\mathcal{D})$;
- each non-leaf p 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 1. *The Positive Induction Operator $\mathcal{PI}_{\mathcal{D}}$ maps partial interpretations I to I' such that $\forall p \in D$:*

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

⁷ This formalisation is mathematically equivalent with the previous one, is more common and leads to more elegant mathematics. Note that in this view, u plays a similar role as null-values in databases: just as a null value, u is not a real truth value, it is a place holder for an (as yet) undefined truth value. Below, I return to the issue of interpretation of u .

The Positive Induction Operator is a monotonic operator w.r.t. the precision order \leq_p , the point-wise extension of $\mathbf{u} \leq_p \mathbf{f}$, $\mathbf{u} \leq_p \mathbf{t}$. Monotonic operators w.r.t. \leq_p have a least fixpoint [10]. Hence, each interpretation M of the non-defined symbols can be extended to a unique least fixpoint $\mathcal{PI}_{\mathcal{D}}\uparrow(M)$.

Definition 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 1. $\mathcal{PI}_{\mathcal{D}}$ and the 3-valued stable model operator [22] 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. $\{p \leftarrow \neg p\}$). Note here the changing role of \mathbf{u} during the fixpoint computation and in the fixpoint. When the truth value of an atom is \mathbf{u} at some stage of the fixpoint computation, it means that the truth value of the atom is yet undetermined at this stage. If its truth value is still \mathbf{u} in the fixpoint, it means \mathcal{D} does not allow to constructively define the truth value of p . Hence, undefined atoms in the fixpoint 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. In this strict view, Definition 2 is to be refined as:

Definition 3. If $\mathcal{PI}_{\mathcal{D}}\uparrow(M)$ is 2-valued, then it is the model $\overline{M}^{\mathcal{D}}$ of \mathcal{D} ; otherwise, \mathcal{D} has no model.

The result is a 2-valued logic. This is a simple strategy because it avoids potential problems with 3-valued models but it has the disadvantage that no sensible information can be extracted from an ambiguous definition since such a definition entails every formula. This situation is analogous to classical logic.

The more permissive treatment is to allow definitions with partial models. The result is a sort of *paraconsistent* definition logic, i.e. a logic in which definitions with local inconsistencies or local ambiguities do not not entail every formula.

4 Concluding remarks

This paper is a study of the concept of (transfinite) inductive definition. The paper investigates how this concept has been formalised in the past in the ID and IID areas; drawbacks of these formalisations were pointed at and an improved

formalisation, inspired by logic programming semantics, is proposed. Strong connections between the formalisations in ID and IID and perfect model semantics but also circumscription semantics have been exposed.

This study is not only relevant as a study of inductive definitions but improves also our understanding of the use of LP for knowledge representation and hence, of the role of LP in Artificial Intelligence. The reading of logic programs as auto-epistemic 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.

In general, a knowledge theoretic study as the one in this paper is relevant for developing a knowledge representation methodology. It is (or once was) a widespread view that the advantage of declarative logic for “encoding” knowledge is in its intuitive *linguistic* reading; in the case of this paper: the reading of a set of rules as an inductive definition. This reading of the logic provides the methodological basis for knowledge representation; the tight connection between formal syntax and semantics and a clear intuitive reading facilitates the explicitation of the expert knowledge. Formulas of the theory can be understood by the experts through the linguistic interpretation, without the need of explicitly constructing the formal semantics. Knowledge theoretic studies like the one in this paper, are important to build natural and systematic methodologies for knowledge representation. One aim of this study was to clarify how logic programs can be used for knowledge representation and what sort of knowledge can be represented in it.

A simple illustration of the impact of the linguistic interpretation on knowledge methodology is as follows. The definition that dead means not alive, is naturally expressed in LP under the definition reading by the singleton definition:

$$\{dead \leftarrow \neg alive\}$$

On the other hand, in Extended Logic Programming [13], which is based on the default and AEL view, a correct representation would be:

$$\begin{aligned} dead &\leftarrow \neg alive \\ \neg dead &\leftarrow alive \end{aligned}$$

A knowledge theoretic study is also relevant for the design or extension of a logic. This is also well-illustrated in the case of LP. With respect to knowledge representation, a major problem of LP under the default or auto-epistemic view is that no definite negative information can be represented. This led Gelfond and Lifschitz in [13] to extend the formalism and re-introduce a form of classical negation in Extended Logic Programming.

In the definition view, a logic program entails plenty of definite negative information. As a matter of fact, the problem with standard LP is the strength of its closure mechanism: an atom is assumed false unless it can be proven to be true. As a consequence, representing uncertainty is a serious problem; this problem has received a lot of attention in recent years. In the definition view on standard

LP, the problem is because all predicates are defined, have a (possibly empty) 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 Open Logic Programming (OLP). I argued there that OLP provides a knowledge theoretic interpretation of Abductive Logic Programming as a definition logic and that abductive solvers (e.g. SLDNFA [8]) designed for this formalism can be seen as special purpose reasoners on definitions for abduction and deduction⁸. A problem of this work is that it is based on completion semantics; completion is not a good formalisation of induction. To extend this study for the semantics defined in this paper is future work.

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 description logics. In [25], Van Belleghem et al. showed a strong correspondence between OLP-FOL and description logics. To large extend, description logic can be considered as a non-recursive subformalism of OLP-FOL. There is correspondence on the intuitive and semantical level; the differences on the syntactic level are syntactic sugar. The specific syntactic restrictions of description logics have allowed to develop highly efficient reasoning techniques.

Also subject for future work is to substantiate the claim in the introduction, that 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 the sense that (part of) the knowledge is present in the form of a recursive recipe, to be interpreted as defined in this paper. The prominent roles of completion and circumscriptive techniques in NMR and knowledge representation hint at this.

5 Acknowledgements

I thank all colleagues that have commented on this or earlier versions of this paper, in particular Danny De Schreye and Kristof Van Belleghem.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-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.

⁸ Note that LP and OLP as definition logics do not provide default negation; as I argued in [6], OLP is not a natural formalism to express some sorts of default reasoning problems such as the well-known train crossing example [13] [17]. In order to represent this sort of domains, an autoepistemic modal operator or a default negation operator should be added to definition logic.

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. M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1997.
9. 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.
10. M. Fitting. A Kripke-Kleene Semantics for Logic Programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
11. M. Gelfond. On Stratified Autoepistemic Theories. In *Proc. of AAAI87*, pages 207–211. Morgan Kaufman, 1987.
12. 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.
13. 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.
14. M. Hagiya and T. Sakurai. Foundation of Logic Programming Based on Inductive Definition. *New Generation Computing*, 2:59–77, 1984.
15. G. Kreisel. Generalized inductive definitions. Technical report, Stanford University, 1963.
16. 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.
17. J. McCarthy. Applications of Circumscription to Formalizing Common-Sense Knowledge. *Artificial Intelligence*, 28:89–116, 1980.
18. Y. N. Moschovakis. *Elementary Induction on Abstract Structures*. North-Holland Publishing Company, Amsterdam- New York, 1974.
19. E. Post. Formal reduction of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–215, 1943.
20. 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.
21. T.C. Przymusinski. On the semantics of Stratified Databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufman, 1988.
22. T.C. Przymusinski. Well founded semantics coincides with three valued Stable Models. *Fundamenta Informaticae*, 13:445–463, 1990.

23. R. Reiter. The Frame Problem in the Situation Calculus: A simple Solution (Sometimes) and a Completeness Result for Goal Regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honour of John McCarthy*, pages 359–380. Academic Press, 1991.
24. C. Spector. Inductively defined sets of natural numbers. In *Infinitistic Methods (Proc. 1959 Symposium on Foundation of Mathematics in Warsaw)*, pages 97–102. Pergamon Press, Oxford, 1961.
25. K. Van Belleghem, M. Denecker, and D. De Schreye. A strong correspondence between description logics and open logic programming. In Lee Naish, editor, *Proc. of the International Conference on Logic Programming, 1997*, pages 346–360. MIT-press, 1997.
26. 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.
27. 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.
28. 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.