An Algebraic Account of Modularity in ID-logic

Joost Vennekens and Marc Denecker*

{joost.vennekens, marc.denecker}@cs.kuleuven.ac.be
Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A
B-3001 Leuven, Belgium

Abstract. ID-logic uses ideas from the field of logic programming to extend second order logic with non-monotone inductive defintions. In this work, we reformulate the semantics of this logic in terms of approximation theory, an algebraic theory which generalizes the semantics of several non-monotonic reasoning formalisms. This allows us to apply certain abstract modularity theorems, developed within the framework of approximation theory, to ID-logic. As such, we are able to offer elegant and simple proofs of generalizations of known theorems, as well as some new results.

1 Introduction

Inductive definitions are common in mathematical practice. For instance, the non-monotone inductive definition of the satisfaction relation $\models$ (see Definition 1 in Section 2.2) can be found in most textbooks on first-order logic. This prevalence of inductive definitions indicates that these offer a natural and well-understood way of representing knowledge. At the same time, inductive definitions cannot easily be expressed in classical logic. For instance, the transitive closure of a graph is one of the simplest concepts typically defined by induction—such a definition might consist of the following two rules: if (x, y) is an edge of the graph, (x, y) belongs to the transitive closure and if $\exists z$ such that both (x, z) and (z, y) belong to the transitive closure, then (x, y) belongs to the transitive closure—yet it can be shown that this concept cannot be defined in first-order logic. While second-order logic does allow the representation of such simple definitions, the resulting formula might not always be very natural and the use of second-order logic itself may be undesirable, e.g., due to computational considerations. Moreover, even this methodology breaks down when faced with non-monotone inductive definitions, such as that of the satisfaction relation.

It turns out, however, that certain knowledge representation logics do allow even non-monotone inductive definitions to be correctly formalized in an intuitive way. Particularly suited for this are logic programs under the well-founded model semantics. In fact, one could even go so far as to explain the semantical foundations of this logic themselves as precisely a formalization of the principle of inductive definition [Den01]. The language of *ID-logic* uses the well-founded semantics to extend classical logic

* This work is supported by FWO-Vlaanderen, European Framework 5 Project WASP, and by GOA/2003/08.

with a new "inductive definition" primitive. In the resulting formalism, all kinds of definitions regularly found in mathematical practice—e.g., monotone inductive definitions, non-monotone inductive definitions over a well-ordered set, and iterated inductive definitions—can be represented in a uniform way. Moreover, this representation neatly corresponds to the form such a definitions would take in a mathematical text. For instance, in ID-logic the transitive closure of a graph can be defined as:

$$\left\{ \begin{array}{l} \forall x, y\, TransCl(x, y) \leftarrow Edge(x, y). \\ \forall x, y\, TransCl(x, y) \leftarrow (\exists z\, TransCl(x, z) \wedge TransCl(z, y)). \end{array} \right\}$$

However, ID-logic is able to handle more than only mathematical concepts. Indeed, inductive definitions are also useful in common-sense reasoning. For instance, in [DT04a], it was shown that situation calculus can be given a natural representation as an iterated inductive definition. The resulting theory is able to correctly handle tricky issues such as recursive ramifications, and is in fact, to the best of our knowledge, the most general representation of this calculus to date. In general, definitions are a distinctive and important form of human expert knowledge; as a uniform and natural way of representing this kind of knowledge, ID-logic provides a useful contribution to the field of knowledge representation.

The goal of this paper is to study modularity properties for ID-logic. Modularity properties deal with the relation between a theory and its components. Typical examples are so-called splitting results, which allow large theories to be rewritten as equivalent sets of sub-theories. Such properties are of interest, because they may offer additional insight into the semantics of a formalism, can be used to guarantee that certain transformations are equivalence preserving, or may lead allow more efficient computations.

Modularity properties have been studied for a large number of different formalisms. Recently, an algebraic theory of modularity [VGD04b,VGD04a] was developed within the framework of *approximation theory*, a general fixpoint theory for arbitrary operators, which naturally captures the semantics of logic programs, auto-epistemic logic, and default logic [DMT03,DMT00]. These abstract results have since been used to unify several concrete splitting theorems: [VGD04a] generalizes results concerning autoepistemic logic [GP92], and [VGD04b] (partially) generalizes results for logic programming [LT94].

Here, we apply this algebraic modularity theory to ID-logic. First, we show how the semantics of this logic can be reformulated in terms of approximation theory. By doing so, we are able to apply the aforementioned splitting theorems (and a small extension thereof) to ID-logic and obtain a generalization of results from [DT04b], as well as some new results.

The structure of this paper is as follows. Section 2 introduces approximation theory and ID-logic. Section 3 summarizes the algebraic modularity results which will be used. In Section 4, we then apply those results to ID-logic.

2 Preliminaries

2.1 Approximation theory

Approximation theory is a general fixpoint theory for arbitrary operators. Our presentation of this theory is based on [DMT00,DMT03].

Let $\langle L, \leq \rangle$ be a lattice. An element (x, y) of the square L^2 of the domain of such a lattice, can be seen as denoting an interval $[x, y] = \{z \in L \mid x \leq z \leq y\}$. Using this intuition, we can derive a *precision* order $\leq_p$ on the set L^2 from the order $\leq$ on L: for each $x, y, x', y' \in L, (x, y) \leq_p (x', y')$ iff $x \leq x'$ and $y' \leq y$. Indeed, if $(x, y) \leq_p (x', y')$, then $[x, y] \supseteq [x', y']$. It can easily be shown that $\langle L^2, \leq_p \rangle$ is also a lattice, which is called the *bilattice* corresponding to L. Moreover, if L is complete, then so is L^2. As an interval $[x, x]$ contains precisely one element, namely x itself, elements (x, x) of L^2 are called *exact*. The set of all exact elements of L^2 forms a natural embedding of L in L^2. A pair (x, y) only corresponds to a non-empty interval if $x \leq y$. Such pairs are called *consistent*.

Approximation theory is based on the study of operators on bilattices L^2 which are monotone w.r.t. the precision order $\leq_p$. Such operators are called *approximations*. For an approximation A and $x, y \in L$, we denote by $A^1(x, y)$ and $A^2(x, y)$ the unique elements of L, for which $A(x, y) = (A^1(x, y), A^2(x, y))$. An approximation *approximates* an operator O on L if for each $x \in L$, $A(x, x)$ contains $O(x)$, i.e. $A^1(x, x) \leq O(x) \leq A^2(x, x)$. An approximation is *symmetric* if for each pair $(x, y) \in L^2$, if $A(x, y) = (x', y')$ then $A(y, x) = (y', x')$.

For an approximation A on L^2, the following two operators on L can be defined: the function $A^1(\cdot, y)$ maps an element $x \in L$ to $A^1(x, y)$, i.e. $A^1(\cdot, y) = \lambda x.A^1(x, y)$, and the function $A^2(x, \cdot)$ maps an element $y \in L$ to $A^2(x, y)$, i.e. $A^2(x, \cdot) = \lambda y.A^2(x, y)$. As all such operators are monotone, they all have a unique least fixpoint. We define an operator $C_A^{\downarrow}$ on L, which maps each $y \in L$ to $lfp(A^1(\cdot, y))$ and, similarly, an operator $C_A^{\uparrow}$, which maps each $x \in L$ to $lfp(A^2(x, \cdot))$. $C_A^{\downarrow}$ is called the *lower stable operator* of A, while $C_A^{\uparrow}$ is the *upper stable operator* of A. Both these operators are anti-monotone. Combining these two operators, the operator $\mathcal{C}_A$ on L^2 maps each pair (x, y) to $(C_A^{\downarrow}(y), C_A^{\uparrow}(x))$. This operator is called the *partial stable operator* of A. Because the lower and upper partial stable operators $C_A^{\downarrow}$ and $C_A^{\uparrow}$ are anti-monotone, the partial stable operator $\mathcal{C}_A$ is monotone. If an approximation A is symmetric, its lower and upper partial stable operators will always be equal, i.e. $C_A^{\downarrow} = C_A^{\uparrow}$.

An approximation A defines a number of different fixpoints: the least fixpoint of an approximation A is called its *Kripke-Kleene fixpoint*, fixpoints of its partial stable operator $\mathcal{C}_A$ are *stable fixpoints* and the least fixpoint of $\mathcal{C}_A$ is called the *well-founded fixpoint* of A. As shown in [DMT00,DMT03], these fixpoints correspond to various semantics of logic programming, auto-epistemic logic and default logic.

Finally, it should also be noted that the concept of an approximation as defined in [DMT00] corresponds to our definition of a *symmetric* approximation.

2.2 ID-Logic

ID-logic [DT04b,DT04a] extends second-order logic with non-monotone inductive definitions. Before defining this logic in its entirity, we first introduce basic second order logic. Following [DT04a], we do this in a slightly non-standard way. In particular, no distinction is made between constant symbols and variables.

We assume an infinite supply of *object symbols* $x, y, \ldots$, *function symbols* $f/n, g/n, \ldots$ of every arity n, and *predicate symbols* $P/n, Q/n, \ldots$ of every arity n. A *vocabulary* Σ is a set of symbols. We denote by Σ_o the object symbols in Σ, by Σ_f the function symbols, and by Σ_P the predicate symbols. *Terms* and *atoms* of Σ are defined in the usual way. A *formula of* Σ is inductively defined as:

- a Σ-atom $P(t_1, \ldots, t_n)$ is a Σ-formula;
- if ϕ is a Σ-formula, then so is $\neg\phi$;
- if ϕ_1 and ϕ_2 are Σ-formulas, then so is $(\phi_1 \vee \phi_2)$;
- if ϕ is a $(\Sigma \cup \{\sigma\})$-formula and σ an (object, function or predicate) symbol, then $(\exists\sigma\ \phi)$ is a Σ-formula.

If in all quantifications $\exists\sigma$ of a formula ϕ, σ is an object symbol, ϕ is called *first order*.

Given a certain domain D, a symbol σ can be assigned a *value* in D:

- if $\sigma \in \Sigma_o$, a value for σ in D is an element of D;
- if $\sigma/n \in \Sigma_f$, a value for σ in D is a function of arity n in D;
- if $\sigma/n \in \Sigma_P$, a value for σ in D is a relation of arity n in D.

A *structure* S *for vocabulary* Σ, or Σ*-structure* S, consists of a domain, denoted S_D, and a mapping from each symbol σ in Σ to a value σ^S in S_D for σ. A vocubalary Σ is a *sub-vocabulary* of Σ' iff $\Sigma \subseteq \Sigma'$. The *restriction* $S'|_\Sigma$ of a Σ'-structure S' to a sub-vocabulary Σ, is the Σ-structure S for which $S_D = S'_D$ and, for each symbol σ of Σ, $\sigma^S = \sigma^{S'}$. Under the same conditions, S' is called an *extension* of S to Σ'. The set of all structures extending S to Σ' is denoted by $\mathcal{S}^S_{\Sigma'}$. For each value a in S_D for a symbol σ, we denote by $S[\sigma/a]$ the extension S' of S to $\Sigma \cup \{\sigma\}$, such that $\sigma^{S'} = a$.We also extend this notation to tuples $\mathbf{x}$ and $\mathbf{a}$, and to pairs (X, Y) of Σ-structures sharing the same domain, i.e., $(X, Y)[\mathbf{x}/\mathbf{a}] = (X[\mathbf{x}/\mathbf{a}], Y[\mathbf{x}/\mathbf{a}])$.

The *value* of a Σ-term t in a Σ-structure S, also denoted t^S, is inductively defined as: $(f(t_1, \ldots, t_n))^S = f^S(t_1^S, \ldots, t_n^S)$, for a function symbol f and terms $t_1, \ldots, t_n$. We now define a satisfaction relation between structures and formulas:

Definition 1. *For a Σ-structure S and Σ-formula ϕ, the relation "S satisfies ϕ", denoted $S \models \phi$, is inductively defined as:*

- $S \models P(\mathbf{t})$ *iff* $\mathbf{t}^S \in P^S$*;*
- $S \models (\phi_1 \vee \phi_2)$ *iff* $S \models \phi_1$ *or* $S \models \phi_2$*;*
- $S \models \neg\phi$ *iff* $S \not\models \phi$*;*
- $S \models (\exists\sigma\ \phi)$ *iff there exists a value a for σ in the domain S_D, such that $S[\sigma/a] \models \phi$;*

A *pre-interpretation H for Σ* is a structure for the language $\Sigma_o \cup \Sigma_f$, i.e., one which interprets only the object and function symbols of Σ. A structure S extending H to Σ is called an *H-interpretation*. Clearly, H-interpretations can only differ in their assignment of relations (over the common domain S_H) to predicate symbols. Given a domain D, a *domain atom* is a pair $(P, \mathbf{a})$, with P/n a predicate of Σ and $\mathbf{a} \in D^n$. We also write such a pair as $P(\mathbf{a})$. The function At_H is defined as mapping an H-interpretation S to the set of all domain atoms $P(\mathbf{a})$ in H_D, for which $\mathbf{a} \in P^S$. At_H is a one-to-one correspondence between H-interpretations and sets of domain atoms for H_D. The set of all H-interpretations is a complete lattice w.r.t. to the truth order $\leq_t$, defined as: $S \leq_t S'$ iff $At_H(S) \subseteq At_H(S')$ (or, equivalently, for each predicate P, $P^S \subseteq P^{S'}$).

Next, we explain how this logic can be extended with inductive definitions. We do this using concepts from approximation theory. In this, our presentation differs from the more direct approach taken in [DT04a].

As a first step, we extend the notion of satisfation to pairs (X, Y) of structures.

Definition 2. *Let H be a pre-interpretation for Σ, X and Y H-interpretations, and ϕ a Σ-formula. The relation "(X, Y)* satisfies *ϕ", denoted $(X, Y) \models \phi$ is inductively defined by:*

- $(X, Y) \models P(\mathbf{t})$ *iff* $\mathbf{t}^H \in P^X$*;*
- $(X, Y) \models (\phi_1 \vee \phi_2)$ *iff* $(X, Y) \models \phi_1$ *or* $(X, Y) \models \phi_2$*;*
- $(X, Y) \models \neg\phi$ *iff* $(Y, X) \not\models \phi$*;*
- $(X, Y) \models (\exists\sigma\ \phi)$ *iff there exists a value a for σ in H_D, such that* $(X, Y)[\sigma/a] \models \phi$*;*

Observe that in the rule for $\neg\phi$, the roles of X and Y are switched. This causes all positively occurring atoms in ϕ to be evaluated in X, while all negatively occurring atoms in ϕ are evaluated in Y. To motivate this definition, let us consider a structure S approximated by (X, Y), i.e. such that $X \leq_t S \leq_t Y$. In the evaluation of ϕ in (X, Y), all positively occurring atoms are evaluated with respect to the underestimate X of S, and all negatively occurring atoms are evaluated with respect to the overestimate Y of S. Therefore, the truth value of ϕ in (X, Y) is an underestimate of the value of ϕ in S. Vice versa, in the evaluation of ϕ in (Y, X), all positively occurring atoms are evaluated in the overestimate Y while all negatively occurring atoms are evaluated in the underestimate X, and hence, the truth value of ϕ in (Y, X) is an overestimate of the value of ϕ in S.

Considering satisfaction in pairs of structures rather than single structures, corresponds to switching to a four-valued logic: ϕ is *true* according to (X, Y) if $(X, Y) \models \phi$ and $(Y, X) \models \phi$, *false* if $(X, Y) \not\models \phi$ and $(Y, X) \not\models \phi$, *unkown* if $(X, Y) \not\models \phi$ and $(Y, X) \models \phi$, and *inconsistent* if $(X, Y) \models \phi$ and $(Y, X) \not\models \phi$.

We now define the ID-logic syntax used for inductive definitions. Let Σ be a vocabulary. A *definitional rule* r of Σ is a formula $\forall \mathbf{x}\ A \leftarrow \phi$, with A a Σ-atom and ϕ a first-order $(\Sigma \cup \mathbf{x})$-formula. The atom A is called the *head, $head(r)$, of* r and ϕ is called the *body, $body(r)$, of* r. Note that the symbol "$\leftarrow$" in such a rule should not be read as material implication, but rather as a new language primitive: the *definitional implication*. A rule r is said to be a *defining rule* of a predicate P if P is the predicate of $head(r)$. A *Σ-definition* Δ is a set of definitional rules. A predicate symbol having

at least one defining rule r in Δ, is called a *defined predicate of* Δ. The set of all such predicates is denoted by $\mathcal{P}^d_\Delta$. Predicates of Σ_P which are not defined by Δ are *open in* Δ and the set of all such predicates is denoted by $\mathcal{P}^o_\Delta$. The notations Σ^o_Δ and Σ^d_Δ are used to denote the vocabularies $\Sigma_o \cup \Sigma_f \cup \mathcal{P}^o_\Delta$ and $\Sigma_o \cup \Sigma_f \cup \mathcal{P}^d_\Delta$, respectively.

Using this syntax, the well-known simultaneous inductive definition of the even and odd numbers (i.e., 0 is an even number, each successor of an even number is an odd number, and vice versa) can be written as:

Example 1.

$$\Delta_{even} = \left\{ \begin{array}{c} Even(0). \\ \forall x\ Even(s(x)) \leftarrow Odd(x). \\ \forall x\ Odd(s(x)) \leftarrow Even(x). \end{array} \right\}$$

Intuitively, such an inductive definition describes a process by which, given some fixed interpretation of the open predicates, new elements of the defined relations can be derived from a set of already known elements. The formal definition of the semantics of ID-logic captures this intuition, by associating a class of operators to a definition Δ. More precisely, for each interpretation O of the open predicates of Δ, an operator $\mathcal{T}^O_\Delta$ is defined, which maps an estimate (X, Y) of the defined relations to a more precise estimate $\mathcal{T}^O_\Delta(X, Y) = (X', Y')$. The new lower bound X' is constructed by underestimating the truth of the bodies of the rules in Δ, i.e., by evaluating these in (X, Y). When constructing the new upper bound Y', on the other hand, the truth of the bodies of these rules is overestimated, i.e., evaluated in (Y, X).

Definition 3. *Let Δ be a Σ-definition and O a Σ^o_Δ-structure. We define a function U^O_Δ from the bilattice $(\mathcal{S}^O_\Sigma)^2$ to $\mathcal{S}^O_\Sigma$ as $U^O_\Delta(X, Y) = S$, with for each $P \in \Sigma^d_\Delta$: $\mathbf{a} \in P^S$ iff there exists a rule $(\forall \mathbf{x}\ P(\mathbf{t}) \leftarrow \phi)$ in Δ and a value $\mathbf{c}$ for $\mathbf{x}$, such that $(X, Y)[\mathbf{x}/\mathbf{c}] \models \phi$ and $\mathbf{a} = \mathbf{t}^{S[\mathbf{x}/\mathbf{c}]}$. The operator $\mathcal{T}^O_\Delta$ on $(\mathcal{S}^O_\Sigma)^2$ is defined as, for all $X, Y \in \mathcal{S}^O_\Sigma$:*

$$\mathcal{T}^O_\Delta(X, Y) = (U^O_\Delta(X, Y), U^O_\Delta(Y, X)).$$

If an estimate (X, Y) is more precise than an estimate (X', Y'), i.e., $X' \leq_t X$ and $Y \leq_t Y'$, then $\mathcal{T}^O_\Delta(X, Y)$ will also be more precise than $\mathcal{T}^O_\Delta(X', Y')$. In other words, each operator $\mathcal{T}^O_\Delta$ is an approximation. As such, each $\mathcal{T}^O_\Delta$ has a well-founded fixpoint. We now use this to define the semantics of the logic.

Definition 4. *Let Σ be a vocabulary. An* ID-logic formula *is inductively defined by extending the definition of a formula with the additional base case:*

- *A definition Δ is an ID-logic formula.*

The corresponding base case for the satisfaction relation is:

- $S \models \Delta$ *iff* $X|_{\Sigma^d_\Delta} = S|_{\Sigma^d_\Delta} = Y|_{\Sigma^d_\Delta}$, *with* (X, Y) *the well-founded fixpoint of* $\mathcal{T}^O_\Delta$, *with* $O = S|_{\Sigma^o_\Delta}$.

Note that, even though this definition uses the operator $\mathcal{T}^S_\Delta$ on pairs of structures, the eventual models of a definition are always single structures S. The intuition here is that a definition should completely define its defined predicates, i.e., there should be no tuples for which it is "unknown" whether they belong to the defined relations or not.

Definition 5. *Let Σ be a vocabulary. A Σ-definition Δ is* total *in a Σ^o_Δ-structure O iff $X = Y$, with (X, Y) the well-founded fixpoint of $\mathcal{T}^O_\Delta$.*

3 Algebraic splitting results

In this section, we summarize and extend results from [VGD04b]. First, we introduce some basic definitions and notations. Let I be a set, which we call the *index set*, and for each $i \in I$, let S_i be a set. The *product set* $\bigotimes_{i\in I} S_i$ is the following set of functions:

$$\bigotimes_{i\in I} S_i = \{f \mid f : I \to \bigcup_{i\in I} S_i \text{ such that } \forall i \in I : f(i) \in S_i\}.$$

If, for instance, I is $\{1, \ldots, n\}$, the product $\bigotimes_{i\in I} S_i$ is (isomorphic to) the cartesian product $S_1 \times \cdots \times S_n$.

If each S_i is partially ordered by some $\leq_i$, this induces the *product order* $\leq_\otimes$ on $\otimes_{i\in I} S_i$: $\forall x, y \in \otimes_{i\in I} S_i$, $x \leq_\otimes y$ iff $\forall i \in I : x(i) \leq_i y(i)$. It can easily be shown that if all $\langle S_i, \leq_i\rangle$ are (complete) lattices, then $\langle \otimes_{i\in I} S_i, \leq_\otimes\rangle$ is also a (complete) lattice; this is the *product lattice* of the lattices S_i.

From now on, we only consider product lattices with a *well-founded* index set, i.e., index sets I with a partial order $\preceq$ such that each non-empty subset of I has a $\preceq$-minimal element. This allows us to use inductive arguments in dealing with elements of product lattices.

The following notations are used. Let L be a product lattice $\otimes_{i\in I} L_i$. For $x \in L$ and $i \in I$, we abbreviate the restriction $x|_{\{j\in I | j\preceq i\}}$ by $x|_{\preceq i}$. We also use similar abbreviations $x|_{\prec i}$, $x|_i$ and $x|_{\not\preceq i}$. If i is a minimal element of the well-founded set I, $x|_{\prec i}$ is defined as the empty function. For any subset J of I, the set $\{x|_J \mid x \in L\}$, ordered by the appropriate restriction $\leq_\otimes|_J$ of the product order, is also a lattice. This sublattice of L is of course equal to the product lattice $\otimes_{j\in J} L_j$. If J is of the form $\{j \in I \mid j \preceq i\}$ for some i, we simply write $L|_{\preceq i}$ for $L|_J$. Similarly, $L|_{\prec i}$ is written for $\otimes_{j\prec i} L_i$.

If f, g are functions $f : A \to B$, $g : C \to D$ and the domains A and C are disjoint, we denote by $f \sqcup g$ the function from $A \cup C$ to $B \cup D$, such that for all $a \in A$, $(f \sqcup g)(a) = f(a)$ and for all $c \in C$, $(f \sqcup g)(c) = g(c)$. Furthermore, for any g whose domain is disjoint from the domain of f, we call $f \sqcup g$ an *extension* of f. For each element x of a product lattice L and each index $i \in I$, the extension $x|_{\prec i} \sqcup x|_i$ of $x|_{\prec i}$ is clearly equal to $x|_{\preceq i}$. To ease notation, we sometimes write $x(i)$ instead of $x|_i$ in such expressions, i.e. we identify an element a of the ith lattice L_i with the function from $\{i\}$ to L_i which maps i to a. Similarly, $x|_{\prec i} \sqcup x(i) \sqcup x|_{\not\preceq i} = x$.

Our goal is now to study operators on product lattices. Let $\langle I, \preceq\rangle$ be a well-founded index set and let $L = \otimes_{i\in I} L_i$ be a product lattice. Intuitively, an operator O on L is stratifiable over $\preceq$, if the value $(O(x))(i)$ of $O(x)$ in the ith level only depends on values $x(j)$ for which $j \preceq i$.

Definition 6. *An operator O on a product lattice L is* stratifiable *iff $\forall x, y \in L, \forall i \in I$: if $x|_{\preceq i} = y|_{\preceq i}$ then $O(x)|_{\preceq i} = O(y)|_{\preceq i}$.*

It is possible to characterize stratifiablity in a more constructive manner. The following proposition shows that stratifiablity of an operator O on a product lattice L is

equivalent to the existence of a family of operators on each lattice L_i (one for each partial element u of $L|_{\prec i}$), which mimics the behaviour of O on this lattice.

Proposition 1. *Let O be an operator on a product lattice L. O is stratifiable iff for each $i \in I$ and $u \in L|_{\prec i}$ there exists a unique operator O_i^u on L_i, such that for all $x \in L$:*

$$\textit{If } x|_{\prec i} = u \textit{ then } (O(x))(i) = O_i^u(x(i)).$$

The operators O_i^u are called the *components* of O. The main results of [VGD04b] are the following correspondences between various kinds of fixpoints of the original operator O and those of its components O_i^u:

Theorem 1. *Let L be a product lattice $\otimes_{i \in I} L_i$.*

- *If O is a stratifiable operator on L, then for each $x \in L$: x is a fixpoint of O iff $\forall i \in I : x(i)$ is a fixpoint of $O_i^{x|_{\prec i}}$.*
- *If O is a monotone stratifiable operator on L, then for each $x \in L$: x is the least fixpoint of O iff $\forall i \in I : x(i)$ is the least fixpoint of $O_i^{x|_{\prec i}}$.*
- *If O is a stratifiable approximation on the bilattice L^2, then for each $x \in L^2$: x is a stable (well-founded) fixpoint of O iff $\forall i \in I : x(i)$ is a stable (well-founded, respectively) fixpoint of $O_i^{x|_{\prec i}}$.*

This theorem allows us to incrementally construct any kind of fixpoint of a stratifiable operator, by constructing the corresponding fixpoints of its components in a bottom-up manner w.r.t. the well-founded order $\preceq$ on the index set.

We now extend this material from [VGD04b] with some additional results. More specifically, we not only want to split a stratifiable operators into its components, but also into sets of "bigger" operators, i.e., operators which may encompass several levels. For a subset J of I and $x \in L|_{I \setminus J}$, we denote by O_J^x the operator on $L|_J$ which maps each $y \in L|_J$ to $O(x \sqcup y)|_J$. Such operators O_J^x are called *recombinations of O*. Our goal is now to show that, for each partition $\mathcal{J}$ of I, a stratifiable operator O can be split into the recombinations O_J^x, with $J \in \mathcal{J}$. We do this, by showing that a recombination O_J^x is also stratifiable and can be split into the components of O itself.

Proposition 2. *Let O be a stratifiable operator. For each $J \subseteq I$ and $x \in L|_{I \setminus J}$, O_J^x is stratifiable.*

Proof. Let O_J^x be as above, $i \in J$, and $y, y' \in L|_J$, such that $y|_{\preceq i} = y'|_{\preceq i}$. By definition, $O_J^x(y) = O(x \sqcup y)|_J$. Because $(x \sqcup y)|_{\preceq i} = (x \sqcup y')|_{\preceq i}$, we have that, by stratifiability of O, $O_J^x(y)|_{\preceq i} = O(x \sqcup y)|_{\{j \in J | j \preceq i\}} = O(x \sqcup y')|_{\{j \in J | j \preceq i\}} = O_J^x(y')|_{\preceq i}$.

Proposition 3. *Let O be a stratifiable operator. For each $J \subseteq I$, $x \in L|_{I \setminus J}$, $i \in J$, and $u \in L|_{\{j \in J | j \prec i\}}$, the component $(O_J^x)_i^u$ of O_J^x equals the component $O_i^{u \sqcup (x|_{\prec i})}$ of O.*

Proof. Let $(O_J^x)_i^u$ be as above and let $y \in L_i$. By definition, for any z extending $u \sqcup y$ to J, $(O_J^x)_i^u(y) = O(x \sqcup z)|_i = (O_i^{(x \sqcup z)|_{\prec i}}(z|_i))|_i = O_i^{x|_{\prec i} \sqcup u}(y)$.

These two propositions now imply the wanted result.

Theorem 2. *Let O be a stratifiable operator and let $\mathcal{J}$ be a partition of I. Then, for each $x \in L$, x is a fixpoint (least fixpoint, stable fixpoint, or well-founded fixpoint) of O (assuming that O is monotone or an approximation, where appropriate) iff for each $J \in \mathcal{J}$, $x|_J$ is a fixpoint (least fixpoint, stable fixpoint, or well-founded fixpoint) of $O_J^{x|_{I \setminus J}}$.*

Proof. We only show the correspondence between fixpoints; the proofs of the other correspondences are similar. Let x be a fixpoint of O. By Theorem 1, this is equivalent to: $\forall i \in I$, $x|_i$ is a fixpoint of $O_i^{x|_{\prec i}}$. Because $\mathcal{J}$ partitions I, this is equivalent to $\forall J \in \mathcal{J}, \forall i \in J$, $x|_i$ is a fixpoint of $O_i^{x|_{\prec i}}$. By Proposition 3, such a component $O_i^{x|_{\prec i}}$ is equal to $(O_J^{x|_{I \setminus J}})_i^{x|_{\{j \in J | j \prec i\}}}$. By Proposition 2 and Theorem 1, $\forall J \in \mathcal{J}, \forall i \in J$, $x|_i$ is a fixpoint $(O_J^{x|_{I \setminus J}})_i^{x|_{\{j \in J | j \prec i\}}}$ iff $\forall J \in \mathcal{J}$, $x|_J$ is a fixpoint of $O_J^{x|_{I \setminus J}}$.

4 Modularity results for ID-logic

Now, we apply the algebraic results presented in Section 3 to ID-logic. We fix a vocabulary Σ and a pre-interpretation H for Σ. Also, we restrict our attention to H-interpretations, which can therefore be viewed as sets of domain atoms.

The basic notion needed to split an ID-logic theory, is that of a *dependence relation* between domain atoms. Roughly speaking, such a relation is supposed to express which domain atoms $Q(\mathbf{c})$ can influence whether an operator $\mathcal{T}_\Delta^O$ will derive a certain $P(\mathbf{a})$ in a pair (X, Y). We require that dependence relations are well-founded.

Definition 7. *A well-founded pre-order $\leqslant$ on domain atoms is called a* dependence relation. *We denote by $\mathcal{E}^{\leqslant}$ the set of all equivalence classes $\overline{P(\mathbf{a})} = \{Q(\mathbf{c}) \mid \underline{P(\mathbf{a})} \leqslant \underline{Q(\mathbf{c})}$ and $Q(\mathbf{c}) \leqslant P(\mathbf{a})\}$, together with the well-founded order $\preccurlyeq$, defined as $\overline{P(\mathbf{a})} \preccurlyeq \overline{Q(\mathbf{c})}$ iff $P(\mathbf{a}) \leqslant Q(\mathbf{c})$.*

Such a dependence relation now gives us a product lattice in which to study stratifiability of the operators $\mathcal{T}_\Delta^O$. Recall that we can only apply the algebraic splitting results, if $\mathcal{T}_\Delta^O$ can be seen as operating on the square of some product lattice $\otimes_{i \in I} L_i$. It turns out that the product of the powersets of all equivalence classes E in $\mathcal{E}^{\leqslant}$ can give us such a lattice. We denote by $\mathcal{S}^{\leqslant}$ the lattice $\bigotimes_{E \in \mathcal{E}^{\leqslant}} 2^E$. Now, $\mathcal{S}^{\leqslant}$ is isomorphic to the powerset of all domain atoms, which is in turn isomorpic to the set of all H-interpretations. An operator $\mathcal{T}_\Delta^O$ can therefore be seen as operating on the square of the set $\mathcal{S}_O^{\leqslant}$ of all elements of $\mathcal{S}^{\leqslant}$ which extend O (or, more precisely, whose image under the appropriate isomorphism extends O).

When dealing with the definition Δ_{even} from Example 1, we will consider the obvious pre-interpretation $H_{\mathbb{N}}$ with domain $\mathbb{N}$. The set of domain atoms then consists of $\{Even(n) \mid n \in \mathbb{N}\} \cup \{Odd(n) \mid n \in \mathbb{N}\}$. We will use the dependence relation $\leqslant$ consisting of: $Odd(n) \leqslant Even(n+1)$ and $Even(n) \leqslant Odd(n+1)$, for all $n \in \mathbb{N}$. The fact that $\leqslant$ is well-founded follows from the fact that $\mathbb{N}$ is well-founded. The set $\mathcal{E}^{\leqslant}$ consists of the equivalence classes $\{\overline{Even(n)} \mid n \in \mathbb{N}\} \cup \{\overline{Odd(n)} \mid n \in \mathbb{N}\}$, which are all singletons, i.e., for all $n \in \mathbb{N}$, $\overline{Even(n)} = \{\underline{Even(n)}\}$ and $\overline{Odd(n)} = \{\underline{Odd(n)}\}$. The relation $\preccurlyeq$ consists of the pairs $\overline{Even(n)} \preccurlyeq \overline{Odd(n+1)}$ and $\overline{Odd(n)} \preccurlyeq \overline{Even(n+1)}$ with $n \in \mathbb{N}$.

Definition 8. *A dependence relation* $\leqslant$ stratifies *a definition* Δ *given an* H*-interpretation* O *of* Σ^o_Δ *iff the operator* $\mathcal{T}^O_\Delta$ *is a stratifiable approximation on the product lattice* $\mathcal{S}^{\leqslant}_O$.

In [DT04b], a dependence relation that stratifies a definition, is called a *reduction relation*. In case of our example, the dependence relation $\leqslant$ defined above stratifies Δ_{even}. Now, the results presented in Section 3 can be used to show the equivalence of a definition Δ and certain partitions of Δ.

Definition 9. *Let* Δ *be a definition and let* $\leqslant$ *be a dependence relation. A partition* $\{\Delta_1, \ldots, \Delta_n\}$ *of* Δ *is a* $\leqslant$-partition *iff, for each* $1 \leq j \leq n$*, if* Δ_j *contains a rule defining a predicate* P*, then* Δ_j *also contains all rules defining a predicate* Q*, for which there exist tuples* $\mathbf{a}, \mathbf{c}$ *of domain elements, such that* $Q(\mathbf{c}) \in \overline{P(\mathbf{a})}$.

In order to show the desired equivalence, we relate the concept of $\leqslant$-partitions to that of recombinations.

Proposition 4. *Let* Δ *be a definition, let* $\leqslant$ *be a dependence relation, and* $\{\Delta_1, \ldots, \Delta_n\}$ *a* $\leqslant$*-partition. Let* O *be an* H*-interpretation of* $\Sigma^o_{\Delta_j}$*, for some* $1 \leq j \leq n$*. Then* $\mathcal{T}^O_{\Delta_j}$ *is equal to the recombination* $(\mathcal{T}^{O_1}_\Delta)^{O_2}_J$*, with* $O_1 = O|_{\Sigma^o_\Delta}$*,* $O_2 = O|_{(\Sigma^o_{\Delta_j} \setminus \Sigma^o_\Delta)}$*, and* $J = \{\overline{P(\mathbf{a})} \mid \Delta_j \text{ defines } P\}$.

Proof. Let $\mathcal{T}^O_{\Delta_j}$ and $(\mathcal{T}^{O_1}_\Delta)^{O_2}_J$ be as above. We first note that an H-interpretation X extends O iff it extends $O_1 \sqcup O_2$. It now follows directly from the definitions of the two operators, that $\mathcal{T}^O_{\Delta_j} = (\mathcal{T}^{O_1}_\Delta)^{O_2}_J$ iff for all X, Y extending O, the following two statements are equivalent:

- There exists a rule $\forall \mathbf{x}\ P(\mathbf{t}) \leftarrow \phi$ in Δ_j, for which there exists a $\mathbf{c} \in H^n_D$, such that $(X, Y)[\mathbf{x}/\mathbf{c}] \models \phi$.
- There exists a rule $\forall \mathbf{x}\ P(\mathbf{t}) \leftarrow \phi$ in Δ, for which there exists a $\mathbf{c} \in H^n_D$, such that $(X, Y)[\mathbf{x}/\mathbf{c}] \models \phi$.

Because, for each $P \in \mathcal{P}^d_{\Delta_j}$, Δ_j contains precisely all rules from Δ defining P, this is the case.

As a direct consequence of this proposition and Theorem 2, we now have the following equivalence between a definition and its $\leqslant$-partitions:

Theorem 3. *Let* Δ *be a definition,* $\leqslant$ *a dependence relation, and* $\{\Delta_1, \ldots, \Delta_n\}$ *a* $\leqslant$*-partition. Let* O *be a* Σ*-structure, such that* $\leqslant$ *stratifies* Δ *given* O*. Then for each* Σ*-structure* S*, such that* $S|_{\Sigma^o_\Delta} = O|_{\Sigma^o_\Delta}$*:*

$$S \models \Delta \text{ iff } S \models \Delta_1 \wedge \cdots \wedge \Delta_n.$$

[DT04b] contains a theorem which corresponds to the restriction of this theorem to those cases where each Δ_j is total given O. Our result is strictly more general.

We can now use this result to split the example Δ_{even}. Recall that above we already defined a dependence relation $\leqslant$ which stratifies Δ_{even}. A corresponding $\leqslant$-partition of Δ_{even} is:

$$\Delta_1 = \left\{ \begin{array}{c} Even(0). \\ \forall x\ Even(s(x)) \leftarrow Odd(x). \end{array} \right\}$$
$$\Delta_2 = \left\{ \forall x\ Odd(s(x)) \leftarrow Even(x). \right\}$$

Therefore, for every H-interpretation S, $S \models \Delta_{even}$ iff $S \models \Delta_1 \wedge \Delta_2$.

We now characterize the components of the operators $\mathcal{T}_\Delta^O$ in more detail. Recall that a stratifiable operator $\mathcal{T}_\Delta^O$ has a component $(\mathcal{T}_\Delta^O)_E^{(U,V)}$ for each level $E \in \mathcal{E}^\leqslant$ and (U,V) in $(\mathcal{S}_O^\leqslant|_{\prec E})^2$. Our goal is now to find a way of deriving some new definition $\Delta_E^{(U,V)}$ from Δ, which characterizes such a component, i.e., such that $(\mathcal{T}_\Delta^O)_E^{(U,V)} = (U_{\Delta_E^{(U,V)}}, U_{\Delta_E^{(V,U)}})$.

Intuitively, there are two main steps in constructing a component-definition $\Delta_E^{(U,V)}$. First, we need to ground Δ w.r.t. to the set of domain atoms E. To do this, we need to assume domain closure, i.e., that for each $a \in H_D$, there exists some term t of Σ, such that $t^H = a$. Such a term is denoted $\hat{a}$; for a tuple $\mathbf{a} = (a_1, \ldots, a_n) \in H_D^n$, we denote $(\hat{a}_1, \ldots, \hat{a}_n)$ by $\hat{\mathbf{a}}$. Roughly speaking, in the grounding step, a rule r should be replaced by all rules that can be obtained by replacing the universally quantified variables $\mathbf{x}$ of r by some $\hat{\mathbf{a}}$, such that the head of this new rule corresponds to a domain atom in E. Additionally, existential quantifiers also need to be eliminated; this can be done by replacing such a quantifier by a disjunction over all domain elements.

In the following definition, the notation $\phi[\mathbf{x}/\mathbf{y}]$ is used to denote the result of substituting in ϕ every free occurence of a symbol $x \in \mathbf{x}$ by the corresponding symbol $y \in \mathbf{y}$.

Definition 10. *Let Δ be a definition, $E \in \mathcal{E}^\leqslant$. For a rule $(\forall \mathbf{x}\ A \leftarrow \phi) \in \Delta$ and domain tuple $\mathbf{a}$, the rule $r^{\mathbf{a}}$ is the rule $A' \leftarrow \phi'$, with $A' = A[\mathbf{x}/\hat{\mathbf{a}}]$ and $\phi' = \gamma(\phi[\mathbf{x}/\hat{\mathbf{a}}])$, with γ defined as:*

- *for each atom A, $\gamma(A) = A$;*
- *$\gamma(\phi_1 \vee \phi_2) = \gamma(\phi_1) \vee \gamma(\phi_2)$ and $\gamma(\neg\phi) = \neg\gamma(\phi)$;*
- *$\gamma(\exists x\ \phi) = \bigvee_{a \in H_D} \gamma(\phi[x/\hat{a}])$;*

The grounding *$\lfloor r \rfloor_E$ of a rule $r = (\forall \mathbf{x}\ P(\mathbf{t}) \leftarrow \phi) \in \Delta$, is the set of rules $r^{\mathbf{a}}$, with $\mathbf{a}$ a domain tuple, such that $P(\mathbf{t}^{H[\mathbf{x}/\mathbf{a}]}) \in E$. The* grounding *$\lfloor \Delta \rfloor_E$ of Δ is $\bigcup_{r \in \Delta} \lfloor r \rfloor_E$.*

In a second step, we now replace ground atoms $P(\mathbf{t})$ for which $\overline{P(\mathbf{t}^H)} \prec E$, by their truth-value according to (U,V); atoms such that $\overline{P(\mathbf{t}^H)} \in E$ are left as they are. We make the small technical assumption that two predicate symbols T and F exist, such that T holds and F does not.

Definition 11. *Let Δ be a definition, $E \in \mathcal{E}^\leqslant$, and $(U,V) \in (\mathcal{S}_O^\leqslant|_{\prec E})^2$. For each rule $r = (A \leftarrow \phi) \in \lfloor \Delta \rfloor_E$, we define $r^{(U,V)}$ as the rule $A \leftarrow \delta^{(U,V)}(\phi)$, with $\delta^{(U,V)}$ inductively defined as:*

- *for each atom $A = P(\mathbf{t})$, such that $P(\mathbf{t}^H) \notin E$:* $\delta^{(U,V)}(A)$ *is T if $(U,V) \models A$ and F otherwise;*
- *for each other atom A,* $\delta^{(U,V)}(A) = A$*;*
- $\delta^{(U,V)}(\phi_1 \vee \phi_2) = \delta^{(U,V)}(\phi_1) \vee \delta^{(U,V)}(\phi_2)$*;*
- $\delta^{(U,V)}(\neg\phi) = \neg\delta^{(V,U)}(\phi)$.

We define $\Delta_E^{(U,V)}$ *as* $\{r^{(U,V)} \mid r \in \lfloor \Delta \rfloor_E\}$.

The proof of the following theorem is omitted, as it follows easily from the various definitions.

Theorem 4. *Let Δ be a definition, $E \in \mathcal{E}^{\leqslant}, U, V \in \mathcal{S}_O^{\leqslant}|_{\prec E}$, and O an H-interpretation of Σ_Δ^o. Then* $(U_\Delta^O)_E^{(U,V)} = U^O_{\Delta_E^{(U,V)}}$ *and* $(U_\Delta^O)_E^{(V,U)} = U^O_{\Delta_E^{(V,U)}}$.

Let us look again at definition Δ_{even} from Example 1, with the obvious pre-interpretation $H_{\mathbb{N}}$. If $E = \{Even(n+1)\}$ for some $n \in \mathbb{N}$, then for all $U, V \in \mathcal{S}^{\leqslant}|_{\prec E}$ the component $(\mathcal{T}_{\Delta_{even}})_E^{(U,V)}$ is the constant function $\{Even(n+1)\}$ if $n \in Odd^U$ and the constant function $\{\}$ otherwise. Similarly, for every level $E = \{Odd(n+1)\}$, $(\mathcal{T}_{\Delta_{even}})_E^{(U,V)}$ is the constant function $\{Odd(n+1)\}$ if $n \in Even^U$ and the constant function $\{\}$ otherwise. The component $(\mathcal{T}_{\Delta_{even}})_{\{Even(0)\}}$ is the constant function $\{Even(0)\}$, while the component $(\mathcal{T}_{\Delta_{even}})_{\{Odd(0)\}}$ is the constant function $\{\}$. From this, it follows that there exists a unique model of Δ_{even} extending $H_{\mathbb{N}}$, namely that which interprets $Even$ by $\{n \in \mathbb{N} \mid n \text{ is even}\}$ and Odd by $\{n \in \mathbb{N} \mid n \text{ is odd}\}$.

While space restrictions prevent us from discussing this here, this characterization of the components of a stratifiable operator promises to be useful for the study of the relation between ID-logic and known classes of mathematical inductive definitions. For instance, we suspect that the class of *well-founded inductions* coincides precisely with the class of ID-logic definitions whose $\mathcal{T}_\Delta^O$-operators can be split into constant components, as witnessed by the above example.

5 Conclusions and related work

Our work extends that from [VGD04b,VGD04a] about algebraic modularity results. Firstly, we have extended these results to also allow operators to be split into recombinations, rather than components. Secondly, our work is the first to apply these results outside a propositional context.

Our work also extends previous work on modularity properties for ID-logic [DT04b], by generalizing existing results in Theorem 3 and by the additional Theorem 4. It is interesting to note that, although in the context of ID-logic we are only interested in the well-founded fixpoints of the operators associated with definitions, our results also suffice to show a similar correspondence between their Kripke-Kleene and stable fixpoints. Indeed, this follows directly from the generality of the algebraic splitting theorem (Theorem 1). As such, our work actually also generalizes the results from [VGD04b], which in turn generalized part of the splitting theorem for the stable model semantics from [LT94].

The work presented here demonstrates that approximation theory and algebraic modularity results can be used to elegantly and easily derive useful results, even in a complex setting. In our opinion, it therefore offers quite a convincing testimony to the power of this approach.

References

[Den01] M. Denecker. Logic programming revisited: logic programs as inductive definitions. In *ACM Transactions on Computational Logic*, 2(4):623-654, 2001.

[DMT00] M. Denecker, V. Marek, and M. Truszczynski. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In *Logic-based Artificial Intelligence*, The Kluwer International Series in Engineering and Computer Science, pages 127–144, 2000.

[DMT03] M. Denecker, V. Marek, and M. Truszczynski. Uniform semantic treatment of default and autoepistemic logics. *Artificial Intelligence*, 143(1):79–122, 2003.

[DT04a] M. Denecker and E. Ternovska. Inductive situation calculus. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004)*, pages 545–553. AAAI Press, 2004.

[DT04b] M. Denecker and E. Ternovska. A logic of non-monotone inductive definitions and its modularity properties. In *7th International Conference on Logic Programming and Nonmonotonic Reasoning*, 2004.

[GP92] M. Gelfond and H. Przymusinska. On consistency and completeness of autoepistemic theories. *Fundamenta Informaticae*, 16(1):59–92, 1992.

[LT94] V. Lifschitz and H. Turner. Splitting a logic program. In *Proceedings of the 11th International Conference on Logic Programming*, pages 23–37, 1994.

[VGD04a] J. Vennekens, D. Gilis, and M. Denecker. Splitting an operator: An algebraic modularity result and its application to auto-epistemic logic. In *Proceedings of International Workshop on Non-Monotonic Reasoning*, 2004.

[VGD04b] J. Vennekens, D. Gilis, and M. Denecker. Splitting an operator: An algebraic modularity result and its applications to logic programming. In *Logic Programming, 20th International Conference, ICLP 2004, Proceedings*, volume 3132 of *Lecture Notes in Computer Science*, pages 195–209. Springer, 2004.