

## Predicate Introduction for Logics with a Fixpoint Semantics. Part I: Logic Programming. \*

### Joost Vennekens

joost.vennekens@cs.kuleuven.be  
Department of Computer Science, K.U. Leuven,  
Celestijnlaan 200A  
B-3001 Leuven, Belgium

### Maarten Mariën

maartenm@cs.kuleuven.be  
Department of Computer Science, K.U. Leuven,  
Celestijnlaan 200A  
B-3001 Leuven, Belgium

### Johan Wittocx<sup>†</sup>

johan@cs.kuleuven.be  
Department of Computer Science, K.U. Leuven,  
Celestijnlaan 200A  
B-3001 Leuven, Belgium

### Marc Denecker

marcd@cs.kuleuven.be  
Department of Computer Science, K.U. Leuven,  
Celestijnlaan 200A  
B-3001 Leuven, Belgium

---

**Abstract.** We study the transformation of “predicate introduction” in non-monotonic logics. By this, we mean the act of replacing a complex formula by a newly defined predicate. From a knowledge representation perspective, such transformations can be used to eliminate redundancy or to simplify a theory. From a more practical point of view, they can also be used to transform a theory into a normal form imposed by certain inference programs or theorems. In this paper, we study predicate introduction in the algebraic framework of “approximation theory”; this is a fixpoint theory for non-monotone operators that generalizes all main semantics of various non-monotonic logics, including logic programming, default logic and autoepistemic logic. We prove an abstract, algebraic equivalence result in this framework. This can then be used to show that, in logic programming, certain transformations are equivalence preserving under, among others, both the stable and well-founded semantics. Based on this result, we develop a general method of eliminating universal quantifiers in the bodies of rules. Our work is, however, also applicable beyond logic programming. In a companion paper, we demonstrate this, by using the same algebraic results to derive a transformation which reduces the nesting depth of the modal operator  $K$  in autoepistemic logic.

---

\*Works supported by FWO-Vlaanderen, IWT-Vlaanderen, and by GOA/2003/08.

<sup>†</sup>Research Assistant of the Fund for Scientific Research-Flanders (Belgium) (FWO-Vlaanderen).

## 1. Introduction

This paper studies the problem of “predicate introduction”. To introduce this topic, let us consider logic programming. In this context, predicate introduction refers to a transformation that introduces a new predicate in order to be able to simplify the expressions in the bodies of certain rules. To motivate our interest in this, we consider a simplified version of a program that occurs in [1]. In this paper, a logic program (under the stable semantics) is constructed to capture the meaning of theories in the action language  $\mathcal{AL}$ . In particular, *static causal laws* of the following form are considered: “ $P$  is caused if  $P_1, \dots, P_N$ ”. Here,  $P, P_1, \dots, P_N$  are propositional symbols. In its logic programming translation, such a causal law  $R$  is represented by the following set of facts:  $\{Head(R, P), Prec(R, 1, P_1), \dots, Prec(R, N, P_N), NbOfPrec(R, N)\}$ . (In logic programs, we use the notational convention that predicates, functions, and constant symbols start with an upper case letter, while variables are all lower case.)

Now, the meaning in  $\mathcal{AL}$  of such a law is that whenever all of  $P_1, \dots, P_N$  hold, then so must  $P$ . Using the predicate  $Holds/1$  to describe which propositions hold, this can be captured by the following rule (we use  $\Leftarrow$  to represent material implication and  $\leftarrow$  for the “rule construct” of logic programming):

$$\forall p \text{ Holds}(p) \leftarrow (\exists r \text{ Head}(r, p) \wedge \forall i \forall q \text{ Prec}(r, i, q) \Rightarrow \text{Holds}(q)). \quad (1)$$

This rule contains universal quantifiers in its body. Even though it is possible to define both stable and well-founded semantics for such programs, current model generation systems such as ASSAT, SModels or DLV cannot handle this kind of rules. Therefore, we would like to eliminate this quantifier. The well-known Lloyd-Topor transformation [13] suggests introducing a new predicate,  $BodyNotSat/1$ , to represent the negation of the subformula  $\varphi = \forall i \forall q \text{ Prec}(r, i, q) \Rightarrow \text{Holds}(q)$ . Because  $\neg\varphi = \exists i \exists q \text{ Prec}(r, i, q) \wedge \neg\text{Holds}(q)$ , we would then get:

$$\begin{aligned} \forall p, r \text{ Holds}(p) &\leftarrow \text{Head}(r, p) \wedge \neg\text{BodyNotSat}(r). \\ \forall r, i, q \text{ BodyNotSat}(r) &\leftarrow \text{Prec}(r, i, q) \wedge \neg\text{Holds}(q). \end{aligned} \quad (2)$$

This transformation preserves equivalence under the (two-valued) completion semantics [13]. However, for stable or well-founded semantics, this is not the case. For instance, consider the  $\mathcal{AL}$  theory  $\mathcal{A} = \{P \text{ is caused if } Q; Q \text{ is caused if } P\}$ . In the original translation (1), neither  $P$  nor  $Q$  holds; in the second version (2), however, we obtain (ignoring the  $Head/2$  and  $Prec/3$  atoms for clarity):

$$\begin{aligned} \text{Holds}(P) &\leftarrow \neg\text{BodyNotSat}(R_1). \\ \text{BodyNotSat}(R_1) &\leftarrow \neg\text{Holds}(Q). \\ \text{Holds}(Q) &\leftarrow \neg\text{BodyNotSat}(R_2). \\ \text{BodyNotSat}(R_2) &\leftarrow \neg\text{Holds}(P). \end{aligned} \quad (3)$$

Under the stable semantics, this program has the following two models:  $\{\text{Holds}(P), \text{Holds}(Q)\}$  and  $\{\text{BodyNotSat}(R_1), \text{BodyNotSat}(R_2)\}$ . As such, even though it might look reasonable at first, the Lloyd-Topor transformation does not preserve stable (or well-founded) models in this case.

Predicate introduction under the well-founded semantics was considered by Van Gelder [17]. That paper, however, imposes strong restrictions on how newly introduced predicates can be defined. In particular, recursive definitions of such a new predicate are not allowed. However, the ability to introduce recursively defined new predicates can be very useful; indeed, it is precisely in this way that [1] manages to eliminate the universal quantifier in (1).

**Example 1.1. (Adapted from [1])**

In order to replace the universally quantified subformula  $\forall i \forall q \text{Prec}(r, i, q) \Rightarrow \text{Holds}(q)$  of (1), we introduce a new predicate  $\text{AllPrecHold}(r)$ , resulting in:

$$\forall r, p \text{Holds}(p) \leftarrow \text{Head}(r, p) \wedge \text{AllPrecHold}(r). \quad (4)$$

This predicate is then defined in terms of another new predicate,  $\text{AllFrom}(r, i)$ , that means that the preconditions  $i, i + 1, \dots, n$  of a rule  $r$  with  $n$  preconditions are satisfied. We then define this predicate by the following recursion:

$$\begin{aligned} \forall r, n \text{AllPrecHold}(r) &\leftarrow \text{AllFrom}(r, 1). \\ \forall r, n, q \text{AllFrom}(r, n) &\leftarrow \text{Prec}(r, n, q) \wedge \text{Holds}(q) \wedge \text{AllFrom}(r, n + 1). \\ \forall r, n, q \text{AllFrom}(r, n) &\leftarrow \text{Prec}(r, n, q) \wedge \text{Holds}(q) \wedge \text{NbOfPrec}(r, n). \end{aligned} \quad (5)$$

In this paper, we prove a generalization of Van Gelder's result, that shows that this translation is indeed equivalence preserving.

So far, we have motivated our interest in predicate introduction by looking at logic programming. However, this same principle would clearly also be useful in other knowledge representation languages. It would therefore be preferable if we could somehow study such transformations in general, without committing to a specific logic. To this end, we will consider the algebraic framework of *approximation theory* [4] (see also Section 2.1). This is a fixpoint theory for arbitrary (non-monotone) operators that generalizes all main semantics of various non-monotonic logics, including logic programming, default logic and autoepistemic logic. As such, it allows properties of these different semantics for all of these logics to be studied in a uniform way. In this paper, we define the general, abstract concept of a *fixpoint extension* of an operator, which captures the notion of predicate introduction at the level of approximation theory. The central result of this paper is then an algebraic theorem that relates the fixpoints of such a fixpoint extension (which, intuitively, correspond to models of the transformed theory) to those of the original operator it extends (which correspond to models of the original theory).

By instantiating this algebraic theorem to the case of logic programming, we will be able to prove certain equivalences under the well-founded and stable model semantics, which generalize the aforementioned result by Van Gelder [17], by also allowing recursively defined new predicates. This has some interesting applications, including a general way of eliminating universal quantifiers. This offers an alternative for the corresponding step from the Lloyd-Topor transformation, which is only valid under completion semantics.

Our results in approximation theory are also applicable beyond logic programming. We demonstrate this in a companion paper [19], by applying the same algebraic theory to autoepistemic logic, to study transformations that introduce new propositions to reduce the nesting level of the modal operator  $K$ .

In summary, the contributions of this paper are the following. We develop a general algebraic theory of fixpoint extensions. We show that this abstracts several useful transformations for non-monotonic logics with a fixpoint semantics. Concretely, it generalizes both a method for eliminating universal quantifiers in logic programming and a way of reducing the nesting depth of the modal operator in autoepistemic logic. Our results allow succinct proofs of the fact that these transformations are equivalence preserving under an entire family of different semantics for the logic in question.

This paper is structured as follows. Section 2 introduces the framework of approximation theory and shows how a number of different semantics for logic programming can be defined in this setting.

We present our algebraic theory of fixpoint extension in Section 3 and, in Section 4, apply this to logic programming. In particular, this leads to a general way of eliminating universal quantifiers, which we discuss in Section 5.

This paper extends and generalizes work originally published in [20].

## 2. Preliminaries

In this section, we introduce some important concepts from approximation theory and show how these can be used to capture the stable and well-founded semantics for several logic programming variants.

### 2.1. Approximation theory

We use the following notations. Let  $\langle L, \leq \rangle$  be a complete lattice. A fixpoint of an operator  $O : L \rightarrow L$  on  $L$  is an element  $x \in L$  for which  $x = O(x)$ ; a prefixpoint of  $O$  is an  $x$  such that  $x \geq O(x)$ . We denote the set of all fixpoints of  $O$  as  $fp(O)$ . If  $O$  is monotone, then it has a unique least fixpoint  $x$ , which is also its unique least prefixpoint. We denote this  $x$  by  $lfp(O)$ .

Our presentation of approximation theory is based on [4, 5]. We consider the square  $L^2$  of the domain of some lattice  $L$ . We will denote an element of  $L^2$  as  $(x \ y)$ . We introduce the following projection functions: for a tuple  $(x \ y)$ , we denote by  $|(x \ y)|$  the first element  $x$  of this pair and by  $|x \ y|$  the second element  $y$ . The obvious point-wise extension of  $\leq$  to  $L^2$  is called the *product order* on  $L^2$ , which we also denote by  $\leq$ : i.e., for all  $(x \ y), (x' \ y') \in L^2$ ,  $(x \ y) \leq (x' \ y')$  iff  $x \leq x'$  and  $y \leq y'$ . An element  $(x \ y)$  of  $L^2$  can be seen as approximating certain elements of  $L$ , namely those in the (possibly empty) interval  $[x, y] = \{z \in L \mid x \leq z \text{ and } z \leq y\}$ . Using this intuition, we can derive a second order, the *precision order*  $\leq_p$ , on  $L^2$ : for each  $(x \ y), (x' \ y') \in L^2$ ,  $(x \ y) \leq_p (x' \ y')$  iff  $x \leq x'$  and  $y' \leq y$ . Indeed, if  $(x \ y) \leq_p (x' \ y')$ , then  $[x, y] \supseteq [x', y']$ , i.e.,  $(x' \ y')$  approximates fewer elements than  $(x \ y)$ . It can easily be seen that  $\langle L^2, \leq_p \rangle$  is also a lattice. The structure  $\langle L^2, \leq, \leq_p \rangle$  is the *bilattice* corresponding to  $L$ . If  $\langle L, \leq \rangle$  is complete, then so are  $\langle L^2, \leq \rangle$  and  $\langle L^2, \leq_p \rangle$ . Elements  $(x \ x)$  of  $L^2$  are called *exact*. The set of exact elements forms a natural embedding of  $L$  in  $L^2$ . We denote the set of all exact elements of a lattice  $L^2$  as  $Diag(L^2)$ .

Approximation theory is based on the study of operators which are monotone w.r.t.  $\leq_p$ . Such operators are called *approximations*. An approximation  $A$  approximates an operator  $O$  on  $L$  if for each  $x \in L$ ,  $A(x \ x)$  contains  $O(x)$ , i.e.  $[A(x \ x)] \leq O(x) \leq |A(x \ x)|$ . An exact approximation is one which maps exact elements to exact elements, i.e., for all  $x \in L$ ,  $[A(x \ x)] = |A(x \ x)|$ . Each exact approximation  $A$  approximates a unique operator  $O$  on  $L$ , namely the one that maps each  $x \in L$  to  $[A(x \ x)] = |A(x \ x)|$ . An approximation  $A$  is *symmetric* if  $\forall (x \ y) \in L^2$ , if  $A(x \ y) = (x' \ y')$  then  $A(y \ x) = (y' \ x')$ . A symmetric approximation is exact.

For an approximation  $A$  on  $L^2$ , we define the operator  $[A(\cdot \ y)]$  on  $L$  that maps an element  $x \in L$  to  $[A(x \ y)]$ , i.e.  $[A(\cdot \ y)] = \lambda x. [A(x \ y)]$ , and  $|A(x \ \cdot)|$  that maps an element  $y \in L$  to  $|A(x \ y)|$ . These operators are monotone. We define an operator  $C_A^\downarrow$  on  $L$ , called the *lower stable operator* of  $A$ , as  $C_A^\downarrow(y) = lfp([A(\cdot \ y)])$ . We also define the *upper stable operator*  $C_A^\uparrow$  of  $A$  as  $C_A^\uparrow(x) = lfp(|A(x \ \cdot)|)$ . Note that if  $A$  is symmetric, both operators are identical. We define the *stable operator*  $\mathcal{C}_A : L^2 \rightarrow L^2$  of  $A$  by  $\mathcal{C}_A(x \ y) = (C_A^\downarrow(y) \ C_A^\uparrow(x))$ . Because both  $C_A^\downarrow$  and  $C_A^\uparrow$  are anti-monotone,  $\mathcal{C}_A$  is  $\leq_p$ -monotone.

An approximation  $A$  defines a number of different fixpoints: the least fixpoint of  $A$  is called its

*Kripke-Kleene fixpoint*, fixpoints of its stable operator  $\mathcal{C}_A$  are *stable fixpoints* and the least fixpoint of  $\mathcal{C}_A$  is called the *well-founded fixpoint* of  $A$ . In [4, 5], it was shown that all main semantics of logic programming, autoepistemic logic and default logic can be characterized in terms of these fixpoints. In the next section, the case of logic programming is recalled.

## 2.2. Rule sets and logic programming semantics

We define a logical formalism generalizing logic programming and several of its extensions. An *alphabet*  $\Sigma$  consists of a set  $\Sigma^o$  of object symbols, a set  $\Sigma^f$  of function symbols, and a set  $\Sigma^p$  of predicate symbols. Note that we make no formal distinction between variables and constants; both will simply be called *object symbols*. The formalism considered here in this paper is that of *rule sets*  $\Delta$ . A rule set consists of rules of the form:

$$\forall \mathbf{x} P(\mathbf{t}) \leftarrow \varphi.$$

Here,  $P$  is a predicate symbol,  $\mathbf{x}$  a tuple of variables,  $\mathbf{t}$  a tuple of terms, and  $\varphi$  a first-order logic formula. We say that  $\Delta$  is a rule set over alphabet  $\Sigma$  if  $\Sigma$  contains all symbols occurring free in  $\Delta$ . For a rule  $r$  of the above form, the atom  $P(\mathbf{t})$  is called the *head* of  $r$ , while  $\varphi$  is known as its *body*. Predicates that appear in the head of a rule are *defined by*  $\Delta$ ; all other symbols are *open*. We denote the set of defined predicates by  $Def(\Delta)$  and that of all open ones by  $Op(\Delta)$ .

We now define a class of semantics for such rule sets. We interpret an alphabet  $\Sigma$  by a  $\Sigma$ -*structure* or  $\Sigma$ -*interpretation*. Such a  $\Sigma$ -interpretation  $I$  consists of a domain  $dom(I)$ , an interpretation of the object symbols  $C$  of  $\Sigma$  by domain elements, an interpretation of each function symbol  $F/n$  of  $\Sigma$  by an  $n$ -ary function on  $dom(I)$ , and an interpretation of each predicate symbol  $P/n$  by an  $n$ -ary relation on  $dom(I)$ . A *pre-interpretation* of  $\Sigma$  consists of a domain and an interpretation of the object and function symbols  $\Sigma^o \cup \Sigma^f$ . If the alphabet  $\Sigma$  is clear from the context, we often omit this from our notation. For any symbol  $\sigma \in \Sigma$ , we denote by  $\sigma^I$  the interpretation of  $\sigma$  by  $I$ . Similarly, for a term  $t$  we denote the interpretation of  $t$  by  $t^I$  and we also extend this notation to tuples  $\mathbf{t}$  of terms. For a structure  $I$ , an object symbol  $x$ , and a  $d \in dom(I)$ , we denote by  $I[x/d]$  the interpretation  $J$  with the same domain as  $I$ , that interprets  $x$  by  $d$  and coincides with  $I$  on everything else. We also extend this notation to tuples  $\mathbf{x}$  and  $\mathbf{d}$ . We define a truth order  $\leq$  on  $\Sigma$ -interpretations by  $I \leq J$  if  $I$  and  $J$  share the same pre-interpretation and, for each predicate  $P \in \Sigma$ ,  $P^I \subseteq P^J$ .

A feature of stable and well-founded semantics of logic programming is that positive and negative occurrences of atoms in rule bodies are treated very differently. The following non-standard truth evaluation function captures this difference.

**Definition 2.1.** Let  $\varphi$  be a formula. Let  $I$  and  $J$  be structures, which share the same pre-interpretation. We now define when a formula  $\varphi$  is satisfied in the pair  $(I J)$ , denoted  $(I J) \models \varphi$ , by induction over the size of  $\varphi$ :

- $(I J) \models P(\mathbf{t})$  iff  $I \models P(\mathbf{t})$ , i.e.,  $\mathbf{t}^I \in P^I$ ;
- $(I J) \models \neg\varphi$  iff  $(J I) \not\models \varphi$ ;
- $(I J) \models \varphi \vee \psi$  iff  $(I J) \models \varphi$  or  $(I J) \models \psi$ ;
- $(I J) \models \exists x \varphi(x)$  iff there is a  $d \in dom(I)$ , such that  $(I[x/d] J[x/d]) \models \varphi(x)$ .

Observe that evaluating the negation connective  $\neg$  switches the roles of  $I$  and  $J$ . Hence, this is the standard evaluation except that positively occurring atoms in  $\varphi$  are interpreted by  $I$ , while negatively occurring atoms in  $\varphi$  are interpreted by  $J$ . This evaluation function has a natural explanation when we view a pair  $(I J)$  as an approximation, i.e., when  $I$  is seen as a lower estimate and  $J$  as an upper estimate of some interpretation  $I'$ . When  $I \leq I' \leq J$ , positive occurrences of atoms are underestimated by using  $I$  and negative occurrences of atoms are overestimated by using  $J$ . It follows that  $(I J) \models \varphi$  implies  $I' \models \varphi$ . Or, if  $(I J) \models \varphi$ , then  $\varphi$  is certainly true in every approximated interpretation, while if  $(I J) \not\models \varphi$ , then  $\varphi$  is possibly false. Vice versa, when computing whether  $(J I) \models \varphi$ , positively occurring atoms are overestimated and negatively occurring ones are underestimated. Hence, if  $(J I) \not\models \varphi$ , then  $\varphi$  is certainly false in approximated interpretations, while if  $(J I) \models \varphi$ , then  $\varphi$  is possibly true. There is a strong link to Belnap's four-valued logic [2]. Indeed, pairs  $(I J)$  (sharing domains and interpretations of object and function symbols) correspond to four-valued interpretations, and the above non-standard truth evaluation is equivalent to the standard four-valued truth evaluation. More precisely, the four-valued valuation  $\varphi^{(I J)}$  of a formula  $\varphi$  can be characterized as follows:

- $\varphi^{(I J)} = \mathbf{t}$  iff  $(I J) \models \varphi$  and  $(J I) \models \varphi$ ;
- $\varphi^{(I J)} = \mathbf{f}$  iff  $(I J) \not\models \varphi$  and  $(J I) \not\models \varphi$ ;
- $\varphi^{(I J)} = \mathbf{u}$  iff  $(I J) \not\models \varphi$  and  $(J I) \models \varphi$ ;
- $\varphi^{(I J)} = \mathbf{i}$  iff  $(I J) \models \varphi$  and  $(J I) \not\models \varphi$ .

It is easy to see that a consistent pair  $(I J)$ , i.e., one for which  $I \leq J$ , corresponds to a three-valued interpretation.

We can offer an alternative explanation of these intuitions in the terminology of *rough sets* [14]. Essentially, a two-valued interpretation assigns to every predicate a crisp set of tuples, whereas a three-valued interpretation assigns each predicate a rough set of tuples, i.e., all tuples in the inside region of the rough set (those that belong to its lower approximation) are  $\mathbf{t}$ , all tuples in the outside region (those that do not belong to its upper approximation) are  $\mathbf{f}$  and all tuples in the boundary region are  $\mathbf{u}$ . Of course, even though our mathematical objects are in this way similar to rough sets, our use of them will be quite different to what is common in rough set theory. Indeed, rough sets are typically used in relation to some given data set, which does not occur in our context, making rough set concepts such as “discernability” of little interest.

From now on, for a formula  $\varphi(\mathbf{x})$  and a tuple  $\mathbf{d}$  of domain elements, we will write  $(I J) \models \varphi(\mathbf{d})$  instead of  $(I[\mathbf{x}/\mathbf{d}] J[\mathbf{x}/\mathbf{d}]) \models \varphi(\mathbf{x})$ .

Given a set of predicates  $\mathbf{P}$ , the class of all  $(\Sigma^o \cup \Sigma^f \cup \mathbf{P})$ -structures that extend some fixed pre-interpretation  $F$  is denoted as  $L_{\mathbf{P}}^F$ . For the order  $\leq$ ,  $L_{\mathbf{P}}^F$  is a complete lattice. Given a pair of interpretations for the open predicates  $(O_1 O_2)$  in  $(L_{Op(\Delta)}^F)^2$ , we will now define an immediate consequence operator  $\mathcal{T}_{\Delta}^{(O_1 O_2)}$  on pairs of interpretations of the defined predicates, i.e., on  $(L_{Def(\Delta)}^F)^2$ . The definition below is an alternative formalization of the standard four-valued immediate consequence operator [10].

**Definition 2.2.** Let  $\Delta$  be a rule set and  $(O_1 O_2) \in (L_{Op(\Delta)}^F)^2$ . We define a function  $U_{\Delta}^{(O_1 O_2)}$  from  $(L_{Def(\Delta)}^F)^2$  to  $L_{Def(\Delta)}^F$  as:  $U_{\Delta}^{(O_1 O_2)}(I J) = I'$ , where for each defined predicate  $P/n$ , for each  $\mathbf{d} \in$

$dom(F)^n$ ,  $\mathbf{d} \in P^I$  iff there exists a rule  $(\forall \mathbf{x} P(\mathbf{t}) \leftarrow \varphi(\mathbf{x})) \in \Delta$  and an  $\mathbf{a} \in dom(F)^n$ , such that  $((O_1 \cup I) (O_2 \cup J)) \models \varphi(\mathbf{a})$  and  $\mathbf{t}^{F[\mathbf{x}/\mathbf{a}]} = \mathbf{d}$ . We define the operator  $\mathcal{T}_\Delta^{(O_1 O_2)}$  on  $(L_{Def(\Delta)}^F)^2$  as  $\mathcal{T}_\Delta^{(O_1 O_2)}(I J) = (U_\Delta^{(O_1 O_2)}(I J) U_\Delta^{(O_2 O_1)}(J I))$ .

When a pair  $(O_1 O_2)$  approximates an interpretation  $O$  for the open predicates, i.e.,  $O_1 \leq O \leq O_2$ , then  $\mathcal{T}_\Delta^{(O_1 O_2)}$  is an approximation of the well-known 2-valued immediate consequence operator  $T_\Delta^O$ , which can be defined as  $T_\Delta^O(I) = [\mathcal{T}_\Delta^{(O O)}(I I)]$ . Because  $\mathcal{T}_\Delta^{(O_1 O_2)}$  is an approximation, it has a stable operator  $\mathcal{C}_{\mathcal{T}_\Delta^{(O_1 O_2)}}$ . The *well-founded model* of  $\Delta$  given  $(O_1 O_2)$  is the least fixpoint of this stable operator. Similarly, a pair  $(I J)$  is a *partial stable model* of  $\Delta$  given  $(O_1 O_2)$  iff  $(I J)$  is a fixpoint of this stable operator. An interpretation  $I$  for which  $(I I)$  is a partial stable model is called an (exact) stable model. Often, the latter are the only kind of stable models that are considered. However, for generality, we will always consider partial stable models. If  $\Delta$  is a logic program, then  $Op(\Delta) = \emptyset$  and  $O_1, O_2$  coincide with the Herbrand pre-interpretation  $F$ . In this case  $\mathcal{C}_{\mathcal{T}_\Delta}$  is symmetric and the upper and lower stable operators are identical to the well-known Gelfond-Lifschitz operator  $\mathcal{GL}_\Delta$  [11]. For the sake of completeness, we can also define some less popular logic programming semantics in terms of these operators. An interpretation  $I$  is a *supported model* of  $\Delta$  iff  $I|_{Def(\Delta)}$  is a fixpoint of  $\mathcal{T}_\Delta^{(I I)}$ . If  $\Delta$  does not contain open predicates, then its supported models are known to coincide with the classical models of Clark's *completion* [3] of  $\Delta$ . The *Kripke-Kleene model* [9] of  $\Delta$  under an interpretation  $(O_1 O_2)$  for the open predicates is the pair of interpretations  $(I J)$  for which  $(I J)$  is the least fixpoint of  $\mathcal{T}_\Delta^{(O_1 O_2)}$ .

A rule set  $\Delta$  is *monotone* iff every  $\mathcal{T}_\Delta^{(O_1 O_2)}$  is a monotone operator (w.r.t. the product order  $\leq$ ). For such rule sets, the well-founded model of  $\Delta$  given some  $(O_1 O_2)$  can be shown to coincide with the Kripke-Kleene model of  $\Delta$  under  $(O_1 O_2)$ , which is also the unique partial stable model for  $\Delta$  given  $(O_1 O_2)$ . A rule set  $\Delta$  is *positive* iff no defined predicate appears negatively in a rule body of  $\Delta$ . Such rule sets are always monotone.

We now introduce the following notations for the models of a rule set under partial stable and well-founded semantics:

**Definition 2.3.** Let  $\Delta$  be a rule set and  $F$  a pre-interpretation. Let  $I, J$  be  $\Sigma$ -structures that extend  $F$ . The pair  $(I J)$  is a model of  $\Delta$  under the well-founded semantics, denoted  $(I J) \models_w \Delta$  iff  $(I J)|_{Def(\Delta)}$  is the well-founded model of  $\Delta$  given  $(I J)|_{Op(\Delta)}$ . Similarly,  $(I J)$  is a model of  $\Delta$  under the partial stable model semantics, denoted  $(I J) \models_s \Delta$  iff  $(I J)|_{Def(\Delta)}$  is a partial stable model of  $\Delta$  under  $(I J)|_{Op(\Delta)}$ .

Using the above definitions, we can now characterize stable and well-founded semantics of the following extensions of logic programming:

- Normal logic programming: the bodies of rules are restricted to conjunctions of literals, there are no open predicates, and the pre-interpretation  $F$  is fixed to the Herbrand pre-interpretation.
- Abductive logic programming: the same, except that open predicates are allowed, which in this context are called *abducible* predicates and whose interpretation is arbitrary. LP-functions [12] are also of this form.
- Deductive Databases, and its extension AFP [17]: intensional predicates are defined, extensional database predicates are open but interpreted by the database  $O$ .

- ID-logic [6]: rule sets are used to represent inductive definitions.

The results of the following sections are applicable to all these formalisms.

### 3. Fixpoint extension

We want to study the following transformation. We start out with a rule set  $\Delta$  in some alphabet  $\Sigma$  and then introduce an additional alphabet  $\sigma$  of new symbols, e.g., the two predicates *AllPrecHold* and *AllFrom* from the example in the introduction, that are defined by some additional rules  $\delta$ . We then use these new predicates to form a new definition  $\Delta'$  over alphabet  $\Sigma \cup \sigma$ . In order to study such transformations in an algebraic setting, we will assume two complete lattices  $\langle L_1, \leq_1 \rangle$  and  $\langle L_2, \leq_2 \rangle$ . Here,  $L_1$  can be thought of as consisting of the interpretations for the original alphabet  $\Sigma$ , while  $L_2$  represents the interpretations for the additional new alphabet  $\sigma$ . We will need to prove a result concerning the stable and well-founded models of  $\Delta'$ , which means that we will need to work with pairs of interpretations of  $\Sigma \cup \sigma$ . As such, in our algebraic setting, we consider the square  $(L_1 \times L_2)^2$  of the Cartesian product  $L_1 \times L_2$ , which is isomorphic to the Cartesian product  $L_1^2 \times L_2^2$  of the squares of these lattices. We denote pairs  $P = ((x \ y) \ (u \ v))$  of this latter Cartesian product as  $\begin{pmatrix} x & y \\ u & v \end{pmatrix}$ , where  $(x \ y) \in L_1^2$  and  $(u \ v) \in L_2^2$ . We define the following projection functions: by  $[P]$  we denote the pair  $\begin{pmatrix} x \\ u \end{pmatrix}$ , by  $|P|$  the pair  $\begin{pmatrix} y \\ v \end{pmatrix}$ , by  $\lceil P \rceil$  the pair  $(x \ y)$ , by  $\lfloor P \rfloor$  the pair  $(u \ v)$ , by  $\lceil P \rceil$  the element  $u$ , by  $\lceil P \rceil$  the element  $x$ , by  $|P|$  the element  $y$ , and by  $\lfloor P \rfloor$  the element  $v$ .

Now, we want to prove a relation between the stable and well-founded fixpoints of the operator  $\mathcal{T}_\Delta$  of the original definition  $\Delta$  and those of the new operator  $\mathcal{T}_{\Delta'}$ . Algebraically, we consider an approximation  $A$  on the square  $L_1^2$  of the original lattice  $L_1$  and an approximation  $B$  on the extended lattice  $L_1^2 \times L_2^2$ . We now impose some conditions to ensure a correspondence between the stable fixpoints of  $A$  and  $B$ .

The main idea behind these conditions is the following. By introducing a new predicate into our original definition  $\Delta$ , we have added an additional “indirection”. For instance, in the original version  $\Delta$  of our example, we had the formula  $\forall i, q \text{ Prec}(r, i, q) \Rightarrow \text{ Holds}(q)$ , that could be evaluated in order to check whether all preconditions  $q$  of rule  $r$  were satisfied. This could be done by the  $\mathcal{T}_\Delta$ -operator in a single step. In our new definition  $\Delta'$ , however, every application of  $\mathcal{T}_{\Delta'}$  only checks whether a single precondition is satisfied. Intuitively, to match the effect of a single application of  $\mathcal{T}_\Delta$  to some pair  $(X, Y)$  of interpretations of the alphabet of  $\Delta$ , we have to iterate  $\mathcal{T}_{\Delta'}$  long enough for the truth assignments of  $(X, Y)$  to propagate throughout all of the new symbols of  $\Delta'$ . Nevertheless, the end result of this iteration of  $\mathcal{T}_{\Delta'}$  should coincide with the result of the single application of  $\mathcal{T}_\Delta$ . We need some more notation to formalize this intuition.

Given the operator  $B$  on  $L_1^2 \times L_2^2$  and a pair  $(x \ y) \in L_1^2$ , we define the operator  $B^{(x \ y)}$  on  $L_2^2$  as  $\lambda(u \ v). \lfloor B \begin{pmatrix} x & y \\ u & v \end{pmatrix} \rfloor$ . Conversely, given a pair  $(u \ v) \in L_2^2$ , we define the operator  $B_{(u \ v)}$  on  $L_1^2$  as  $\lambda(x \ y). \lceil B \begin{pmatrix} x & y \\ u & v \end{pmatrix} \rceil$ . Intuitively, the idea is that one application of the operator  $B^{(x \ y)}$  will correspond to the act of checking a single precondition in our example. Now, an important property is that the rule set  $\delta$  defining the new predicates contains only *positive* recursion, i.e.,  $\delta$  is a monotone definition. In our algebraic setting, this leads to a property which we will call *part-to-part monotonicity*, and which states that each operator  $B^{(x \ y)}$  is monotone.

By itself, however, this form of monotonicity will not suffice. Intuitively, the reason for this is that we should make sure that our transformation does not introduce “too much” non-monotonicity. In the context of logic programming, this means that we should not introduce an additional cycle over negation. Consider, for instance, the singleton rule set  $\{P \leftarrow \neg\neg P\}$ . The transformation of this rule set into  $\{P \leftarrow \neg N, N \leftarrow \neg P\}$  (i.e., replacing  $\neg P$  by a new atom  $N$ , defined as  $N \leftarrow \neg P$ ) would clearly not preserve stable or well-founded semantics. To avoid this case, we need to make sure that we either replace a positively occurring formula, e.g., replace  $P$  in  $\{P \leftarrow \neg\neg P\}$ , or that our new atom depends only positively on the original atoms. This last case would cover, for instance, the transformation of  $\{P \leftarrow \neg P\}$  into  $\{P \leftarrow \neg N, N \leftarrow P\}$ . In our algebraic setting, we introduce two different ways of strengthening the aforementioned part-to-part monotonicity. The first, called part-to-whole monotonicity, states that both the new and the old predicates depend positively on the new predicates; this covers, e.g., the replacement of  $P$  in  $\{P \leftarrow \neg\neg P\}$ . The second is called whole-to-part monotonicity and states that the new predicates should depend positively on both the new and the old predicates; this covers the transformation of  $\{P \leftarrow \neg P\}$  to  $\{P \leftarrow \neg N, N \leftarrow P\}$ . Note that none of these cases cover the forbidden transformation from  $\{P \leftarrow \neg\neg P\}$  to  $\{P \leftarrow \neg N, N \leftarrow \neg P\}$ . In summary, we get the following algebraic definitions.

**Definition 3.1.** Let  $B$  be an approximation on  $L_1^2 \times L_2^2$ .

- $B$  is *part-to-part monotone* iff for each  $(x \ y) \in L_1^2$  and  $(u \ v) \leq (u' \ v') \in L_2^2$ ,

$$\lfloor B \begin{pmatrix} x & y \\ u & v \end{pmatrix} \rfloor \leq \lfloor B \begin{pmatrix} x & y \\ u' & v' \end{pmatrix} \rfloor;$$

- $B$  is *part-to-whole monotone* iff for each  $(x \ y) \in L_1^2$  and  $(u \ v) \leq (u' \ v') \in L_2^2$ ,

$$B \begin{pmatrix} x & y \\ u & v \end{pmatrix} \leq B \begin{pmatrix} x & y \\ u' & v' \end{pmatrix};$$

- $B$  is *whole-to-part monotone* iff for all  $\begin{pmatrix} x & y \\ u & v \end{pmatrix} \leq \begin{pmatrix} x' & y' \\ u' & v' \end{pmatrix} \in L_1^2 \times L_2^2$ ,

$$\lfloor B \begin{pmatrix} x & y \\ u & v \end{pmatrix} \rfloor \leq \lfloor B \begin{pmatrix} x' & y' \\ u' & v' \end{pmatrix} \rfloor.$$

It is easy to see that both part-to-whole and whole-to-part monotonicity imply part-to-part monotonicity. Therefore, if any of these three properties is satisfied, then every operator  $B^{(x \ y)}$  will be monotone w.r.t. the product order  $\leq$  and, as such, have a  $\leq$ -least fixpoint  $\text{lfp}(B^{(x \ y)})$ . If we are extending a definition  $\Delta$  with some new predicates, defined by a monotone rule set  $\delta$ , it is now precisely this least fixpoint that will tell us what we can obtain by iteratively applying the rules of  $\delta$ . As explained above, once this derivation using the rules of  $\delta$  has reached its limit, the operator  $B$  should behave as  $A$  does on  $L_1^2$ .

**Definition 3.2. (Fixpoint extension)**

Let  $B$  be an approximation on  $L_1^2 \times L_2^2$  and  $A$  an approximation on  $L_1^2$ .  $B$  is a *fixpoint extension* of  $A$  iff it is part-to-part monotone and, for all  $x, y \in L_1$ ,  $B_{\text{lfp} \leq (B^{(x \ y)})}(x \ y) = A(x \ y)$ .

Our goal is now to prove a correspondence between fixpoints of an approximation  $A$  and a fixpoint extension  $B$  of  $A$ . We begin by making the trivial observation that any fixpoint  $(x \ y)$  of  $A$  can be extended to a fixpoint  $\begin{pmatrix} x & y \\ u & v \end{pmatrix}$  of  $B$ , by choosing  $u$  and  $v$  in an appropriate way.

**Theorem 3.1.** Let  $B$  be a fixpoint extension of  $A$ . A pair  $(x \ y) \in L_1^2$  is a fixpoint of  $A$  iff  $\begin{pmatrix} x & y \\ \text{lfp}(B(x \ y)) \end{pmatrix}$  is a fixpoint of  $B$ .

This theorem shows that we can find the fixpoints of  $A$  by first constructing the fixpoints of  $B$  and then checking for which of these fixpoints  $\begin{pmatrix} x & y \\ u & v \end{pmatrix}$  it is the case that  $(u \ v) = \text{lfp}(B(x \ y))$ . The following example demonstrates that it is indeed necessary to check this additional condition, because fixpoints of  $B$  for which it does not hold might not correspond to fixpoints of  $A$ .

**Example 3.1.** Consider the following rule set:

$$\Delta = \left\{ \begin{array}{l} P \leftarrow Q. \\ Q \leftarrow \mathbf{false}. \end{array} \right\}.$$

Let us try to replace  $Q$  in the first rule by  $R$ , defined by:

$$\delta = \left\{ \begin{array}{l} R \leftarrow R. \\ R \leftarrow Q. \end{array} \right\}.$$

The resulting rule set  $\Delta'$  is then of course:

$$\Delta' = \left\{ \begin{array}{l} P \leftarrow R. \\ Q \leftarrow \mathbf{false}. \\ R \leftarrow R. \\ R \leftarrow Q. \end{array} \right\}.$$

As will become clear in Section 4, where we discuss predicate introduction for rule sets, the operator  $\mathcal{T}_{\Delta'}$  is a fixpoint extension of the operator  $\mathcal{T}_{\Delta}$ . However,  $\mathcal{T}_{\Delta'}$  has a fixpoint in which  $P$  holds (namely, the pair  $(\{P, R\}, \{P, R\})$ ), while  $\mathcal{T}_{\Delta}$  does not. The reason for the discrepancy is, of course, the rule  $R \leftarrow R$  in  $\delta$ . This positive recursion has the effect that—at least under completion semantics— $R$  might become true, even when  $Q$  is false.

This example demonstrates that, given some fixpoint  $\begin{pmatrix} x & y \\ u & v \end{pmatrix}$  of  $B$ , it is indeed necessary to first check whether  $(u \ v) = \text{lfp}(B(x \ y))$ , before concluding that  $(x \ y)$  is a fixpoint of  $A$ . Of course, if  $B$  happens to be such that, for some reason, we can always be sure that this condition is satisfied, then we can safely ignore it, and Theorem 3.1 tells us that the fixpoints  $\text{fp}(A)$  coincide precisely with the set  $\lceil \text{fp}(B) \rceil$  of all restrictions to  $L_1$  of fixpoints of  $B$ , which implies that also  $\text{lfp}(A) = \lceil \text{lfp}(B) \rceil$ . A sufficient condition for this is that every operator  $B^{(x \ y)}$  has only a single fixpoint; indeed, since for each fixpoint  $\begin{pmatrix} x & y \\ u & v \end{pmatrix}$  of  $B$ ,  $(u \ v)$  is obviously a fixpoint of  $B^{(x \ y)}$ , the equality  $(u \ v) = \text{lfp}(B^{(x \ y)})$  then immediately follows.

An interesting special case of this is when every  $B^{(x\ y)}$  is a constant operator. In this case, we will call the operator  $B$  *part-to-part constant*. In logic programming, we can get a part-to-part constant operator by simply disallowing recursion in the rule set defining our new predicates. As such, this result directly applies to, for instance, any transformation where we replace a subformula  $\varphi(\mathbf{x})$  of the body of some rule by a new atom  $P(\mathbf{x})$ , which we then define  $\forall \mathbf{x} P(\mathbf{x}) \leftarrow \varphi(\mathbf{x})$ . For such transformations, we will therefore get a correspondence between the completion and Kripke-Kleene semantics.

So far, we have considered the relation between fixpoints of  $A$  and  $B$ , and showed that, in particular, every fixpoint of  $A$  has a corresponding fixpoint of  $B$ . We now turn our attention to the stable fixpoints of these operators. We first prove that if a fixpoint of  $A$  happens to be stable, then the corresponding fixpoint of  $B$  will be stable as well.

**Theorem 3.2.** Let  $B$  be a fixpoint extension of  $A$ . If  $(x\ y)$  is a fixpoint of the stable operator  $\mathcal{C}_A$  of  $A$ , then  $\binom{x}{\text{lfp}(B^{(x\ y)})}$  is a fixpoint of the stable operator  $\mathcal{C}_B$  of  $B$ .

To prove this theorem, we first study some more properties of  $B$  and  $\text{lfp}(B^{(x\ y)})$ . The operators  $B^{(x\ y)}$  are quite special, in the sense that they are both  $\leq_p$  and  $\leq$ -monotone. It can easily be shown that, in general, for any such operator  $O$  on a lattice  $L^2$ , the first and second component of  $O$  are completely independent.

**Lemma 3.1.** Let  $O$  be an operator on a lattice  $L^2$ , such that  $O$  is both  $\leq$ -monotone and  $\leq_p$ -monotone. For every  $(a\ b) \in L^2$ ,  $[O(a\ b)] = [O(a\ \top)]$  and  $|O(a\ b)| = |O(\top\ b)|$ .

**Proof:**

Let  $a, b \in L$ . Because  $(a\ b) \leq (a\ \top)$ , we have that  $O(a\ b) \leq O(a\ \top)$  and, specifically,  $[O(a\ b)] \leq [O(a\ \top)]$ . Because  $(a\ b) \geq_p (a\ \top)$ , we have that  $O(a\ b) \geq_p O(a\ \top)$  and, specifically,  $[O(a\ b)] \geq [O(a\ \top)]$ . Therefore,  $[O(a\ b)] = [O(a\ \top)]$ . Similarly, from  $(a\ b) \leq (\top\ b)$  and  $(a\ b) \leq_p (\top\ b)$ , it follows that, respectively,  $|O(a\ b)| \leq |O(\top\ b)|$  and  $|O(a\ b)| \geq |O(\top\ b)|$ , which proves the result.  $\square$

Let us denote by  $B_1^{(x\ y)}$  and  $B_2^{(x\ y)}$  the operators on  $L_2$ , that map, respectively, any  $u \in L_2$  to  $[B^{(x\ y)}(u\ \top)]$  and any  $v \in L_2$  to  $|B^{(x\ y)}(\top\ v)|$ , i.e., for all  $u \in L_2$ ,  $B_1^{(x\ y)}(u) = [B^{(x\ y)}(u\ \top)] = [B^{(x\ y)}(u\ \top)]$  and, similarly, for all  $v \in L_2$   $B_2^{(x\ y)}(v) = |B^{(x\ y)}(\top\ v)| = |B^{(x\ y)}(\top\ v)|$ . Clearly, these two operators are monotone.

Because each  $B^{(x\ y)}$  is both  $\leq$ -monotone (since  $B$  is part-to-part monotone) and  $\leq_p$ -monotone (since  $B$  is an approximation), Lemma 3.1 now implies the following result.

**Lemma 3.2.** Let  $B$  be part-to-part monotone. Then for all  $x, y \in L_1$  and  $u, v \in L_2$ ,

$$B^{(x\ y)}(u\ v) = (B_1^{(x\ y)}(u)\ B_2^{(x\ y)}(v)).$$

It follows directly from this lemma that  $\text{lfp}(B^{(x\ y)}) = (\text{lfp}(B_1^{(x\ y)})\ \text{lfp}(B_2^{(x\ y)}))$ . We can now use this result to show that extending a pair  $(x\ y) \in L_1^2$  with  $\text{lfp}(B^{(x\ y)})$  preserves the precision order.

**Lemma 3.3.** Let  $B$  be part-to-part monotone. For all  $x, x', y, y' \in L_1$ ,

$$(x\ y) \leq_p (x'\ y') \text{ iff } \binom{x}{\text{lfp}(B^{(x\ y)})} \leq_p \binom{x'}{\text{lfp}(B^{(x'\ y')})}.$$

**Proof:**

It is clear that the right hand side of this equivalence directly implies the left. Let  $x, x', y, y'$  be as above and let  $(u \ v) = \text{lfp}(B^{(x \ y)})$  and  $(u' \ v') = \text{lfp}(B^{(x' \ y')})$ . We have to show that  $(u \ v) \leq_p (u' \ v')$ . We first show that  $u \leq u'$ . By Lemma 3.2,  $u = \text{lfp}(B_1^{(x \ y)})$ . Because this implies that  $u$  is also the least prefixpoint of  $B_1^{(x \ y)}$ , it now suffices to show that  $u'$  is also a prefixpoint of  $B_1^{(x \ y)}$ , i.e., that  $u' \geq B_1^{(x \ y)}(u')$ . Because  $(\begin{smallmatrix} x' & y' \\ u' & \top \end{smallmatrix}) \geq_p (\begin{smallmatrix} x & y \\ u' & \top \end{smallmatrix})$ , we have that  $u' = \lfloor B(\begin{smallmatrix} x' & y' \\ u' & \top \end{smallmatrix}) \rfloor \geq \lfloor B(\begin{smallmatrix} x & y \\ u' & \top \end{smallmatrix}) \rfloor = B_1^{(x \ y)}(u')$ . The fact that  $v \geq v'$  can be shown in a similar way, by proving that  $v$  is a prefixpoint of the operator  $B_2^{(x' \ y')}$ , of which  $v'$  is the least prefixpoint.  $\square$

The stable operator  $\mathcal{C}_B$  of an approximation  $B$  is defined in terms of its downward and upward stable operators  $C_B^\downarrow$  and  $C_B^\uparrow$ . We show the following relation between these operators and the operators  $B_1^{(x \ y)}$  and  $B_2^{(x \ y)}$ .

**Lemma 3.4.** Let  $B$  be a part-to-part monotone approximation on  $L_1^2 \times L_2^2$ . If  $(\begin{smallmatrix} x \\ u \end{smallmatrix}) = C_B^\downarrow(\begin{smallmatrix} y \\ v \end{smallmatrix})$ , then  $u = \text{lfp}(B_1^{(x \ y)})$ . If  $(\begin{smallmatrix} y \\ v \end{smallmatrix}) = C_B^\uparrow(\begin{smallmatrix} x \\ u \end{smallmatrix})$ , then  $v = \text{lfp}(B_2^{(x \ y)})$ .

**Proof:**

Let  $B$  be as above. We only prove the first implication; the proof of the second one is analogous. Let  $(\begin{smallmatrix} x \\ u \end{smallmatrix}) = C_B^\downarrow(\begin{smallmatrix} y \\ v \end{smallmatrix})$  and let  $u' = \text{lfp}(B_1^{(x \ y)})$ . We will show that  $u = u'$ . We start by showing that  $u' \leq u$ . By definition of  $C_B^\downarrow$ ,  $(\begin{smallmatrix} x \\ u \end{smallmatrix}) = \lceil B(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix}) \rceil$ . In particular,  $u = B_1^{(x \ y)}(u)$ , i.e.,  $u$  is a fixpoint of  $B_1^{(x \ y)}$ . Because  $u'$  was chosen to be the least fixpoint of this operator,  $u' \leq u$ .

Now, we prove that also  $u \leq u'$ . We do this by constructing an element  $x' \in L_1$ , such that  $(\begin{smallmatrix} x' \\ u' \end{smallmatrix})$  is a prefixpoint of  $\lceil B(\begin{smallmatrix} \cdot & y \\ \cdot & v \end{smallmatrix}) \rceil$ . Because  $(\begin{smallmatrix} x \\ u \end{smallmatrix})$  is the least such prefixpoint, it will then follow that  $(\begin{smallmatrix} x \\ u \end{smallmatrix}) \leq (\begin{smallmatrix} x' \\ u' \end{smallmatrix})$  and, in particular,  $u \leq u'$ . To construct this  $x'$ , we consider the operator  $\lceil B(\begin{smallmatrix} \cdot & y \\ u' & v \end{smallmatrix}) \rceil$  on  $L_1$  that maps every  $z \in L_1$  to  $\lceil B(\begin{smallmatrix} z & y \\ u' & v \end{smallmatrix}) \rceil$ . Because  $B$  is  $\leq_p$ -monotone, this is a monotone operator and, therefore, it must have a least fixpoint. Let  $x'$  be this least fixpoint. Because  $u' \leq u$ ,  $(\begin{smallmatrix} x & y \\ u' & v \end{smallmatrix}) \leq_p (\begin{smallmatrix} x & y \\ u & v \end{smallmatrix})$  and  $\lceil B(\begin{smallmatrix} x & y \\ u' & v \end{smallmatrix}) \rceil \leq \lceil B(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix}) \rceil = x$ , i.e.,  $x$  is a prefixpoint of the operator  $\lceil B(\begin{smallmatrix} \cdot & y \\ u' & v \end{smallmatrix}) \rceil$ . Therefore,  $x' \leq x$ . Now, because  $(\begin{smallmatrix} x' & y \\ u' & v \end{smallmatrix}) \leq_p (\begin{smallmatrix} x & y \\ u' & v \end{smallmatrix})$ ,  $\lfloor B(\begin{smallmatrix} x' & y \\ u' & v \end{smallmatrix}) \rfloor \leq \lfloor B(\begin{smallmatrix} x & y \\ u' & v \end{smallmatrix}) \rfloor = B_1^{(x \ y)}(u') = u'$ . Because, by construction,  $x' = \lceil B(\begin{smallmatrix} x' & y \\ u' & v \end{smallmatrix}) \rceil$ , we have that  $\lfloor B(\begin{smallmatrix} x' & y \\ u' & v \end{smallmatrix}) \rfloor \leq (\begin{smallmatrix} x' \\ u' \end{smallmatrix})$  and  $(\begin{smallmatrix} x' \\ u' \end{smallmatrix})$  is indeed a prefixpoint of  $\lceil B(\begin{smallmatrix} \cdot & y \\ \cdot & v \end{smallmatrix}) \rceil$ . Therefore,  $u \leq u'$ .  $\square$

We now have all the material needed to prove that a stable fixpoint of  $A$  can be extended to a stable fixpoint of  $B$ .

**Proof:**

[Proof of theorem 3.2] Let  $(x \ y) = \mathcal{C}_A(x \ y)$  and let  $(u \ v)$  be  $\text{lfp}(B^{(x \ y)})$ . We have to show that  $(\begin{smallmatrix} x & y \\ u & v \end{smallmatrix})$

is a fixpoint of  $\mathcal{C}_B$ , i.e., that  $\binom{x}{u} = C_B^\uparrow(y)$  and  $\binom{y}{v} = C_B^\downarrow(x)$ . We only prove the first equality; the proof of the other one is analogous. Let  $\binom{x'}{u'}$  be  $\text{lfp}([B(\cdot \ y)])$ . We will show that  $\binom{x'}{u'} = \binom{x}{u}$ . By Theorem 3.1, we have that  $\binom{x \ y}{u \ v}$  is a fixpoint of  $B$ , which implies that  $\binom{x}{u}$  is a fixpoint of  $[B(\cdot \ y)]$ . Therefore,  $\binom{x'}{u'} \leq \binom{x}{u}$ . We now show that also  $\binom{x}{u} \leq \binom{x'}{u'}$ . First, we prove that  $x \leq x'$ , by showing that  $x'$  is a prefixpoint of the operator  $[A(\cdot \ y)]$ , of which  $x = C_A^\downarrow(y)$  is the least prefixpoint. If we let  $\binom{u'' \ v''}{u' \ v''} = \text{lfp}(B(\binom{x' \ y}{u'' \ v''}))$ , then, because  $B$  is a fixpoint extension of  $A$ ,  $[B(\binom{x' \ y}{u'' \ v''})] = A(x' \ y)$ . Moreover, because  $\binom{x'}{u'}$  is a fixpoint of  $[B(\cdot \ y)]$ ,  $u'$  is a fixpoint of  $B_1^{(x' \ y)}$ . Since  $u''$  is the least fixpoint of this operator,  $u'' \leq u'$  and therefore  $\binom{x' \ y}{u'' \ v''} \leq_p \binom{x' \ y}{u' \ v''}$ . By Lemma 3.3, the fact that  $x' \leq x$  implies that  $v'' = \text{lfp}(B(x' \ y)) \geq \text{lfp}(B(x \ y)) = v$ . Consequently, we also have that  $\binom{x' \ y}{u' \ v''} \leq_p \binom{x' \ y}{u' \ v}$  and, therefore, by  $\leq_p$ -monotonicity of  $B$ :

$$[A(x' \ y)] = [B(\binom{x' \ y}{u'' \ v''})] \leq [B(\binom{x' \ y}{u' \ v''})] \leq [B(\binom{x' \ y}{u' \ v})] = x'$$

Hence,  $x'$  is a prefixpoint of  $[A(\cdot \ y)]$  and  $x \leq x'$ . Therefore,  $x = x'$ . Since  $\binom{x'}{u'} = C_B^\downarrow(y)$ , Lemma 3.4 states that  $u' = \text{lfp}(B_1^{(x' \ y)})$ , which we now know to be identical to  $\text{lfp}(B_1^{(x \ y)}) = u$ . We conclude that  $\binom{x}{u} = \binom{x'}{u'}$ .  $\square$

So far, we have shown that for every stable fixpoint  $(x \ y)$  of  $A$ , it must be the case that  $B$  has some stable fixpoint  $\binom{x \ y}{u \ v}$ . We are of course also interested in the converse question, i.e., in whether, for every stable fixpoint  $\binom{x \ y}{u \ v}$  of  $B$ , the pair  $(x \ y)$  is a stable fixpoint of  $A$ . In the beginning of this section, we already encountered an example showing that this is not always the case: transforming  $\{P \leftarrow \neg\neg P\}$  into  $\{P \leftarrow \neg N; N \leftarrow \neg P\}$  generates additional stable fixpoints, which do not correspond to fixpoints of the original rule set. To exclude such cases, we require  $B$  to be either part-to-whole or whole-to-part monotone.

**Theorem 3.3.** Let  $B$  be a fixpoint extension of  $A$  that is either part-to-whole or whole-to-part monotone. If  $\binom{x \ y}{u \ v}$  is a fixpoint of the stable operator  $\mathcal{C}_B$ , then  $(x \ y)$  is a fixpoint of the stable operator  $\mathcal{C}_A$  of  $A$  and  $(u \ v) = \text{lfp}(B(x \ y))$ .

Our proof of this result will make use of the following property of whole-to-part monotone operators.

**Lemma 3.5.** Let  $B$  be an approximation on  $L_1^2 \times L_2^2$ . If  $B$  is whole-to-part monotone, then, for all  $(x \ y) \in L_1^2$ , the operator  $B_1^{(x \ y)}$  coincides with  $B_1^{(x \top)}$  and the operator  $B_2^{(x \ y)}$  coincides with  $B_2^{(\top \ y)}$ .

**Proof:**

Let  $B$  and  $(x \ y)$  be as above. To prove that  $B_1^{(x \ y)} = B_1^{(x \top)}$ , we observe that  $\binom{x \ y}{u \ \top} \geq_p \binom{x \ \top}{u \ \top}$  and

$\begin{pmatrix} x & y \\ u & \top \end{pmatrix} \leq \begin{pmatrix} x & \top \\ u & \top \end{pmatrix}$ . It now follows from the  $\leq_p$ -monotonicity and whole-to-part monotonicity of  $B$ , that  $|B(\begin{pmatrix} x & y \\ u & \top \end{pmatrix})| = |B(\begin{pmatrix} x & \top \\ u & \top \end{pmatrix})|$ . The proof that  $B_2^{(x y)} = B_2^{(\top y)}$  is analogous.  $\square$

We can now prove Theorem 3.3.

**Proof:**

[Proof of Theorem 3.3.] Let  $B$  be a fixpoint extension of  $A$ . We need to show that if  $\begin{pmatrix} x & y \\ u & v \end{pmatrix}$  is a fixpoint of  $\mathcal{C}_B$ , then  $(x y)$  is a fixpoint of  $\mathcal{C}_A$  and  $(u v)$  is  $lfp(B^{(x y)})$ . By definition,  $\begin{pmatrix} x & y \\ u & v \end{pmatrix}$  is a fixpoint of  $\mathcal{C}_B$  iff  $\begin{pmatrix} x \\ u \end{pmatrix} = C_B^\downarrow(\begin{pmatrix} y \\ v \end{pmatrix})$  and  $\begin{pmatrix} y \\ v \end{pmatrix} = C_B^\uparrow(\begin{pmatrix} x \\ u \end{pmatrix})$ . By Lemma 3.4, if this is the case, then  $u = lfp(B_1^{(x y)})$  and  $v = lfp(B_2^{(x y)})$ . Therefore, by Lemma 3.2,  $(u v) = lfp(B^{(x y)})$ . What remains to be shown is that  $x = C_A^\downarrow(y) = lfp(|A(\cdot y)|)$  and  $y = C_A^\uparrow(x) = lfp(|A(x \cdot)|)$ .

We will only prove the first equality; the proof of the second one is analogous. Because  $|A(x y)| = |B(\begin{pmatrix} x & y \\ u & v \end{pmatrix})| = x$ ,  $x$  is a fixpoint of  $|A(\cdot y)|$ . Let us now assume that there exists a fixpoint  $x'$  of this operator such that  $x' \leq x$ . We will show that  $x \leq x'$ , by constructing some  $u'$  for which  $\begin{pmatrix} x' \\ u' \end{pmatrix}$  is a prefixpoint of the operator  $|B(\cdot \begin{pmatrix} y \\ v \end{pmatrix})|$ , of which  $\begin{pmatrix} x \\ u \end{pmatrix}$  is the least fixpoint. Let  $(u' v')$  be  $lfp(B^{(x' y)})$ . Observe that, by construction, we have that both  $u' = |B(\begin{pmatrix} x' & y \\ u' & v' \end{pmatrix})|$  and  $x' = |A(x' y)| = |B(\begin{pmatrix} x' & y \\ u' & v' \end{pmatrix})|$ , that is,  $\begin{pmatrix} x' \\ u' \end{pmatrix} = |B(\begin{pmatrix} x' & y \\ u' & v' \end{pmatrix})|$ . We now need to distinguish between the case where  $B$  is part-to-whole monotone and the case where  $B$  is whole-to-part monotone.

- Suppose  $B$  is whole-to-part monotone. By Lemma 3.5, we have that the operators  $B_2^{(x y)}$  and  $B_2^{(x' y)}$  both coincide with  $B_2^{(\top y)}$  and, therefore, they must have the same least fixpoint, i.e.,  $v = v'$ . Therefore,  $\begin{pmatrix} x' \\ u' \end{pmatrix} = |B(\begin{pmatrix} x' & y \\ u' & v' \end{pmatrix})| = |B(\begin{pmatrix} x' & y \\ u' & v \end{pmatrix})|$ . Because  $\begin{pmatrix} x \\ u \end{pmatrix}$  is the least fixpoint of  $|B(\cdot \begin{pmatrix} y \\ v \end{pmatrix})|$ , this implies that  $\begin{pmatrix} x \\ u \end{pmatrix} \leq \begin{pmatrix} x' \\ u' \end{pmatrix}$  and, in particular,  $x \leq x'$ .
- Suppose  $B$  is part-to-whole monotone. Because  $(x' y) \leq_p (x y)$ , Lemma 3.3 implies that  $(u' v') \leq_p (u v)$ . In particular,  $v' \geq v$ . Because  $(u' v') \geq (u' v)$ , by part-to-whole monotonicity,  $\begin{pmatrix} x' \\ u' \end{pmatrix} = |B(\begin{pmatrix} x' & y \\ u' & v' \end{pmatrix})| \geq |B(\begin{pmatrix} x' & y \\ u' & v \end{pmatrix})|$ . Therefore  $\begin{pmatrix} x' \\ u' \end{pmatrix}$  is a prefixpoint of  $|B(\cdot \begin{pmatrix} y \\ v \end{pmatrix})|$  and, because  $\begin{pmatrix} x \\ u \end{pmatrix}$  is the least prefixpoint of this operator,  $\begin{pmatrix} x \\ u \end{pmatrix} \leq \begin{pmatrix} x' \\ u' \end{pmatrix}$  and, in particular,  $x \leq x'$ .  $\square$

Putting the above results together, we get the following theorem.

**Theorem 3.4.** Let  $B$  be a fixpoint extension of  $A$ , such that  $B$  is either part-to-whole or whole-to-part monotone.  $\begin{pmatrix} x & y \\ u & v \end{pmatrix}$  is a stable fixpoint of  $B$  iff  $(x y)$  is a stable fixpoint of  $A$  and  $(u v) = lfp(B^{(x y)})$ . Moreover,  $\begin{pmatrix} x & y \\ u & v \end{pmatrix}$  is the well-founded fixpoint of  $B$  iff  $(x y)$  is the well-founded fixpoint of  $A$  and  $(u v) = lfp(B^{(x y)})$ .

**Proof:**

The correspondence between stable fixpoints follows from Theorems 3.2 and 3.3. By Lemma 3.3, this correspondence between stable fixpoints also implies the correspondence between well-founded fixpoints, as these are simply the  $\leq_p$ -least stable fixpoints.  $\square$

## 4. Predicate introduction for rule sets

In this section, we use the algebraic results of Section 3 to derive a concrete equivalence theorem for rule sets. Recall that we are interested in transformations from some original rule set  $\Delta$  over an alphabet  $\Sigma$  into a new rule set  $\Delta'$  over an alphabet  $\Sigma' = \Sigma \cup \sigma$ . More concretely,  $\Delta'$  is the result of replacing a subformula  $\varphi(\mathbf{x})$  of some rule of  $\Delta$  by a new predicate  $P(\mathbf{x})$  and adding a new rule set  $\delta$  to  $\Delta$  to define this new predicate  $P$ . We will assume that  $\Sigma \cap Def(\delta)$  is empty. Note that  $\delta$  and  $\Delta'$  may contain new open predicates. We will denote the result of replacing (some fixed set of occurrences of)  $\varphi(\mathbf{x})$  in  $\Delta$  by  $P(\mathbf{x})$  as  $\Delta[\varphi(\mathbf{x})/P(\mathbf{x})]$ , i.e.,  $\Delta' = \Delta[\varphi(\mathbf{x})/P(\mathbf{x})] \cup \delta$ .

We will now use our algebraic results to prove the equivalence of  $\Delta'$  and  $\Delta$  under a number of different semantics. Recall that these results relate the fixpoints of an approximation  $A$  on the square of some lattice  $L_1$  to those of an operator  $B$  on the square of a product lattice  $L_1 \times L_2$ . In our case, we need to consider an pre-interpretation  $F$  and an interpretation  $O$  extending  $F$  to the open predicates of  $\Delta'$ . Our initial operator  $A$  will then be the operator  $\mathcal{T}_{\Delta}^{(O \ O)}$  on the square of the lattice  $L = L_{Def(\Delta)}^F$  of interpretations for the defined predicates of  $\Delta$ . The extended operator  $B$  will then be the operator  $\mathcal{T}_{\Delta'}^{(O \ O)}$  on the square of the lattice  $L_{Def(\Delta')}^F$  of interpretations for  $Def(\Delta')$ . Because  $Def(\Delta') = Def(\Delta) \cup Def(\delta)$ , this last lattice is clearly isomorphic to the product lattice  $L \times L'$ , with  $L'$  the lattice  $L_{Def(\delta)}^F$  of interpretations for  $Def(\delta)$ . Therefore, our lattices and operators are of the right form for our results to be applied. Throughout this section, we will continue to use the notations  $L$  and  $L'$  as introduced in this paragraph.

Our first task is now to show that, under suitable conditions,  $\mathcal{T}_{\Delta'}^{(O \ O)}$  is a fixpoint extension of  $\mathcal{T}_{\Delta}^{(O \ O)}$ .

**Theorem 4.1.** Let  $\Delta$  be a rule set and let  $\Delta'$  be the result  $\Delta[\varphi(\mathbf{x})/P(\mathbf{x})] \cup \delta$  of replacing some  $\varphi(\mathbf{x})$  in  $\Delta$  by a new predicate  $P(\mathbf{x})$  defined by some  $\delta$ . Let  $O$  be an interpretation for the object symbols, the function symbols and the open predicates of  $\Delta'$ , with  $D$  the domain of  $O$ . Let  $\mathcal{C}$  be the class of all structures that extend  $O$  to  $\Sigma'$ . If the following conditions are satisfied:

1.  $\delta$  is a monotone rule set and
2. for all  $I, J \in \mathcal{C}$  such that  $(I \ J) \models_s \delta$  it holds that  $\forall \mathbf{a} \in D^n, P(\mathbf{a})^{(I \ J)} = \varphi(\mathbf{a})^{(I \ J)}$ ,

then  $\mathcal{T}_{\Delta'}^{(O \ O)}$  is a fixpoint extension of  $\mathcal{T}_{\Delta}^{(O \ O)}$ .

**Proof:**

Let  $\Delta, \delta, \Delta', \varphi(\mathbf{x}), P(\mathbf{x}), O$ , and  $\mathcal{C}$  be as above. Let us first observe that, because the rules of  $\Delta'$  with a new predicate in their head are precisely the rules of  $\delta$ , we have that, for any  $\binom{I_1 \ J_1}{I_2 \ J_2} \in (L \times L')^2$ :

$$[\mathcal{T}_{\Delta'}^{(O \ O)}\left(\binom{I_1 \ J_1}{I_2 \ J_2}\right)] = (\mathcal{T}_{\Delta'}^{(O \ O)})^{(I_1 \ J_1)(I_2 \ J_2)} = \mathcal{T}_{\delta}^{(I_1 \ J_1)}(I_2 \ J_2). \quad (6)$$

Because  $\delta$  is a monotone rule set, this shows that  $\mathcal{T}_{\Delta'}^{(O\ O)}$  is part-to-part monotone. Now, let  $(I_2\ J_2)$  be  $\text{lfp}((\mathcal{T}_{\Delta'}^{(O\ O)})^{(I_1\ J_1)})$ . We now need to show that  $(\mathcal{T}_{\Delta'}^{(O\ O)})_{(I_2\ J_2)}(I_1\ J_1) = \mathcal{T}_{\Delta}^{(O\ O)}(I_1\ J_1)$ . Because none of the rules in  $\delta$  have a predicate from  $\Sigma$  in their head, these can safely be ignored, i.e.,  $(\mathcal{T}_{\Delta'}^{(O\ O)})_{(I_2\ J_2)}(I_1\ J_1) = (\mathcal{T}_{\Delta' \setminus \delta}^{(O\ O)})_{(I_2\ J_2)}(I_1\ J_1)$ . Now,  $\Delta' \setminus \delta$  and  $\Delta$  are completely identical, apart from the fact that in some rule bodies of  $\Delta' \setminus \delta$ , the formula  $\varphi(\mathbf{x})$  has been replaced by  $P(\mathbf{x})$ . Therefore, it now suffices to show that, for all tuples of domain elements  $\mathbf{a} \in D^n$ ,  $P(\mathbf{a})^{(I_2\ J_2)} = \varphi(\mathbf{a})^{(I_1\ J_1)}$ . By equation (6), we clearly have that  $(I_2\ J_2) = \text{lfp}(\mathcal{T}_{\delta}^{(I_1\ J_1)})$ . Because  $\delta$  is monotone, this implies that  $(\begin{smallmatrix} I_1 & J_1 \\ I_2 & J_2 \end{smallmatrix}) \models_s \delta$ . Condition 2 of the theorem now gives us precisely the equality we need.  $\square$

Note that, because we only consider monotone rule sets  $\delta$ , we could have equivalently used  $\models_w$  instead of  $\models_s$  in our formulation of condition 2 of this theorem.

To see that this result indeed applies to Example 1.1 from the introduction, let  $\delta$  be the rules given in (5). Clearly, this  $\delta$  is a positive rule set and, therefore, also monotone. Now, if we restrict our attention to those interpretations  $O$  for  $\text{Open}(\Delta')$  that actually correspond to  $\mathcal{AL}$ -rules<sup>1</sup>, then it is easy to see that for all  $r \in \text{dom}(O)$ ,  $\text{AllPrecHold}(r)^{(I\ J)}$  iff  $\varphi(r)^{(I\ J)}$ . Hence, for such interpretations  $O$ , the second condition of Theorem 4.1 is also satisfied, and, therefore, the immediate consequence operator of the extended rule set is a fixpoint extension of the original operator.

Let us now look at some implications of Theorem 4.1. We first consider supported model and Kripke-Kleene semantics. By Theorem 3.1, we have that, if the definition  $\delta$  is non-recursive, i.e., no new predicates appear in rule bodies, then the supported models and Kripke-Kleene model of  $\Delta$  coincide with, respectively, the restriction of the supported models and Kripke-Kleene model of  $\Delta'$  to the original alphabet  $\Sigma$ .

As we recall from Section 3, in order to also get a similar result for stable and well-founded semantics, we need some monotonicity properties. To this end, we will prove two results. The first result states that we get part-to-whole monotonicity if we replace only positively occurring subformulas, i.e., if after the transformation, the new predicates appear positively in the bodies of the original rules.

**Theorem 4.2.** Let  $\Delta$  be a rule set and let  $\Delta'$  be the result  $\Delta[\varphi(\mathbf{x})/P(\mathbf{x})] \cup \delta$  of replacing some  $\varphi(\mathbf{x})$  in  $\Delta$  by a new predicate  $P(\mathbf{x})$  defined by some monotone rule set  $\delta$ . Let  $O$  be an interpretation for the object symbols, the function symbols and the open predicates of  $\Delta'$ . If only positive occurrences of  $\varphi(\mathbf{x})$  are replaced, then  $\mathcal{T}_{\Delta'}^{(O\ O)}$  is part-to-whole monotone.

**Proof:**

Let  $\Delta, \delta, \Delta', P(\mathbf{x}), \varphi(\mathbf{x})$  and  $O$  be as above. Let  $(I_1\ J_1)$  extend  $O$  to  $\Sigma$  and let  $(I_2\ J_2)$  and  $(I'_2\ J'_2)$  extend  $O$  to  $\sigma$ , such that  $(I_2\ J_2) \leq (I'_2\ J'_2)$ . As can be seen from the proof of Theorem 4.1, the fact that  $\delta$  is monotone implies that  $\mathcal{T}_{\Delta'}^{(O\ O)}$  is part-to-part monotone. Therefore, it suffices to prove that  $\lceil \mathcal{T}_{\Delta'}^{(O\ O)}(\begin{smallmatrix} I_1 & J_1 \\ I_2 & J_2 \end{smallmatrix}) \rceil \leq \lceil \mathcal{T}_{\Delta'}^{(O\ O)}(\begin{smallmatrix} I_1 & J_1 \\ I'_2 & J'_2 \end{smallmatrix}) \rceil$ . Because the new predicate  $P(\mathbf{x})$  appears only positively in the rules that define the old predicates, this is the case.  $\square$

<sup>1</sup>More specifically, for every  $r$  there should be a unique  $n$  such that  $(r, n) \in \text{NbOfPrec}^O$  and for every  $1 \leq i \leq n$  there should a unique  $q$  such that  $(r, i, q) \in \text{Prec}^O$ .

For the purpose of eliminating universal quantifiers, we are only interested in replacing positively occurring subformulas, because a negatively occurring universal quantifier can of course simply be transformed into an existential one. Therefore, this kind of monotonicity will suffice for that particular purpose. However, in other applications, the following monotonicity result might also be useful. It states that we get whole-to-part monotonicity if the old predicates appear positively in the rules defining the new predicates.

**Theorem 4.3.** Let  $\Delta$  be a rule set and let  $\Delta'$  be the result  $\Delta[\varphi(\mathbf{x})/P(\mathbf{x})] \cup \delta$  of replacing some  $\varphi(\mathbf{x})$  in  $\Delta$  by a new predicate  $P(\mathbf{x})$  defined by some  $\delta$ . Let  $O$  be an interpretation for the object symbols, the function symbols and the open predicates of  $\Delta'$ . If the rules of  $\delta$  contain only positive occurrences of atoms of  $Def(\Delta)$ , then  $T_{\Delta'}^{(O O)}$  is whole-to-part monotone.

**Proof:**

Let  $\Delta, \delta, \Delta', P(\mathbf{x}), \varphi(\mathbf{x})$  and  $O$  be as above. Because  $Def(\Delta') = Def(\delta) \cup Def(\Delta)$  and  $\delta$  is monotone, the condition of this theorem implies that  $\delta$  is actually monotone in all of the predicates  $Def(\Delta')$ . Because the rules of  $\delta$  are the only rules of  $\Delta'$  with a new predicate in their head, this shows that  $T_{\Delta'}^{(O O)}$  is whole-to-part monotone.  $\square$

Put together, these two results tell us that the partial stable models and well-founded model of  $\Delta$  coincide with the restrictions of, respectively, the partial stable models and well-founded model of  $\Delta'$  to the original alphabet  $\Sigma$ , if the following conditions are satisfied. Firstly,  $\delta$  should be a monotone definition and all of its partial stable models should satisfy the four-valued equivalence between the new predicate  $P(\mathbf{x})$  and the original formula  $\varphi(\mathbf{x})$  (Condition 2 of Theorem 4.1). Secondly, it should either be the case that only positive occurrences of a formula are replaced, or that the  $\delta$  contains only positive occurrences of the original predicates.

Let us consider, for instance, the following rule, representing an inertia property:

$$\forall p, t \text{ Holds}(p, t + 1) \leftarrow \text{Holds}(p, t) \wedge \neg(\exists a \text{ Occurs}(a, t) \wedge \text{Terminates}(a, p)).$$

By our first monotonicity result, i.e., Theorem 4.2, we can replace the positively occurring formula  $\neg\exists a \text{ Occurs}(a, t) \wedge \text{Terminates}(a, p)$  by a new predicate  $Unclipped(p, t)$ , which gives us:

$$\begin{aligned} \forall p, t \text{ Holds}(p, t + 1) &\leftarrow \text{Holds}(p, t) \wedge \text{Unclipped}(p, t). \\ \forall p, t \text{ Unclipped}(p, t) &\leftarrow \neg(\exists a \text{ Occurs}(a, t) \wedge \text{Terminates}(a, p)). \end{aligned}$$

By the second monotonicity result of Theorem 4.3, on the other hand, we can replace the formula  $\exists a \text{ Occurs}(a, t) \wedge \text{Terminates}(a, p)$  of the original inertia property—a negatively occurring formula that does not contain negation—by a new predicate  $Clipped(a, p)$  in the following way:

$$\begin{aligned} \forall p, t \text{ Holds}(p, t + 1) &\leftarrow (\text{Holds}(p, t) \wedge \neg\text{Clipped}(p, t)). \\ \forall p, t \text{ Clipped}(p, t) &\leftarrow (\exists a \text{ Occurs}(a, t) \wedge \text{Terminates}(a, p)). \end{aligned}$$

The results of this section show that both of these transformations preserve partial stable and well-founded models, and, since neither the rule for  $Clipped$  nor  $Unclipped$  is recursive, also that they preserve supported and Kripke-Kleene models.

As a final remark in this section, it is useful to come back to this four-valued equivalence between  $P(\mathbf{x})$  as defined by  $\delta$  and the original formula  $\varphi(\mathbf{x})$ , that is a condition of Theorem 4.1. One might wonder whether this is really necessary, i.e., whether it would suffice to check only the following two-valued equivalence:

$$\text{For all } I \in C \text{ such that } (I \ I) \models_s \delta : \forall \mathbf{a} \in D^n, I \models P(\mathbf{a}) \text{ iff } I \models \varphi(\mathbf{a}). \quad (7)$$

In general, this is not the case. For instance, consider an attempt to replace in  $\Delta = \{R \leftarrow Q \vee \neg Q. Q \leftarrow \neg Q.\}$  the formula  $\varphi = Q \vee \neg Q$  by a new predicate  $P$ , defined by a definition  $\delta = \{P.\}$ . The above equivalence would then be satisfied and we would also get both part-to-whole and whole-to-part monotonicity, but there still is no correspondence between the models of  $\Delta$  and  $\Delta'$ . Indeed, the well-founded model of  $\Delta' = \{R \leftarrow P. P. Q \leftarrow \neg Q.\}$  is  $(\{R, P\} \{R, P, Q\})$ , while that of  $\Delta$  is  $(\{ \} \{R, Q\})$ .

The four-valued way of interpreting formulas is an integral part of both stable and well-founded semantics. Therefore, it makes sense that, as the above example shows, a four-valued equivalence is required in order to preserve either of these semantics. In practice, however, this should not pose too much of a problem, since most common transformations from classical logic, e.g. the De Morgan and distributivity laws, are still equivalence preserving in the four-valued case.

## 5. Applications and Related Work

The kind of transformations considered in this paper have a long history in logic programming. In particular, we consider three related investigations:

- Lloyd and Topor [13] introduced transformations that preserve equivalence under the 2-valued completion semantics. It is well-known that these transformations do not preserve equivalence under the well-founded or stable model semantics. As such, our result can be seen as an attempt to provide Lloyd-Topor-like transformations for these semantics.
- Van Gelder [17] presented a logic of alternating fixpoints to generalize the well-founded semantics to arbitrary rule bodies. In the same paper, he established the result given below as Theorem 5.1. We discuss the relation of our work with this result in Section 5.1
- The ‘‘Principle of Partial Evaluation’’ (PPE) was introduced by Dix in a study of properties for classifying logic programming semantics [7]. As we discuss in Section 5.2, this principle has a strong relation with our work.

More recently, there has been a lot of work in Answer Set programming on the topic of strong equivalence. Two rule sets  $\Delta$  and  $\Delta'$  are called strongly equivalent iff for all rule set  $\Delta''$ , it is the case that  $\Delta \cup \Delta''$  and  $\Delta' \cup \Delta''$  are equivalent, i.e., have the same stable models. Technically speaking, the transformations we consider here do not even preserve normal equivalence, due to the fact that they introduce new predicates. Indeed, our results only concern equivalence w.r.t. the original alphabet  $\Sigma$  of a rule set. Even this equivalence, however, can be lost if we allow the introduction of additional rules, which have some of the new predicates in their head. For instance, our result shows that  $\Delta = \{P \leftarrow \neg Q. Q \leftarrow Q.\}$  is equivalent to  $\Delta' = \{P \leftarrow R. R \leftarrow \neg Q. Q \leftarrow Q.\}$  w.r.t. the original alphabet  $\{P, Q\}$ .

However, if we now add to both  $\Delta$  and  $\Delta'$  the rule set  $\Delta'' = \{Q \leftarrow \neg R. R \leftarrow R.\}$ , we see that the restrictions to  $\Sigma$  of the stable models of  $\Delta \cup \Delta''$  and  $\Delta' \cup \Delta''$  are not the same. Indeed,  $\Delta' \cup \Delta''$  has a stable model in which  $P$  holds, whereas  $\Delta \cup \Delta''$  does not. It would seem, however, that this cannot occur if we only consider the addition of rules in the original alphabet.

### 5.1. Predicate extraction and eliminating $\forall$

The following result is due to Van Gelder:

#### Theorem 5.1. ([17])

Let  $\Delta$  be a rule set containing a rule  $r = \forall \mathbf{x} P(\mathbf{t}) \leftarrow \psi$ . Let  $\varphi(\mathbf{y})$  be an existentially quantified conjunction of literals, and let  $Q$  be a new predicate symbol. If  $\varphi(\mathbf{y})$  is a positively occurring subformula of  $\psi$ , then  $\Delta$  is equivalent under the partial stable and well-founded semantics to the rule set  $\Delta'$ , that results from replacing  $\varphi(\mathbf{y})$  in  $r$  by  $Q(\mathbf{y})$  and adding the rule  $\forall \mathbf{y} Q(\mathbf{y}) \leftarrow \varphi(\mathbf{y})$  to  $\Delta$ .

Because the rule set  $\delta = \{\forall \mathbf{y} Q(\mathbf{y}) \leftarrow \varphi(\mathbf{y}).\}$  clearly satisfies the conditions of Theorem 4.1, Van Gelder's theorem follows directly from ours. This result provides a theoretical justification for the common programming practice of *predicate extraction*: replacing a subformula that occurs in multiple rules by a new predicate to make the program more concise and more readable. In [15], predicate extraction is considered to be an important *refactoring* operation (i.e., an equivalence preserving transformation to improve maintainability) for logic programming.

Our result extends Van Gelder's theorem by allowing the new predicate  $Q$  to be defined by an additional rule set  $\delta$ , instead of allowing only the definition  $\{\forall \mathbf{y} Q(\mathbf{y}) \leftarrow \varphi(\mathbf{y}).\}$ . In particular, *recursive* definitions of  $Q$  are also allowed. This significantly increases the applicability of the theorem. Indeed, as we already illustrated in the introduction, this allows us to eliminate universal quantifiers. The general idea behind this method is that we can replace a universal quantifier by a recursion over some total order on the domain. Of course, this can only be done if the domain in question is finite.

#### Definition 5.1. (Domain iterator)

Let  $C$  be a set of  $\Sigma$ -structures with domain  $D$ . Let  $First/1$ ,  $Next/2$  and  $Last/1$  be predicate symbols of  $\Sigma$ . The triple  $\langle First, Next, Last \rangle$  is a *domain iterator* in  $C$  iff for each structure  $S \in C$ : the transitive closure of  $Next^S$  is a total order on  $D$  and there exists elements  $f, l \in D$  such that  $First^S = \{f\}$  and  $Last^S = \{l\}$  and, for all  $x \in D \setminus \{l\}$  there exists a unique  $y \in D \setminus \{f\}$  such that  $(x y) \in Next^S$ .

Given such a domain iterator  $It = \langle First, Next, Last \rangle$ , we can introduce the following rule set  $\delta_\varphi^{It}$  to define a new predicate  $Forall(\mathbf{x})$  as a replacement for some positive occurrence of a formula  $\varphi(\mathbf{x}) = \forall y \psi(\mathbf{x}, y)$ :

$$\begin{aligned} \forall \mathbf{x}, y \text{ Forall}(\mathbf{x}) &\leftarrow First(y) \wedge \psi(\mathbf{x}, y) \wedge AllFrom(\mathbf{x}, y). \\ \forall \mathbf{x}, y \text{ AllFrom}(\mathbf{x}, y) &\leftarrow Next(y, y') \wedge \psi(\mathbf{x}, y') \wedge AllFrom(\mathbf{x}, y'). \\ \forall \mathbf{x}, y \text{ AllFrom}(\mathbf{x}, y) &\leftarrow Last(y). \end{aligned} \tag{8}$$

It is quite obvious that, for non-empty domains, this transformation satisfies the conditions of Theorems 4.1 and 4.2. Therefore, we directly get the following result.

**Theorem 5.2. ( $\forall$  elimination)**

Let  $\Delta$  be a rule set and  $\varphi(\mathbf{x})$  be a formula of the form  $\forall y \psi(\mathbf{x}, y)$ , that appears only positively in the bodies of rules of  $\Delta$ . For a set of structures  $C$  with finite domain, if  $It$  is a domain iterator, then  $\Delta[\varphi/Forall] \cup \delta_\varphi^{It}$  is equivalent to  $\Delta$  under partial stable and well-founded semantics.

In this theorem, we assume a total order on the entire domain and this same order can be used to eliminate all universally quantified formulas, that satisfy the condition of the theorem. This is not precisely what happened in our example. Indeed, there, the universally quantified formula  $\varphi(\mathbf{x})$  was of the form:  $\forall \mathbf{y} \Psi_1(\mathbf{x}, \mathbf{y}) \Rightarrow \Psi_2(\mathbf{x}, \mathbf{y})$ . Using the above theorem, we would replace  $\varphi(\mathbf{x})$  by a recursion that says that the implication  $\Psi_1(\mathbf{x}) \Rightarrow \Psi_2(\mathbf{x})$  must hold for every element in the domain. However, in our original version of this example, we actually replaced  $\varphi(\mathbf{x})$  by a recursion which says that for all  $\mathbf{y}$  that satisfy  $\Psi_1(\mathbf{x}, \mathbf{y})$  (i.e., for all  $i, q$  such that  $Prec(r, i, q)$ ) the consequent  $\Psi_2(\mathbf{x}, \mathbf{y})$  (i.e.,  $Hold(q)$ ) is satisfied. This is a more fine-grained approach, which we can also prove in general. A *restricted iterator* for  $\mathbf{y}$  of  $\psi_1(\mathbf{x}, \mathbf{y})$  in a structure  $I$  is a triple of predicates  $\langle First(\mathbf{x}, \mathbf{y}), Next(\mathbf{x}, \mathbf{y}, \mathbf{y}'), Last(\mathbf{x}, \mathbf{y}) \rangle$ , such that for every tuple  $\mathbf{d}$  of elements of the domain  $D$  of  $I$ ,  $\langle First(\mathbf{d}, \mathbf{y}), Next(\mathbf{d}, \mathbf{y}, \mathbf{y}'), Last(\mathbf{d}, \mathbf{y}) \rangle$  is an iterator over  $\{\mathbf{e} \in D^n \mid I \models \Psi_1(\mathbf{d}, \mathbf{e})\}$ .

Given such a restricted iterator, we can define the following replacement  $Forall(\mathbf{x})$  for  $\varphi(\mathbf{x})$ :

$$\begin{aligned} \forall \mathbf{x}, \mathbf{y} \text{ Forall}(\mathbf{x}) &\leftarrow First(\mathbf{x}, \mathbf{y}) \wedge \Phi_2(\mathbf{x}, \mathbf{y}) \wedge AllFrom(\mathbf{x}, \mathbf{y}). \\ \forall \mathbf{x}, \mathbf{y}, \mathbf{y}' \text{ AllFrom}(\mathbf{x}, \mathbf{y}) &\leftarrow Next(\mathbf{x}, \mathbf{y}, \mathbf{y}') \wedge \Psi_2(\mathbf{x}, \mathbf{y}) \wedge AllFrom(\mathbf{x}, \mathbf{y}'). \\ \forall \mathbf{x}, \mathbf{y} \text{ AllFrom}(\mathbf{x}, \mathbf{y}) &\leftarrow Last(\mathbf{x}, \mathbf{y}). \end{aligned}$$

Again, Theorem 4.1 can be used to show that, if there is at least one tuple that satisfies  $\Phi_1$ ,  $\varphi(\mathbf{x})$  can be replaced by  $Forall(\mathbf{x})$ .

In [17], Van Gelder also considered predicate extraction for *negatively* occurring subformulas. His results on this topic are, however, substantially different from our Theorem 4.3. Indeed, we prove that, if  $\Delta'$  is the result of performing predicate introduction on a rule set  $\Delta$ , then, under certain conditions, the restriction of the well-founded model of  $\Delta'$  to the original alphabet coincides with the the well-founded model of  $\Delta$ . Van Gelder's results, on the other hand, prove that in all cases certain *parts* of the well-founded model of  $\Delta$  and  $\Delta'$  will be the same. This result is not implied by ours, nor vice versa. We have actually found no results similar to Theorem 5 in the literature.

**5.2. Principle of Partial Evaluation**

In one of a series of papers that gave properties by which major logic programming semantics could be classified, Dix introduced the “(Generalized) Principle of Partial Evaluation” ((G)PPE) [7]. The PPE basically states that a positive occurrence of  $P$  in a rule body can be “unfolded” (i.e.,  $P$ 's definition can be substituted) if  $P$  is defined non-recursively. Here, we recall the weak version of this property.

**Definition 5.2. (Weak PPE [7])**

Let  $\Delta$  be a ground rule set, and let an atom  $P$  occur only positively in the bodies of rules of  $\Delta$ . Let  $P \leftarrow \varphi_1, \dots, P \leftarrow \varphi_N$  be all the rules with head  $P$ , and assume that none of the  $\varphi_i$  contains  $P$ . We denote by  $\Delta_P$  the rule set obtained from  $\Delta$  by deleting all rules with head  $P$  and replacing each rule “ $Head \leftarrow P \wedge \psi$ ” containing  $P$  by the rules:

$$Head \leftarrow \varphi_1 \wedge \psi. \quad \dots \quad Head \leftarrow \varphi_N \wedge \psi. \quad (9)$$

The *weak principle of partial evaluation* states that there is a 1-1 correspondence between models of  $\Delta_P$  and models of  $\Delta$  (with removal of  $\{P, \neg P\}$ ).

Using Theorem 4.1, we can show that the stable and well-founded semantics exhibit this property: we rewrite (9) as “ $Head \leftarrow (\varphi_1 \vee \dots \vee \varphi_N) \wedge \psi$ ”. The replacement  $\delta = \{P \leftarrow \varphi_1 \vee \dots \vee \varphi_N.\}$  then satisfies all conditions of Theorem 4.1.

In the GPPE,  $P$  is allowed to have arbitrary occurrences in  $\Delta$ , and the substitution of  $P$  for its definition need not be applied for *each* rule containing  $P$ . On the other hand, the definition of  $P$  has to be present both before and after the transformation, making the precise relation with our result unclear. It is shown in [8] that the stable and well-founded semantics satisfy GPPE.

## 6. Conclusion

We have developed a theory of fixpoint extension in the algebraic framework of approximation theory and studied the applications of these results to logic programming. Concretely, we investigated transformations for a general class of logic programming variants, under the supported model, Kripke-Kleene, stable, and well-founded model semantics. One of our most interesting results here was a general way of eliminating universal quantifiers from rule bodies under stable and well-founded semantics. In a companion paper [19], we will also investigate how the same algebraic results can be applied to autoepistemic logic. This will further demonstrate that our algebraic results indeed provide a meaningful and generally applicable abstraction of the common knowledge representation methodology of predicate introduction.

The work in this paper is part of a larger research effort to define and study important knowledge representation concepts in the framework of approximation theory. In this paper, we have done so for predicate introduction. In [18], we did the same for certain modularity properties, proving an algebraic theorem that generalizes a number of known splitting results for logic programming, autoepistemic logic, and default logic. More recently, the author of [16] studied the concept of strong equivalence in the setting of approximation theory.

## References

- [1] Balduccini, M., Gelfond, M.: Diagnostic reasoning with A-Prolog., *Theory and Practice of Logic Programming (TPLP)*, **3**(4-5), 2003, 425–461.
- [2] Belnap, N. D.: A useful four-valued logic, in: *Modern uses of multiple-valued logic*, Reidel, Dordrecht, NL, 1977, 5–37.
- [3] Clark, K. L.: Negation as failure, in: *Logic and Databases* (H. Gallaire, J. Minker, Eds.), Plenum Press, 1978, 293–322.
- [4] Denecker, M., Marek, V., Truszczyński, M.: Approximating operators, stable operators, well-founded fix-points and applications in nonmonotonic reasoning, in: *Logic-based Artificial Intelligence* (J. Minker, Ed.), chapter 6, Kluwer Academic Publishers, 2000, 127–144.
- [5] Denecker, M., Marek, V., Truszczyński, M.: Uniform semantic treatment of default and autoepistemic logics, *Artificial Intelligence*, **143**(1), 2003, 79–122.

- [6] Denecker, M., Ternovska, E.: A Logic of Non-Monotone Inductive Definitions and its Modularity Properties, *Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'7)* (V. Lifschitz, I. Niemelä, Eds.), 2004.
- [7] Dix, J.: A Classification Theory of Semantics of Normal Logic Programs: II. Weak Properties., *Fundamenta Informaticae*, **22**(3), 1995, 257–288.
- [8] Dix, J., Müller, M.: Partial Evaluation and Relevance for Approximations of Stable Semantics., *ISMIS* (Z. W. Ras, M. Zemankova, Eds.), 869, Springer, 1994, ISBN 3-540-58495-1.
- [9] Fitting, M.: A Kripke-Kleene Semantics for Logic Programs, *Journal of Logic Programming*, **2**(4), 1985, 295–312.
- [10] Fitting, M.: Fixpoint semantics for logic programming - a survey, *Theoretical Computer Science*, **278**, 2002, 25–51.
- [11] Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming, *International Joint Conference and Symposium on Logic Programming (JICSLP'88)*, MIT Press, 1988.
- [12] Gelfond, M., Przymusinska, H.: Towards a Theory of Elaboration Tolerance: Logic Programming Approach, *Journal on Software and Knowledge Engineering*, **6**(1), 1996, 89–112.
- [13] Lloyd, J., Topor, R.: Making Prolog more expressive, *Journal of Logic Programming*, **1**(3), 1984, 225–240.
- [14] Pawlak, Z.: *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Norwell, MA, USA, 1992, ISBN 0792314727.
- [15] Schrijvers, T., Serebrenik, A.: Improving Prolog Programs: Refactoring for Prolog, *Logic Programming, 20th International Conference, ICLP 2004, Proceedings*, 3132, 2004.
- [16] Truszczyński, M.: Strong and uniform equivalence of nonmonotonic theories — an algebraic approach, *Principles of Knowledge Representation and Reasoning, Proceedings of the Tenth International Conference (KR2006)*, 2006.
- [17] Van Gelder, A.: The Alternating Fixpoint of Logic Programs with Negation, *Journal of Computer and System Sciences*, **47**(1), 1993, 185–221.
- [18] Vennekens, J., Gilis, D., Denecker, M.: Splitting an operator: Algebraic modularity results for logics with fixpoint semantics, *ACM Transactions on computational logic (TOCL)*, **7**(4), 2006, 765–797.
- [19] Vennekens, J., Wittocx, J., Mariën, M., Denecker, M., Bruynooghe, M.: Predicate Introduction for Logics with Fixpoint Semantics. Part II: Autoepistemic Logic, *Fundamenta Informaticae*.
- [20] Wittocx, J., Vennekens, J., Mariën, M., Denecker, M., Bruynooghe, M.: Predicate Introduction under Stable and Well-founded Semantics, *Proceedings of the 22nd International Conference on Logic Programming (ICLP'06)* (S. Etalle, M. Truszczyński, Eds.), 4079, Springer, 2006.