The Journal of Logic Programming 40 (1999) 89–123

THE JOURNAL OF LOGIC PROGRAMMING

An algebraic framework for the definition of compositional semantics of normal logic programs

Paqui Lucio, Fernando Orejas *, Elvira Pino

Dpto de L.S.I., Universidad Politécnica de Catalunya. Campus Nord. Modul C6, Jordi Girona 1-3, 08034 Barcelona, Spain

Received 8 May 1997; accepted 27 August 1998

Abstract

The aim of our work is the definition of compositional semantics for modular units over the class of normal logic programs. In this sense, we propose a declarative semantics for normal logic programs in terms of model classes that is monotonic in the sense that $\mathrm{Mod}(P \cup P') \subseteq \mathrm{Mod}(P)$, for any programs P and P', and we show that in the model class associated to every program there is a least model that can be seen as the semantics of the program, which may be built upwards as the least fix point of a continuous immediate consequence operator. In addition, it is proved that this least model is "typical" for the class of models of Clark–Kunen's completion of the program. This means that our semantics is equivalent to Clark–Kunen's completion. Moreover, following the approach defined in a previous paper, it is shown that our semantics constitutes a "specification frame" equipped with the adequate categorical constructions needed to define compositional and fully abstract (categorical) semantics for a number of program units. In particular, we provide a categorical semantics of arbitrary normal logic program fragments which is compositional and fully abstract with respect to the (standard) union.

Keywords: Normal logic programs; Model-theoretic semantics; Compositionality; Institutions; Monotonic semantics; Constructive negation; Modular logic programs

1. Introduction

Despite the amount of papers on the semantics of negation (see, e.g. Ref. [3]), there are several semantic issues that are insufficiently explored. One such basic issue is modularity. The reason is that a proper semantics for any kind of modular unit

* Corresponding author. Tel.: +34 93 401 3018; fax: +34 93 401 3014; e-mail: orejas@lsi.upc.es

0743-1066/99/$ – see front matter
PII: S0743-1066(98)10039-0

must be shown to be compositional with respect to the kind of module operations considered, but the non-monotonic nature of negation in Logic Programming does not seem to fit too well with compositionality. In particular, for different reasons, none of the various operational semantics [13,11,32], neither the different model-theoretic approaches (see e.g. Ref. [3]), nor the completion semantics [12,25], seems to be adequate to be the basis for defining a compositional semantics for normal logic program units. To our knowledge, only Refs. [17,19,27,35,9] provide some compositional semantic constructs for normal logic programs. In Section 6 we compare the results presented in this paper and these approaches. It must be noted that compositionality is a very important property for defining the semantics of a modular unit. In particular, if the semantics of a unit is not compositional with respect to the given module operations this means that, for reasoning about a program consisting of several such units, we would need to previously "flatten" the program (or its semantics) "forgetting" the modular structure of the program. Also, when dealing with modular units, another important property is full abstraction with respect to composition, which holds if the semantics of two modules coincide if and only if the two modules "behave" equally in every context. In particular, if a semantics is fully abstract this guarantees that our notion of program equivalence is the right one for reasoning about implementation, i.e., a program unit could be substituted by another unit implementing the same abstraction if and only if they have the same semantics.

In Ref. [29], a methodology is presented for the semantic definition of modular logic programs ensuring compositionality and full abstraction, and it is applied to study several kinds of program units for the class of definite logic programs. The approach is based on the fact that most modular constructions can be defined and studied independently of the underlying formalism used "inside" the modules, as far as this formalism is an "institution" [23] or a "specification frame" [16] (or some similar notion) equipped with some categorical constructions. In particular, the proposed methodology for defining the semantics of a certain kind of modular unit consists, essentially, of three steps. Firstly, one has to study the given unit, and the associated composition operations, at the general level. This means defining the meaning of the construction in terms of the categorical constructions that the specification frames will be assumed to provide. Secondly, one has to define the given class of logic programs as an institution or specification frame with the needed constructions. At this point one may already obtain a compositional and fully abstract semantic definition for the given unit. The categorical constructions obtained at this stage may be more abstract than required. A further third step can be the definition of an equivalent, more concrete semantics.

Even if the intermediate categorical machinery is discarded at the end, the three-step approach is instrumental in avoiding arbitrary and unfortunate choices in the concrete semantics, which then fail to have critical properties, such as monotonicity. Applying this methodology not only may save some work (since some results must be proved just once, independently of the classes of logic programs considered) but, what is more important, it provides clear guidelines about how the concrete semantics for the various constructions must be defined. In particular, these guidelines were extremely valuable for the work reported in this paper. In principle, the main problem found in order to apply this methodology to study modularity and compositionality issues for the class of normal logic programs is (the lack of) monotonicity. Institutions and specification frames can be seen as characterizations of monotonic

formalisms. This seems to be in contradiction with the non-monotonic nature of negation as failure and constructive negation. However, if we look at the simpler case of the class of definite logic programs with negative queries, then we could see one of the basic ideas of our proposal: the class of definite logic programs (Horn Clause Logic) is, obviously, a monotonic logic; the non-monotonicity of the negative queries is related to the selection of an arbitrary model (the least one) to define what is assumed to be "false". Similarly, in this paper we propose a declarative semantics for normal logic programs in terms of model classes that is monotonic in the sense that $\mathtt{Mod}(P \cup P') \subseteq \mathtt{Mod}(P)$ for any programs P and P'. This is enough for defining a specification frame of normal logic programs equipped with the categorical constructions needed to apply the results in Ref. [29] to the class of normal logic programs, obtaining compositional and fully abstract (categorical) semantics for a number of program units [8,21]. In particular, we apply these results to provide a (categorical) semantics of arbitrary program fragments which is compositional and fully abstract with respect to (standard) union. In addition, we show that in the model class associated to every program there is a least model that can be seen as the (non-compositional) semantics of the program. This least model is "typical" for the class of models of the Clark–Kunen's completion of the program. In that sense, our semantics is equivalent to Clark–Kunen's completion. Moreover, we provide a continuous immediate consequence operator and show that this least model can be built "upwards" as the least fixpoint of that operator.

In addition, in Section 5, it is proved that the class of models of a given program P forms a complete lattice. For this reason, we are convinced that, not only with respect to compositionality issues, our semantics is just the "right" kind of model-theoretic semantics for normal logic programs. In particular, if model-theoretic semantics are usually the most adequate tool for meta-logical reasoning (e.g. for proving completeness of operational approaches), the structure of our classes of models, together with the closeness to ranked resolution, makes our semantics adequate for the proof of such kind of properties.

Moreover, ranked structures can be seen as a special case of (a three-valued version of) Beth structures, used to provide semantics to intuitionistic logic (e.g., see Ref. [33]). In this sense, our semantics suggests a link, already mentioned by other authors (e.g., see Ref. [31]), between logic programming negation and intuitionistic logic that may be worthwhile to study. In particular, it could serve as a basis for extending with negation those approaches to modularity based on the use of an intuitionistic implication (e.g., see Ref. [28]). In this line, the only work we know is Ref. [6] where a semantics of programs including an intuitionistic implication and negation as failure is defined in terms of Kripke models under some severe restrictions. In particular, programs must be stratified and signatures may only contain predicate symbols, i.e., function symbols are not allowed.

The paper is organized as follows: in the next section we introduce some basic notions and notation; in Section 3, we review the basic definitions and results about specification frames and, as illustration, their application to the class of definite logic programs; in Section 4, we present the declarative model-theoretic semantics for normal logic programs, including a fix point construction of least models, and show its connection with Clark–Kunen's completion; in Section 5, we discuss the results presented with respect to compositionality and full abstraction issues; finally, in Section 6, we give some conclusions and relationships with other works.

The reader is assumed to have certain familiarity with basic constructs from category theory. For details one may consult any basic text on the subject (e.g., Refs. [5,4]).

2. Preliminaries

A countable signature Σ consists of a pair of sets (FS_Σ, PS_Σ) of function and predicate symbols, respectively, with some arity associated. Σ-terms and Σ-atoms are built using functions and predicates from Σ and, also, variables from a prefixed countable set X of variable symbols. Terms will be denoted by $t, s, \ldots$, and $var(t)$ will denote the variables appearing in t.

Normal programs over a signature Σ (or Σ-programs) are sets of Σ-clauses

$$a : -l_1, \ldots l_k,$$

where a is a Σ-atom, $k \geqslant 0$, and each l_i is a Σ-literal, that is b or $\neg b$ where b is an atom. For simplifying some technical constructions, we consider that any Σ-program is written as its equivalent constraint normal program with flat head. That is, any clause

$$p(t_1, \ldots, t_n) : -l_1, \ldots l_k$$

is written as the constraint clause

$$p(x_1, \ldots, x_n) : -l_1, \ldots l_k \square x_1 = t_1, \ldots, x_n = t_n;$$

Moreover, we suppose the identical tuple $x_1, \ldots, x_n$ of *fresh variables* occurs in all clauses (in a program) with predicate p in its head. We denote by $Hd_P(p(\bar{x}))$ the set $\{p(\bar{x}) : -\bar{l}^k \square \bar{x} = \bar{t}^k \mid k = 1, \ldots, m\}$ of all clauses with head p appearing in P.

Constraints appearing in programs are a special kind of simple constraints. In general, we consider that Σ-constraints are arbitrary first order Σ-formulas over equality atoms. That is, formulas composing equality atoms with the connectives: $\neg, \wedge, \vee, \rightarrow$, and the quantifiers: $\forall, \exists$. For a formula φ, in particular for a constraint, $free(\varphi)$ is the set of all free variables in φ, and $\varphi(\bar{x})$ specifies that $free(\varphi) \subseteq \bar{x}$. We will identify the list of constraints in any program clause with the corresponding conjunction (i.e., a formula). We denote constraints by $c, d, \ldots$ (possibly with sub- or superscripts). Formulas $\varphi^\forall, \varphi^\exists$ stand for the universal and existential closures of φ, respectively. The atomic formulas naming the two classical truth values are **T** and **F**.

We will handle constraints in a logical way, using logical consequence of the *free equality theory*. The free equality theory FET_Σ for a signature Σ is the following set of formulas:

$\forall x(x = x)$	
$\forall \bar{x} \forall \bar{y}(\bar{x} = \bar{y} \leftrightarrow f(\bar{x}) = f(\bar{y}))$	for each $f \in FS_\Sigma$,
$\forall \bar{x} \forall \bar{y}(\bar{x} = \bar{y} \rightarrow (p(\bar{x}) \leftrightarrow p(\bar{y})))$	for each $p \in PS_\Sigma \cup \{=\}$,
$\forall \bar{x} \forall \bar{y}(f(\bar{x}) \neq g(\bar{y}))$	for each pair $f, g \in FS_\Sigma$ such that $f \not\equiv g$,
$\forall x(x \neq t)$	for each Σ-term t and variable x such that $x \in var(t)$ and $x \not\equiv t$.

Besides, whenever Σ is finite, FET_Σ also includes the *weak closure domain axiom*:

$$\forall x\left(\bigvee_{f\in FS_\Sigma}\exists\bar{y}(x=f(\bar{y}))\right),$$

Then FET_Σ is a complete theory, that is $FET_\Sigma \models \varphi$ or $FET_\Sigma \models \neg\varphi$ for any Σ-sentence φ. Therefore all models of FET_Σ are elementary equivalent.

A constraint c is satisfiable (resp. unsatisfiable) if and only if $FET_\Sigma \models c^\exists$ (resp. $FET_\Sigma \models \neg(c^\exists)$). A ground substitution $\bar{x}=\bar{t}$ (where t_i are closed terms) is called a solution of a constraint c if and only if $FET_\Sigma \models (\bar{x}=\bar{t}\rightarrow c)^\forall$. A constraint d is less general than c iff $FET_\Sigma \models (d\rightarrow c)^\forall$.

From a logical point of view, programs are sets of formulas. There are, mainly, two logical ways of interpreting a normal program P. The first one, denoted by $P^\forall$, interprets every clause as the universal closure of the formula which results from substituting "commas" and $\square$ (in the clause body) by logical conjunction, and the symbol ":-" by logical implication (right-to-left). The second one is Clark's program completion, denoted by $Comp(P)$. The completion of a Σ-program P consists of the free equality theory FET_Σ together with, for each $p\in PS_\Sigma$, a *predicate completion formula*:

$$\forall\bar{x}\left(p(\bar{x})\leftrightarrow\bigvee_{k=1}^{m}\exists\bar{y}^k(\bar{x}=\bar{t}^k\wedge\bar{l}^k)\right),$$

where $\bar{y}^k$ are the variables appearing in $\bar{t}^k$ and $\bar{l}^k$ which do not belong to $\bar{x}$, and $Hd_P(p(\bar{x}))=\{p(\bar{x}):-\bar{l}^k\ \square\ \bar{x}=\bar{t}^k \mid k=1,\ldots,m\}$. In both interpretations, conjunction (resp. disjunction) of an empty set is simplified to the atomic formula **T** (resp. **F**).

However, clauses like $p:-\neg p$ are inconsistent when program completion is considered. To avoid this problem [25] proposed to interpret Clark's program completion in three-valued logic. In particular, in this logic the three truth values are true (**t**), false (**f**) or undefined (**u**); the connectives $\neg,\wedge,\vee$ are interpreted in Kleene's partial logic [24], $\leftrightarrow$ is interpreted as the identity of truth values, so it is two-valued; finally, existential quantification can be seen as infinite disjunction, and universal quantification is treated as infinite conjunction. Equality is two-valued. Currently, this interpretation of Clark's completion (from now on the Clark–Kunen completion) is considered the standard declarative meaning of normal logic programs. Anyhow, it must be noted that, in the context of completion, any three valued extension of classical implication can be considered. The reason is that implication does not appear in predicate completion formulas and FET_Σ contains only implication between two-valued formulas, i.e., the choice of a three-valued semantics for implication becomes an important matter when the program itself is treated as a logical theory. In this sense, we will use Przymusinski's implication [30]:

$\rightarrow$	$\underline{t}$	$\underline{f}$	$\underline{u}$
$\underline{t}$	$\underline{t}$	$\underline{f}$	$\underline{f}$
$\underline{f}$	$\underline{t}$	$\underline{t}$	$\underline{t}$
$\underline{u}$	$\underline{t}$	$\underline{f}$	$\underline{t}$

whose intuitive meaning is "$\varphi\rightarrow\psi$ is true if and only if whenever φ is true ψ is also true and whenever ψ is false φ is also false". Then, $\varphi\leftrightarrow\psi$ is equivalent to $(\varphi\rightarrow\psi)\wedge(\psi\rightarrow\varphi)$ and, in particular, we have that $Comp(P)\models P^\forall$. Note that the

classical equivalence $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$ does not hold. However, in the case of φ being two-valued (e.g. an equality formula) $\varphi \rightarrow \psi$ is true iff $\neg\varphi \vee \psi$ is true, so that $\varphi \rightarrow \psi$ is false iff $\neg\varphi \vee \psi$ is false or undefined.

A three-valued Σ-structure $\mathcal{A}$ consists of a universe of values A and an interpretation of every function symbol by a (total) function from A^n to A (of adequate arity n) and of every predicate symbol by a partial relation, which can be seen as a (total) function from A^n to the set of the three boolean values $\{\underline{\mathbf{t}}, \underline{\mathbf{f}}, \underline{\mathbf{u}}\}$. In that way, every closed Σ-term can be interpreted as a value belonging to the universe of a Σ-structure (they cannot be undefined), every equality ground atom $t_1 = t_2$ is associated to one of the classical truth values, but every ground atom $p(t_1, \ldots, t_n)$ is associated to one of the three boolean values: $\{\underline{\mathbf{t}}, \underline{\mathbf{f}}, \underline{\mathbf{u}}\}$.

A Herbrand three-valued structure $\mathcal{H}$ is a three-valued Σ-structure whose universe H is the Herbrand universe for Σ, function symbols are trivially interpreted and the predicate interpretation is given by a pair of disjoint sets: H^+ of true ground atoms and H^- of false ground atoms, so that any other ground atom is undefined.

The value of any first order sentence φ in a three-valued structure $\mathcal{A}$ will be denoted by $\mathcal{A}(\varphi)$. A three-valued structure $\mathcal{A}$ is a model of a set of formulas Φ, denoted by $\mathcal{A} \models \Phi$, iff $\mathcal{A}(\varphi) = \underline{\mathbf{t}}$ for any formula $\varphi \in \Phi$. Three-valued logical consequence $\Phi \models \varphi$ means that for all three-valued structure $\mathcal{A}$ if $\mathcal{A} \models \Phi$ then $\mathcal{A} \models \varphi$.

3. The algebraic framework

In this section we review some basic notions on algebraic specification needed in this paper (for further detail see e.g. Refs. [15,36] and also Refs. [22,23,14] for more detail on institutions and specification frames).

In Section 3.1, we introduce the notion of specification frame. A specification frame can be seen as a formal description of a logic formalism with certain compositionality properties. From our point of view, this notion provides an adequate theoretical framework for studying structuring issues in logic programming. In the following section, we introduce some algebraic properties of specification frames which are specially interesting for our work. In particular, these properties allow us to study different structuring constructs at the abstract level, that means, independently of the concrete class of logic programs used to build modular or structured logic programs. In order to show the gains of using this framework, we present, in Section 3.3, a compositional and fully abstract semantics with respect to the union of logic programs [29]. These results are obtained independently from the concrete class of logic programs considered as long as the required algebraic properties are satisfied. Throughout the section, we illustrate the introduced notions with the results obtained in Ref. [29] for Horn Clause Logic.

3.1. Specification frames

The notion of specification frame was introduced in Ref. [16] to axiomatize formalisms with certain basic compositionality properties, in order to study the structuring and modularization of specifications with independence of any logic formalism. The notion was defined as a "slight" abstraction of the notion of institu-

tion [22] defined, some years before, by Goguen and Burstall with similar aims. That idea was connected with the design of the Clear specification language [10]. In particular, Clear was defined as providing operations for structuring specifications independently of the underlying logic.

Specification frames are indexed categories that satisfy some additional structural properties:

Definition 3.1 *A specification frame* $\mathcal{SF}$ is a pair (Spec, Mod), where

- Spec is a category of abstract specifications (or programs), and
- Mod : $\mathrm{Spec}^{op} \rightarrow$ Cat is a functor, that associates to every specification SP in Spec its category of models Mod(SP), and to every specification morphism $f : SP1 \rightarrow SP2$ a functor Mod(f) : Mod($SP2$) $\rightarrow$ Mod($SP1$), usually denoted by V_f,

such that the following two properties are satisfied:

(a) Spec has pushouts

(b) Mod transforms pushouts in Spec into pullbacks in Cat (i.e., $\mathcal{SF}$ has amalgamations).

Remark 3.1. (1) Pushouts are the operations that allow us to combine specifications, while amalgamation is the semantic counterpart to pushouts.

In particular, pushouts in the category of specifications correctly capture the required notion of combination of specifications with a common sub-specification, in a general way. Pushouts are diagrams in the category of specifications. Essentially, if we want to put together two specifications $SP1$ and $SP2$, having a common subspecification $SP0$, the pushout $SP3$ (of $SP1$ and $SP2$, with respect to $SP0$) would provide the right combination. Almost all logics of practical interest have pushouts (see Ref. [15] for more detail).

Amalgamation allows us to define the semantics of a combined specification purely on the semantic level as the amalgamation of the model classes of the specifications which are combined. The reason is that, as we show below, given a pushout of specifications as in the diagram of Fig. 1, amalgamation can be characterized as an operation for "building" the models of $SP3$ in terms of the models of $SP0$, $SP1$ and $SP2$.

Most logics have amalgamation. This is the case, for instance, of Horn Clause Logic ($\mathcal{HCL}$), Equational Logic ($\mathcal{EQL}$), Conditional Equational Logic ($\mathcal{CEQL}$), Clausal Logic ($\mathcal{CL}$), and First Order Logic ($\mathcal{FOL}$).

(2) It must be noted that the functorial character of Mod, usually, implies that specification frames are monotonic formalisms. In particular, if we consider a

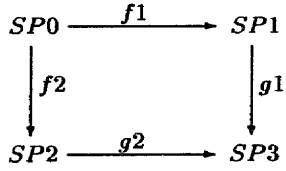

Fig. 1. Pushout diagram.

specification frame where specifications are pairs (Σ, Φ) (where Σ is some kind of signature and Φ is a set of axioms over that signature), then for any sets of formulas Φ and Φ' over Σ:

$$\mathtt{Mod}(\Sigma, \Phi \cup \Phi') \subseteq \mathtt{Mod}(\Sigma, \Phi)$$

when specification inclusions, as $(\Sigma, \Phi) \subseteq (\Sigma, \Phi \cup \Phi')$, are considered morphisms in Spec.

Theorem 3.1 [16]. *Given* $\mathscr{SF} = (\mathtt{Spec}, \mathtt{Mod})$, Mod *transforms pushouts in* Spec *into pullbacks in* Cat *iff for every pushout diagram in* Spec, *given in Fig.* 1, *the following three facts hold*:

(i) *For every* $\mathscr{A}i \in \mathtt{Mod}(SPi)$ $(i = 0, 1, 2)$ *such that* $V_{f1}(\mathscr{A}1) = \mathscr{A}0 = V_{f2}(\mathscr{A}2)$ *there is a unique* $\mathscr{A}3 \in \mathtt{Mod}(SP3)$, *called amalgamation of* $\mathscr{A}1$ *and* $\mathscr{A}2$ *via* $\mathscr{A}0$, *written* $\mathscr{A}3 = \mathscr{A}1 +_{\mathscr{A}0} \mathscr{A}2$, *such that we have*:

$$V_{g1}(\mathscr{A}3) = \mathscr{A}1 \quad \textit{and} \quad V_{g2}(\mathscr{A}3) = \mathscr{A}2.$$

(ii) *Conversely, every* $\mathscr{A}3 \in \mathtt{Mod}(SP3)$ *has a unique decomposition*

$$\mathscr{A}3 = V_{g1}(\mathscr{A}3) +_{V_{g1} \circ f1(\mathscr{A}3)} V_{g2}(\mathscr{A}3).$$

(iii) *Similar properties to* 1 *and* 2 *above hold if we replace objects* $\mathscr{A}i$ *by morphisms* h_i *in* $\mathtt{Mod}(SPi)$ *(for* $0 \leqslant i \leqslant 3$*), leading to a unique amalgamated sum of morphisms* $h3 = h1 +_{h0} h2$ *with* $V_{g1}(h3) = h1$ *and* $V_{g2}(h3) = h2$.

The next example defines a specification frame for Horn Clause Logic over a predefined (universal) signature of functions. After defining it, we will analyze its properties.

Example 3.1. Horn Clause Logic over the functions signature *FS* can be defined as the specification frame, $\mathscr{HCL} = (\underline{HCL}, \mathtt{Mod})$, where:

- Specifications are pairs (PS, Φ) formed by a signature of predicates and a set of Horn clauses over *FS* and *PS*, and specification morphisms $h: (PS, \Phi) \to (PS', \Phi')$, are mappings, $h: PS \to PS'$ such that (1) arities are preserved and (2) $h^{\#}(\Phi) \subseteq \Phi'$, up to renaming of variables, where $h^{\#}$ denotes the translation induced by h.
- (PS, Φ)-models, in $\mathscr{HCL}$ are Herbrand structures, i.e. sets of atoms over *FS* and *PS*, that satisfy the axioms in Φ (according to the standard notion of satisfaction). A (PS, Φ)-homomorphism between (PS, Φ)-models, $f: \mathscr{A}1 \to \mathscr{A}2$, is just an inclusion, $\mathscr{A}1 \subseteq \mathscr{A}2$. Then, $\mathtt{Mod}: \underline{HCL}^{op} \to \mathtt{Cat}$ maps every specification (PS, Φ) in $\mathscr{HCL}$ into the category of all (PS, Φ)-models and (PS, Φ)-homomorphisms, and every specification morphism $h: (PS, \Phi) \to (PS', \Phi')$ into the corresponding forgetful functor $V_h: \mathtt{Mod}(PS', \Phi') \to \mathtt{Mod}(PS, \Phi)$ defined as usual, i.e. for every PS'-model $\mathscr{A}'$ we define $V_h(\mathscr{A}')$ as the set of atoms whose translation via h is in A', i.e.:

$$V_h(\mathscr{A}') = \{a \in Atoms(PS) / h^{\#}(a) \in \mathscr{A}'\}$$

- Also, if $f': \mathscr{A}1' \to \mathscr{A}2'$ is a homomorphism in $\mathtt{Mod}(PS', \Phi')$, i.e. $\mathscr{A}1'$ is included in $\mathscr{A}2'$, then $V_h(f')$ is the inclusion $V_h(\mathscr{A}1') \subseteq V_h(\mathscr{A}2')$.

Let $P0 = (PS0, \mathscr{C}0)$, $P1 = (PS1, \mathscr{C}1)$ and $P2 = (PS2, \mathscr{C}2)$ be programs in *HCL*, with $h1 : P0 \to P1$ and $h2 : P0 \to P2$. If $h1$ and $h2$ are inclusions and $PS1 \cap PS2 = PS0$ then the pushout of $P1$ and $P2$ is just $P3 = P1 \cup P2$, i.e. $(PS1 \cup PS2, \mathscr{C}1 \cup \mathscr{C}2)$. In the general case, a pushout is a kind of disjoint union where the symbols in $PS2$, but not in $PS0$, are renamed adequately and the morphisms $g1$ and $g2$ map each symbol in $P1$ and $P2$, respectively, into the corresponding symbol in $P3$.

Given the pushout diagram of Fig. 2, for every $\mathscr{A}i \in \mathtt{Mod}(Pi)$ $(0 \leqslant i \leqslant 2)$, with $V_{h1}(\mathscr{A}1) = V_{h2}(\mathscr{A}2) = \mathscr{A}0$, the amalgamation of $\mathscr{A}1$ and $\mathscr{A}2$ via $\mathscr{A}0$, that is $\mathscr{A}3 = \mathscr{A}1 +_{\mathscr{A}0} \mathscr{A}2$, is defined just as $\mathscr{A}1 \cup \mathscr{A}2$, whenever $h1$ and $h2$ are inclusions and $PS1 \cap PS2 = PS0$. In the general case, $\mathscr{A}3$ would be $\mathscr{A}3 = g1^{\#}(\mathscr{A}1) \cup g2^{\#}(\mathscr{A}2)$.

Horn Clause Logic, $\mathscr{HCL}$, seems to be the most obvious choice for a specification frame for defining the (declarative) semantics of definite logic programs. Actually, this is (implicitly) done by most authors. In particular, the "standard" declarative meaning of a logic program P is defined as the least Herbrand model of P (see, for instance, Refs. [26,1]). In algebraic terms, this is equivalent to defining the semantics of P as the least (initial) model in $\mathtt{Mod}(P)$. However, if we are interested in logic programming languages *as programming languages*, then a reasonable choice would be one in which the input/output behaviour of programs were better captured. In that sense, Ref. [29] provides the definition of another specification frame, $\mathscr{DLP}$ for Definite Logic Programs, which, obviously, shares the syntax with $\mathscr{HCL}$, i.e., it has the same category of programs, but it is based on different notions of model and satisfaction.

3.2. Other properties of specification frames

In this subsection, we present some other properties of specification frames that may be required when studying specific structuring or modular constructs. As we have already mentioned, the satisfaction of these properties provides the adequate setting for proving some usually desired semantic properties for these constructs. Moreover, that can be made independently of the underlying logic formalism (used to build specifications or programs) whenever this formalism is a specification frame. In what follows, we also sketch, as an example, that these properties hold for the specification frame $\mathscr{HCL}$.

Definition 3.2. A specification frame $\mathscr{SF} = (\underline{\mathtt{Spec}}, \mathtt{Mod} : \underline{\mathtt{Spec}}^{op} \to \underline{\mathtt{Cat}})$ *has free constructions* iff for every specification morphism $f : SP1 \to SP2$ in $\underline{\mathtt{Spec}}$ there is a

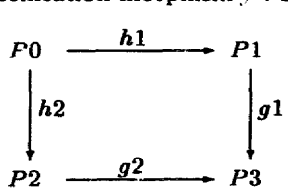

Fig. 2. $\mathscr{HCL}$-pushout diagram.

free functor F_f : Mod($SP1$) → Mod($SP2$) which is left adjoint to V_f. F_f (and, in general, any functor F : Mod($SP1$) → Mod($SP2$)) is *strongly persistent* iff $V_f \circ F_f = !D$.

The intuition of the free construction, in this context, is quite simple. Consider the case where f is an inclusion of programs (specifications): $P \subseteq P'$. The free construction associated to this inclusion would build, for each model $\mathscr{A}$ of P, the least P'-model that can be built over $\mathscr{A}$, i.e., if P and P' are definite logic programs $F(\mathscr{A})$ is the least model associated to $P' \cup \mathscr{A}^*$, where $\mathscr{A}^*$ denotes the program consisting of all the atoms in $\mathscr{A}$. If the morphism is more general than an inclusion (i.e., it defines some form of translation between the signatures of P and P') then, similarly, $F(\mathscr{A})$ could also be defined as the least model associated to $P' \cup \mathscr{A}^*$, where $\mathscr{A}^*$ would mean here the program consisting of the corresponding translation of all atoms in $\mathscr{A}$.

It may be noticed that the existence of free constructions in a given specification frame, in general, implies the existence of "initial" models (least models). Since the least model of a program P can be defined as $F(\emptyset)$ where $\emptyset$ denotes the empty model and F is the free construction associated to the inclusion $\mathscr{E} \subseteq P$ where $\mathscr{E}$ denotes the empty program.

Conversely, it can be shown that for most specification frames the existence of initial (least) models associated to every specification (or program) ensures the existence of free constructions.

Example 3.2 (*Properties of* $\mathscr{HCL}$, [29]). $\mathscr{HCL}$ has free constructions.

It is easy to see that, given a program $P = (PS, \mathscr{C})$ of $\mathscr{HCL}$, the category Mod(P) is closed under intersection. This means that there is a least model $\mathscr{M}_P$ in Mod(P) which happens to be trivially initial, according to the notion of homomorphism used (inclusions) in the categories of models. Therefore, in the case of $\mathscr{HCL}$, the existence of free constructions is a consequence of the existence of initial objects. In particular, given a morphism $h : P \to P'$, with $P = (PS, \mathscr{C})$ and $P' = (PS', \mathscr{C}')$, the free construction F_h : Mod(P) → Mod(P') can be defined for every $\mathscr{A}$ in Mod(P) as the initial model of the program $(PS', \mathscr{C}' \cup h^{\#}(\mathscr{A}))$, denoted $\mathscr{M}_{P(\mathscr{A})}$, where $h^{\#}(\mathscr{A})$ is the program consisting of the translation through h of all the atoms in $\mathscr{A}$.

Free constructions have been used at the model level to give semantics to parameterized specifications. In Ref. [29] free constructions are considered as the semantics of the different kinds of open (or modular) logic programs. Horn Clause Logic ($\mathscr{HCL}$), Equational Logic ($\mathscr{E}2\mathscr{L}$) and Conditional Equational Logic ($\mathscr{CE}2\mathscr{L}$) have free constructions (see Ref. [15]). In contrast Clausal Logic ($\mathscr{CL}$) and First Order Logic ($\mathscr{FOL}$), in general, do not.

Definition 3.3. A specification frame $\mathscr{SF} = (\underline{\text{Spec}}, \text{Mod})$ *has free extensions* iff for every pushout diagram in $\underline{\text{Spec}}$ as Fig. 1, if $F : \text{Mod}(SP0) \to \text{Mod}(SP1)$ is a strongly persistent free functor with respect to $f1$, then there is a strongly persistent functor $F^* : \text{Mod}(SP2) \to \text{Mod}(SP3)$, called the *extension of* F *via* $f2$, such that:

(a) F^* is free with respect to $g2$.

(b) The diagram of Fig. 3 commutes.

Extension may be, in some cases, a key construction for proving compositionality and full abstraction results. This is the case, in particular, when the semantics of the

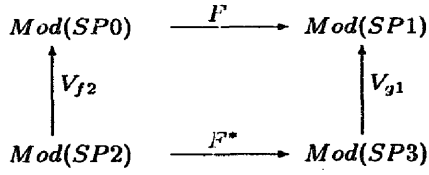

Fig. 3. Free extension diagram.

given construction is expressed as a persistent free functor. Every logic having amalgamations has also free extensions.

Theorem 3.2 [16]. *Specification frames have free extensions.*

This result is a consequence of the existence of amalgamation. Being more concrete, if F : Mod($SP0$) → Mod($SP1$) is a strongly persistent free functor with respect to $f1$ then the extension of F via $f2$ is the strongly persistent free functor F^* : Mod($SP2$) → Mod($SP3$), such that for each model $\mathscr{A}2$ in Mod($SP2$), $F^*(\mathscr{A}2)$ is the amalgamated sum $\mathscr{A}2 +_{V_{f2}(\mathscr{A}2)} F(V_{f2}(\mathscr{A}2))$.

Example 3.3 (*Properties of* $\mathscr{HCL}$, [29]). $\mathscr{HCL}$ has free extensions, since it has amalgamations.

The existence of extensions for strongly persistent free functors can be generalized to the non-persistent case under certain circumstances:

Definition 3.4. A specification frame $\mathscr{SF} = (\underline{\text{Spec}}, \text{Mod})$ *has generalized free extensions* iff for every pushout diagram as in Fig. 1, if F : Mod($SP0$) → Mod($SP1$) is a free functor with respect to $f1$, then there is a functor F^* : Mod($SP2$) → Mod($SP3$), called the *generalized extension of* F *via* $f2$, such that:

(a) F^* is free with respect to $g2$.

(b) There is a natural transformation v : $F \circ V_{f2} \to V_{g1} \circ F^*$ such that the diagram of natural transformations in Fig. 4 commutes, where $f3 = g1 \circ f1 = g2 \circ f2$ and u and u^* are, respectively, the universal transformations associated to F and F^*.

Theorem 3.3 [23]. *If a specification frame* $\mathscr{SF}$ *has free constructions and pushouts in all model categories* Mod(SP), *for all abstract specifications* SP *in* $\underline{\text{Spec}}$, *then* $\mathscr{SF}$ *has generalized free extensions.*

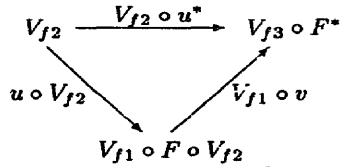

Fig. 4. Natural transformations associated to a generalized free extension.

Example 3.4 (*Properties of $\mathscr{HCL}$*, [29]). $\mathscr{HCL}$ has generalized free extensions because, according to Definition 4, it is enough to check that for every program P, there are pushouts in $\mathtt{Mod}(P)$: Given models $\mathscr{A}0, \mathscr{A}1, \mathscr{A}2$ in $\mathtt{Mod}(P)$, with $f1 : \mathscr{A}0 \subseteq \mathscr{A}1$ and $f2 : \mathscr{A}0 \subseteq \mathscr{A}2$, we can define the pushout of $\mathscr{A}1$ and $\mathscr{A}2$ along $f1$ and $f2$ as just the join $\mathscr{A}1 \sqcup \mathscr{A}2$.

3.3. Standard union of logic programs

In this section, we present compositionality and full abstraction results [29] for a semantics of the standard union of logic programs, which are general in the sense that they are independent of the class of logic programs considered, as long as it is a specification frame with the properties introduced in Section 3.2.

As is well known, the least model semantics of logic programs is neither compositional nor fully abstract (in a compositional way). As a result, some form of more complex semantics must be considered if we intend to capture a compositional behaviour. For instance, Ref. [21] studies the (standard) union of logic programs and the composition of *logic modules*, where a logic module can be seen as a logic program including an additional *importlexport interface*, with the restriction that clauses in the module do not include imported predicates in their heads. In both cases, the meaning of these constructions is defined in terms of sets of minimal clauses, that are logical consequences of the given program. In our context, we can see these meanings as concrete representatives of our general algebraic constructions. In this sense, the full abstraction results in Ref. [21] can be seen just as ad hoc versions of variations of the results obtained in Ref. [29].

In our approach, for studying the operation of union, we consider that a logic program $P = (PS, \mathscr{C})$ may be seen as a special kind of open program where all predicates are partially defined, in the sense that more information about the predicates in PS_{PS}, can be added by union with other programs. In our context, this implies that the meaning of a program P can be seen as a mapping that given a PS-structure $\mathscr{A}$ (that can be seen as including the "missing" definitions of the predicates in P), yields as result the "complete" interpretation of P, i.e. we may consider that the meaning of P is the free construction associated to the program inclusion: $(PS, \emptyset) \subseteq P$.

Definition 3.5. The *semantics of a program* $P = (PS, \mathscr{C})$, noted by $Sem(P)$, is the free functor $F : \mathtt{Mod}(PS, \emptyset) \to \mathtt{Mod}(P)$, associated to the inclusion $(PS, \emptyset) \subseteq P$.

It may be noted that, in this case, the semantics of P is never a persistent functor, since given a program P and a PS-model $\mathscr{A}$, $F(\mathscr{A})$ is in general different from $\mathscr{A}$.

Definition 3.6. Let $P1 = (PS1, \mathscr{C}1)$ and $P2 = (PS2, \mathscr{C}2)$ be programs, the standard union of $P1$ and $P2$, $P1 \cup P2$, is the program $(PS1 \cup PS2, \mathscr{C}1 \cup \mathscr{C}2)$.

It must be noted that $P1 \cup P2$ coincides with the result of the pushout diagram, in the category of programs of the underlying specification frame, given by Fig. 5. Fig. 6

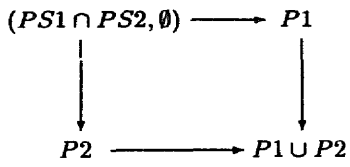

Fig. 5. Standard union of programs.

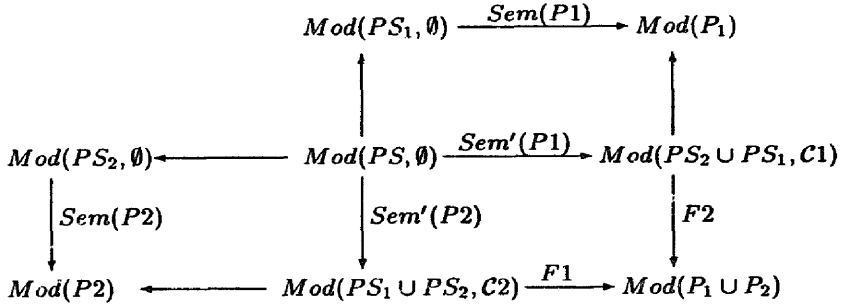

Fig. 6. Union compositionality.

Dealing with programs whose semantics is persistent, compositionality of our semantics, with respect to standard union, is a direct consequence of the existence of free extensions in the specification frame. However, in the general case, we have to use the more complex construction of generalized free extension.

Theorem 3.4 (Compositionality, [29]). *The semantics of* $P1 \cup P2$ *can be obtained as*: $Sem(P1 \cup P2) = F1 \circ Sem'(P2) = F2 \circ Sem'(P1)$, *where*

(i) $PS = PS1 \cup PS2$.

(ii) $Sem'(P1)$ *and* $Sem'(P2)$ *are the generalized extensions of* $Sem(P1)$ *and* $Sem(P2)$ *via the inclusions* $(PS1, \emptyset) \subseteq (PS, \emptyset)$ and $(PS2, \emptyset) \subseteq (PS, \emptyset)$, *respectively*.

(iii) $F1$ *and* $F2$ *are the generalized extensions of* $Sem'(P1)$ *and* $Sem'(P2)$ *via the inclusions* $(PS, \emptyset) \subseteq (PS, \mathscr{C}1)$ *and* $(PS, \emptyset) \subseteq (PS, \mathscr{C}2)$, *respectively*.

It must be noted that Theorem 3.4 really proves the compositionality of *Sem* with respect to standard union, in the sense that the meaning of $P1 \cup P2$ is defined in terms of the meaning of $P1$ and $P2$, since the generalized extension of free functor F via an inclusion i, is uniquely determined by F and i.

On the other hand, the following lemma is a consequence of the fact that free constructions are unique up to natural isomorphism:

Lemma 3.1 [29]. *Given two programs* $P1$ *and* $P2$,

$$Sem(P1) = Sem(P2) \quad \textit{iff for every } P : Sem(P \cup P1) = Sem(P \cup P2).$$

This lemma can be used to prove full abstraction of the given semantics. In particular, a semantic definition of a program unit is fully abstract with respect to a

given composition operation, for instance $\cup$, and a given observation criteria *Obs* if and only if for all programs $P1$ and $P2$

$$Sem(P1) = Sem(P2) \quad \text{iff for every } P : Obs(P \cup P1) = Obs(P \cup P2).$$

Now, there are several observation criteria that may be used in the context of logic programming. The most obvious one is to consider two programs $P1$ and $P2$ observationally equivalent if and only if the ground consequences of the two programs coincide, or equivalently if and only if their associated least models coincide (in Ref. [29] it is also considered observations associated to the computed answers of the given programs). In this sense, full abstraction can be reformulated as:

$$Sem(P1) = Sem(P2) \quad \text{iff for every } P : T_{P \cup P1} = T_{P \cup P2}.$$

where T_P denotes the initial model of P in the corresponding specification frame, for example, if the underlying specification frame is $\mathscr{HCL}$ then T_P is the minimal Herbrand model of P, that is, $T_P = \mathscr{H}_P$.

The abstract result of full abstraction works for all "algebraic" specification frames (in particular $\mathscr{HCL}$ is algebraic).

Definition 3.7. A specification frame $\mathscr{SF} = (\underline{\texttt{Spec}}, \texttt{Mod})$ is *algebraic* if for each specification SP in $\underline{\texttt{Spec}}$ and for each model $\mathscr{A}$ in $\texttt{Mod}(P)$ there exists a specification $SP0$ such that $\mathscr{A} = T_{SP0}$, where T_{SP0} denotes the initial model of $SP0$.

Theorem 3.5 (Full abstraction, [29]). *Let $\mathscr{SF} = (\underline{Prog}, \texttt{Mod})$ be an algebraic specification frame. Then, given two programs $P1$ and $P2$ in $\underline{Prog}$,*

$$Sem(P1) = Sem(P2) \quad \textit{iff for every } P : T_{P \cup P1} = T_{P \cup P2}.$$

In Ref. [29], these results are used to analyze and improve previous ones. More specifically, with respect to standard union, it is proved that the semantics proposed in Ref. [21] is equivalent to the above "abstract" semantics: this allows us to conclude that their semantics is not only fully abstract, as they prove, but also compositional. Being more concretely, the semantics of a logic program $P = (PS, \mathscr{C})$, as defined in Ref. [21], can be seen as a specific representative of the free construction associated to the inclusion $(PS, \emptyset) \subseteq (PS, \mathscr{C})$ in the specification frame $\mathscr{HCL}$. Then, the full abstraction results of Ref. [21] are just a consequence of the results in Section 3.2 applied to the specification frame $\mathscr{HCL}$. On the other hand, according to these results, the compositionality of the semantics, with respect to the union, is a consequence of Theorem 3.4.

4. A model-theoretic semantics for normal logic programs

As said in Section 1, our aim is to define a model-theoretic semantics for normal logic programs (i.e., the meaning of a program P is the set $\texttt{Mod}(P)$ of all models of P, for a given notion of model), such that the following monotonicity property holds

$$\texttt{Mod}(P) \supseteq \texttt{Mod}(P \cup P') \quad \text{for all } P, P'.$$

In addition, we also want this semantics to be adequate for applying the general results presented in the previous section. This means that it must be possible, based on this semantics, to define a specification frame satisfying all the properties needed for defining the meaning of the kind of programs units considered. In particular, this means that this specification frame must have free constructions and, as a consequence, every program P must have a least model, denoted $\mathcal{M}_P$, that could be considered its standard meaning. On the other hand, obviously, this semantics should be proved equivalent to the standard meaning associated to normal logic programs.

An obvious choice is to consider that the models of a program P are three-valued structures. Then, one would try to find some ordering $\preceq$ among models satisfying that there is a least element that can be proved equivalent to the intended meaning of P. Unfortunately, as the following counter-example shows, this is not possible.

Example 4.1. Let us consider the normal program $P1 \equiv \{a : -\neg b\}$, its least model $\mathcal{M}_{P1}$ should be the pair $(\{a\}, \{b\})$, and consider $P1' \equiv \{b : -\}$, then $\mathcal{M}_{P1 \cup P1'} \equiv (\{b\}, \{a\})$. Then we must have $(\{a\}, \{b\}) \preceq (\{b\}, \{a\})$.

Now, by considering the program $P2 \equiv \{b : -\neg a\}$ and extending it with the clause $\{a : -\}$ we obtain that $(\{b\}, \{a\}) \preceq (\{a\}, \{b\})$ should hold.

From our point of view, the problem in this counter-example is that $\mathcal{M}_{P1 \cup P1'}$ and $\mathcal{M}_{P2}$ should not be identical and should reflect, in some sense, the "dependences from negative information" which make a given atom be in the model. For instance, $\mathcal{M}_{P2}$ includes b as a consequence of the negative information provided by a, while $\mathcal{M}_{P1 \cup P1'}$ includes b without any dependency of negative information. This consideration has led us to consider models having "layers" that reflect these dependencies. We call these models *ranked structures* because of their relation with ranked resolution. For instance, if we consider again the above Example 4.1, the "intended" model for $P1$ has a first layer given by $(\emptyset, \{b\})$ and a second layer $(\{a\}, \{b\})$. However for $P1 \cup P1'$ the first layer is $(\{b\}, \emptyset)$, and the second layer $(\{b\}, \{a\})$. Similarly, for $P2$ the first layer is $(\emptyset, \{a\})$ and the second layer is $(\{b\}, \{a\})$. Now the intended models associated to $P1 \cup P1'$ and to $P2$ are different, since their first layers differ.

In what follows, first we sketch the propositional case to provide some intuition. In Section 4.2 we extend the already presented semantical notions to the class of all normal logic programs. Then, in Section 4.3 we prove the existence of a least model and we provide a continuous immediate consequence operator for obtaining it in a bottom–up constructive way. Finally, we show the equivalence of our semantics with Clark–Kunen semantics, proving that our least model is a "typical" element in the class of all models of program completion.

4.1. A first approach: The propositional case

In the propositional case, it is enough to consider sequences of Herbrand three-valued Σ-structures. In the next section we extend this notion of semantical structure to deal with normal programs with variables.

Definition 4.1 (*Propositional case*). A *ranked three-valued Σ-structure* $\mathcal{M}$ is an infinite sequence of pairs $\langle (A_i^+, A_i^-) \rangle_{i \in \mathbb{N}}$ such that for any $i \in \mathbb{N}$:

- $A_i^+ \subseteq A_{i+1}^+$ and $A_i^- \subseteq A_{i+1}^-$
- $A_i^+ \cap A_i^- = \emptyset$.

We will just write a finite number n of layers, whenever the rest of the layers are equal to the nth layer.

The layers of our structures could also be related to the notion of stratification [2,34], but stratification is a syntactic restriction on the class of programs for ensuring the existence of certain semantic constructions, whereas ranked structures are models. Actually, as it can be seen below, we do not impose any restriction on the kind of programs we deal with (they do not have to be stratified in any sense).

Now, we define when one of these structures is a model of a program. In order to distinguish the satisfaction relation between ranked structures and programs and the logical consequence relation in the three-valued logic, the former will be denoted by $\models_R$ and the latter by $\models_3$.

Definition 4.2. A ranked three-valued Σ-structure $\mathcal{M}$ is a *model of a propositional normal Σ-program P* (denoted by $\mathcal{M} \models_R P$) iff the following four conditions are satisfied:

(a) If $P^\forall \cup A_0^+ \models_3 a$ then $a \in A_0^+$ (in particular if $a: - \in P$).
(b) If $a \in A_0^-$ then there is not any clause $a: -\bar{l}$ in P.
(c) If $P^\forall \cup A_{i+1}^+ \cup \neg A_i^- \models_3 a$ then $a \in A_{i+1}^+$, where $\neg A_i^-$ *means* $\{\neg a | a \in A_i^-\}$.
(d) If $a \in A_{i+1}^-$ then for every $a: -\bar{l} \in P$ one of the following two facts holds:
- there exists $b \in \bar{l}$ such that $b \in A_i^-$,
- there exists $\neg b \in \bar{l}$ such that $b \in A_i^+$.

Notice that for the program $P1 \equiv \{a: -\neg b\}$ of Example 4.1, the following are some of its models:

$\mathcal{M}1 = \langle(\emptyset, \{b\}), (\{a\}, \{b\})\rangle$
$\mathcal{M}2 = \langle(\{a\}, \{b\})\rangle$
$\mathcal{M}3 = \langle(\{a, b\}, \emptyset)\rangle$
$\mathcal{M}4 = \langle(\emptyset, \emptyset)\rangle$
$\mathcal{M}5 = \langle(\{b\}, \emptyset), (\{b\}, \{a\})\rangle$

but $\mathcal{M}6 = \langle(\{b\}, \{a\})\rangle$ is not a model of P.

Our model notion allows us to include (in any layer) more positive information than what is supported as logical consequence of the previous layers, but the negative information of each layer must be supported (in that sense). Thus, if we want to define an ordering $\preceq$ on ranked structures such that the least model is the one having, at each layer, the least amount of positive information and the greatest amount of negative information supported by the previous layer, it suffices to take $\preceq$ to be the lexicographic extension over sequences $\langle(A_i^+, A_i^-)\rangle_{i \in \mathbb{N}}$ of the standard ordering over three-valued structures:

$$(A^+, A^-) \preceq (B^+, B^-) \quad \text{iff } A^+ \subseteq B^+ \quad \text{and} \quad A^- \supseteq B^-.$$

It is easy to see that, for the above program $P1$, $\mathcal{M}1$ is the $\preceq$-least model in $\mathrm{Mod}(P1)$. Now, consider the case where we add the clause $b: -b$ to $P1$, then $\mathcal{M}1$ and $\mathcal{M}2$ are not models of the new program. In this case, the least model is $\mathcal{M}4$. Furthermore, by adding a third clause $b: -$, $\mathcal{M}4$ is not a model of the new program $\{a: -\neg b,\ b: -b,\ b: -\}$, and the least model would now be $\mathcal{M}5$.

4.2. Normal logic programs

In this section, we extend the model-theoretic semantics to the general case of normal programs with variables. Firstly, it must be noticed that this extension can not just be based on seeing normal programs with variables as abbreviations for programs including all possible ground instances of the given clauses. For instance, the programs

$$P1 \equiv \{nat(0) : -,\ nat(s(x)) : -nat(x)\} \text{ and } P2 = \{nat(x) : -\}$$

have exactly the same instances (considering the signature including, as unique function symbols, the constant 0 and the unary function symbol s), but they have a completely different behavior. In particular, the query

$$? - \neg nat(x)$$

would be undefined for $P1$ and false for $P2$. The solution proposed is rather to handle the first-order case in a similar manner to the propositional case, by considering ranked structures including not just ground atoms but constrained atoms with variables.

Definition 4.3. *A ranked three-valued* Σ-structure is an infinite sequence

$$\mathscr{A} = \langle (A_i^+, A_i^-) \rangle_{i \in \mathbb{N}}$$

such that for any $i \in \mathbb{N}$:

- A_i^+ and A_i^- are sets of pairs $p(\bar{x})\Box c(\bar{x})$, where $p \in PS_\Sigma$ and $c(\bar{x})$ is a satisfiable Σ-constraint.
- A_i^+ and A_i^- are closed under renaming of variables.
- $A_i^+ \subseteq A_{i+1}^+$ and $A_i^- \subseteq A_{i+1}^-$.
- (Consistency Property) For any $p \in PS_\Sigma$, if there exists Σ-constraints c and d such that $p(\bar{x})\Box c \in A_i^+$ and $p(\bar{x})\Box d \in A_i^-$, then $c \wedge d$ is unsatisfiable.

We will not make explicit the free variables of a constrained atom whenever they are not relevant and we will just write a finite number n of layers whenever rest of the layers are identical to the nth layer.

A pair $p\Box c \in A_i^+$ is logically interpreted as the formula $(c \rightarrow p)^\forall$, and a pair $p\Box c \in A_i^-$ has the logical meaning of $(c \rightarrow \neg p)^\forall$. Consequently, we define the sets:

$$A_i^{+\forall} \equiv \{(c \rightarrow p)^\forall | p\Box c \in A_i^+\},$$
$$A_i^{-\forall} \equiv \{(c \rightarrow \neg p)^\forall | p\Box c \in A_i^-\},$$
$$A_i^\forall \equiv A_i^{+\forall} \cup A_i^{-\forall}.$$

Definition 4.4. A ranked three-valued Σ-structure $\mathscr{A}$ is a *model of a normal* Σ*-program* P (denoted by $\mathscr{A} \models_R P$) iff the following four conditions are satisfied:

(a) If $FET_\Sigma \cup P^\forall \cup A_0^{+\forall} \models_3 (c \rightarrow p)^\forall$ and c is satisfiable, then $p\Box c \in A_0^+$.

(b) If $p(\bar{x})\Box c(\bar{x}) \in A_0^-$ then $c \wedge c'$ is unsatisfiable for every (properly renamed) clause $p(\bar{x}) : -l\Box c' \in P$.

(c) If $FET_\Sigma \cup P^\forall \cup A_{i+1}^{+\forall} \cup A_i^{-\forall} \models_3 (c \rightarrow p)^\forall$ and c is satisfiable, then $p\Box c \in A_{i+1}^+$.

(d) If $p(\bar{x})\Box c \in A_{i+1}^-$ then $FET_\Sigma \cup A_i^\forall \models_3 ((c \wedge c') \rightarrow \neg \bar{l}))^\forall$ for every (properly renamed) clause $p(\bar{x}) : -\bar{l}\Box c' \in P$.

Remark 4.1. Conditions (a) and (c) can be slightly simplified to:

(a') If $FET_\Sigma \cup P^\forall \models_3 (c \rightarrow p)^\forall$ and c is satisfiable, then $p\Box c \in A_0^+$

(c') If $FET_\Sigma \cup P^\forall \cup A_i^{-\forall} \models_3 (c \rightarrow p)^\forall$ and c is satisfiable, then $p\Box c \in A_{i+1}^+$

if we would not have the aim of proving that this semantics defines a specification frame. Unfortunately, properties (a) and (c) are needed for proving the so-called amalgamation property of specification frames.

Now, we can define a model theoretic semantics for normal programs, in terms of the class of models, for a program P:

$$\mathtt{Mod}(P) = \{\mathscr{M} | \mathscr{M} \models_R P\}.$$

This semantics is monotonic with respect to program extension.

Theorem 4.1. *For all Σ-programs P, P'. $\mathtt{Mod}(P) \supseteq \mathtt{Mod}(P \cup P')$.*

Proof. Suppose that $\mathscr{M} \models_R P \cup P'$, for proving that $\mathscr{M} \models_R P$ conditions (b) and (d) are trivial. In order to prove conditions (a) and (c), it is enough to observe that $(P \cup P')^\forall \equiv P^\forall \cup P'^\forall$, which means that, for any set of formulas $\Phi \cup \{\varphi\}$, the following holds: if $FET_\Sigma \cup P^\forall \cup \Phi \models_3 \varphi$ then $FET_\Sigma \cup (P \cup P')^\forall \cup \Phi \models_3 \varphi$. □

Likewise in the propositional case, the ordering considered over $\mathtt{Mod}(P)$ is the lexicographical extension $\preceq$ over sequences $\langle (A_i^+, A_i^-) \rangle_{i \in \mathbb{N}}$ of the standard ordering.

As in the propositional case, we have the following theorem.

Theorem 4.2. *For any Σ-program P there exists a $\preceq$-least Σ-model $\mathscr{M}_P$ in the class $\mathtt{Mod}(P)$.*

Proof. Let P be any Σ-program, we define $\mathscr{M}_P = \langle (M_i^+, M_i^-) \rangle_{i \in \mathbb{N}}$ as the ranked Σ-structure such that

- M_0^+ is the $\subseteq$-least set satisfying condition (a).
- M_0^- is the $\subseteq$-greatest set satisfying condition (b).
- M_{i+1}^+ is the $\subseteq$-least set satisfying condition (c).
- M_{i+1}^- is the $\subseteq$-greatest set satisfying condition (d).

By definition $\mathscr{M}_P$ is a model of P. In order to prove that it is the least one, suppose any other $\mathscr{M} \in \mathtt{Mod}(P)$ such that $\mathscr{M} \prec \mathscr{M}_P$. Then, there is some $i \in \mathbb{N}$ such that $A_j^+ = M_j^+$ and $A_j^- = M_j^-$ for any $j < i$, but one of the following two facts holds:

(i) there is $p\Box c \in A_i^- \setminus M_i^-$

(ii) there is $p\Box c \in M_i^+ \setminus A_i^+$.

We will prove that both facts are not possible.

(i) Suppose that $p\Box c \in A_i^-$. If $i = 0$ then for every clause $p(\bar{x}) : -\bar{l}\Box \bar{x} = \bar{t}$ in P, the constraint $c \wedge \bar{x} = \bar{t}$ is unsatisfiable, but this is a sufficient condition for $p\Box c \in M_0^-$.

For $i > 0$, the case $p\Box c \in A_j^-$ for some $j < i$ is trivial since $A_i^- \supseteq A_j^- = M_j^-$. Otherwise, we have that for every clause $p(\bar{x}) : -\bar{l}\Box\bar{x} = \bar{t}$ in P:

$$FET_\Sigma \cup A_{i-1}^\forall \models_3 ((c \wedge \bar{x} = \bar{t}) \rightarrow \neg\bar{l})^\forall$$

since $M_{i-1}^\forall = A_{i-1}^\forall$, we have that $p\Box c$ also belongs to M_i^-.

(ii) Now, suppose that $p\Box c \in M_i^+$. For $i = 0$ that means $FET_\Sigma \cup P^\forall \cup M_0^{+\forall} \models_3 (c \rightarrow p)^\forall$, since $\mathcal{M} \in \mathrm{Mod}(P)$ and $M_0^+ \subseteq A_0^+$, this suffices to ensure that $p\Box c \in A_0^+$. For $i > 0$ there are two cases. First, $p\Box c \in M_j^+$ for some $j < i$, but $A_i^+ \supseteq A_j^+ = M_j^+$ and $A_{i-1}^- \supseteq A_{j-1}^- = M_{j-1}^-$. Otherwise, we have that

$$FET_\Sigma \cup P^\forall \cup M_i^{+\forall} \cup M_{i-1}^{-\forall} \models_3 (c \rightarrow p)^\forall$$

since (i), we have that $A_{i-1}^{-\forall} = M_{i-1}^{-\forall}$, and because of the construction of $\mathcal{M}_P$ we also have that $A_i^{+\forall} \supseteq M_i^{+\forall}$, so that we obtain $p\Box c \in A_i^+$. □

4.3. The least model

In this section we study some interesting properties of least models. In particular, different characterizations by logical consequence closure and its constructive definition through an immediate consequence operator.

From now on, $\langle (M_i^+, M_i^-) \rangle_{i \in \mathbb{N}}$ will denote the least model $\mathcal{M}_P$ of a given program P.

Our first least model characterization is made in terms of a logical consequence closure of the equality theory and the standard logical interpretation of the program.

Lemma 4.1. *For any Σ-program P:*

(i) $p\Box c \in M_0^+ \iff FET_\Sigma \cup P^\forall \models_3 (c \rightarrow p)^\forall$.
(ii) $p\Box c \in M_{i-1}^- \iff FET_\Sigma \cup P^\forall \cup M_i^{+\forall} \models_3 (c \rightarrow p)^\forall$.
(iii) $p\Box c \in M_i^- \iff FET_\Sigma \cup P^\forall \cup M_i^{+\forall} \models_3 (c \rightarrow \neg p)^\forall$.

Proof. Right-to-left implication of (i) and (ii), as well as (iii)-left-to-right, are trivial. We will prove the others by simultaneous induction on i.

For the converse implication of (i), we define the set

$$B \equiv \{q\Box d \mid FET_\Sigma \cup P^\forall \models_3 (d \rightarrow q)^\forall \text{ and } d \text{ is satisfiable}\}.$$

Now, using the fact that for every set of formulas $\Phi \cup \{\psi\}$:

$$\Phi \cup \{\varphi \mid \Phi \models_3 \varphi\} \models_3 \psi \implies \Phi \models_3 \psi$$

it is easy to see that B satisfies Definition (a). Therefore, $M_0^+ \subseteq B$.

The proof for the converse implication of (ii) is similar, but taking the set

$$B \equiv \{q\Box d \mid FET_\Sigma \cup P^\forall \cup M_i^{+\forall} \models_3 (d \rightarrow q)^\forall \text{ and } d \text{ is satisfiable}\}.$$

For the right-to-left implication of (iii), the key idea is that the program P cannot "add" new negative logical consequences. In particular, if we assume that $a\Box c$ is not in M_i^-, then we can build a model of $FET_\Sigma \cup P^\forall \cup M_i^{+\forall}$ which is not a model of $(c \rightarrow \neg a)^\forall$: it is enough to consider the Herbrand structure (A^+, A^-) where A^- consists of all atoms $b\sigma$ such that $b\Box d \in M_i^-$ and σ is a (ground substitution) solution of d, and A^+ includes the rest of the atoms. □

A trivial consequence of the previous lemma is that $\mathcal{M}_P$ is closed with respect to less general constraints.

Lemma 4.2. *For any Σ-program P and for every $i \in \mathbb{N}$:*
(i) *if $p\Box c \in M_i^+$ and $FET_\Sigma \models_3 (d \to c)^\forall$ and d is satisfiable, then $p\Box d \in M_i^+$;*
(ii) *if $p\Box c \in M_i^-$ and $FET_\Sigma \models_3 (d \to c)^\forall$ and d is satisfiable, then $p\Box d \in M_i^-$.*

Proof. It is enough to notice that for any set of formulas $\Phi \cup \{\varphi\}$, if $FET_\Sigma \cup \Phi \models_3 (c \to \varphi)^\forall$ and $FET_\Sigma \models_3 (d \to c)^\forall$, then $FET_\Sigma \cup \Phi \models_3 (d \to \varphi)^\forall$. □

Now, we are going to characterize the least model in the usual constructive way: as the least fixpoint of a monotonic and continuous immediate consequence operator. For that purpose we order ranked structures by the trivial extension of Fitting's ordering

$$\mathcal{A} \preceq_F \mathcal{B} \text{ iff } A_i^+ \subseteq B_i^+ \quad \text{and} \quad A_i^- \subseteq B_i^- \quad \text{for all } i \in \mathbb{N}.$$

It is easy to see that ranked structures are a cpo with respect to $\preceq_F$, whose bottom is the infinite sequence of pairs of empty sets and the least upper bound, for every infinite increasing chain of ranked structures, is the level-by-level union of positive and negative parts of all of them. We define an immediate consequence operator in the following way.

Definition 4.5. Let P be a Σ-program and $\mathcal{A}$ a ranked Σ-structure, $T_P(\mathcal{A}) = \mathcal{B}$ where $\mathcal{B}$ is the ranked structure defined for each $i \in \mathbb{N}$ by:

$$B_i^+ = V_P(A_i^+, A_{i-1}^-) \quad \text{and} \quad B_i^- = R_P(A_{i-1}^+, A_{i-1}^-),$$

where $A_{-1}^+ \equiv A_{-1}^- \equiv \emptyset$, by convention, and V_P and R_P are the following two operators over pairs of sets of constrained atoms:

$$\begin{aligned} V_P(C,D) = \{p(\bar{x})\Box c(\bar{x}) \mid & \text{For some } n \geqslant 1, \\ & \text{some satisfiable constraints } c_1, \ldots, c_n, \\ & \text{and some subset of properly renamed clauses of } P \\ & \{p(\bar{x}) : -\bar{l}^k \Box d_k \mid 1 \leqslant k \leqslant n, free(\bar{l}^k \wedge d_k) = \bar{x}, \bar{y}^k\} : \\ & FET_\Sigma \models_3 (c \to \vee_{k=1}^n \exists \bar{y}^k (d_k \wedge c_k))^\forall \text{ and} \\ & FET_\Sigma \cup C^\forall \cup D^\forall \models_3 (c_k \to \bar{l}^k)^\forall \text{ for all } k = 1, \ldots, n\}, \end{aligned}$$

$$\begin{aligned} R_P(C,D) = \{p(\bar{x})\Box c(\bar{x}) \mid & \text{For every properly renamed clause } p(\bar{x}) : -\bar{l}\Box d \in P: \\ & FET_\Sigma \cup C^\forall \cup D^\forall \models_3 ((c \wedge d) \to \neg \bar{l}))^\forall\}. \end{aligned}$$

The powers (or iterations) of T_P are defined by

$$T_P^0(\mathcal{A}) = \mathcal{A} \quad \text{and} \quad T_P^{k+1}(\mathcal{A}) = T_P(T_P^k(\mathcal{A})).$$

The monotonicity of logical consequence trivially implies that V_P and R_P are monotonic with respect to $\subseteq$, hence T_P is monotonic with respect to $\preceq_F$. Moreover, we will prove that it is continuous and, therefore, we will obtain M_P at the ω iteration of T_P over the always empty ranked structure.

Lemma 4.3. *For any Σ-program P, T_P is continuous.*

Proof. Consider any infinite chain of ranked structures

$$\mathscr{A}0 \preceq_F \mathscr{A}1 \preceq_F \ldots \preceq_F \mathscr{A}n \preceq_F \ldots$$

By monotonicity of T_P it is enough to prove that $T_P(\sqcup\mathscr{A}n) \preceq_F \sqcup T_P(\mathscr{A}n)$. Consider, firstly, $p(\bar{x})\Box c(\bar{x}) \in (T_P(\sqcup\mathscr{A}n))_0^+$, then there exists $n \geqslant 1$, constraints $c_1, \ldots, c_n$, and a subset $\{p(\bar{x}) : -\bar{l}^k \Box d_k \mid 1 \leqslant k \leqslant n\}$ of P, such that

$$FET_\Sigma \models_3 \left(c \rightarrow \bigvee_{k=1}^{n} \exists \bar{y}^k (d_k \wedge c_k)\right)^\forall$$

and

$$FET_\Sigma \cup (\sqcup\mathscr{A}n)_0^{+\forall} \models_3 (c_k \rightarrow \bar{l}^k)^\forall \quad \text{for all } k = 1, \ldots, n.$$

Hence, by compactness of the logic, there exists some $\mathscr{A}r$ (in the chain) such that

$$FET_\Sigma \cup (\mathscr{A}r)_0^{+\forall} \models_3 (c_k \rightarrow \bar{l}^k)^\forall \quad \text{for all } k = 1, \ldots, n.$$

Therefore $p(\bar{x})\Box c(\bar{x}) \in (T_P(\mathscr{A}r))_0^+$, and so, it belongs to $(\sqcup T_P(\mathscr{A}n))_0^+$.

Similarly, by considering that $p(\bar{x})\Box c(\bar{x}) \in (T_P(\sqcup\mathscr{A}n))_{i+1}^+$, we have that:

$$FET_\Sigma \models_3 \left(c \rightarrow \bigvee_{k=1}^{n} \exists \bar{y}^k (d_k \wedge c_k)\right)^\forall$$

and

$$FET_\Sigma \cup (\mathscr{A}r)_{i+1}^{+\forall} \cup (\mathscr{A}r)_i^{-\forall} \models_3 (c_k \rightarrow \bar{l}^k)^\forall \quad \text{for all } k = 1, \ldots, n.$$

so that $p(\bar{x})\Box c(\bar{x}) \in (T_P(\mathscr{A}r))_{i+1}^+$.

The proof for the inclusion of negative parts is similar and easier. □

The following lemma provides a useful induction principle for reasoning about the least model. In particular, it is the basis for the least model characterization in terms of least fixpoints and for comparing T_P with Fitting's operator.

Lemma 4.4. *For any Σ-program P:*

(i) $M_0^+ = V_P^\omega(\emptyset, \emptyset)$,
(ii) $M_0^- = R_P(\emptyset, \emptyset)$,
(iii) $M_{i+1}^+ = V_P^\omega(M_i^+, M_i^-)$,
(iv) $M_{i+1}^- = R_P(M_i^+, M_i^-)$,

where $V_P^0(C, D) = C$ *and* $V_P^{i+1}(C, D) = V_P(V_P^i(C, D), D)$.

Proof. Facts (ii) and (iv) are trivial since M_0^- and M_{i+1}^- are, respectively, the $\subseteq$-greatest sets satisfying conditions (b) and (d) in Definition 4. The right-to-left inclusions of (i) and (iii) are also trivial from the fact that $\mathscr{M}_P$ is a model of P. Since $\mathscr{M}_P$ is the least model of P, we will prove the left-to-right inclusions of (i) and (iii) by proving that $V_P^\omega(\emptyset, \emptyset)$ and $V_P^\omega(M_i^+, M_i^-)$ satisfy, respectively, Definition 4(a) and (c), that is, for any satisfiable constraint c:

(1) $FET_\Sigma \cup P^\forall \cup V_P^\omega(\emptyset,\emptyset)^\forall \models_3 (c \to p)^\forall \implies p\Box c \in V_P^\omega(\emptyset,\emptyset)$.
(2) $FET_\Sigma \cup P^\forall \cup V_P^\omega(M_i^+, M_i^-)^\forall \cup M_i^{-\forall} \models_3 (c \to p)^\forall \implies p\Box c \in V_P^\omega(M_i^+, M_i^-)$.

In order to prove (1), let us consider a constrained atom $p(\bar{x})\Box c(\bar{x}) \notin V_P^\omega(\emptyset,\emptyset)$ with c satisfiable, we will prove the existence of a three-valued Σ-model of $FET_\Sigma \cup P^\forall \cup (V_P^\omega(\emptyset,\emptyset))^\forall$ which is a counter-model of $\forall\bar{x}(c(\bar{x}) \to p(\bar{x}))$.

Let Φ be the following set of formulas over the signature Σ':

$$\Phi \equiv FET_\Sigma \cup \{c[\bar{a}/\bar{x}]\} \cup \{\neg d[\bar{a}/\bar{x}] \mid p\Box d \in V_P^\omega(\emptyset,\emptyset)\},$$

where Σ' is the extension of Σ by the new constant symbols $\bar{a}$.

Now, let $\mathcal{A}$ be the least 3-valued Herbrand Σ'-model of Φ satisfying, in addition:

$$q^{\mathcal{A}}(\bar{s}) = \begin{cases} \underline{\mathbf{t}} & \text{if } \Phi \models (\bar{x} = \bar{s} \to e)^\forall \text{ for some } q(\bar{x})\Box e(\bar{x}) \in V_P^\omega(\emptyset,\emptyset), \\ \underline{\mathbf{u}} & \text{otherwise,} \end{cases}$$

for each n-ary predicate symbol $q \in PS_\Sigma$ and each $\bar{s} \in (H_{\Sigma'})^n$.

It is trivial, by construction, that $\mathcal{A}$ is a model of $FET_\Sigma \cup (V_P^\omega(\emptyset,\emptyset))^\forall$, and a counter-model of $\forall\bar{x}(c(\bar{x}) \to p(\bar{x}))$.

To prove that $\mathcal{A}$ is also a model of $P^\forall$, let $q(\bar{x}) : -l_1, \ldots, l_k\Box g$ be a clause in P (with $\bar{x}, \bar{y}$ as free variables in the body), and let $\bar{s} \in (H_{\Sigma'})^n$ and $\bar{r} \in (H_{\Sigma'})^m$, such that $\mathcal{A} \models (l \wedge g)[\bar{s},\bar{r}/\bar{x},\bar{y}]$. Then, $l_1, \ldots, l_k$ must be atoms (because our model does not satisfy any negated atom) such that for all $i = 1, \ldots, k$ there exists a constraint d_i such that $l_i(\bar{x},\bar{y})\Box d_i(\bar{x},\bar{y}) \in V_P^\omega(\emptyset,\emptyset)$ and

$$\Phi \models \left(\bar{x} = \bar{s} \to \exists\bar{y}(g(\bar{x},\bar{y}) \wedge \bigwedge_{i=1}^{k} d_i(\bar{x},\bar{y}))\right)^\forall.$$

Moreover, by logical compactness and the definition of V_P, the following constrained atom belongs to $V_P^\omega(\emptyset,\emptyset)$:

$$q(\bar{x})\Box\exists\bar{y}\left(g(\bar{x},\bar{y}) \wedge \bigwedge_{i=1}^{k} d_i(\bar{x},\bar{y})\right).$$

So that, $q^{\mathcal{A}}(\bar{s}) = \underline{\mathbf{t}}$.

Finally, the Σ'-structure $\mathcal{A}$ must be transformed into a Σ-structure, by interpreting (over the same universe) only function symbols in FS_Σ (but not the new constant symbols).

The proof for (2) is very similar. For a constrained atom $p\Box c \notin V_P^\omega(M_i^+, M_i^-)$ with c satisfiable, we obtain the initial Σ'-model $\mathcal{A}$ of

$$\Phi \equiv FET_\Sigma \cup \{c[\bar{a}/\bar{x}]\} \cup \{\neg d[\bar{a}/\bar{x}] \mid p\Box d \in V_P^\omega(M_i^+, M_i^-)\}$$

with interpretation for predicate symbols given by:

$$q^{\mathscr{M}}(\bar{s}) = \begin{cases} \underline{\mathbf{t}} & \text{if } \Phi \models (\bar{x} = \bar{s} \rightarrow e)^{\forall} \text{ for some } q\Box e \in V_P^{\omega}(M_i^+, M_i^-), \\ \underline{\mathbf{f}} & \text{if } \Phi \models (\bar{x} = \bar{s} \rightarrow e)^{\forall} \text{ for some } q\Box e \in M_i^-, \\ \underline{\mathbf{u}} & \text{otherwise.} \end{cases}$$

Proving that $\mathscr{M}$ is a model of P is very similar to the previous case. □

From now on, we will denote by $T_P \uparrow k$ the kth power (or iteration) of T_P over the ranked structure $\langle(\emptyset, \emptyset), (\emptyset, \emptyset), \ldots\rangle$. Now it can be shown that $\mathscr{M}_P$ coincides with $T_P \uparrow \omega$, which is the least fixpoint of T_P.

Lemma 4.5. *For any Σ-program P: $T_P \uparrow \omega = \mathscr{M}_P$.*

Proof. It is trivial that for all $j \in \mathbb{N}$: $T_P \uparrow j \preceq_F \mathscr{M}_P$, hence $T_P \uparrow \omega \preceq_F \mathscr{M}_P$. We will prove the opposite inclusion, that is, for all $i \in \mathbb{N}$:

$$M_i^+ \subseteq (T_P \uparrow \omega)_i^+ \quad \text{and} \quad M_i^- \subseteq (T_P \uparrow \omega)_i^-$$

by induction on i. Using Lemma 4.4, for all $i \in \mathbb{N}$ (where $M_{-1}^+ = M_{-1}^- = \emptyset$), we know that:

$$M_i^+ = V_P^{\omega}(M_{i-1}^+, M_{i-1}^-) \quad \text{and} \quad M_i^- = R_P(M_{i-1}^+, M_{i-1}^-).$$

For $i = 0$ it is trivial by definition of T_P. In the inductive step for $i + 1$, the inclusion of negative parts is trivial. For the positive ones, it is easy to prove that for all $j \in \mathbb{N}$: $V_P^j(M_i^+, M_i^-) \subseteq (T_P \uparrow \omega)_{i+1}^+$, using induction on j and the induction hypothesis about M_i^+ and M_i^-. Hence, $\mathscr{M}_P \preceq_F T_P \uparrow \omega$. □

4.4. Equivalence with the Clark–Kunen semantics

In Ref. [25] was proved that the finite powers of Fitting's operator coincide with the three-valued logical consequences of Clark's completion (the Clark–Kunen semantics). This result was adapted to the Constraint Logic Programming framework in Ref. [32]. Here we are going to show that the finite powers of our continuous operator T_P essentially coincide with those of Fitting's operator. Hence, our model-theoretic semantics is equivalent to the Clark–Kunen semantics. In particular, the least model of every program P is a three-valued model of $Comp(P)$ and is typical in the class of all three-valued models of $Comp(P)$. Firstly, we recall the definition of Fitting's operator Φ_P and show its relationship with our T_P by means of one example.

Definition 4.6 ([20,25,32]). Let P be a normal Σ-program, the immediate consequence operator Φ_P, ranging over Herbrand three-valued Σ-structures (or standard three-valued interpretations) $\mathscr{H} = (H^+, H^-)$, is given by:

$$\Phi_P(\mathscr{H})^+ = \{p(\bar{t}) \in B_\Sigma \mid \text{There exists a clause } p(\bar{x}): -\bar{l}\Box d \text{ in P with free variables } \bar{x}, \bar{y} \text{ and a tuple } \bar{s} \text{ of closed } \Sigma - \text{terms such that: } \mathscr{H} \models (d \wedge \bar{l})[\bar{t}/\bar{x}, \bar{s}/\bar{y}]\},$$

$$\Phi_P(\mathscr{H})^- = \{p(\bar{t}) \in B_\Sigma \mid \text{For every clause } p(\bar{x}): -\bar{l}\Box d \text{ in P with free variables } \bar{x}, \bar{y} \text{ and every tuple } \bar{s} \text{ of closed } \Sigma - \text{terms: } \mathscr{H} \models \neg(d \wedge \bar{l})[\bar{t}/\bar{x}, \bar{s}/\bar{y}]\}.$$

Φ_P is not continuous and obtains information without taking into account the negative dependences, whereas T_P is continuous and ranges over ranked structures placing information at layers. Finite powers of both operators obtain essentially (in spite of layers) the same information. In order to illustrate the relationship between both operators let us consider the following example (extending the usual program to show the non-continuity of Fitting's operator with two more clauses).

Example 4.2. Let P be the following program of the signature with constant 0, 1-ary function s, 0-ary predicate q and 1-ary predicates p, a, b:

$$p(x) : - p(y)\square x = s(y)$$
$$q : - p(x)$$
$$a(x) : - \neg p(x)$$
$$b(x) : - a(y)\square x = s(y)$$

For this program P, the iterations $\Phi_P \uparrow k$ over the $(\emptyset, \emptyset)$ three-valued interpretation can be described as follows: [1]

k	$\Phi_P \uparrow k = ((\Phi_P \uparrow k)^+, (\Phi_P \uparrow k)^-)$
0	$(\emptyset, \underline{\emptyset})$
1	$(\emptyset, \underline{\{b(0), p(0)\}})$
2	$(\{a(0)\}, \underline{\{b(0), p(0), p(s(0))\}})$
3	$(\{a(0), b(s(0)), a(s(0))\}, \underline{\{b(0), p(0), p(s(0)), p(s^2(0))\}})$
$\vdots$	$\vdots$
ω	$(\{a(s^i(0)) \mid i \geqslant 0\} \cup \{b(s^i(0)) \mid i \geqslant 1\}, \underline{\{b(0)\} \cup \{p(s^i(0)) \mid i \geqslant 0\}})$
$\omega + 1$	$(\{a(s^i(0)) \mid i \geqslant 0\} \cup \{b(s^i(0)) \mid i \geqslant 1\}, \underline{\{b(0)\} \cup \{p(s^i(0)) \mid i \geqslant 0\} \cup \{q\}})$.

To describe the iterations $T_P \uparrow k$, since each layer of our fixpoint is closed with respect to less general constraints, in each layer we will only write its most general constrained atoms.

k	$T_P \uparrow k = \langle (M_0^+, M_0^-), \ldots, (M_i^+, M_i^-), \ldots \rangle$
0	$\langle (\emptyset, \underline{\emptyset}) \rangle$
1	$\langle (\emptyset, \underline{\{b(x)\square x = 0, p(x)\square x = 0\}}) \rangle$
2	$\langle (\emptyset, \underline{\{b(x)\square x = 0, p(x)\square x = 0\}}),$ $(\{a(x)\square x = 0\}, \underline{\{b(x)\square x = 0, p(x)\square x = 0 \vee x = s(0)\}}) \rangle$
3	$\langle (\emptyset, \underline{\{b(x)\square x = 0, p(x)\square x = 0\}}),$ $(\{a(x)\square x = 0, b(x)\square x = s(0)\}, \underline{\{b(x)\square x = 0, p(x)\square x = 0 \vee x = s(0)\}})$ $(\{a(x)\square x = 0 \vee x = s(0), b(x)\square x = s(0)\},$ $\underline{\{b(x)\square x = 0, p(x)\square x = 0 \vee x = s(0) \vee x = s^2(0)\}}) \rangle$

[1] In order to make the reading easier we underline the negative parts.

$$
\begin{array}{ll}
\vdots & \vdots \\
\omega & \langle(\emptyset, \{b(x)\square x = 0, p(x)\square x = 0\}), \\
& (\{a(x)\square x = 0, b(x)\square x = s(0)\}, \{b(x)\square x = 0, p(x)\square x = 0 \vee x = s(0)\}), \\
& \qquad \vdots \\
& (\{a(x)\square x = 0 \vee \ldots \vee x = s^i(0) \mid i \geqslant 0\} \cup \\
& \{b(x)\square x = s(0) \vee \ldots \vee x = s^i(0) \mid i \geqslant 1\}, \\
& \qquad \{b(x)\square x = 0\} \cup \{p(x)\square x = 0 \vee \ldots \vee x = s^i(0) \mid i \geqslant 0\}) \\
& \qquad \vdots \\
& \qquad \ldots \qquad \rangle
\end{array}
$$

Notice that all positive facts $b(x)\square x = s^{i+1}(0)$ are placed in the same layer as $a(x)\square x = s^i(0)$, but negative facts $p(x)\square x = s^{i+1}(0)$ are placed one layer after the first occurrence of $p(x)\square x = s^i(0)$.

The operator $\Phi_P^{\mathscr{A}}$ given in Ref. [32] is a non-ground version of Φ_P relative to a structure $\mathscr{A}$ where the constraints are interpreted. It ranges over (non-ranked) partial constrained interpretations and is neither continuous. The continuous operator defined in Ref. [18], to obtain a fully abstract fixpoint semantics characterizing the operational semantics with respect to answer constraints, is in some sense closer to our T_P. However, there are two differences that may be remarked. Firstly, it also ranges over (non-ranked) partial constrained interpretations, and is defined relative to a given structure. Secondly, only the negative part of the resulting fixpoint is closed with respect to finite disjunction of constraints. Remember that in our case both parts of every layer are closed with respect to less general constraints.

Now, we will show that our fixpoint semantics essentially coincides with cutting off at step ω the iteration of Φ_P, in the sense that we are going to relate $\Phi_P \uparrow \omega$ with the three-valued interpretation obtained from our (ranked) fixpoint model by forgetting layers. We build the positive (respectively negative) part of this interpretation as the set all ground instances of the constrained atoms in the positive (respectively negative) part of any layer.

Definition 4.7. Let P be a Σ-program, $[\mathscr{M}_P]$ (or equivalently $[T_P \uparrow \omega]$) is the three-valued interpretation given by ($\oplus \in \{+, -\}$):

$$[\mathscr{M}_P]^{\oplus} = \{p(\bar{t}) \in B_\Sigma \mid p(\bar{x})\square\bar{x} = \bar{t} \in M_i^{\oplus} \text{ for some } i \in \mathbb{N}\}.$$

It is worthwhile noting that by closure with respect to less general constraints the above membership requirement is equivalent to ask for some $p(\bar{x})\square c(\bar{x})$ such that $FET_\Sigma \models c[\bar{t}/\bar{x}]$. Moreover, by completeness of the theory FET_Σ, the latter is equivalent to satisfaction in some arbitrary fixed model of FET_Σ, since all its models are elementary equivalent.

Lemma 4.6. *For any Σ-program P and any $k \in \mathbb{N}$: $[T_P \uparrow k] = \Phi_P \uparrow k$.*

Proof. The proof is made by induction on k, using the induction principle provided in Lemma 4.4 for $[T_P \uparrow k]$. We also use, along the proof, the fact that a sentence is a logical consequence of FET_Σ iff it is satisfied in some specific model of this theory.

For $k = 0$ it trivially holds. For the inductive step we are going to show the four inclusions needed for proving $[T_P \uparrow (k+1)] = \Phi_P \uparrow (k+1)$, assuming the induction hypothesis $[T_P \uparrow k] = \Phi_P \uparrow k$.

We first consider $p(\bar{t}) \in [T_P \uparrow (k+1)]^+$. Then there exists $i, j \in \mathbb{N}$ such that

$$p(\bar{x})\Box\bar{x} = \bar{t} \in V_P^j((T_P \uparrow k)_i^+, (T_P \uparrow k)_i^-).$$

Now, we use induction on j. The basic case $j = 0$ holds simply by induction hypothesis and $(\Phi_P \uparrow k)^+ \subseteq (\Phi_P \uparrow (k+1))^+$. For the inductive case suppose that $p(\bar{x})\Box\bar{x} = \bar{t} \in V_P^{j+1}((T_P \uparrow k)_i^+, (T_P \uparrow k)_i^-)$. Therefore for some $n \geqslant 1$, some satisfiable constraints $c_1, \ldots, c_n$, some subset $\{p(\bar{x}): -\bar{l}'\Box d_r \mid 1 \leqslant r \leqslant n\}$ of properly renamed clauses of P, with $free(\bar{l}' \wedge d_r) = \bar{x}, \bar{y}'$:

$$FET_\Sigma \models \left(\bar{x} = \bar{t} \rightarrow \bigvee_{r=1}^{n} \exists \bar{y}'(d_r \wedge c_r)\right)^\forall$$

and for all $r = 1, \ldots, n$:

$$FET_\Sigma \cup (V_P^j((T_P \uparrow k)_i^+, (T_P \uparrow k)_i^-)^\forall \cup (T_P \uparrow k)_i^{-\forall} \models (c_r \rightarrow \bar{l}')^\forall.$$

It is clear that $\Phi_P \uparrow k \models FET_\Sigma$. Besides by the induction hypothesis:

$$\Phi_P \uparrow k \models (V_P^j((T_P \uparrow k)_i^+, (T_P \uparrow k)_i^-))^\forall \text{ and also } \Phi_P \uparrow k \models (T_P \uparrow k)_i^{-\forall}.$$

Then we obtain $\Phi_P \uparrow k \models (\bar{l}' \wedge d_r)[\bar{t}/\bar{x}, \bar{s}/\bar{y}']$ for some $1 \leqslant r \leqslant n$ and some closed $\bar{s}$. Hence, $p(\bar{t}) \in (\Phi_P \uparrow (k+1))^+$.

Now, consider $p(\bar{t}) \in (\Phi_P \uparrow (k+1))^+$. Then $\Phi_P \uparrow k \models (d \wedge \bar{l})[\bar{t}/\bar{x}, \bar{s}/\bar{y}]$ for some clause $p(\bar{x}): -\bar{l}\Box d$ in P and some tuples $\bar{t}, \bar{s}$ of closed Σ-terms, hence

$$FET_\Sigma \models \forall\bar{x}(\bar{x} = \bar{t} \rightarrow \exists\bar{y}(d \wedge \bar{x} = \bar{t} \wedge \bar{y} = \bar{s}))$$

and also, by the induction hypothesis, there exists some $i \in \mathbb{N}$ such that

$$FET_\Sigma \cup (T_P \uparrow k)^\forall \models ((\bar{x} = \bar{t} \wedge \bar{y} = \bar{s}) \rightarrow \bar{l})^\forall.$$

Then $p(\bar{x})\Box\bar{x} = \bar{t} \in (T_P \uparrow (k+1))_{i+1}^+$.

For the negative parts, we first prove that $[T_P \uparrow (k+1)]^- \subseteq (\Phi_P \uparrow (k+1))^-$. Suppose that $p(\bar{x})\Box\bar{x} = \bar{t} \in R_P((T_P \uparrow k)_i^+, (T_P \uparrow k)_i^-)$ holds for some $i \in \mathbb{N}$. Then

$$FET_\Sigma \cup (T_P \uparrow k)_i^\forall \models ((\bar{x} = \bar{t} \wedge d) \rightarrow \neg\bar{l})^\forall.$$

for each clause $p(\bar{x}): -\bar{l}\Box d$ in P. In particular, by the induction hypothesis $\Phi_P \uparrow k$ satisfies all of these sentences. Hence $\Phi_P \uparrow k \models (\neg(d \wedge \bar{l})[\bar{t}/\bar{x}, \bar{s}/\bar{y}]$ for all of these clauses and any tuple of closed terms $\bar{s}$. Then $p(\bar{t}) \in (\Phi_P \uparrow (k+1))^-$.

Conversely, let us suppose $p(\bar{t}) \in (\Phi_P \uparrow (k+1))^-$, then

$$\Phi_P \uparrow k \models \neg(d \wedge \bar{l})[\bar{t}/\bar{x}, \bar{s}/\bar{y}]$$

for all clauses $p(\bar{x}): -\bar{l}\Box d$ in P and all closed terms $\bar{s}$. By the induction hypothesis there exists $i \in \mathbb{N}$ such that

$$FET_\Sigma \cup (T_P \uparrow k)_i^\forall \models ((\bar{x} = \bar{t} \wedge d) \rightarrow \neg\bar{l})^\forall$$

and therefore $p(\bar{x})\Box\bar{x} = \bar{t} \in \cup(T_P \uparrow (k+1))_{i+1}^-$. □

A direct consequence of the previous lemma is the equivalence between our least model and the finite powers of Fitting's operator.

Theorem 4.3. *For any Σ-program P:* $[\,.\,/\!/_P] = \Phi_P \uparrow \omega$.

It is well-known (cf. Ref. [25]) that $[\,.\,/\!/_P]$ could not be a model of *Comp*(*P*). Therefore, in order to relate our least model $.\,/\!/_P$ with program completion, we have to see it as a "standard" three-valued structure rather than as the Herbrand structure given by $[\,.\,/\!/_P]$. We should first define the truth-value of first-order sentences in ranked three-value structures. For that, we begin by assigning truth-values (in $\mathcal{A}$) to constrained atoms with the key (and obvious) definition:

$$\mathcal{A}(p(\bar{x})\Box c(\bar{x})) = \begin{cases} \underline{\mathbf{t}} & \text{if } p(\bar{x})\Box c(\bar{x}) \in A_i \text{ for some } i \in \mathbb{N}, \\ \underline{\mathbf{f}} & \text{if } p(\bar{x})\Box c(\bar{x}) \in A_i^- \text{ for some } i \in \mathbb{N}, \\ \underline{\mathbf{u}} & \text{otherwise.} \end{cases}$$

This definition can be extended, in a direct way to any arbitrary (constrained) formula. Here, we omit this definition due to the lack of space; however a very similar extension, to interpret goals of CLP-programs, is made in Ref. [32].

Theorem 4.4. *For every Σ-program P :* $.\,/\!/_P \models Comp(P)$.

Proof. One has to prove that every axiom $\varphi \in Comp(P)$, $.\,/\!/_P(\varphi\Box\mathbf{T}) = \underline{\mathbf{t}}$ holds. For the axioms in FET_Σ this is trivial. For the axioms of the form:

$$\forall\bar{x}\left(p(\bar{x}) \leftrightarrow \bigvee_{k=1}^{m} \exists\bar{y}^k(d_k \wedge \bar{l}^k)\right)$$

we proceed by case-analysis of the three possible truth-values (in $.\,/\!/_P$) of the constrained formula $p(\bar{x})\Box c$ (for some arbitrary satisfiable constraint c), showing that it coincides with the truth value of $\bigvee_{k=1}^{m} \exists\bar{y}^k(d_k \wedge \bar{l}^k)\Box c$ in $.\,/\!/_P$. In each case, we made use of the definition of the truth-value corresponding to constrained formulas with the connectives involved. □

Now, we prove that $.\,/\!/_P$ is "typical" in the class of all models of *Comp*(*P*).

Theorem 4.5. *For any normal program P and any constrained literal* $l\Box c$:

$$.\,/\!/_P \models l\Box c \;\Rightarrow\; Comp(P) \models l\Box c.$$

Proof. We will prove (by simultaneous induction on n) that for all $n \in \mathbb{N}$:

(i) $p\Box c \in M_n^+ \;\Rightarrow\; Comp(P) \models (c \rightarrow p)^\forall$,

(ii) $p\Box c \in M_n^- \;\Rightarrow\; Comp(P) \models (c \rightarrow \neg p)^\forall$.

For $n = 0$, if $p\Box c \in M_0^+$ then, by Lemma 4.1, $FET_\Sigma \cup P^\forall \models (c \rightarrow p)^\forall$. Since $Comp(P) \models FET_\Sigma \cup P^\forall$ (see Section 2), $Comp(P) \models (c \rightarrow p)^\forall$. If $p(\bar{x})\Box c(\bar{x}) \in M_0^-$ then $c \wedge d_k$ is unsatisfiable for all $p(\bar{x}) : -\bar{l}^k\Box d_k \in P$. Then, for all $k = 1 \ldots m$ (where m is the number of clauses with head p): $FET_\Sigma \models \forall\bar{x}(c \rightarrow \neg\exists\bar{y}^k d_k)$. Therefore, $Comp(P) \models \forall\bar{x}(c(\bar{x}) \rightarrow \neg p(\bar{x}))$.

For the inductive step, suppose that $Comp(P) \models M_i^\forall$. In the part (i), we have that $Comp(P) \models FET_\Sigma \cup P^\forall \cup M_i^\forall$. If $p\Box c \in M_{i+1}^+$ then, by Lemma 4.1, $FET_\Sigma \cup P^\forall \cup M_i^\forall \models (c \rightarrow p)^\forall$. Hence, the last formula is also a logical consequence of *Comp*(*P*). For part (ii), if $p(\bar{x})\Box c(\bar{x}) \in M_{i+1}^-$, then $\forall\bar{x}(c(\bar{x}) \rightarrow \neg\bigvee_{k=1}^{m} \exists\bar{y}^k(d_k \wedge \bar{l}^k))$ is a logical consequence of $FET_\Sigma \cup M_i^\forall$. Therefore $Comp(P) \models \forall\bar{x}(c(\bar{x}) \rightarrow \neg p(\bar{x}))$. □

5. A specification frame for normal logic programs

In this section, we show how the model-theoretic semantics defined in Section 4 can be the basis for defining a specification frame, which has the additional properties of ensuring the existence of compositional and fully abstract semantics for most kinds of modular units. In particular, we prove, as a consequence of these properties, the existence of a compositional and fully abstract semantics for the standard union of normal logic programs.

Definition 5.1. Let $\Sigma = (FS_\Sigma, PS_\Sigma)$ be some prefixed signature. Let $\underline{NLP_\Sigma}$ be the category of normal logic programs over Σ, whose objects are the pairs (Σ, Φ), with Φ being a set of normal clauses over Σ and whose morphisms are just inclusions, up to renaming of variables, of sets of normal clauses.

We define the model functor $\mathtt{Mod}$ mapping every program P in $\underline{NLP_\Sigma}$ into the category $\mathtt{Mod}(P)$, whose objects are ranked Σ-structures satisfying P and where a morphism is just the ordering relation between two ranked structures. For every morphism $h: P \to P'$, $V_h = \mathtt{Mod}(h)$ is just the identity.

Now, we show that the above defined pair is, in fact, a specification frame.

Lemma 5.1. *$\mathscr{SP}_\Sigma = (\underline{NLP_\Sigma}, \mathtt{Mod}_\Sigma : \underline{NLP_\Sigma}^{op} \to \underline{\mathtt{Cat}})$ is a specification frame, i.e. it satisfies:*

(i) *$\underline{NLP_\Sigma}$ has pushouts;*

(ii) *$\mathtt{Mod}$ transforms pushouts in $\underline{NLP_\Sigma}$ into pullbacks in $\underline{\mathtt{Cat}}$.*

Proof. The pushout of three programs $(\Sigma, \Phi 0)$, $(\Sigma, \Phi 1)$ and $(\Sigma, \Phi 2)$, with $\Phi 0 \subseteq \Phi 1$ and $\Phi 0 \subseteq \Phi 2$ is just $P1 \cup P2$, i.e. $(\Sigma, \Phi 1 \cup \Phi 2)$.

On the other hand, to show existence of amalgamation, on the current context, is trivial, since

$$\mathscr{A} \models_R P1 \quad \text{and} \quad \mathscr{A} \models_R P2 \quad \Longrightarrow \quad \mathscr{A} \models_R P1 \cup P2$$

is an obvious consequence of Definition 4 (although it does not hold for the simplification discussed in Remark 3.1). Therefore

$$\mathtt{Mod}(P1) + \mathtt{Mod}(P0)\mathtt{Mod}(P2) = \mathtt{Mod}(P1) \cap \mathtt{Mod}(P2). \quad \square$$

Remark 5.1. It may be noted that we consider a fixed signature for all programs in the specification frame. The main reason for this is technical, as the counter-example below shows. In particular, in the general case we can not define a forgetful functor. It can be argued that this is highly inconvenient with respect to modularity issues, however we do not think that this is important insofar as visibility is treated completely at the static semantics level. On the other hand, we believe that this situation is in some sense related to the nature of negation-as-failure where one can always expect to obtain (negative) answers to queries over predicates which are not in the signature of the given program.

Example 5.1. Let $\Sigma 1$ and $\Sigma 2$ be two signatures with $PS_{\Sigma 1} = \{p\}$ and $PS_{\Sigma 2} = \{p, q\}$. Let $\mathscr{A} = \langle(\emptyset, \{q\}), (\{p\}, \{q\})\rangle$ and $\mathscr{B} = \langle(\emptyset, \emptyset), (\emptyset, \emptyset)\rangle$. Then $\mathscr{A} \preceq \mathscr{B}$ in $\Sigma 2$, but

$\mathscr{B}|_{\Sigma 1} \preceq \mathscr{A}|_{\Sigma 1}$ in $\mathtt{Mod}(\Sigma 1, \emptyset)$, where $\mathscr{A}|_{\Sigma 1}$ (resp. $\mathscr{B}|_{\Sigma 1}$) is $V_i(\mathscr{A})$ (resp. $V_i(\mathscr{B})$, and V_i is the forgetful functor associated to the inclusion i: $(\Sigma 1, \emptyset) \subseteq (\Sigma 2, \emptyset)$. That is, $\mathscr{A}|_{\Sigma 1}$ (resp. $\mathscr{B}|_{\Sigma 1}$) is obtained from $\mathscr{A}$ (resp. $\mathscr{B}$) by deleting all atoms including symbols not in $\Sigma 1$, which is the most obvious definition of a forgetful functor in this context.

Before proving further "structural" properties of this specification frame we will show that the class of models associated to a given program forms a complete lattice. On the other hand, this result will be used as a lemma for showing the other properties of the specification frame.

Lemma 5.2. *For any program P,* $\mathtt{Mod}(P)$ *is a complete lattice.*

Proof. In order to show that $\mathtt{Mod}(P)$ is a complete lattice we have to prove that, for each subset $\mathscr{S}$ of $\mathtt{Mod}(P)$, we can define the join and meet of the models of $\mathscr{S}$, $\sqcup\mathscr{S}$ and $\sqcap\mathscr{S}$.

(a) The join $\mathscr{C} = \sqcup\mathscr{S}$ can be defined as follows:

$$C_0^+ = \{a\Box c \mid FET_\Sigma \cup P^\forall \cup \bigcup\{A_0^{+\forall} | \mathscr{A} \in \mathscr{S}_0\} \models_3 (c \to a)^\forall\}$$

$$C_0^- = \bigcap\{A_0^- | \mathscr{A} \in \mathscr{S}_0\}$$

where $\mathscr{S}_0 = \mathscr{S}$.

For all layers $i > 0$ such that $\mathscr{S}_i \neq \emptyset$:

$$C_i^+ = \{a\Box c \mid FET_\Sigma \cup P^\forall \cup C_{i-1}^\forall \cup \bigcup\{A_i^{+\forall} | \mathscr{A} \in \mathscr{S}_i\} \models_3 (c \to a)^\forall\}$$

$$C_i^- = \bigcap\{A_i^- | \mathscr{A} \in \mathscr{S}_i\}$$

where $\mathscr{S}_i = \mathscr{S}_{i-1} \setminus \{\mathscr{A} | A_{i-1} \neq C_{i-1}\}$.

If there is $k \in \mathbb{N}$ such that $\mathscr{S}_{k-1} \neq \emptyset$ but $\mathscr{S}_k = \emptyset$, then for all layers i with $i \geqslant k$:

$$C_i^+ = \{a\Box c \mid FET_\Sigma \cup P^\forall \cup C_{i-1}^\forall \models_3 (c \to a)^\forall\}$$

$$C_i^- = \{a\Box c \mid \text{For all } a\text{: } -\bar{l}\Box d \in P\text{: } FET_\Sigma \cup C_{i-1}^\forall \models_3 ((c \wedge d) \to \neg\bar{l})^\forall\}.$$

(b) The meet $\mathscr{D} = \sqcap\mathscr{S}$ is defined as follows:

For all layers $i \in \mathbb{N}$ such that $\mathscr{R}_i \neq \emptyset$:

$$D_i^+ = \bigcap\{A_i^+ | \mathscr{A} \in \mathscr{R}_i\}$$

$$D_i^- = \bigcup\{A_i^- | \mathscr{A} \in \mathscr{R}_i\}$$

where $\mathscr{R}_0 = \mathscr{S}$ and $\mathscr{R}_i = \mathscr{R}_{i-1} \setminus \{\mathscr{A} | A_{i-1} \neq D_{i-1}\}$.

If there exists $k \in \mathbb{N}$ such that $\mathscr{R}_{k-1} \neq \emptyset$, and $\mathscr{R}_k = \emptyset$, then for all layers $i \in \mathbb{N}$ such that $i \geqslant k$:

$$D_i^+ = \{a\Box c \mid \text{ if for every } a\Box d \in D_i^-, c \wedge d \text{ is unsatisfiable}\}$$

$$D_i^- = D_{i-1}^-.$$

(a) First of all, we have to prove the consistency property: Let us suppose that $a\Box c1 \in C_i^+$ and $a\Box c2 \in C_i^-$ for some $i < k$, such that $c1 \wedge c2$ is satisfiable. Let $c = c1 \wedge c2$, then $a\Box c \in C_i^+ \cap C_i^-$ because the construction guarantees that layers are closed with respect to less general constraints. If $a\Box c \in C_i^-$ then $a\Box c \in A_i^-$ for all $\mathscr{A} \in \mathscr{S}_i$, what means: $FET_\Sigma \cup A_{i-1}^\forall \models_3 (c \wedge d) \to \neg\bar{l})^\forall$, for all a: $-\bar{l}\Box d \in P$. But $A_{i-1} = C_{i-1}$ holds for all $\mathscr{A} \in \mathscr{S}_i$. Then, by monotonicity:

$$FET_\Sigma \cup C_{i-1}^{\forall} \cup \bigcup\{A_i^{+\forall} | \mathscr{A} \in \mathscr{S}_i\} \models_3 ((c \wedge d) \rightarrow \neg \bar{l})^{\forall}, \text{ for all } a: -\bar{l}\Box d \in P,$$

contradicting $a\Box c \in C_i^+$, because $(c \rightarrow a)^{\forall}$ can never be a consequence of $FET_\Sigma \cup P^{\forall} \cup C_{i-1}^{\forall} \cup \bigcup\{A_i^{+\forall} | \mathscr{A} \in \mathscr{S}_i\}$. The consistency property is guaranteed for all layers $i \geqslant k$ because they are just an "if and only if" version of the satisfaction condition in Definition 4.

In order to prove that $\mathscr{C}$ is a model of P, it suffices to note that C_i^+, for any layer i, contains all the positive information that is supported as logical consequence of the previous layers, and that C_i^- only contains supported negative information in the same sense.

Finally, we prove that $\mathscr{C}$ is the least model which is greater than every model in $\mathscr{S}$. The construction of $\mathscr{C}$ implies that for each model $\mathscr{A} \in \mathscr{S}$, it holds either $\mathscr{A} \in \mathscr{S}_{i-1} \setminus \mathscr{S}_i$, for some $i \in \mathbb{N}$, implying $A_{i-1}^+ \subseteq C_{i-1}^+$ and $A_{i-1}^- \supseteq C_{i-1}^-$ and $A_{i-1} \neq C_{i-1}$ and $A_j = C_j$ for all $j < i-1$; or $A \in \mathscr{S}_i$, for all $i \in \mathbb{N}$, but in this case the definition ensures $A_i^+ \subseteq C_i^+$ and $A_i^- \supseteq C_i^-$, for each i. Hence, $\mathscr{A} \preceq \mathscr{C}$.

In order to prove that $\mathscr{C}$ is the least model satisfying $\mathscr{A} \preceq \mathscr{C}$ for all $\mathscr{A} \in \mathscr{S}$, let us suppose that $\mathscr{B}$ is a model satisfying $\mathscr{A} \preceq \mathscr{B}$ for all $\mathscr{A} \in \mathscr{S}$. First, it may be noted that, according to the definition of $\mathscr{C}$, if the given k does not exist then, for every layer i, there is an $\mathscr{A} \in \mathscr{S}$ such that for each j, $0 \leqslant j < i$, $A_j = C_j$. Then, for every i, $C_i^+ \subseteq B_i^+$ and $C_i^- \supseteq B_i^-$. Hence $\mathscr{B} \preceq \mathscr{C}$. If the given k exists then, similarly, there is an $\mathscr{A} \in \mathscr{S}$ such that for each j, $0 \leqslant j < k$, $A_j = C_j$. Therefore, for every j, $0 \leqslant j < k$, $C_j^+ \subseteq B_j^+$ and $C_j^- \supseteq B_j^-$. On the other hand, the construction of $\mathscr{C}$ ensures that for all layers $i \geqslant k$, C_i contains the least positive information and the greatest negative information supported by the previous layers. This means that also for each $i \geqslant k$, $C_i^+ \subseteq B_i^+$ and $C_i^- \supseteq B_i^-$.

(b) In this case, the consistency of $\mathscr{D}$ is a trivial consequence of the consistency of the models in $\mathscr{S}$. Let us prove that $\mathscr{D}$ is a model of P. Firstly, suppose that

$$FET_\Sigma \cup P^{\forall} \cup D_{i-1}^{-\forall} \cup D_i^{+\forall} \models_3 (c \rightarrow a)^{\forall},$$

where $i < k$, if k exists, and $i \in \mathbb{N}$ is arbitrary, otherwise. We know that all $\mathscr{A} \in \mathscr{R}_i$ satisfies: $A_i^+ \supseteq D_i^+$ and $A_{i-1}^- = D_{i-1}^-$ so, by monotonicity

$$FET_\Sigma \cup P^{\forall} \cup A_{i-1}^{-\forall} \cup A_i^{+\forall} \models_3 (c \rightarrow a)^{\forall}.$$

This means $a\Box c \in A_i^+$ for all $\mathscr{A} \in \mathscr{R}_i$, so $a\Box c \in D_i^+$. Now, suppose that $a\Box c \in D_i^-$ for any i ($i < k$ if the given k exists). Then, $a\Box c \in A_i^-$ for some $\mathscr{A} \in \mathscr{R}_i$, and $FET_\Sigma \cup D_{i-1}^{\forall} \models_3 ((c \wedge d) \rightarrow \neg \bar{l})^{\forall}$ for all $a: -\bar{l}\Box d \in P$, because every $\mathscr{A} \in \mathscr{R}_i$ satisfies $A_{i-1}^+ = D_{i-1}^+$ and $A_{i-1}^- = D_{i-1}^-$. If there exists the given layer k then, for any layer D_i with $i \geqslant k$, the satisfaction condition trivially holds, since they contain more positive information than what is supported by the previous layers, but just the negative information from D_{i-1}.

It is not difficult to see that $\mathscr{D}$ is the greatest model which is smaller than all models in $\mathscr{S}$, because it is trivial that $\bigcap\{A_i^+ | \mathscr{A} \in \mathscr{R}_i\}$ is the greatest set such that $B_i^+ \supseteq \bigcap\{A_i^+ | \mathscr{A} \in \mathscr{R}_i\}$, and that $\bigcup\{(A_i^-) | \mathscr{A} \in \mathscr{R}_i\}$ is the least set such that $B_k^+ \subseteq \bigcup\{(A_i^-) | \mathscr{A} \in \mathscr{R}_i\}$ for all $\mathscr{B} \in \mathscr{S}$. If there exists the given layer k then D_i, for each $i \geqslant k$, contains the greatest positive information and the least negative information supported by the previous layers. □

The following example gives some hints about the constructions in the previous proof.

Example 5.2. Let the program $P = \{p : -q\}$, the symbols $PS_\Sigma = \{p, q, r, s\}$ and the following structures:

$$\mathscr{M}1 = \langle(\{p\}, \emptyset), (\{p\}, \emptyset)\rangle$$
$$\mathscr{M}2 = \langle(\{p\}, \{q\}), (\{p\}, \{q\})\rangle$$
$$\mathscr{M}3 = \langle(\{p\}, \emptyset), (\{p, r\}, \emptyset)\rangle$$
$$\mathscr{M}4 = \langle(\emptyset, \{q\}), (\emptyset, \{q\})\rangle$$
$$\mathscr{M}5 = \langle(\emptyset, \{q\}), (\{p\}, \{q, r\})\rangle$$

Then

$$\mathscr{M}1 \sqcup \mathscr{M}2 \sqcup \mathscr{M}3 = \langle(\{p\}, \emptyset), (\{p, r\}, \emptyset)\rangle$$
$$\mathscr{M}1 \sqcap \mathscr{M}4 \sqcap \mathscr{M}5 = \langle(\emptyset, \{q\}), (\emptyset, \{q, r\}), (\{p, s\}, \{q, r\})\rangle$$

Now, using the previous lemma we can prove that $\mathscr{NLP}_\Sigma$ satisfies all the properties needed for giving adequate compositional semantics to the intended program units.

Theorem 5.1 (Properties of $\mathscr{NLP}_\Sigma$). *$\mathscr{NLP}_\Sigma$ has free constructions, free extensions and generalized free extensions.*

Proof. $\mathscr{NLP}_\Sigma$ has free constructions, since given a morphism $h : P \to P'$, with $P = (\Sigma, \mathscr{C})$ and $P' = (\Sigma, \mathscr{C}')$, the free construction $F_h : \mathtt{Mod}(P) \to \mathtt{Mod}(P')$ is defined for every $\mathscr{M}$ in $\mathtt{Mod}(P)$ as $\mathscr{C} = F_h(\mathscr{M})$ such that:

$$C_0^+ = \{a \square c \mid FET_\Sigma \cup P'^\forall \cup A_0^{+\forall} \models_3 (c \to a)^\forall\}$$
$$C_0^- = A_0^- \cap M_0^-$$

- For all layers $i > 0$ such that $A_{i-1} = C_{i-1}$:

$$C_i^+ = \{a \square c \mid FET_\Sigma \cup P'^\forall \cup C_{i-1}^\forall \cup A_i^{+\forall} \models_3 (c \to a)^\forall\}$$
$$C_i^- = A_i^- \cap M_i^-$$

- If there exists $k \in \mathbb{N}$ such that $A_{k-1} \neq C_{k-1}$, then for all layers i with $i \geqslant k$:

$$C_i^+ = \{a \square c \mid FET_\Sigma \cup P'^\forall \cup C_{i-1}^\forall \models_3 (c \to a)^\forall\}$$
$$C_i^- = \{a \square c \mid \text{ For all } a : -\bar{l} \square d \in P' : FET_\Sigma \cup C_{i-1}^\forall \models_3 ((c \wedge d) \to \neg \bar{l})^\forall\}.$$

Note that if $\mathscr{M} \in \mathtt{Mod}(P')$, then the above construction coincides with the definition of the join model in Lemma 5.2, for the particular case when $\mathscr{S} = \{\mathscr{M}, \mathscr{M}_{P'}\}$, that is $\sqcup\{\mathscr{M}, \mathscr{M}_{P'}\}$. Nevertheless, it is quite easy to see that even in this case, the result is the least model of P' greater than $\mathscr{M}$. The reason is that the definition guarantees that, at any layer i ($i < k$ if k exist), C_i^+ contains the least positive information supported by the previous layers and A_i^+, and C_i^- contains the greatest negative information supported by the previous layers which belongs to A_i^-. When k exists, C_i, for all layers $i \geqslant k$, contains the least positive information and the greatest negative information supported by the previous layers.

Now, we have to prove that $F_h(\mathscr{A})$ satisfies the universal property of free constructions: for each model $\mathscr{A}'$ in $\mathtt{Mod}(P')$, such that $\mathscr{A} \preceq V_h(\mathscr{A}')(=\mathscr{A}')$, it holds that $\sqcup\{\mathscr{A}, \mathscr{M}_{P_1}\} \preceq \mathscr{A}'$. This property holds by definition of the join operation $\sqcup$.

$\mathscr{NLP}_\Sigma$ has free extensions since it has amalgamations.

To see that $\mathscr{NLP}_\Sigma$ has generalized free extensions, according to Theorem 3.3, it is enough to prove that for every program P, there are pushouts in $\mathtt{Mod}(P)$. Given models $\mathscr{A}0, \mathscr{A}1, \mathscr{A}2$ in $\mathtt{Mod}(\mathrm{P})$, with $f1 : \mathscr{A}0 \preceq \mathscr{A}1$ and $f2 : \mathscr{A}0 \preceq \mathscr{A}2$, the pushout $\mathscr{A}3$ of $\mathscr{A}1$ and $\mathscr{A}2$ via $f1$ and $f2$ must be the least model greater than $\mathscr{A}1$ and $\mathscr{A}2$, thus again $\mathscr{A}3$ is just the join $\mathscr{A}1 \sqcup \mathscr{A}2$. □

Once proved the required properties of $\mathscr{NLP}$ we can provide a categorical semantics for programs fragments which is compositional with respect to standard union. The compositionality result is just a consequence of Theorem 3.4. However, as we can see below, full abstraction is not a direct consequence of Theorem 3.5. Nevertheless, in this case we were also able to prove full abstraction making use of the specific properties of our semantics.

Theorem 5.2 (Compositionality). *For any normal logic program P, the semantics $Sem(P) = F$ such that F is the free construction associated to the inclusion $(\Sigma, \emptyset) \subseteq P$, is compositional with respect to the standard union of programs.*

Proof. Is a direct consequence of Theorems 3.4 and 5.1. □

Let us now see a counter-example showing that $\mathscr{NLP}$ is not algebraic:

Example 5.3. The model $\mathscr{A} = \langle(\emptyset, \emptyset), (\{q\}, \emptyset), \ldots\rangle$ can never be a least model of any normal logic program. In particular in $\mathscr{A}$, the fact q is not supported by the previous layer.

Nevertheless, as said above, we can still prove full abstraction using the specific properties of our semantic constructions.

Theorem 5.3 (Full abstraction). *Given two normal programs $P1$ and $P2$, the following three facts are equivalent*:

(i) $Sem(P1) = Sem(P2)$.

(ii) *For every program P, $Sem(P \cup P1) = Sem(P \cup P2)$.*

(iii) *For every program P, $\mathscr{M}_{P \cup P1} = \mathscr{M}_{P \cup P2}$.*

Proof. It is enough to prove that (iii) implies (i), because the other implications are direct consequences of Lemma 3.1 and Theorem 5.1.

Let us suppose that there exists a model $\mathscr{A}$ in $\mathtt{Mod}(\Sigma, \emptyset)$ such that $F1(\mathscr{A}) \neq F2(\mathscr{A})$, where $F1 = Sem(P1)$ and $F2 = Sem(P2)$. Then, we will show that there exists a program P such that $\mathscr{M}_{P \cup P1} \neq \mathscr{M}_{P \cup P2}$. Let $j \in \mathbb{N}$ be the least layer such that $F1(\mathscr{A})^+_j \neq F2(\mathscr{A})^+_j$ or $F1(\mathscr{A})^-_j \neq F2(\mathscr{A})^-_j$. Then we can consider two cases. First, if there exists the given level $k \in \mathbb{N}$, and $F1(\mathscr{A})^+_j \neq F2(\mathscr{A})^+_j$, for some $j < k$, then $F1(\mathscr{B}) \neq F2(\mathscr{B})$ for all models $\mathscr{B} \in \mathtt{Mod}(\Sigma, \emptyset)$ such that $\mathscr{A}^+_j = \mathscr{B}^+_i$ and $\mathscr{A}^-_{j-1} = \mathscr{B}^-_{i-1}$ for some layer i. This is the case for the model $\mathscr{B}$ such that, for all $i \in \mathbb{N}$:

$$B_i^+ = A_j^+,$$

$$B_i^- = A_{j-1}^-.$$

In any other case, $F1(\mathscr{B}) \neq F2(\mathscr{B})$ for all models $\mathscr{B} \in \mathrm{Mod}(\Sigma, \emptyset)$ such that $B_i = F1(\mathscr{A})_{j-1} = F2(\mathscr{A})_{j-1}$ for some layer i. Now, we choose the model $\mathscr{B}$ such that, for all $i \in \mathbb{N}$:

$$B_i^+ = F1(\mathscr{A})_{j-1}^+ = F2(\mathscr{A})_{j-1}^+,$$

$$B_i^- = F1(\mathscr{A})_{j-1}^- = F2(\mathscr{A})_{j-1}^-.$$

It is easy to see that, in both cases, $\mathscr{B} = \mathscr{M}_P$, for P being the program: $P = (\Sigma, B_i^{+\forall} \cup \{a: -a\Box d / a\Box c \in B_i^- \text{ and } c \wedge d \text{ is unsatisfiable}\})$.

Hence, we can conclude that $F1(\mathscr{M}_{P\cup P}) = \mathscr{M}_{P\cup P1} \neq \mathscr{M}_{P\cup P2} = F2(\mathscr{M}_{P\cup P})$. □

6. Conclusions and related work

We have presented a new monotonic semantic framework for normal logic programs. The main characteristics of this semantics are the following ones: We do not consider any restrictions on programs (e.g., stratification). We associate to every program a class of models which forms a complete lattice whose least element is shown to be typical for the class of models of the Clark–Kunen's completion of the program. As a consequence, this least model can be seen as the standard semantics of the given program. Finally, the models of a program are a special case of Beth structures, where the ordering relating the "worlds" of the structure is total. Actually, our semantics could have been defined, without any problem, in terms of general Beth structures. In this sense, we believe that our semantics could also be valuable for knowledge representation considering the intuition behind Beth (and also Kripke) structures where each world in a model represents the knowledge one has at a given moment (see e.g. Ref. [33]).

The motivation for this new semantics was the definition of a specification frame of normal logic programs that could be used for defining compositional semantics to a variety of program units. In this sense, we have shown that the proposed semantics defines indeed a specification frame with the required properties. In particular, we have provided a categorical semantics for arbitrary program fragments which is compositional and fully abstract with respect to standard program union. Actually, other kind of units and composition operations can be seen just as a special case.

The kind of compositionality results obtained are quite more powerful than the results presented in Refs. [17,19,27,35,9]. In Refs. [17,19,27] different semantic definitions are provided for certain kinds of modular units which are shown to be compositional. However, they all impose (at least) the restriction (not needed in our work) that, for putting together (through the corresponding composition operation) two units, the sets of predicates defined in each unit must be disjoint. This means that, there can not be clauses defining the same predicate p (i.e. having p in the head of a clause) in both units. This restriction rules out the application of those results to approaches where the given system of modules supports the incremental definition of predicates through some form of inheritance (e.g. Ref. [7]). In Ref. [35] a slightly

more general framework is considered. In particular they study open programs where the open predicates can be axiomatized by arbitrary first order axioms. They provide a semantic definition based on well-founded semantics and show its compositionality under certain sufficient conditions which are quite close to the restrictions imposed in Ref. [17]. Finally, Ref. [9] proves that Fittings's immediate consequence operator can be used for defining a semantics for arbitrary program fragments which is compositional with respect to union, intersection and filtering. The main problem here is that, if only union is considered, the given semantics is too concrete to be of any use.

We have not directly related our approach with other kinds of semantics, although the relation established with completion implies, by transitivity, that our semantics can be considered equivalent to constructive negation approaches as Refs. [13,32]. Actually, the relation to Ref. [13] is quite more direct, in the sense that the construction of our least model is closely related to ranked resolution as defined there. There is also a certain relation between the construction of our least model and Fitting's fix point semantics [20], or rather with the version defined in Ref. [18], although not as close as it may seem: notice that in each layer of our least model we add not just the immediate consequences of the previous layer, but all logical consequences.

Acknowledgements

The authors would like to thank the referees for their detailed comments that have greatly contributed to improve the final version of this paper. This work has been partially supported by the Spanish CICYT project COSMOS (ref. TIC95-1016-C02-01) and the UPV-project 141.226-EA209/94 TIC.

References

[1] K.R. Apt, Logic programming, in: Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Ch. 10, Elsevier, Amsterdam, 1990.
[2] K.R. Apt, H. Blair, A. Walker, Towards a theory of declarative Knowledge, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programs, Morgan Kaufmann, Los Altos, CA, 1988.
[3] K.R. Apt, R.N. Bol, Logic programming and negation: a survey, Journal of Logic Programming 19 (1994) 9–71.
[4] A. Asperti, G. Longo, Categories, Types and Structures, MIT Press, Cambridge, MA, 1991.
[5] M. Barr, C. Wells, Category Theory for Computing Science, Prentice-Hall, Englewood Cliffs, NJ, 1990.
[6] A.J. Bonner, L.T. McCarty, Adding negation-as-failure to intuitionistic logic programming, in: Proc. of NACLP'90, 1990.
[7] A. Bossi, M. Bugliesi, M. Gabbrielli, G. Levi, M.C. Meo, Differential logic programming, in: Proc. ACM SIGPLAN-SIGACT Symp. on Princ. of Programming Languages POPL'93, ACM, 1993.
[8] A. Bossi, M. Gabrielli, G. Levi, M.C. Meo, Contributions to the semantics of open logics programs, in: Proc. Fifth Gen. Comp. Systems, 1992.
[9] A. Brogi, S. Contiero, F. Turini, Programming by combining general logic programs, Technical Report 97/02, Dept. of Comp. Science, Univ. of Pisa, 1997.
[10] R.M. Burstall, J.A. Goguen, The semantics of Clear, a specification language, in: Proc. Copenhagen Winter School on Abstract Software Specification, Springer LNCS 86, 1980, pp. 292–332.

[11] D. Chang, Constructive negation based on the completed database, in: R.A. Kowalski, K.A. Bowen (Eds.), Proc. Fifth Int. Conf. and Symp. on Logic Programmming, MIT Press, Cambridge, MA, 1988.
[12] K.L. Clark, Negation as failure, in: H. Gallaire, J. Minker (Eds.), Logic and Databases, Plenum Press, New York, 1978.
[13] W. Drabent, What is failure? An approach to constructive negation, Acta Informatica 32 (1995) 27–59.
[14] H. Ehrig, M. Baldamus, F. Orejas, New concepts for amalgamation and extension in the framework of specification logics, in: G. Rozenberg, A. Salomaa (Eds.), Current Trends in Theoretical Computer Science, World Scientific, Singapore, 1993.
[15] H. Ehrig, B. Mahr, Fundamentals of Algebraic Specification 1, Springer, Berlin, 1985.
[16] H. Ehrig, P. Pepper, F. Orejas, On recent trends in algebraic specifications, in: Proc. ICALP'89, LNCS 372, Springer, Berlin, 1989.
[17] S. Etalle, F. Teusink, A compositional semantics for normal open programs, in: Proc. Int. Conf. and Symp. on Logic Programming'96, MIT Press, Cambridge, MA, 1996.
[18] F. Fages, Constructive negation by pruning, Journal of Logic Programming 32 (1997) 11–85.
[19] G. Ferrand, A. Lallouet, A compositional proof method of partial correctness for normal logic programs, in: J. Lloyd (Ed.), Proc. Int. Logic Programming Symp., 1995.
[20] M. Fitting, A Kripke–Kleene semantics for logic programs, Journal of Logic Programming 4 (1985) 295–312.
[21] H. Gaifman, E. Shapiro, Fully abstract compositional semantics for logic programs, in: Proc. 16th Annual ACM Symp. on Principles of Programming Languages, 1989.
[22] J.A. Goguen, R.M. Burstall, Introducing institutions, in: Proc. Logics of Programming Workshop, Carnegie-Mellon University, LNCS 164, Springer, Berlin, 1984.
[23] J.A. Goguen, R.M. Burstall, Institution: abstract model theory for specification and programming, Journal of the ACM 39 (1992) 95–146.
[24] S.C. Kleene, Introduction to Metamathematics, North-Holland, Amsterdam, 1952.
[25] K. Kunen, Negation in logic programming, Journal of Logic Programming 4 (1987) 289–308.
[26] J.W. Lloyd, Foundations of Logic Programming, Springer, Berlin, 1987.
[27] M. Maher, A logic programming view of CLP, in: D.S. Warren (Ed.), Proc. Tenth Int. Conf. on Logic Programming, 1993.
[28] D. Miller, A logical analysis of modules in logic programming, Journal of Logic Programming 6 (1989) 79–108.
[29] F. Orejas, E. Pino, H. Ehrig, Institutions for logic programmming, Theoretical Computer Science 173 (1997) 485–511.
[30] T.C. Przymusinski, On the declarative semantics of deductive databases and logic programs, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programs, Morgan Kaufmann, Los Altos, CA, 1988.
[31] J.C. Shepherdson, Negation in logic programming, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programs, Morgan Kaufmann, Los Altos, CA, 1988.
[32] P.J. Stuckey, Negation and constraint logic programmming, Information and Computation 118 (1995) 12–23.
[33] D. van Dalen, A.S. Troelstra, Constructivism in Mathematics: An Introduction, vols. 1 and 2, Elsevier, Amsterdam, 1988.
[34] A. van Gelder, Negation as failure using tight derivations for general logic programs, in: J. Minker (Ed.), Foundations of Deductive Databases and Logic Programs, Morgan Kaufmann, Los Altos, CA, 1988.
[35] S. Verbaeten, M. Denecker, D. De Schreye, Compositionality of normal open logic programs, to appear in Proc. of ILPS'97.
[36] M. Wirsing, Algebraic specification, in: Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Elsevier, Amsterdam, 1990.