A Functorial Framework for Constraint Normal Logic Programming

P. Lucio[2], F. Orejas[1], E. Pasarella[1], and E. Pino[1]

[1] Departament LSI
Universitat Politècnica de Catalunya,
Campus Nord, Mòdul C5, Jordi Girona 1-3, 08034 Barcelona, Spain
[2] Departament LSI
Univ. Pais. Vasco,
San Sebastián, Spain
jiplucap@si.ehu.es, {orejas,edelmira,pino}@lsi.upc.es

Abstract. The semantic constructions and results for definite programs do not extend when dealing with negation. The main problem is related to a well-known problem in the area of algebraic specification: if we fix a constraint domain as a given model, its free extension by means of a set of Horn clauses defining a set of new predicates is semicomputable. However, if the language of the extension is richer than Horn clauses its free extension (if it exists) is not necessarily semicomputable. In this paper we present a framework that allows us to deal with these problems in a novel way. This framework is based on two main ideas: a reformulation of the notion of constraint domain and a functorial presentation of our semantics. In particular, the semantics of a logic program P is defined in terms of three *functors*: $(\mathcal{OP}_P, \mathcal{ALG}_P, \mathcal{LOG}_P)$ that apply to constraint domains and provide the operational, the least fixpoint and the logical semantics of P, respectively. To be more concrete, the idea is that the application of $\mathcal{OP}_P$ to a specific constraint solver, provides the operational semantics of P that uses this solver; the application of $\mathcal{ALG}_P$ to a specific domain, provides the least fixpoint of P over this domain; and, the application of $\mathcal{LOG}_P$ to a theory of constraints, provides the logic theory associated to P. In this context, we prove that these three functors are in some sense equivalent.

1 Introduction

Constraint logic programming was introduced in ([9]) as a powerful and conceptually simple extension of logic programming. Following that seminal paper, the semantics of definite (constraint) logic programs has been studied in detail (see, e.g. [10], [11]). However, the constructions and results for definite programs do not extend when dealing with negation. The main problem is related to a well-known problem in the area of algebraic specification: if we fix a constraint domain as a given model, its free extension by means of a set of Horn clauses defining a set of new predicates is semicomputable. However, if the language of the extension is richer than Horn clauses its free extension (if it exists) is not necessarily semicomputable ([8]). Now, when working without negation we are in the former case, but when working with negation we are in the latter case. In particular, this implies that the results about the soundness and completeness of the operational semantics with respect to the logical and algebraic semantics of a definite constraint logic program do not extend to the case of programs with negation, except when we impose some restrictions to these programs.

The only approach that we know that has dealt with this problem is ([19]). In that paper, Stuckey presents one of the first operational semantics which is proven complete for programs that include (constructive) negation. Although we use a different operational semantics, that paper has had an important influence in our work on negation. The results in ([19]) were very important when applied to the case of standard (non-constrained) logic programs because they provided some good insights about constructive negation. However, the general version (i.e., logic programs over an arbitrary constraint domain) is not so interesting (in our opinion). The reason is that the completeness results are obtained only for programs over *admissible* constraints. We think that this restriction on the constraints that can be used in a program is not properly justified.

In our opinion, the problem when dealing with negation is not on the class of constraints considered, but rather, in the notion of constraint domain used. In particular, we argue that the notion of constraint domain used in the context of definite programs is not adequate when dealing with negation. Instead, we propose and justify a small reformulation of the notion of constraint domain. To be precise, we propose that a domain should be defined in terms of a class of elementarily equivalent models and not in terms of a single model. With this variation we are able to show the equivalence of the logical, operational, and fixpoint semantics of programs with negation without needing to restrict the class of constraints.

The logical semantics that we have used is the standard Clark-Kunen 3-valued completion of programs (see, e.g. [19]). The fixpoint semantics that we are using is a variation of other well-known fixpoint semantics used to deal with negation ([5, 19, 6, 15]). Finally, the operational semantics that we are using is an extension of a semantics called BCN that we have defined in previous work ([16]) for the case of programs without constraints. The main reason for using this semantics and not Stuckey's semantics is that our semantics, is in our opinion, simpler. This implies having simpler proofs for our results. In particular, we do not claim that our semantics is better than Stuckey's (nor that it is worse). A proper comparison of these two semantics and of others like [5, 6] would need experimental work. We have a prototype implementation of BCN ([1]), but we do not know if the other approaches have been implemented. Anyhow, the pragmatic virtues of the various operational approaches to constructive negation are not a relevant issue in this paper.

In addition, our semantics is functorial. We consider that a constraint logic program is a program that is parameterized by the given constraint domain. Then, we think that the semantics of a program should be some kind of mapping. However, we also think that working in a categorical setting provides some additional advantages that are shown in the paper.

The paper is organized as follows. In the following section we give a short introduction to the semantics of (definite) constraint logic programs. In Section three, we discuss the inadequacy of the standard notion of constraint domain when dealing with negation and propose a new one. In Section four we study the semantics of programs when defined over a given arbitrary constraint domain. Then, in the following section we define several categories for defining the various semantic domains involved and define the functorial semantics of logic programs. Finally, in Section 6 we prove the equivalence of the logical, fixpoint and operational semantics.

Due to lack of space, the paper includes no proofs. However, the detailed proofs can be found at our web page (`http://www.lsi.upc.edu/~orejas/`) in the extended version of the paper.

2 Preliminaries

2.1 Basic Notions and Notation

A signature Σ consists of a pair of sets (FS_Σ, PS_Σ) of function and predicates symbols, respectively, with some associated arity. $T_\Sigma(X)$ denotes the set of all *first-order* Σ*-terms* over variables from X, and T_Σ denotes the set of all ground terms. A literal is either an atom $p(t_1,\ldots,t_n)$ (namely a positive literal) or a negated atom $\neg p(t_1,\ldots,t_n)$ (namely a negative literal). The set $Form_\Sigma$ is formed by all *first-order* Σ*-formulas* written (from atoms) using connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ and quantifiers $\forall, \exists$. We denote by $free(\varphi)$ the set of all free variables occurring in φ. $\varphi(\bar{x})$ specifies that $free(\varphi) \subseteq \bar{x}$. $Sent_\Sigma$ is the set of all $\varphi \in Form_\Sigma$ such that $free(\varphi) = \emptyset$, called Σ-sentences. By $\varphi^{\forall \setminus \bar{z}}$ (resp. $\varphi^{\exists \setminus \bar{z}}$) we denote the formula $\forall x_1 \ldots \forall x_n(\varphi)$ (resp. $\exists x_1 \ldots \exists x_n(\varphi)$), where $x_1 \ldots x_n$ are the variables in $free(\varphi) \setminus \bar{z}$. In particular, the universal (resp. existential) closure, that is $\varphi^{\forall \setminus \emptyset}$ (resp. $\varphi^{\exists \setminus \emptyset}$) is denoted by $\varphi^{\forall}$ (resp. $\varphi^{\exists}$).

To define the semantics of normal logic programs and their completion, it becomes necessary to use a concrete *three-valued* extension of the classical two-valued interpretation of logical symbols. The connectives $\neg, \wedge, \vee$ and quantifiers ($\forall$, $\exists$) are interpreted as in Kleene's logic ([12]). However, $\leftrightarrow$ is interpreted as the identity of truth-values (hence, $\leftrightarrow$ is two-valued) Moreover, to make $\varphi \leftrightarrow \psi$ logically equivalent to $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$, Przymusinski's interpretation ([17]) of $\rightarrow$ is required. It is also two-valued and gives the value $\underline{\mathtt{f}}$ exactly in the following three cases: $\underline{\mathtt{t}} \rightarrow \underline{\mathtt{f}}$, $\underline{\mathtt{t}} \rightarrow \underline{\mathtt{u}}$ and $\underline{\mathtt{u}} \rightarrow \underline{\mathtt{f}}$. Equality is two-valued also. Following [3], it is easy to see that the above detailed three-valued logic satisfies (as classical first-order logic does) all of the basic *metalogical properties*, in particular completeness and compactness.

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 (of adequate arity), and of every predicate symbol by a total function on the set of the three boolean values $\{\underline{\mathtt{t}}, \underline{\mathtt{f}}, \underline{\mathtt{u}}\}$ (i.e., a partial relation). Hence, terms cannot be undefined, but atoms can be interpreted as $\underline{\mathtt{u}}$. Classical (two-valued) first-order Σ-structures can be seen as a special case of three-valued ones, where every predicate symbol is interpreted by a total relation. Mod_Σ denotes the set of all three-valued Σ-structures. A Σ-structure $\mathcal{A} \in \mathsf{Mod}_\Sigma$ is a model of (or satisfies) a set of sentences Φ if, and only if, $\mathcal{A}(\varphi) = \underline{\mathtt{t}}$ for any sentence $\varphi \in \Phi$. This is also denoted by $\mathcal{A} \models \Phi$. We will denote by $\mathcal{A} \models_\sigma \Phi$ that $\mathcal{A}$ satisfies the sentence $\sigma(\Phi)$, resulting from the valuation $\sigma : free(\Phi) \rightarrow \mathcal{A}$ of the formula Φ. Given a set Φ of Σ-sentences $\mathsf{Mod}_\Sigma(\Phi)$ is the subclass of Mod_Σ formed by the models of Φ. Logical consequence $\Phi \models \varphi$ means that $\mathcal{A} \models \varphi$ holds for all $\mathcal{A} \in \mathsf{Mod}_\Sigma(\Phi)$. We say that two

Σ-structures $\mathcal{A}$ and $\mathcal{B}$ are *elementarily equivalent*, denoted $\mathcal{A} \simeq \mathcal{B}$ if $\mathcal{A}(\varphi) = \mathcal{B}(\varphi)$ for every first-order Σ-sentence φ. We denote by $EQ(\mathcal{A})$ the set of all Σ-structures that are elementarily equivalent to $\mathcal{A}$.

A Σ-*theory* is a set of Σ-sentences closed under logical consequence. A theory can be presented *semantically* or *axiomatically*. A semantic presentation is a class $\mathcal{C}$ of Σ-structures. Then, the theory semantically presented by $\mathcal{C}$ is the set of all Σ-sentences which are satisfied by $\mathcal{C}$:

$$Th(\mathcal{C}) = \{\varphi \in Sent_\Sigma \mid \ for\ all\ \mathcal{A} \in \mathcal{C}\ \mathcal{A}(\varphi) = \underline{t}\}$$

An *axiomatic* presentation is a decidable set of axioms $Ax \subseteq Sent_\Sigma$. Then, the theory axiomatically presented by Ax is the set of all logical consequences of Ax:

$$Th(Ax) = \{\varphi \in Sent_\Sigma \mid Ax \models \varphi\}$$

A Σ-theory $\mathcal{T}$ is said to be *complete* if, and only if, $\varphi \in \mathcal{T}$ or $\neg\varphi \in \mathcal{T}$ holds for every Σ-sentence φ.

Example 1 *Given a signature Σ, the free-equality theory $\mathcal{FEA}(\Sigma)$ is complete and can be presented by the following axioms:*
(1) $\forall x(x = x)$
(2) $\forall \bar{x} \forall \bar{y}(\bar{x} = \bar{y} \leftrightarrow f(\bar{x}) = f(\bar{y}))$ for each $f \in FS_\Sigma$
(3) $\forall \bar{x} \forall \bar{y}(\bar{x} = \bar{y} \rightarrow (p(\bar{x}) \leftrightarrow p(\bar{y})))$ for each $p \in PS_\Sigma$ (in part. $=$)
(4) $\forall \bar{x} \forall \bar{y} \neg(f(\bar{x}) = g(\bar{y}))$ for each pair $f, g \in FS_\Sigma$ such that $f \not\equiv g$
(5) $\forall x \neg(x = t)$ for each $t \in T_\Sigma(X)$ and $x \in X$ such that $x \in var(t)$ and $x \not\equiv t$.
(6) $\forall x(\bigvee_{f \in FS_\Sigma} \exists \bar{y}(x = f(\bar{y})))$ if Σ is finite. ■

2.2 Constraint Domains

A constraint logic program can be seen as a program where some function and predicate symbols have a predefined meaning on a given domain, called the constraint domain. To be more precise, a constraint domain determines the interpretation of the given predefined symbols. In particular, according to the standard approach for defining the class of $CLP(\mathcal{X})$ programs ([10], [11]), a constraint domain $\mathcal{X}$ consists of five parts:

$$\mathcal{X} = (\Sigma_\mathcal{X}, \mathcal{L}_\mathcal{X}, Ax_\mathcal{X}, \mathcal{D}_\mathcal{X}, solv_\mathcal{X})$$

where $\Sigma_\mathcal{X} = (FS_\mathcal{X}, PS_\mathcal{X})$ is the constraint signature, i.e., the set of symbols that are considered to be predefined; $\mathcal{L}_\mathcal{X}$ is the constraint language, i.e., the class of $\Sigma_\mathcal{X}$-formulas that can be used in programs; $\mathcal{D}_\mathcal{X}$ is the domain of computation, i.e., a model defining the semantics of the symbols in $\Sigma_\mathcal{X}$; $Ax_\mathcal{X}$ is an axiomatization of the domain, i.e., a decidable set of $\Sigma_\mathcal{X}$-sentences such that $\mathcal{D}_\mathcal{X} \models Ax_\mathcal{X}$; and, finally, $solv_\mathcal{X}$ is a constraint solver, i.e., an oracle that answers queries about constraints and that is used for defining the operational semantics of programs. In general, constraint solvers are expected to solve constraints, i.e., given a constraint c, one would expect that the solver will provide the values that satisfy the constraint or that it returns an equivalent constraint in *solved form*. However, in our case, we just need the solver to answer (un)satisfiability queries.

We consider that, given a constraint c, $solv_{\mathcal{X}}(c)$ may return $\mathbf{F}$, meaning that c is not satisfiable or it may answer $\mathbf{T}$, meaning that c is valid in the constraint domain, i.e., that $\neg c$ is unsatifiable. The solver may also answer $\mathbf{U}$ meaning that either the solver does not know the right answer or that the constraint is neither valid nor unsatifiable.

In addition, a constraint domain $\mathcal{X}$ must satisfy:

- $\mathbf{T}, \mathbf{F}, t_1 = t_2 \in \mathcal{L}_{\mathcal{X}}$ (hence the equality symbol $=$ belongs to $PS_{\mathcal{X}}$) and $\mathcal{L}_{\mathcal{X}}$ is closed under variable renaming, existential quantification and conjunction. Moreover, the equality symbol $=$ is interpreted as the equality in $\mathcal{D}_{\mathcal{X}}$, and $Ax_{\mathcal{X}}$ includes the equality axioms for $=$.
- The solver does not take variable names into account, that is, for all renamings ρ, $solv_{\mathcal{X}}(c) = solv_{\mathcal{X}}(\rho(c))$
- $Ax_{\mathcal{X}}, \mathcal{D}_{\mathcal{X}}$ and $solv_{\mathcal{X}}$ agree in the sense that:
  1. $\mathcal{D}_{\mathcal{X}}$ is a model of $Ax_{\mathcal{X}}$.
  2. For all $c \in \mathcal{L}_{\mathcal{X}} \cap Sent_{\Sigma_{\mathcal{X}}}$: $solv_{\mathcal{X}}(c) = \mathbf{T} \Rightarrow Ax_{\mathcal{X}} \models c$.
  3. For all $c \in \mathcal{L}_{\mathcal{X}} \cap Sent_{\Sigma_{\mathcal{X}}}$: $solv_{\mathcal{X}}(c) = \mathbf{F} \Rightarrow Ax_{\mathcal{X}} \models \neg c$.

Moreover, $solv_{\mathcal{X}}$ must be well-behaved, i.e., for any constraints c_1 and c_2:

1. $solv_{\mathcal{X}}(c_1) = solv_{\mathcal{X}}(c_2)$ if $\models c_1 \leftrightarrow c_2$.
2. If $solv_{\mathcal{X}}(c_1) = \mathbf{F}$ and $\models c_1 \leftarrow c_2^{\exists \setminus free(c_1)}$ then $solv_{\mathcal{X}}(c_2) = \mathbf{F}$.

In what follows, a constraint domain $\mathcal{X} = (\Sigma_{\mathcal{X}}, \mathcal{L}_{\mathcal{X}}, Ax_{\mathcal{X}}, \mathcal{D}_{\mathcal{X}}, solv_{\mathcal{X}})$ will be called a $(\Sigma_{\mathcal{X}}, \mathcal{L}_{\mathcal{X}})$-constraint domain.

2.3 Constraint Logic Programs

A constraint logic program over a $(\Sigma_{\mathcal{X}}, \mathcal{L}_{\mathcal{X}})$-constraint domain $\mathcal{X}$ can be seen as a generalization of a definite logic program. In particular, a constraint logic program consists of rules $p :- q_1, ..., q_n$, where each q_i is either an atom or a constraint in $\mathcal{L}_{\mathcal{X}}$ and where atoms have the form $q(t_1, \ldots, t_n)$ where q is a user-defined predicate and $t_1, \ldots, t_n$ are terms over $\Sigma_{\mathcal{X}}$. A constraint logic program rule

$$p(t_1, \ldots, t_n) :- q_1, ..., q_n$$

can be written, equivalently, in flat form

$$p(X_1, \ldots, X_n) :- q_1, ..., q_n, X_1 = t_1, \ldots X_n = t_n$$

where $X_1, \ldots, X_n$ are fresh new variables. In what follows we will assume that constraint logic programs consist only of flat rules. We will also assume that the rules are written as follows:

$$p :- q_1, ..., q_n \Box c_1, ..., c_m$$

where the $q_1, ..., q_n$ are atoms and the $c_1, ..., c_m$ are constraints. Moreover we will also assume that all clauses defining the same predicate p have exactly the same head $p(X_1, \ldots, X_m)$.

The semantics of a $(\Sigma_{\mathcal{X}}, \mathcal{L}_{\mathcal{X}})$-logic program P can be also seen as a generalization of the semantics of a (non-constrained) logic program. In particular, in [10, 11], the meaning of P is given in terms of the usual three kinds of semantics.

The *operational semantics* is defined in terms of finite or infinite derivations

$$S_1 \rightsquigarrow S_2 \rightsquigarrow \ldots \rightsquigarrow S_n \ldots$$

where the states S_i in these derivations are tuples $G_i \Box C_i$, where G_i is a goal (i.e., a sequence of atoms) and C_i is a sequence of constraints (actually a constraint, since constraints are closed under conjunction). In particular, from a state $S = G \Box C$ we can derive the state $S' = G' \Box C'$ if there is an atom $p(t_1, \ldots, t_n)$ in G, and a rule $p(X_1, \ldots, X_n) : - G_0 \Box C_0$, where $X_1, \ldots, X_n$ are fresh new variables not occurring in $G \Box C$, such that $G' =< G_0, (G \backslash p(t_1, \ldots, t_n)) >$ and $C' =< C, C_0, X_1 = t_1, \ldots X_n = t_n >$ is satisfiable. Then, given a derivation $S_1 \rightsquigarrow S_2 \rightsquigarrow \ldots \rightsquigarrow S_n$, with $S_n = G_n \Box C_n$, we say that C_n is an answer to the query $S_1 = G_1 \Box C_1$ if G_n is the empty goal.

The *logical semantics* of P is defined as the theory presented by $P \cup Ax_{\mathcal{X}}$.

Finally its *algebraic semantics*, $M(P, \mathcal{X})$, is defined as the least model of P extending $\mathcal{D}_{\mathcal{X}}$, in the sense that this model agrees with $\mathcal{D}_{\mathcal{X}}$ in the corresponding universe of values and in the interpretation of the symbols in $\Sigma_{\mathcal{X}}$. It may be noted that Σ-structures extending $\mathcal{D}_{\mathcal{X}}$ can be seen as subsets of $Base_P(\mathcal{D}_{\mathcal{X}})$, where $Base_P(\mathcal{D}_{\mathcal{X}})$ is the set of all atoms of the form $p(\alpha_1, \ldots, \alpha_n)$, where p is a user-defined predicate and $\alpha_1, \ldots, \alpha_n$ are values in $\mathcal{D}_{\mathcal{X}}$.

As in the standard case, the algebraic semantics of P can be defined as the least fixpoint of the immediate consequence operator $T_P^{\mathcal{X}} : 2^{Base_P(\mathcal{D}_{\mathcal{X}})} \to 2^{Base_P(\mathcal{D}_{\mathcal{X}})}$ defined as follows:

$$T_P^{\mathcal{X}}(I) = \{\sigma(p) \mid \sigma : free(p) \to \mathcal{D}_{\mathcal{X}} \text{ is a valuation, } (p :- \overline{a} \Box c) \in P,\ I \models_\sigma \overline{a} \text{ and } \mathcal{D}_{\mathcal{X}} \models_\sigma c\}$$

In [11] it is proved that the above three semantics are equivalent in the sense that:

– The operational semantics is sound with respect to the logical semantics. That is, if a goal G has answer c then
$$P \cup Ax_{\mathcal{X}} \models c \to G$$
– The operational semantics is also sound with respect to the algebraic semantics. That is, if a goal G has answer c then
$$M(P, \mathcal{X}) \models c \to G$$
– The operational semantics is complete with respect to the logical semantics. That is, if
$$P \cup Ax_{\mathcal{X}} \models c \to G$$
then G has answers $c_1, \ldots, c_n$ such that
$$Ax_{\mathcal{X}} \models c \leftrightarrow c_1 \vee \ldots \vee c_n$$
– The operational semantics is complete with respect to the algebraic semantics. That is, if
$$M(P, \mathcal{X}) \models_\sigma G$$
where $\sigma : free(G) \to \mathcal{D}_{\mathcal{X}}$ is a valuation, then G has an answer c such that
$$\mathcal{D}_{\mathcal{X}} \models_\sigma c$$

2.4 A functorial semantics for constraint logic programs

The semantic definitions sketched in the previous subsection are, in our opinion, not fully satisfactory. On one hand, a constraint logic program can be seen as a logic program parameterized by the constraint domain. Then, we think that its semantics should also be parameterized by the domain. This is not explicit in the semantics sketched above. On the other hand, we think that the formulation of some of the previous equivalence results could be found to be, in some sense, not fully satisfactory. Let us consider, for instance, the last result, i.e., the completeness of the operational semantics with respect to the algebraic semantics. In our opinion, a fully satisfactory result would have said something like:

if $M(P,\mathcal{X}) \models_\sigma G$ *where* $\sigma : free(G) \to \mathcal{D}_\mathcal{X}$ *is a valuation, then* G *has an answer* c *such that* $solv_\mathcal{X}(c) \neq \mathbf{F}$

However this property will not hold unless the constraint solver $solv_\mathcal{X}$ is also complete with respect to the computation domain. A similar situation would occur with the result stating the completeness of the operational semantics with respect to the logical semantics. In that case we would need that the $solv_\mathcal{X}$ is complete with respect to the domain theory.

In our opinion, each of the three semantics (logical, algebraic and operational semantics) of a constraint logic program should be some kind of mapping. Moreover, we can envision that the parameters of the logical definitions would be constraint theories. Similarly, the parameters for algebraic definitions would be computation domains. Finally, the parameters for the operational definitions would be constraint solvers.

In this context, proving the soundness and completeness of one semantics with respect to another one would mean comparing the corresponding mappings. In particular, a given semantics would be sound and complete with respect to another one if the two semantic mappings are in some sense equivalent. Or, in more detail, if the two mappings when applied to the same (or equivalent) argument return an equivalent result. On the other hand, we believe that these mappings are better studied if the given domains and codomains are not just sets or classes but categories, which means taking care of their underlying structure. As a consequence, these mappings would be defined as functors and not just as plain set-theoretic functions, which means that they must be structure-preserving mappings.

In Section 5 the above ideas are fully developed for the case of constraint normal logic programs. Then, the case of constraint logic programs can be seen as a particular case.

3 Domain constraints for constraint normal logic programs

In this section, we provide a notion of constraint domain for constraint normal logic programming. The idea, as discussed in the introduction, is that this notion, together with a proper adaptation of the semantic constructions used for (unconstrained) normal logic programs, will provide an adequate semantic definition for constraint normal logic programs. In particular, the idea is that the logical semantics of a program should be given in terms of the (3-valued) Clark-Kunen completion of the program, the operational semantics in terms of some form of constructive negation [19, 5, 6], and the algebraic semantics in terms of some form of fixpoint construction (as, for example, in [19, 6, 15]).

The main problem is that a straightforward extension (as it may be just the inclusion of negated atoms in the constraint languages) of the notion of constraint domain introduced in Subsection 2.2 will not work, as the following example shows.

Example 2 *Let P be the CLP($\mathcal{N}$) program:*

$q(z) :- \square z = 0$
$q(v) :- q(x) \square v = x + 1$

and assume that its logical semantics is given by its completion:

$$\forall z(q(z) \leftrightarrow (z = 0 \vee \exists x(q(x) \wedge v = x + 1))).$$

This means, obviously, that $q(n)$ should hold for every n. Actually, the model defined by the algebraic semantics seen in Subsection 2.2 would satisfy $\forall z q(z)$.

Now consider that P is extended by the following definitions:

$r :- \neg q(x)$
$s :- \neg r$

whose completion is:

$$(r \leftrightarrow \exists x(\neg q(x))) \;\wedge\; (s \leftrightarrow \neg r).$$

Now, the operational semantics, and also the ω-iteration of the Fitting's operator [7], would correspond to a three-valued structure extending $\mathcal{N}$, where both r and s are undefined and where, as before, $q(n)$ holds for every n. Unfortunately, such a structure would not be a model of the completion of the program since this structure satisfies $\forall z q(z)$ but it does not satisfy either $\neg r$ or s. ■

The problem with the example above is that, if the algebraic semantics is defined by means of the ω-iteration of an immediate consequence operator, then, in many cases, the resulting structure would not be a model of the completion of the program. Otherwise, if we define the algebraic semantics in terms of some least (with respect to some order relation) model of the completion extending $\mathcal{N}$, then, in many cases, the operational semantics would not be complete with respect to that model. Actually, in some cases this model could be non (semi-)computable ([2], [8]).

The situation could be considered similar to what happens in the case of (non-constrained) normal logic programs, where the least fixpoint of Fitting's operator may not agree with the operational semantics of a given program. However, the situation is worse in the current case. On one hand, in the non-constrained case one may define other immediate consequence operators (e.g. [6, 15]) whose least fixpoint is equivalent to the operational semantics of a given program and provides a model of the 3-valued completion of the program. Unfortunately these operators would not be adequate in the constrained case. For instance, in the example above they would build models which are not extensions of $\mathcal{N}$. On the other hand, if when defining the logical semantics of a program we restrict our attention to the structures extending $\mathcal{N}$ (i.e., if we consider that the class of models of a program P is the class of all 3-valued structures satisfying $Comp(P)$ and extending $\mathcal{N}$) then we cannot expect the operational semantics to be complete with respect to the logical consequences of this class of models.

In our opinion, the problem is related to the following observation. Let us suppose, in the example above, that the computation domain would have been any other algebra which is elementarily equivalent to the algebra of the natural numbers, instead of $\mathcal{N}$ itself. Then, no difference should have been noticed, since both algebras satisfy exactly the same constraints, i.e., we may consider that two structures that are elementarily equivalent should be considered indistinguishable as domains of computation for a given constraint domain. As a consequence, we may consider that the semantics of a program over two indistinguishable constraint domains should also be indistinguishable. However, if $\mathcal{X} = (\Sigma, \mathcal{L}, Ax, D, solv)$ and $\mathcal{X}' = (\Sigma, \mathcal{L}, Ax, D', solv)$ are two constraint domains such that D and D' are elementarily equivalent and P is a $(\Sigma, \mathcal{L})$-program, then $M(P,\mathcal{X})$ and $M(P,\mathcal{X}')$ are not necessarily elementarily equivalent. In particular if we consider the program P of Example 2 and we consider as constraint domain a non-standard model of the natural numbers $\mathcal{N}'$, then we would have that $M(P,\mathcal{N}) \models \forall z q(z)$ but $M(P,\mathcal{N}') \not\models \forall z q(z)$.

In this sense, we think that this problem is caused by considering that the domain of computation, $\mathcal{D}_\mathcal{X}$, of a constraint domain is a single structure. In the case of programs without negation this apparently works fine and it seems quite reasonable from an intuitive point of view. For instance, if we are writing programs over the natural numbers, it seems reasonable to think that the computation domain is the algebra of natural numbers. However, when dealing with negation, we think that the computation domain of a constraint domain should be defined in terms of the class of all the structures which are elementarily equivalent to a given one. To be precise, we reformulate the notion of constraint domain as follows:

Definition 3 *A constraint domain $\mathcal{X}$ is a 5-tuple:*

$$\mathcal{X} = (\Sigma_\mathcal{X}, \mathcal{L}_\mathcal{X}, Ax_\mathcal{X}, Dom_\mathcal{X}, solv_\mathcal{X})$$

where $\Sigma_\mathcal{X} = (FS_\mathcal{X}, PS_\mathcal{X})$ is the constraint signature, $\mathcal{L}_\mathcal{X}$ is the constraint language, $Dom_\mathcal{X} = EQ(\mathcal{D}_\mathcal{X})$ is the domain of computation, i.e., the class of of all $\Sigma_\mathcal{X}$-structures which are elementarily equivalent to a given structure $\mathcal{D}_\mathcal{X}$, $Ax_\mathcal{X}$ is a decidable set of $\Sigma_\mathcal{X}$-sentences such that $\mathcal{D}_\mathcal{X} \models Ax_\mathcal{X}$, and $solv_\mathcal{X}$ is a constraint solver, such that:

- $\mathbf{T}, \mathbf{F}, t_1 = t_2 \in \mathcal{L}_\mathcal{X}$ *(hence the equality symbol $=$ belongs to $PS_\mathcal{X}$) and $\mathcal{L}_\mathcal{X}$ is closed under variable renaming, existential quantification, conjunction and negation. Moreover, the equality symbol $=$ is interpreted as the equality in $Dom_\mathcal{X}$ and $Ax_\mathcal{X}$ includes the equality axioms for $=$.*
- *The solver does not take variable names into account, that is, for all variable renamings ρ, $solv_\mathcal{X}(c) = solv_\mathcal{X}(\rho(c))$*
- *$Ax_\mathcal{X}, Dom_\mathcal{X}$ and $solv_\mathcal{X}$ agree in the sense that:*
  1. *$\mathcal{D}_\mathcal{X}$ is a model of $Ax_\mathcal{X}$.*

2. *For all $c \in \mathcal{L}_\mathcal{X} \cap Sent_\Sigma$: $solv_\mathcal{X}(c) = \textbf{T} \Rightarrow Ax_\mathcal{X} \models c$.*
3. *For all $c \in \mathcal{L}_\mathcal{X} \cap Sent_\Sigma$: $solv_\mathcal{X}(c) = \textbf{F} \Rightarrow Ax_\mathcal{X} \models \neg c$.*

In addition, we assume that $solv_\mathcal{X}$ is well-behaved, i.e., that for any constraints c_1 and c_2:

1. *$solv_\mathcal{X}(c_1) = solv_\mathcal{X}(c_2)$ if $\models c_1 \leftrightarrow c_2$.*
2. *If $solv_\mathcal{X}(c_1) = \textbf{F}$ and $\models c_1 \leftarrow c_2^{\exists\setminus free(c_1)}$ then $solv_\mathcal{X}(c_2) = \textbf{F}$.*

As before, a constraint domain $\mathcal{X} = (\Sigma_\mathcal{X}, \mathcal{L}_\mathcal{X}, Ax_\mathcal{X}, Dom_\mathcal{X}, solv_\mathcal{X})$ is called a $(\Sigma_\mathcal{X}, \mathcal{L}_\mathcal{X})$-constraint domain.

4 Semantic constructions for constraint normal logic programs

Analogously to constraint logic programs, given a signature $\Sigma = (PS_\Sigma, FS_\Sigma)$, normal constraint logic Σ-programs over a constraint domain $\mathcal{X} = (\Sigma_\mathcal{X}, \mathcal{L}_\mathcal{X}, Ax_\mathcal{X}, Dom_\mathcal{X}, solv_\mathcal{X})$, can be seen as a generalization of a normal logic programs. So, a Σ-program now consists of clauses of the form

$$a :- \ell_1, ..., \ell_m \square c_1, \ldots, c_n$$

where a and the ℓ_i, $i \in \{1, \ldots, m\}$, are a flat atom and flat literals, respectively, whose predicate symbols belong to $PS_\Sigma \setminus PS_\mathcal{X}$ and the c_j, $j \in \{1, \ldots, n\}$ belong to $\mathcal{L}_\mathcal{X}$. For this class of programs, we also assume that all clauses defining the same predicate p have exactly the same head $p(X_1, \ldots, X_m)$.

4.1 Logical semantics

The standard logical meaning of a Σ-program P is its (generalized) Clark's completion $Comp_\mathcal{X}(P) = Ax_\mathcal{X} \cup P^*$, where P^* includes a sentence

$$\forall \bar{z}(q(\bar{z}) \leftrightarrow ((G_1 \wedge c_1)^{\exists\setminus\bar{z}} \vee \ldots \vee (G_k \wedge c_k)^{\exists\setminus\bar{z}}))$$

for each $q \in PS_\Sigma \setminus PS_\mathcal{X}$, and where $\{(q(\bar{z}) :- G_1 \square c_1), \ldots, (q(\bar{z}) :- G_k \square c_k)\}$ is the set [3] of all the clauses in P with head predicate q. In what follows, this set will be denoted by $Def_P(q)$. Intuitively, in this semantics we are considering that $Def_P(q)$ is a *complete definition* of the predicate q. A weaker logical meaning for the program P is obtained by defining its semantics as $Ax_\mathcal{X} \cup P^\forall$, where $P^\forall$, is the set including a sentence

$$\forall \bar{z}(q(\bar{z}) \leftarrow ((G_1 \wedge c_1)^{\exists\setminus\bar{z}} \vee \ldots \vee (G_k \wedge c_k)^{\exists\setminus\bar{z}}))$$

for each $q \in PS_\Sigma \setminus PS_\mathcal{X}$, and where, as above, $Def_P(q) = \{(q(\bar{z}) :- G_1 \square c_1), \ldots, (q(\bar{z}) :- G_k \square c_k)\}$.

[3] If there are no clauses in P with head predicate q, i.e., the set is empty, then the above sentence is simplified to $\forall \bar{z}(q(\bar{z}) \leftrightarrow \textbf{F}$

4.2 The *BCN* operational semantics

In this section we generalize the *BCN* operational semantics introduced in [16] and refined in [1] in such a way that it can be used for any constraint domain. The *BCN* operational semantics is based on two operators originally introduced by Shepherdson [18] to characterize Clark-Kunen's semantics in terms of satisfaction of (equality) constraints. Such operators exploit the definition of literals in the completion of programs and associate a constraint formula to each query. As a consequence, the answers are computed, on one hand, by a symbolic manipulation process that obtains the associated constraint(s) of the given query and, on the other hand, by a constraint checking process that deals with such constraint(s). In particular, the original version ([16]) of the *BCN* operational semantics works with programs restricted to the constraint domain of terms with equality. In that case, *BCN* uses the equality theory $\mathcal{FEA}$, defined by Clark [4] (or any equation solver) as a solver. Here, we generalize this semantics to arbitrary constraint domains.

Definition 4 *For any program P, the operators T_k^P and F_k^P associate a constraint to each query, as follows:*

Let $Def_P(q)=\{q(\overline{z}) :- \overline{\ell}_i \Box c_i \mid 1 \leq i \leq m\}$

$$T_0^P(q(\overline{z})) = \mathbf{F} \qquad T_{k+1}^P(q(\overline{z})) = \bigvee_{i=1}^m \exists \overline{y}^i (c_i \wedge T_k^P(\overline{\ell}_i))$$

$$F_0^P(q(\overline{z})) = \mathbf{F} \qquad F_{k+1}^P(q(\overline{z})) = \bigwedge_{i=1}^m \forall \overline{y}^i (\neg c_i \vee F_k^P(\overline{\ell}_i))$$

For all $k \in \mathbb{N}$:

$$T_k^P(\mathbf{T}) = \mathbf{T} \qquad F_k^P(\mathbf{T}) = \mathbf{F}$$

$$T_k^P(\neg q(\overline{z})) = F_k^P(q(\overline{z})) \qquad F_k^P(\neg q(\overline{z})) = T_k^P(q(\overline{z}))$$

$$T_k^P(\bigwedge_{j=1}^n \ell_j) = \bigwedge_{j=1}^n T_k^P(\ell_j) \qquad F_k^P(\bigwedge_{j=1}^n \ell_j) = \bigvee_{j=1}^n F_k^P(\ell_j)$$

For any $c \in \mathcal{L}_{\mathcal{X}}$, for any $k \in \mathbb{N}$:

$$T_k^P(c) = c \qquad F_k^P(c) = \neg c$$

Definition 5 *Let P be a program and $solv_{\mathcal{X}}$ a constraint solver. A $BCN(P, solv_{\mathcal{X}})$-*derivation step *is obtained by applying the following derivation rule:*

(R) $\overline{\ell}_1, \overline{\ell}_2 \Box d$ *is $BCN(P, solv_{\mathcal{X}})$-*derived *from* $\overline{\ell}_1, \underline{\ell(\overline{x})}, \overline{\ell}_2 \Box c$ *if there exists $k > 0$ such that* $d = T_k^P(\ell(\overline{x})) \wedge c$ *and* $solv_{\mathcal{X}}(d^{\exists}) \neq \mathbf{F}$.

Definition 6 *Let P be a program and $solv_{\mathcal{X}}$ a constraint solver.*

1. *A $BCN(P, solv_{\mathcal{X}})$-*derivation *from the query L is a succession of $BCN(P, solv_{\mathcal{X}})$-derivation steps of the form*

$$L \rightsquigarrow_{(P, solv_{\mathcal{X}})} \ldots \rightsquigarrow_{(P, solv_{\mathcal{X}})} L'$$

Then, $L \overset{n}{\rightsquigarrow}_{(P, solv_{\mathcal{X}})} L'$ means that the query L' is $BCN(P, solv_{\mathcal{X}})$-derived from the query L in n $BCN(P, solv_{\mathcal{X}})$-derivation steps.

2. *A finite $BCN(P,solv_{\mathcal{X}})$-derivation $L \overset{n}{\rightsquigarrow}_{(P,solv_{\mathcal{X}})} L'$ is a* successful $BCN(P,solv_{\mathcal{X}})$-derivation *if $L' = \square c$.*
   In this case, $c^{\exists \setminus free(L)}$ is the corresponding $BCN(P,solv_{\mathcal{X}})$-computed answer.
3. *A query $L = \bar{\ell} \square c$ is a $BCN(P,solv_{\mathcal{X}})$*-failed query *if $solv_{\mathcal{X}}((c \rightarrow F_k^P(\bar{\ell}))^{\forall}) = \mathbf{T}$ for some $k > 0$ such that $solv_{\mathcal{X}}(F_k^P(\bar{\ell})^{\forall}) \neq \mathbf{F}$.*

A *selection rule* is a function selecting a literal in a query and, whenever *Solvx* is well-behaved, $BCN(P,solv_{\mathcal{X}})$ is independent of the selection rule used. To prove this assertion we follow the strategy used in [14, 11], so we first prove the next lemma.

Lemma 7 (Switching Lemma) *Let P be a program and Solvx be a well-behaved solver. Let L be a query, ℓ_1, ℓ_2 be literals in L and let $L \rightsquigarrow_{(P,solv_{\mathcal{X}})} L_1 \rightsquigarrow_{(P,solv_{\mathcal{X}})} L'$ be a non-failed derivation in which ℓ_1 has been selected in L and ℓ_2 in L_1. Then there is a derivation $L \rightsquigarrow_{(P,solv_{\mathcal{X}})} L_2 \rightsquigarrow_{(P,solv_{\mathcal{X}})} L''$ in which ℓ_2 has been selected in L and ℓ_1 in L_2, and L' and L'' are identical up to reordering of their constraint component.*

Theorem 8 (Independence of the selection rule) *Let P be a program and $solv_{\mathcal{X}}$ a well-behaved solver. Let L be a query and suppose that there exists a successful $BCN(P,solv_{\mathcal{X}})$-derivation from L with computed answer c. Then, using any selection rule R there exists another successful $BCN(P,solv_{\mathcal{X}})$-derivation from L of the same length with an answer which is a reordering of c.*

Next, we establish the basis for relating the $BCN(P,solv_{\mathcal{X}})$ operational semantics to the logical semantics of a particular class of constraint logic programs. The proposition below provides the basis for proving soundness and completeness of the semantics.

Proposition 9 *Let $\Sigma = (FS_{\mathcal{X}}, PS_{\mathcal{X}} \cup PS)$ be an extension of a given signature of constraints $\Sigma_{\mathcal{X}} = (FS_{\mathcal{X}}, PS_{\mathcal{X}})$ by a set of predicates PS, and let P be a Σ-program. Then, for each $\Sigma_{\mathcal{X}}$-theory of constraints $Ax_{\mathcal{X}}$, each conjunction of Σ-literals $\bar{\ell}$ and each k in $\mathbb{N}$:*

$$P^* \cup Th(Ax_{\mathcal{X}}) \models (T_k^P(\bar{\ell}) \rightarrow \bar{\ell})^{\forall}$$

4.3 Fixpoint semantics

According to what is argued in Section 3, we consider the domain $(Dom_{\Sigma}/_{\equiv}, \preceq)$ for computing immediate consequences defined as follows: Let Dom_{Σ} be the class of three-valued Σ-interpretations which are extensions of models in $Dom_{\mathcal{X}}$. Then, as it is done in [19] to extend [13] to the general constraint case, we consider the Fitting's ordering on Dom_{Σ} interpreted in the following sense: For all partial interpretations $\mathcal{A}, \mathcal{B} \in Dom_{\Sigma}$, for each $\Sigma_{\mathcal{X}}$-constraint $c(\bar{x})$ and each Σ-literal $\ell(\bar{x})$:

$$\mathcal{A} \preceq \mathcal{B} \;\; iff \;\; \mathcal{A}((c \rightarrow \ell)^{\forall}) = \underline{t} \Rightarrow \mathcal{B}((c \rightarrow \ell)^{\forall}) = \underline{t}$$

It is quite easy to see that $(Dom_{\Sigma}, \preceq)$ is a preorder. Therefore, we consider the equivalence relation $\equiv$ induced by $\preceq$ ($\mathcal{A} \equiv \mathcal{B}$ if, and only if, $\mathcal{A} \preceq \mathcal{B}$ and $\mathcal{B} \preceq \mathcal{A}$), and the induced partial order

$$[\mathcal{A}], [\mathcal{B}] \in Dom_{\Sigma}/_{\equiv} : [\mathcal{A}] \preceq [\mathcal{B}] \;\; iff \;\; \mathcal{A} \preceq \mathcal{B}$$

to build a cpo $(Dom_\Sigma/_\equiv, \preceq)$ with a bottom class $[\bot_\Sigma]$ such that for each $\mathcal{A} \in [\bot_\Sigma]$ we have that $\mathcal{A}((c \to \ell)^\forall) \neq \underline{t}$ for all Σ_X-constraint $c(\bar{x})$ and all Σ-literal $\ell(\bar{x})$. That is, the set of goals of the form $(c \to \ell)^\forall$ satisfied by the models in $[\bot_\Sigma]$ is empty.

Proposition 10 *$(Dom_\Sigma/_\equiv, \preceq)$ is a cpo with respect to $\preceq$, and, the equivalence class $[\bot_\Sigma]$ is its bottom element.*

Remark 11

1. *The relation $\equiv$ builds classes of models which are indistinguishable with respect to satisfaction of goal formulas.*
2. *Moreover, it is easy to see that all the models in a $\equiv$-class are elementarily equivalent in its restrictions to Σ_X.*

Definition 12 (Immediate consequence operator $\mathcal{T}_P^{Dom_X}$) *Let P be a Σ-program, then the immediate consequence operator $\mathcal{T}_P^{Dom_X} : Dom_\Sigma/_\equiv \longrightarrow Dom_\Sigma/_\equiv$ is defined for each $[\mathcal{A}] \in Dom_\Sigma/_\equiv$, as*

$$\mathcal{T}_P^{Dom_X}([\mathcal{A}]) = [\Phi_P^{\mathcal{D}_X}(\mathcal{A})]$$

where $\mathcal{D}_X$ is the distinguished domain model in the class Dom_X, $\mathcal{A}$ is any model in $[\mathcal{A}]$, and $[\Phi_P^{\mathcal{D}_X}(\mathcal{A})]$ is the $\equiv$-class of models such that for each Σ_X-constraint $c(\bar{x})$ and each Σ-atom $p(\bar{x})$,

(i) $\Phi_P^{\mathcal{D}_X}(\mathcal{A})((c \to p)^\forall) = \underline{t}$ *if, and only if, there are (renamed versions of) clauses* $\{p(\bar{x}) :- \ell_1^i, \ldots, \ell_{n_i}^i \Box d_i \mid 1 \leq i \leq m\} \subseteq Def_P(p)$ *and $\mathcal{D}_X$-satisfiable constraints* $\{c_j^i \mid 1 \leq i \leq m \wedge 1 \leq j \leq n_i\}$ *such that*
- $\mathcal{A}((c_j^i \to \ell_j^i)^\forall) = \underline{t}$
- $\mathcal{D}_X((c \to \bigvee_{1 \leq i \leq m} \exists \bar{y}_i (\bigwedge_{1 \leq j \leq n_i} c_j^i \wedge d_i))^\forall) = \underline{t}$

(ii) $\Phi_P^{\mathcal{D}_X}(\mathcal{A})((c \to \neg p)^\forall) = \underline{t}$ *if, and only if, for each (renamed version) clause in* $\{p(\bar{x}) :- \ell_1^i, \ldots, \ell_{n_i}^i \Box d_i \mid 1 \leq i \leq m\} = Def_P(p(\bar{x}))$ *there is a $J_i \subseteq \{1, \ldots n_i\}$ and $\mathcal{D}_X$-satisfiable constraints* $\{c_j^i \mid 1 \leq i \leq m \wedge j \in J_i\}$ *such that*
- $\mathcal{A}((c_j^i \to \neg \ell_j^i)^\forall) = \underline{t}$
- $\mathcal{D}_X((c \to \bigwedge_{1 \leq i \leq m} \forall \bar{y}_i (\bigvee_{j \in J_i} c_j^i \vee \neg d_i))^\forall) = \underline{t}$

where, for each $i \in \{1, \ldots, m\}$, $\bar{y}_i$ are the free variables in $\{\ell_1^i, \ldots, \ell_{n_i}^i, d_i\}$ not in $\bar{x}$.

Remark 13

1. *In the definition of the operator $\Phi_P^{\mathcal{D}_X}$, we could choose any other model in Dom_X, instead of $\mathcal{D}_X$, since all of them are elementarily equivalent, and the domain is just used for constraint satisfaction checking. Similarly, $\mathcal{A}$ could be any other model in $[\mathcal{A}]$ since it is used for checking satisfaction of sentences of the form $(c \to \ell)^\forall$.*
2. *Moreover, models in a $\equiv$-class $[\Phi_P^{\mathcal{D}_X}(\mathcal{A})]$ are elementarily equivalent in its restrictions to Σ_X. In fact, $[\Phi_P^{\mathcal{D}_X}(\mathcal{A})]|_{\Sigma_X} = Dom_X$ since, all classes in Dom_Σ are (conservative) predicative extensions of Dom_X and, the operator $\mathcal{T}_P^{Dom_X}$ does not compute new consequences from $\mathcal{L}_X$. However, neither $[\mathcal{A}]$ nor $\mathcal{T}_P^{Dom_X}([\mathcal{A}]) = [\Phi_P^{\mathcal{D}_X}(\mathcal{A})]$ are elementarily equivalence classes in general.*

In what follows we will prove that $\mathcal{T}_P^{Dom_\mathcal{X}}$ is continuous in the cpo $Dom_\Sigma/_\equiv$. As a consequence, it has an effectively computable least fixpoint:

$$lfp(\mathcal{T}_P^{Dom_\mathcal{X}}) = \mathcal{T}_P^{Dom_\mathcal{X}} \uparrow \omega = \bigsqcup[\Phi_P^{\mathcal{D}_\mathcal{X}} \uparrow n]$$

However, it is important to notice that, as we will show in example 14,

$$\bigsqcup[\Phi_P^{\mathcal{D}_\mathcal{X}} \uparrow n] \neq [\Phi_P^{\mathcal{D}_\mathcal{X}} \uparrow \omega]$$

In fact, the operator $\Phi_P^{\mathcal{D}_\mathcal{X}}$ can be considered a variant of the Stuckey's immediate consequence operator in [19], so, it inherits its drawbacks. On one hand, $\Phi_P^{\mathcal{D}_\mathcal{X}}$ is monotonic but not continuous. On the other hand, it will have different behavior depending on the constraint domain in $Dom_\mathcal{X}$ that may be predicatively extended. As argued in Section 3, the key to solve these problems is to use the whole class $Dom_\mathcal{X}$ as domain of computation instead of a single model. In fact, the key technical point is using its predicative extension, Dom_Σ, in defining the target and the source, $Dom_\Sigma/_\equiv$, of $\mathcal{T}_P^{Dom_\mathcal{X}}$, as the following example aims to illustrate.

Example 14 *Consider the CNLP($\mathcal{N}$)-program from example 2:*

$q(z) :- \square z = 0$
$q(v) :- q(x) \square v = x + 1$
$r :- \neg q(x)$

First, let us look at the behaviour of the operator Φ:

- $\Phi_P^{\mathcal{N}} \uparrow \omega$ *would be the model extending $\mathcal{N}$ where r is undefined and all the sentences*

$$\{(z = n \rightarrow q(z))^\forall | n \geq 0\}$$

  are true, so, the sentence $\forall z.q(z)$ will be evaluated as true in $\Phi_P^{\mathcal{N}} \uparrow \omega$. This is not a fixpoint since we can iterate once more, to obtain a different model $\Phi_P^{\mathcal{N}} \uparrow (\omega + 1)$ where $\neg r$ is true.
- *In contrast, if we consider any non-standard model $\mathcal{M}$ elementarily equivalent to $\mathcal{N}$, the sentence $\forall z.q(z)$ will be evaluated as undefined in $\Phi_P^{\mathcal{M}} \uparrow \omega$, so, no more consequences will be obtained if we iterate once more.*

Now we can compare with the behaviour of $\mathcal{T}$:

Similar to the first case, $\mathcal{T}_P^{EQ(\mathcal{N})} \uparrow \omega$ is the class of $\equiv$-equivalent models extending $EQ(\mathcal{N})$, where r is undefined and all the sentences

$$\{(z = n \rightarrow q(z))^\forall | n \geq 0\}$$

are true. But now, this is a fixpoint in contrast to what happens with any other operator working over just one standard model. In particular, it is not difficult to see that the *sentence $\forall z.q(z)$ is never satisfied (by models) in $[\Phi_P^{\mathcal{N}} \uparrow k]$ for any k. This is because we are considering also non standard models (as the predicative extension of the above $\mathcal{M}$) at each iteration. Therefore, as a consequence of the definition of $\bigsqcup$, we have that $\forall z.q(z)$ is not satisfied in*

$$\mathcal{T}_P^{EQ(\mathcal{N})} \uparrow \omega = \bigsqcup [\Phi_P^{\mathcal{N}} \uparrow k]$$

That is, $\neg r$ is not a consequence that can be added (or satisfied) if the iteration forward proceeds.

Theorem 15 *$\mathcal{T}_P^{Dom_X}$ is continuous in the cpo $(Dom_\Sigma/_\equiv, \preceq)$, so it has a least fixpoint $\mathcal{T}_P^{Dom_X} \uparrow \omega$.*

Finally, as a consequence of the continuity of $\mathcal{T}_P^{Dom_X}$, we can extend a result from Stuckey [19] related to the satisfaction of the logical consequences of the completion in any ordinal iteration of $\Phi_P^{\mathcal{D}_X}$, until the ω iteration of $\mathcal{T}_P^{Dom_X}$, that is, until its least fixpoint:

Theorem 16 (Extended Theorem of Stuckey)
Let $Th(Dom_X)$ be the complete theory of Dom_X. For each Σ-goal $\overline{\ell}\Box c$:

1. *$P^* \cup Th(Dom_X) \models_3 (c \rightarrow \overline{\ell})^\forall \Leftrightarrow \forall \mathcal{A} \in \mathcal{T}_P^{Dom_X} \uparrow \omega : \mathcal{A}((c \rightarrow \overline{\ell})^\forall) = \underline{t}$*
2. *$P^* \cup Th(Dom_X) \models_3 (c \rightarrow \neg\overline{\ell})^\forall \Leftrightarrow \forall \mathcal{A} \in \mathcal{T}_P^{Dom_X} \uparrow \omega : \mathcal{A}((c \rightarrow \neg\overline{\ell})^\forall) = \underline{t}$*

5 Functorial semantics

As introduced in Subsection 2.4, one basic idea in this work is to formulate the constructions associated to the definition of the operational, least fixpoint and logical semantics of constraint normal logic programs in functorial terms. This allows us to separate the study of the properties satisfied by these three semantic constructions, from the classic comparisons of three kinds of semantics of programs over a specific constraint domain. Moreover, once the equivalence of semantic constructions is (as intended) obtained, the classical *soundness* and *completeness* results that can be obtained depending on the relations among solvers, theories and domains, are just consequences of the functorial properties.

However, comparing these semantic functors is not straightforward since, intuitively, their domains and codomains are different categories. In particular, we can see that the logical semantics of a $(\Sigma_X, \mathcal{L}_X)$-constraint logic program P as a mapping (a functor), let us denote it by $\mathcal{LOG}_P$, whose arguments are logical theories and whose results are also logical theories. The algebraic semantics of P, denoted $\mathcal{ALG}_P$, can be seen as a functor that takes as arguments logical structures and returns as results logical structures. Finally, the operational semantics of P, denoted $\mathcal{OP}_P$ can be considered to take as arguments constraint solvers and return as results (for instance) interpretations of computed answers.

Now, comparing the algebraic and the logical semantics is not too difficult, since we can consider logical theories not as sets of formulas but, equivalently, as classes of logical structures. In this way, the domains and codomains of $\mathcal{LOG}_P$ and $\mathcal{ALG}_P$ would be, in both cases, (classes of) logical structures. Of course, we could also associate classes of models to solvers, but given this semantics to solvers would not be adequate. In particular, this would be equivalent to closing the solver (the associated set of non unsatisfiable constraints) up to logical consequence. The problem is that the class of all models that satisfy a given set of formulas (constraints) would also satisfy all its logical consequences. However, solvers may not show a logical behaviour (even if they are well-behaved according to Section 2.2). A solver may say that certain constraints are unsatisfiable but may be unable to say that some other constraint is unsatisfiable, even if its unsatisfiability is a logical consequence of the unsatisfiability of the former constraints.

We take actually the dual approach: we will represent all the semantic domains involved in terms of sets of formulas. This is a quite standard approach in the area of Logic Programming where, for instance, (finitely generated) models are often represented as Herbrand structures (i.e., as classes of ground atoms) rather than as algebraic structures. One could criticize this approach in the framework of constraint logic programming, since a class does not faithfully represents a single model (the constraint domain of computation $Dom_{\mathcal{X}}$) but a class of models. However, we have argued previously that, when dealing with negation, a constraint domain of computation should not be a single model, but the class of models which are elementarily equivalent to $Dom_{\mathcal{X}}$. In this sense, one may note that a class of elementarily equivalent models is uniquely represented by a complete theory. However, since we are dealing with three-valued logic, we are going to represent model classes, theories and solvers as pairs of sets of sentences, rather than just as single sets.

In what follows, we present the categorical setting required for our purposes. Being more precise, first of all, we need to define the categories associated to solvers, computation domains and theories (axiomatizable domains). Then, we will define the category which properly represents the semantics of programs. Finally, we will define the three functors that respectively represent the operational, logical and algebraic semantics of a constraint normal logic programs.

Definition 17 *Given a signature $\Sigma_{\mathcal{X}}$, a $\Sigma_{\mathcal{X}}$-pre-theory $\mathcal{M}$ is a pair of sets of $\Sigma_{\mathcal{X}}$-sentences $(\mathcal{M}_{\underline{t}}, \mathcal{M}_{\underline{f}})$.*

Remarks and Definitions 18

1. *Given a solver $solv_{\mathcal{X}}$ of a given language $\mathcal{L}_{\mathcal{X}}$ of $\Sigma_{\mathcal{X}}$-constraints, we will denote by $\mathcal{M}_{solv_{\mathcal{X}}}$ the pre-theory associated to $solv_{\mathcal{X}}$, i.e., the pair $(\mathcal{M}_{\underline{t}}, \mathcal{M}_{\underline{f}})$ where $\mathcal{M}_{\underline{t}}$ is the set of all constraints $c \in \mathcal{L}_{\mathcal{X}}$ such that $solv_{\mathcal{X}}(c) = \mathbf{T}$ and $\mathcal{M}_{\underline{f}}$ is the set of all constraints $c \in \mathcal{L}_{\mathcal{X}}$ such that $solv_{\mathcal{X}}(c) = \mathbf{F}$.*
2. *Similarly, given a set of axioms $Ax_{\mathcal{X}}$ of a given language $\mathcal{L}_{\mathcal{X}}$ of $\Sigma_{\mathcal{X}}$-constraints, we will denote by $\mathcal{M}_{Ax_{\mathcal{X}}}$ the theory associated to $Ax_{\mathcal{X}}$.*
3. *Finally, given a computation domain $Dom_{\mathcal{X}}$ of a given language $\mathcal{L}_{\mathcal{X}}$ of $\Sigma_{\mathcal{X}}$-constraints, we will denote by $\mathcal{M}_{Dom_{\mathcal{X}}}$ the theory associated to $Dom_{\mathcal{X}}$, i.e., the pair $(\mathcal{M}_{\underline{t}}, \mathcal{M}_{\underline{f}})$ where $\mathcal{M}_{\underline{t}}$ is the set of sentences satisfied by $Dom_{\mathcal{X}}$ and $\mathcal{M}_{\underline{f}}$ is the set of sentences which are false in $Dom_{\mathcal{X}}$. Note that, since constraint domains are typically two-valued, $\mathcal{M}_{\underline{t}}$ would typically be a complete theory and, therefore, $\mathcal{M}_{\underline{f}}$ is the complement of $\mathcal{M}_{\underline{t}}$.*

For the sake of simplicity, given a pre-theory $\mathcal{M}$, we will write $\mathcal{M}(c) = \underline{\mathtt{t}}$, to mean $c \in \mathcal{M}_{\underline{\mathtt{t}}}$; $\mathcal{M}(c) = \underline{\mathtt{f}}$, to mean $c \in \mathcal{M}_{\underline{\mathtt{f}}}$; and $\mathcal{M}(c) = \underline{\mathtt{u}}$, otherwise.

Now, according to the above ideas, we will define categories to represent constraint solvers, computation domains and domain axiomatizations. Also, following similar ideas we are going to define a category of semantic domains for programs. In this case, we will define the semantics in terms of sets of formulas. However, we will restrict ourselves to sets of answers, i.e., formulas with the form $c \to G$, where G is any goal.

Definition 19 (Categories for Constraint Domains and Program Interpretations) *Given a signature $\Sigma_{\mathcal{X}}$ we can define the following categories:*

1. *the category of $\Sigma_{\mathcal{X}}$-pre-theories, $\underline{PreTh}_{\Sigma_{\mathcal{X}}}$(or just $\underline{PreTh}$ if $\Sigma_{\mathcal{X}}$ is clear from the context) is defined as follows:*
   - *Its class of objects is the class of $\Sigma_{\mathcal{X}}$-pre-theories.*
   - *For each pair of objects $\mathcal{M}$ and $\mathcal{M}'$ there is a morphism from $\mathcal{M}$ to $\mathcal{M}'$, noted just by $\mathcal{M} \preceq_c \mathcal{M}'$, if $\mathcal{M}_{\underline{\mathtt{t}}} \subseteq \mathcal{M}'_{\underline{\mathtt{t}}}$ and $\mathcal{M}_{\underline{\mathtt{f}}} \subseteq \mathcal{M}'_{\underline{\mathtt{f}}}$*
2. *$\underline{Th}_{\Sigma_{\mathcal{X}}}$ (or just $\underline{Th}$) is the full subcategory of $\underline{PreTh}_{\Sigma_{\mathcal{X}}}$ whose objects are theories.*
3. *$\underline{CompTh}_{\Sigma_{\mathcal{X}}}$ (or just $\underline{CompTh}$) is the full subcategory of $\underline{PreTh}_{\Sigma_{\mathcal{X}}}$ whose objects are complete theories*
4. *Given a constraint language $\mathcal{L}_{\mathcal{X}}$ and a signature Σ extending $\Sigma_{\mathcal{X}}$, $\underline{ProgInt}^{\Sigma}_{(\Sigma_{\mathcal{X}},\mathcal{L}_{\mathcal{X}})}$ (or just $\underline{ProgInt}$ if $\Sigma, \Sigma_{\mathcal{X}}$ and $\mathcal{L}_{\mathcal{X}}$ are clear from the context) is the category where:*
   - *Its objects are sets of sentences $(c \to \overline{\ell})^{\forall}$ or $(c \to \neg\overline{\ell})^{\forall}$, where $c \in \mathcal{L}_{\mathcal{X}}$ and $\overline{\ell}$ is a conjunction of Σ-literals.*
   - *For each pair of objects $\mathcal{A}$ and $\mathcal{A}'$ there is a morphism from $\mathcal{A}$ to $\mathcal{A}'$, noted just by $\mathcal{A} \preceq \mathcal{A}'$ if $\mathcal{A} \subseteq \mathcal{A}'$*
     - *$\Sigma_{\mathcal{X}} \subseteq \Sigma_{\mathcal{X}'}$ and*
     - *for each $(FS_{\mathcal{X}}, PS_{\mathcal{X}} \cup PS)$-literal $\ell(\overline{x})$ and $\Sigma_{\mathcal{X}}$-formula $c(\overline{x})$, $\mathcal{A}((c \to \overline{\ell})^{\forall}) = \underline{\mathtt{t}}$ implies $\mathcal{A}'((c \to \overline{\ell})^{\forall}) = \underline{\mathtt{t}}$, and $\mathcal{A}((c \to \neg\overline{\ell})^{\forall}) = \underline{\mathtt{t}}$ implies $\mathcal{A}'((c \to \neg\overline{\ell})^{\forall}) = \underline{\mathtt{t}}$.*

As pointed out before, this categorical formulation allows us to speak about relations among solvers, domains and theories by establishing morphisms among them in the common category $\underline{PreTh}$, in such a way that the morphism between two objects represents the relation *"agrees with"* (or *completeness* if they are seen in the reverse sense). To be more precise, given a constraint (domain) parameter $\mathcal{X} = (\Sigma_{\mathcal{X}}, \mathcal{L}_{\mathcal{X}}, Ax_{\mathcal{X}}, Dom_{\mathcal{X}}, solv_{\mathcal{X}})$, we can reformulate the conditions (in Section 2.2) required among $solv_{\mathcal{X}}$, $Dom_{\mathcal{X}}$ and $Ax_{\mathcal{X}}$ as:

$$\mathcal{M}_{solv_{\mathcal{X}}} \preceq_c \mathcal{M}_{Ax_{\mathcal{X}}} \preceq_c \mathcal{M}_{Dom_{\mathcal{X}}}$$

in $\underline{PreTh}$. That is, since $Dom_{\mathcal{X}}$ must be a model of $Ax_{\mathcal{X}}$, there is a morphism from $\mathcal{M}_{Ax_{\mathcal{X}}}$ to $\mathcal{M}_{Dom_{\mathcal{X}}}$. Moreover, since $solv_{\mathcal{X}}$ must agree with $Ax_{\mathcal{X}}$, there is a morphism from $\mathcal{M}_{solv_{\mathcal{X}}}$ to $\mathcal{M}_{Ax_{\mathcal{X}}}$. Then, by transitivity, *$solv_{\mathcal{X}}$ agrees with $Dom_{\mathcal{X}}$*, so there is a morphism $\mathcal{M}_{solv_{\mathcal{X}}}$ to $\mathcal{M}_{Dom_{\mathcal{X}}}$. In addition, we can also reformulate other conditions in these terms:

– $solv_{\mathcal{X}}$ is *$Ax_{\mathcal{X}}$-complete* (respectively, *$Dom_{\mathcal{X}}$-complete*) if, and only if, $\mathcal{M}_{Ax_{\mathcal{X}}} \preceq_c \mathcal{M}_{solv_{\mathcal{X}}}$ (respectively, $\mathcal{M}_{Dom_{\mathcal{X}}} \preceq_c \mathcal{M}_{solv_{\mathcal{X}}}$).
– $Ax_{\mathcal{X}}$ *completely* axiomatizes $Dom_{\mathcal{X}}$ if, and only if, $\mathcal{M}_{Dom_{\mathcal{X}}} \preceq_c \mathcal{M}_{Ax_{\mathcal{X}}}$, so, as expected $\mathcal{M}_{Ax_{\mathcal{X}}} = \mathcal{M}_{Dom_{\mathcal{X}}}$.

Finally, we will define the three functors that represent, for a given program P, its operational, its algebraic or least fixpoint, and its logical semantics.

Definition 20 (Functorial semantics) *Let P be a Σ-program. We can define three functors $\mathcal{OP}_P : \underline{PreTh} \rightarrow \underline{ProgInt}$, $\mathcal{ALG}_P : \underline{CompTh} \rightarrow \underline{ProgInt}$ and $\mathcal{LOG}_P : \underline{Th} \rightarrow \underline{ProgInt}$ such that:*

a) $\mathcal{OP}_P$, $\mathcal{ALG}_P$ and $\mathcal{LOG}_P$ assign objects $\mathcal{M}$ in its corresponding source category to objects in $\underline{ProgInt}$, in the following way

1. $\underline{\text{Operational Semantics}}$:

$$\begin{aligned}\mathcal{OP}_P(\mathcal{M}) = \{&(c \rightarrow \overline{\ell})^{\forall} \mid (\mathcal{M}(c^{\exists}) \neq \underline{\mathtt{f}}) \text{ and there is a } BCN(P,\mathcal{M}) - \text{derivation for}\\ &\overline{\ell}\square\mathbf{T} \text{ with computed answer } d \text{ such that } \mathcal{M}((c \rightarrow d)^{\forall}) = \underline{\mathtt{t}}\} \cup\\ &\{(c \rightarrow \neg\overline{\ell})^{\forall} \mid \overline{\ell}\square c \text{ is a } BCN(P,\mathcal{M}) - \text{failed goal}\}\end{aligned}$$

2. $\underline{\text{Least Fixpoint Semantics}}$:

$$\begin{aligned}\mathcal{ALG}_P(\mathcal{M}) = \{&(c \rightarrow \overline{\ell})^{\forall} \mid (\mathcal{M}(c^{\exists}) \neq \underline{\mathtt{f}}) \wedge \mathcal{T}_P^{\mathcal{M}} \uparrow \omega \models (c \rightarrow \overline{\ell})^{\forall}\} \cup\\ &\{(c \rightarrow \neg\overline{\ell})^{\forall} \mid (\mathcal{M}(c^{\exists}) \neq \underline{\mathtt{f}}) \wedge \mathcal{T}_P^{\mathcal{M}} \uparrow \omega \models (c \rightarrow \neg\overline{\ell})^{\forall}\}\end{aligned}$$

3. $\underline{\text{Logical Semantics}}$:

$$\begin{aligned}\mathcal{LOG}_P(\mathcal{M}) = \{&(c \rightarrow \overline{\ell})^{\forall} \mid (\mathcal{M}(c^{\exists}) \neq \underline{\mathtt{f}}) \wedge P^* \cup Th(\mathcal{M}) \models (c \rightarrow \overline{\ell})^{\forall}\} \cup\\ &\{(c \rightarrow \neg\overline{\ell})^{\forall} \mid (\mathcal{M}(c^{\exists}) \neq \underline{\mathtt{f}}) \wedge P^* \cup Th(\mathcal{M}) \models (c \rightarrow \neg\overline{\ell})^{\forall}\}\end{aligned}$$

b) To each pair of objects $\mathcal{M}$ and $\mathcal{M}'$ such that $\mathcal{M} \preceq_c \mathcal{M}'$ in the corresponding source category, $\mathcal{F} \in \{\mathcal{ALG}_P, \mathcal{LOG}_P\}$ assigns the morphism $\mathcal{F}(\mathcal{M}) \preceq \mathcal{F}(\mathcal{M}')$ in $\underline{ProgInt}$. However, $\mathcal{OP}_P$ is contravariant, i.e., $\mathcal{M} \preceq_c \mathcal{M}'$ in $\underline{PreTh}$ implies $\mathcal{F}(\mathcal{M}') \preceq \mathcal{F}(\mathcal{M})$ in $\underline{ProgInt}$.

It is easy to see that $\mathcal{ALG}_P$ and $\mathcal{LOG}_P$ are functors as a straightforward consequence of the fact that morphisms are partial orders and the monotonicity of the operator $\mathcal{T}_P^{\mathcal{M}}$ and the logic, respectively. The contravariance of $\mathcal{OP}_P$ is a consequence of the fact that the *BCN*-derivation process only makes unsatisfiability queries to the solver to prune derivations. This means that when $\mathcal{M}_{\underline{\mathtt{f}}}$ is larger the derivation process prunes more derivation sequences.

Now, given a $(\Sigma_{\mathcal{X}}, \mathcal{L}_{\mathcal{X}})$-program P, we can define the semantics of P as

$$[\![P]\!] = (\mathcal{OP}_P, \mathcal{ALG}_P, \mathcal{LOG}_P)$$

6 Equivalence of semantics

In this subsection, we will first prove that the semantic constructions represented by the functors $\mathcal{OP}_P$, $\mathcal{ALG}_P$ and $\mathcal{LOG}_P$ are equivalent in the sense that for each object $\mathcal{M}$ in $\underline{\text{CompTh}}$, $\mathcal{OP}_P(\mathcal{M})$, $\mathcal{ALG}_P(\mathcal{M})$, and $\mathcal{LOG}_P(\mathcal{M})$ are the same object in $\underline{ProgInt}$.

Then, we will show the completeness of the operational semantics with respect to the algebraic and logical semantics just as a consequence of the fact that functors preserve the relations from its domains into its codomains.

Theorem 21 *Let P be a Σ-program. For each object $\mathcal{M}$ in $\underline{CompTh}$,*

$$\mathcal{OP}_P(\mathcal{M}) = \mathcal{ALG}_P(\mathcal{M}) = \mathcal{LOG}_P(\mathcal{M})$$

in $\underline{ProgInt}$.

Finally, we present the usual completeness results of the operational semantics that can be obtained when the domains, theories and solvers are not equivalent. As we pointed out before, these results can be obtained just as a consequence of working with functors. In particular, since $\mathcal{M}_{solv_X} \preceq_c \mathcal{M}_{Dom_X}$ the contravariance of $\mathcal{OP}_P$ implies that $\mathcal{ALG}_P(\mathcal{M}_{Dom_X}) \preceq_c \mathcal{OP}_P(\mathcal{M}_{solv_X})$, and similarly for the logical semantics. That is:

Corollary 22 (Completeness of the operational semantics) *For any program P, $\mathcal{OP}_P$ is complete with respect to $\mathcal{ALG}_P$ and with respect to $\mathcal{LOG}_P$. That is, for each constraint domain $(\Sigma_X, \mathcal{L}_X, Ax_X, Dom_X, solv_X)$:*

– $\mathcal{ALG}_P(\mathcal{M}_{Dom_X}) \preceq_c \mathcal{OP}_P(\mathcal{M}_{solv_X})$
– $\mathcal{LOG}_P(\mathcal{M}_{Ax_X}) \preceq_c \mathcal{OP}_P(\mathcal{M}_{solv_X})$

Acknowledgements: The authors would like to thank an anonymous referee for his work in improving this paper. This work has been partially supported by the Spanish CICYT project GRAMMARS (ref. TIN2004-07925-C03).

References

1. Javier Álvez, Paqui Lucio, and Fernando Orejas. Constructive negation by bottom-up computation of literal answers. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 1468–1475, 2004.
2. Jan A. Bergstra, Manfred Broy, J. V. Tucker, and Martin Wirsing. On the power of algebraic specifications. In Jozef Gruska and Michal Chytil, editors, *Mathematical Foundations of Computer Science 1981, Strbske Pleso, Czechoslovakia, August 31 - September 4, 1981, Proceedings MFCS*, volume 118 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 1981.
3. W. A. Carnielli. Sistematization of finite many-valued logics through the method of tableaux. *Journal of Symbolic Logic*, 52(2):473–493, 1987.
4. K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press. New York, 1978.
5. W. Drabent. What is a failure? An approach to constructive negation. *Acta Informática*, 32:27–59, 1995.

6. F. Fages. Constructive Negation by pruning. *Journal of Logic Programming*, 32:85–118, 1997.
7. M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 4:295–312, 1985.
8. J. Goguen and J. Meseguer. *Initiality, Induction and Computability*. in *Algebraic Methods in Semantics*, (M. Nivat and J. Reynolds, eds.). Cambridge Univ. Press:459–540, 1985.
9. Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *POPL*, pages 111–119, 1987.
10. J. Jaffar and M. Maher. Constraint logic programming: a survey. *The Journal of Logic Programming*, (19/20):503–581, 1994.
11. J. Jaffar, M. Maher, K. Marriot, and P. Stukey. The semantics of constraint logic programs. *The Journal of Logic Programming*, (37):1–46, 1998.
12. S. C. Kleene. *Introduction to Metamathematics*. Van Nostrand, 1952.
13. K. Kunen. Signed data dependencies in logic programs. *Journal of Logic Programming*, 7:231–245, 1989.
14. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.
15. P. Lucio, F. Orejas, and E. Pino. An algebraic framework for the definition of compositional semantics of normal logic programs. *Journal of Logic Programming*, 40:89–123, 1999.
16. E. Pasarella, E. Pino, and F. Orejas. Constructive Negation without subsidiary trees. In *9th International Workshop on Functional and Logic Programming (WFLP'00)*, Benicssim, Spain. 2000.
17. T. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Progamming*, pages 193–216. Morgan Kaufmann, 1988.
18. J.C. Shepherdson. Language and equality theory in logic programming. Technical Report PM-91-02, University of Bristol, 1991.
19. P. J. Stuckey. Negation and constraint logic programmming. *Information and Computation*, 118:12–23, 1995.

7 Appendix

Proof of Lemma 7 Let L be $\overline{\ell}_1, \ell_1, \overline{\ell}_2, \ell_2, \overline{\ell}_3 \square c$. Then, $L_1 = \overline{\ell}_1, \overline{\ell}_2, \ell_2, \overline{\ell}_3 \square c \wedge T_k^P(\ell_1)$, $k > 0$, and $solv_{\mathcal{X}}((c \wedge T_k^P(\ell_1))^{\exists}) \neq \mathbf{F}$, and, $L' = \overline{\ell}_1, \overline{\ell}_2, \overline{\ell}_3 \square c \wedge T_k^P(\ell_1) \wedge T_{k'}^P(\ell_2)$, $k > 0$, $k' > 0$ and $solv_{\mathcal{X}}((c \wedge T_k^P(\ell_1) \wedge T_{k'}^P(\ell_2))^{\exists}) \neq \mathbf{F}$.

Now, to construct the derivation $L \rightsquigarrow_{(P, solv_{\mathcal{X}})} L_2 \rightsquigarrow_{(P, solv_{\mathcal{X}})} L''$ in which ℓ_2 is select first in L_1 we choose $L_2 = \overline{\ell}_1, \ell_1, \overline{\ell}_2, \overline{\ell}_3 \square c \wedge T_{k'}^P(\ell_2)$ and $L'' = \overline{\ell}_1, \overline{\ell}_2, \overline{\ell}_3 \square c \wedge T_{k'}^P(\ell_2) \wedge T_k^P(\ell_1)$. Since $solv_{\mathcal{X}}((c \wedge T_k^P(\ell_1) \wedge T_{k'}^P(\ell_2))^{\exists}) \neq \mathbf{F}$, by the well-behavedness property of $solv_{\mathcal{X}}$, we know that $solv_{\mathcal{X}}((c \wedge T_{k'}^P(\ell_2))^{\exists}) \neq \mathbf{F}$ and $solv_{\mathcal{X}}((c \wedge T_{k'}^P(\ell_2) \wedge T_k^P(\ell_1))^{\exists}) \neq \mathbf{F}$. Hence, $L \rightsquigarrow_{(P, solv_{\mathcal{X}})} L_2 \rightsquigarrow_{(P, solv_{\mathcal{X}})} L''$ is a valid $BCN(P, solv_{\mathcal{X}})$-derivation. ■

Proof of Theorem 8 The proof follows by induction on the length, n, of the $BCN(P, solv_{\mathcal{X}})$-derivation. The base step, $n = 0$, trivially holds. Assume that the statement holds for $n' < n$. Now, to prove the inductive step, consider the $BCN(P, solv_{\mathcal{X}})$-derivation

$$L \rightsquigarrow_{(P, solv_{\mathcal{X}})} L_1 \rightsquigarrow_{(P, solv_{\mathcal{X}})} \ldots \rightsquigarrow_{(P, solv_{\mathcal{X}})} L_{n-1} \rightsquigarrow_{(P, solv_{\mathcal{X}})} \square c$$

Since this is a successful derivation, each literal in L is selected at some point of the derivation. Let us consider the literal ℓ in L and suppose that it is selected in L_i. By applying Lemma 7 i times we can reorder the above derivation to obtain the following one $L \leadsto_{(P,solv_{\mathcal{X}})} L'_1 \leadsto_{(P,solv_{\mathcal{X}})} \dots \leadsto_{(P,solv_{\mathcal{X}})} L'_{n-1} \leadsto_{(P,solv_{\mathcal{X}})} \Box c'$, such that ℓ is selected in L and c' is a reordering of c. Assume that the selection rule R selects literal ℓ when considering the singleton derivation L. From the induction hypothesis, there is another $BCN(P, solv_{\mathcal{X}})$-derivation $L'_1 \overset{n-1}{\leadsto}_{(P,solv_{\mathcal{X}})} \Box c''$, using the selection rule R', where R' selects literals as they are selected by the rule R when considering the derivation $L \leadsto_{(P,solv_{\mathcal{X}})} L'_1 \overset{n-1}{\leadsto}_{(P,solv_{\mathcal{X}})} \Box c''$. So, c'' is a reordering of c' and hence of c. Thus, $L \leadsto_{(P,solv_{\mathcal{X}})} L'_1 \leadsto_{(P,solv_{\mathcal{X}})} \dots \leadsto_{(P,solv_{\mathcal{X}})} L'_{n-1} \leadsto_{(P,solv_{\mathcal{X}})} \Box c''$ is the $BCN(P, solv_{\mathcal{X}})$-derivation we were looking for. ■

Proof of proposition 9 Actually we are going to prove that for each $k \in \mathbb{N}$

$$P^* \cup Th(Ax_{\mathcal{X}}) \models (T_k^P(\ell) \rightarrow \ell)^{\forall}$$

since it is easy to see that the general case is a straightforward consequence of Definition 4.

The proof follows by induction on k and it merely relies on standard syntactical properties of first-order logic. For the base case, $k = 0$, the proposition trivially holds. Assume that the statement holds for $k' < k$. Now we have to prove it for k. There are two situations: either $T_k^P(\ell)$ is satisfiable or is not. The proof for the latter case is analogous to the base step. Assume $T_k^P(\ell)$ is satisfiable. There are two cases:

1. $\ell = p(\bar{x})$. Then, applying twice the definition of T_k^P, the first time for atoms and the second time for the conjunction of literals, we obtain the following:

$$T_k^P(p(\bar{x})) = \bigvee_{i=1}^{m} \exists \bar{y}^i (c^i \wedge T_{k-1}^P(\bar{\ell}^i)) = \bigvee_{i=1}^{m} \exists \bar{y}^i (c^i \wedge \bigwedge_{j=1}^{n_i} T_{k-1}^P(\ell_j^i))$$

   Now, from the induction hypothesis we have that, for all $i \in \{1, \dots, m\}$ and for all $j \in \{1, \dots, n_i\}$:

$$P^* \cup Th(Ax_{\mathcal{X}}) \models (T_{k-1}^P(\ell_j^i) \rightarrow \ell_j^i)^{\forall}$$

   Then, it follows logically that,

$$P^* \cup Th(Ax_{\mathcal{X}}) \models (\bigvee_{i=1}^{m} \exists \bar{y}^i (c^i \wedge \bigwedge_{j=1}^{n_i} T_{k-1}^P(\ell_j^i)) \rightarrow \bigvee_{i=1}^{m} \exists \bar{y}^i (c^i \wedge \bigwedge_{j=1}^{n_i} \ell_j^i))^{\forall}$$

   And, again, applying the definition of T_k^P we obtain the following:

$$P^* \cup Th(Ax_{\mathcal{X}}) \models (T_k^P(p) \rightarrow \bigvee_{i=1}^{m} \exists \bar{y}^i (c^i \wedge \bigwedge_{j=1}^{n_i} \ell_j^i))^{\forall} \quad (1)$$

   In addition, by the completion of predicate $p(\bar{x})$, we have that,

$$P^* \cup Th(Ax_{\mathcal{X}}) \models (\bigvee_{i=1}^{m} \exists \bar{y}^i (c^i \wedge \bigwedge_{j=1}^{n_i} \ell_j^i) \rightarrow p(\bar{x}))^{\forall} \quad (2)$$

Hence, by (1) and (2), we can conclude that

$$P^* \cup Th(Ax_X) \models (T_k^P(p(\bar{x})) \rightarrow p(\bar{x}))^\forall, \quad k > 0$$

The proof for the second case is quite similar to the previous one.

2. $\ell = \neg p(\bar{x})$. Then, $T_k^P(\neg p(\bar{x})) = F_k^P(p(\bar{x}))$, and applying the definition of F_k^P we obtain the following:

$$F_k^P(p(\bar{x})) = \bigwedge_{i=1}^{m} \forall \bar{y}^i(\neg c^i \vee F_{k-1}^P(\bar{\ell}^i)) = \bigwedge_{i=1}^{m} \forall \bar{y}^i(\neg c^i \vee \bigvee_{j=1}^{n_i} F_{k-1}^P(\ell_j^i))$$

Now, using the induction hypothesis we have that, for all $i \in \{1, \ldots, m\}$ and for all $j \in \{1, \ldots, n_i\}$:

$$P^* \cup Th(Ax_X) \models (F_{k-1}^P(\ell_j^i) \rightarrow \neg \ell_j^i)^\forall$$

Therefore, it follows logically that,

$$P^* \cup Th(Ax_X) \models (\bigwedge_{i=1}^{m} \forall \bar{y}^i(\neg c^i \vee \bigvee_{j=1}^{n_i} F_{k-1}^P(\ell_j^i)) \rightarrow \bigwedge_{i=1}^{m} \forall \bar{y}^i(\neg c^i \vee \bigvee_{j=1}^{n_i} \neg \ell_j^i))^\forall$$

Again, applying the definition of F_k^P, we have that,

$$P^* \cup Th(Ax_X) \models (F_k^P(p(\bar{x})) \rightarrow \bigwedge_{i=1}^{m} \forall \bar{y}^i(\neg c^i \vee \bigvee_{j=1}^{n_i} \neg \ell_j^i))^\forall \tag{3}$$

Finally, as in the previous case, we use the completion of the predicate $p(\bar{x})$ to obtain:

$$P^* \cup Th(Ax_X) \models (\bigwedge_{i=1}^{m} \forall \bar{y}^i(\neg c^i \vee \bigvee_{j=1}^{n_i} \neg \ell_j^i) \rightarrow \neg p(\bar{x}))^\forall \tag{4}$$

Hence, by (3) and (4), we can conclude that

$$P^* \cup Th(Ax_X) \models (F_k^P(p(\bar{x})) \rightarrow \neg p(\bar{x}))^\forall, \quad k > 0 \quad \blacksquare$$

Proof of proposition 10 To prove that $(Dom_\Sigma/_\equiv, \preceq)$ is a cpo, we show that each increasing chain $\{[\mathcal{A}_i]\}_{i \in I} \subseteq Dom_\Sigma/_\equiv$

$$[\mathcal{A}_1] \preceq \ldots \preceq [\mathcal{A}_n] \preceq \ldots$$

has a least upper bound $\bigsqcup [\mathcal{A}_n]$. Let $[\mathcal{A}]$ be such that $\mathcal{A}((c \rightarrow \ell)^\forall) = \underline{t}$ iff, for some n, $\mathcal{A}_n((c \rightarrow \ell)^\forall) = \underline{t}$. Then, it is almost trivial to see that

– for each n, $[\mathcal{A}_n] \preceq [\mathcal{A}]$
– for any other $[\mathcal{B}]$ such that $[\mathcal{A}_n] \preceq [\mathcal{B}]$ for each n, $[\mathcal{A}] \preceq [\mathcal{B}]$.

Finally, it is trivial to see that $[\bot_\Sigma] \preceq [\mathcal{A}]$ for all $[\mathcal{A}] \in Dom_\Sigma/_\equiv$. $\blacksquare$

Proof of Theorem 15 First of all, $\mathcal{T}_P^{Dom_X}$ is monotonic, that is, for all $[\mathcal{A}]$ and $[\mathcal{B}]$ in $Dom_\Sigma/_\equiv$

$$[\mathcal{A}] \preceq [\mathcal{B}] \Rightarrow \mathcal{T}_P^{Dom_X}([\mathcal{A}]) \preceq \mathcal{T}_P^{Dom_X}([\mathcal{B}])$$

as a consequence of the fact that $\Phi_P^{\mathcal{D}_X}$ is monotonic:

$$[\mathcal{A}] \preceq [\mathcal{B}] \Rightarrow \mathcal{A} \preceq \mathcal{B} \Rightarrow \Phi_P^{\mathcal{D}_X}(\mathcal{A}) \preceq \Phi_P^{\mathcal{D}_X}(\mathcal{B}) \Rightarrow [\Phi_P^{\mathcal{D}_X}(\mathcal{A})] \preceq [\Phi_P^{\mathcal{D}_X}(\mathcal{B})]$$

Then, being $\mathcal{T}_P^{Dom_X}$ monotonic, to prove that it is continuous it is enough to prove that is is finitary. That is: For each increasing chain $\{[\mathcal{A}_n]\}_{n\in I}$, $[\mathcal{A}_1] \preceq \ldots \preceq [\mathcal{A}_n] \preceq \ldots$

$$\mathcal{T}_P^{Dom_X}(\bigsqcup[\mathcal{A}_n]) \preceq \bigsqcup \mathcal{T}_P^{Dom_X}([\mathcal{A}_n])$$

Let $[\mathcal{A}] = \bigsqcup[\mathcal{A}_n]$ and $[\mathcal{B}] = \mathcal{T}_P^{Dom_X}(\bigsqcup[\mathcal{A}_n]) = [\Phi_P^{\mathcal{D}_X}(\mathcal{A})]$. Let us assume $\mathcal{B}((c \rightarrow \ell)^\forall) = \underline{\mathtt{t}}$. We have two cases:

(a) If $\ell = p(\bar{x})$ then, by the definition of the operator $\Phi_P^{\mathcal{D}_X}$, we know there are (renamed versions of) clauses $\{p(\bar{x}) :- \ell_1^i, \ldots, \ell_{n_i}^i \Box d_i \mid 1 \le i \le m\}$ in P and $\mathcal{D}_X$-satisfiable constraints $\{c_j^i \mid 1 \le i \le m \wedge 1 \le j \le n_i\}$ such that
- $\mathcal{A}((c_j^i \rightarrow \ell_j^i)^\forall) = \underline{\mathtt{t}}$
- $\mathcal{A}((c \rightarrow \bigvee_{1\le i\le m} \exists \bar{y}_i(\bigwedge_{1\le j\le n_i} c_j^i \wedge d_i))^\forall) = \underline{\mathtt{t}}$

In such a situation, by definition of $\bigsqcup$, we know that for each $1 \le i \le m$ and $1 \le j \le n_i$ there is a $[\mathcal{A}_k] \in \{[\mathcal{A}_n] \mid n \in I\}$ such that $\mathcal{A}_k((c_j^i \rightarrow \ell_j^i)^\forall) = \underline{\mathtt{t}}$. Then, since $(Dom_\Sigma/_\equiv, \preceq)$ is a cpo, we know that each finite sub-chain has a least upper bound in $\{[\mathcal{A}_n]\}_{n\in I}$. Let it be $[\mathcal{A}_s]$. In addition, since all models in $\mathcal{D}_\Sigma$ are elementarily equivalent we can state that
- $\mathcal{A}_s((c_j^i \rightarrow \ell_j^i)^\forall) = \underline{\mathtt{t}}$
- $\mathcal{A}_s((c \rightarrow \bigvee_{1\le i\le m} \exists \bar{y}_i(\bigwedge_{1\le j\le n_i} c_j^i \wedge d_i))^\forall) = \underline{\mathtt{t}}$

Therefore, $\Phi_P^{\mathcal{D}_X}(\mathcal{A}_s)((c \rightarrow p(\bar{x}))^\forall) = \underline{\mathtt{t}}$ so for all models $\mathcal{C} \in [\Phi_P^{\mathcal{D}_X}(\mathcal{A}_s)]$ we have that $\mathcal{C}((c \rightarrow p(\bar{x}))^\forall) = \underline{\mathtt{t}}$. Thus, by definition of $\bigsqcup$, this implies that for all $\mathcal{C}' \in \bigsqcup[\Phi_P^{\mathcal{D}_X}(\mathcal{A}_n)] = \bigsqcup \mathcal{T}_P^{Dom_X}([\mathcal{A}_n])$ we have that $\mathcal{C}'(c \rightarrow p(\bar{x})))^\forall = \underline{\mathtt{t}}$.

(b) The proof for $\ell = \neg p(\bar{x})$ proceeds in the same way. That is, by the definition of the operator $\Phi_P^{\mathcal{D}_X}$, we know that for each (renamed version) clause in $\{p(\bar{x}) :- \ell_1^i, \ldots, \ell_{n_i}^i \Box d_i \mid 1 \le i \le m\} = Def_P(p(\bar{x})))$ there is a $J_i \subseteq \{1, \ldots n_i\}$ and $\mathcal{D}_X$-satisfiable constraints $\{c_j^i \mid 1 \le i \le m \wedge j \in J_i\}$ such that
- $\mathcal{A}((c_j^i \rightarrow \neg\ell_j)^\forall) = \underline{\mathtt{t}}$
- $\mathcal{A}((c \rightarrow \bigwedge_{1\le i\le m} \forall \bar{y}_i(\bigvee_{j\in J_i} c_j^i \vee \neg d_i))^\forall) = \underline{\mathtt{t}}$

Again, by definition of $\bigsqcup$, we know that for each $j \in J$ there is a $[\mathcal{A}_j] \in \{[\mathcal{A}_n] \mid n \in I\}$ such that $\mathcal{A}_j((c_j \rightarrow \neg\ell_j)^\forall) = \underline{\mathtt{t}}$. Then, as a consequence of $(Dom_\Sigma/_\equiv, \preceq)$ being a cpo, and all models in $\mathcal{D}_\Sigma$ being elementarily equivalent, there is a class $[\mathcal{A}_s]$ in the chain such that
- $\mathcal{A}_s((c_j^i \rightarrow \neg\ell_j)^\forall) = \underline{\mathtt{t}}$
- $\mathcal{A}_s((c \rightarrow \bigwedge_{1\le i\le m} \forall \bar{y}_i(\bigvee_{j\in J_i} c_j^i \vee \neg d_i))^\forall) = \underline{\mathtt{t}}$

Therefore $\Phi_P^{\mathcal{D}_X}(\mathcal{A}_s)((c \rightarrow \neg p(\bar{x}))^\forall) = \underline{\mathtt{t}}$ so, for all models $\mathcal{C} \in [\Phi_P^{\mathcal{D}_X}(\mathcal{A}_s)]$ we have that $\mathcal{C}((c \rightarrow \neg p(\bar{x}))^\forall) = \underline{\mathtt{t}}$. And, finally, by definition of $\bigsqcup$, this implies that for all $\mathcal{C}' \in \bigsqcup[\Phi_P^{\mathcal{D}_X}(\mathcal{A}_n)] = \bigsqcup \mathcal{T}_P^{Dom_X}([\mathcal{A}_n])$ we have that $\mathcal{C}'(c \rightarrow \neg p(\bar{x})))^\forall = \underline{\mathtt{t}}$. ■

Proof of Theorem 16 We prove that 1 and 2 hold for a goal $\ell \square c$. Then, the general case for $\overline{\ell} \square c$ easily follows from the logical definition of the truth-value of $(c \rightarrow \overline{\ell})^{\forall}$ and $(c \rightarrow \neg\overline{\ell})^{\forall}$.
The Stuckey's result states that $P^* \cup Th(Dom_{\mathcal{X}}) \models_3 (c \rightarrow \ell)^{\forall}$ if, and only if,

$$\Phi_P^{\mathcal{D}_X} \uparrow k((c \rightarrow \ell)^{\forall}) = \underline{\mathrm{t}}$$

for some finite k. So, by definition of $\mathcal{T}_P^{Dom_X}$, this is equivalent to

$$\forall \mathcal{A} \in \mathcal{T}_P^{Dom_X} \uparrow k : \mathcal{A}((c \rightarrow \ell)^{\forall}) = \underline{\mathrm{t}}$$

for some finite k. And, by definition of $\bigsqcup$, to

$$\forall \mathcal{A} \in \bigsqcup \mathcal{T}_P^{Dom_X} \uparrow k : \mathcal{A}((c \rightarrow \ell)^{\forall}) = \underline{\mathrm{t}}$$

■

Proof of Theorem 21 First of all, we have that $\mathcal{LOG}_P(\mathcal{M}) = \mathcal{ALG}_P(\mathcal{M})$ as a direct consequence of Theorem 16 (Extended Theorem of Stuckey).
In what follows, we will prove that

– $\mathcal{ALG}_P(\mathcal{M}) \preceq \mathcal{OP}_P(\mathcal{M})$ and
– $\mathcal{OP}_P(\mathcal{M}) \preceq \mathcal{LOG}_P(\mathcal{M})$

(a) To prove that $\mathcal{ALG}_P(\mathcal{M}) \preceq \mathcal{OP}_P(\mathcal{M})$ we use induction on the number of iterations of $\mathcal{T}_P^{\mathcal{M}}$. We just consider goals such that $\overline{\ell} = p(\overline{x})$ and $\overline{\ell} = \neg p(\overline{x})$, since the general case follows from the properties of operators T_k^P and F_k^P and the fact that BCN is independent of the selection rule.
The base case $n = 0$ is trivial, since $\mathcal{T}_P^{\mathcal{M}} \uparrow 0 = [\bot_\Sigma]$ and $[\bot_\Sigma]((c \rightarrow \ell)^{\forall}) \neq \underline{\mathrm{t}}$ for all $\Sigma_{\mathcal{X}}$-constraint $c(\overline{x})$ and all Σ-literal $\ell(\overline{x})$.
Assume that for all $k \leq n$, $\mathcal{T}_P^{\mathcal{M}} \uparrow k((c \rightarrow \ell)^{\forall}) = \underline{\mathrm{t}}$ implies $\mathcal{OP}_P(\mathcal{M})((c \rightarrow \ell)^{\forall}) = \underline{\mathrm{t}}$.
   i) If $\overline{\ell} = p(\overline{x})$ then, by the definition of $\mathcal{T}_P^{\mathcal{M}}$, we know that there are (renamed versions of) clauses $\{p(\overline{x}) :- \ell_1^i, \ldots, \ell_{n_i}^i \square d_i \mid 1 \leq i \leq m\}$ in P and $\mathcal{M}$-satisfiable constraints $\{c_j^i \mid 1 \leq i \leq m \,\wedge\, 1 \leq j \leq n_i\}$ such that $\mathcal{T}_P^{\mathcal{M}} \uparrow n((c_j^i \rightarrow \ell_j^i)^{\forall}) = \underline{\mathrm{t}}$ and

$$\mathcal{M}((c \rightarrow \bigvee_{i=1}^{m} \exists \overline{y}_i (\bigwedge_{j=1}^{n_i} c_j^i \wedge d_i))^{\forall}) = \underline{\mathrm{t}}$$

   Then, by the induction hypothesis we have that $\mathcal{OP}_P(\mathcal{M})((c_j^i \rightarrow \ell_j^i)^{\forall}) = \underline{\mathrm{t}}$ for all $1 \leq i \leq m$ and $1 \leq j \leq n_i$. Thus, there exist successful $BCN(P,\mathcal{M})$-derivations for each $1 \leq i \leq m$ and $1 \leq j \leq n_i$:

$$\ell_j^i \square d_i \rightsquigarrow_{(P,\mathcal{M})} \square T_{k_j^i}^P(\ell_j^i) \wedge d_i$$

   such that $\mathcal{M}((T_{k_j^i}^P(\ell_j^i)) \wedge d_i)^{\exists}) \neq \underline{\mathrm{f}}$ and $\mathcal{M}(c_j^i \rightarrow T_{k_j^i}^P(\ell_j^i))^{\forall}) = \underline{\mathrm{t}}$.

Let $k > 0$ be the largest number in $\{k^i_j \mid 1 \leq i \leq m \wedge 1 \leq j \leq n_i\}$. Then, as a consequence of the monotonicity of the operator T^P_-, we know $\mathcal{M}((\bigwedge_{j=1}^{n_i} T^P_k(\ell^i_j)) \wedge d_i)^\exists) \neq \underline{\mathtt{f}}$. And since

$$T^P_k(\bigwedge_{j=1}^{n_i} \ell^i_j) = \bigwedge_{j=1}^{n_i} T^P_k(\ell^i_j)$$

and

$$\mathcal{M}((\bigwedge_{j=1}^{n_i} c^i_j \rightarrow T^P_k(\bigwedge_{j=1}^{n_i} \ell^i_j))^\forall) = \underline{\mathtt{t}}$$

we have that

$$\mathcal{M}((c \rightarrow \bigvee_{i=1}^{m} \exists \overline{y}_i(T^P_k(\bigwedge_{j=1}^{n_i} \ell^i_j) \wedge d_i))^\forall) = \underline{\mathtt{t}}$$

That is, $\mathcal{M}(T^P_{k+1}(p(\overline{x}))^\exists) \neq \underline{\mathtt{f}}$ and

$$\mathcal{M}((c \rightarrow T^P_{k+1}(p(\overline{x})))^\forall) = \underline{\mathtt{t}}$$

Therefore, we can guarantee the existence of a successful $BCN(P, \mathcal{M})$-derivation:

$$p(\overline{x}) \square \underline{\mathtt{t}} \rightsquigarrow_{(P,\mathcal{M})} \square T^P_{k+1}(p(\overline{x}))$$

such that $\mathcal{OP}_P(\mathcal{M})((c \rightarrow p(\overline{x}))^\forall) = \underline{\mathtt{t}}$.

ii) The proof for $\overline{\ell} = \neg p(\overline{x})$ proceeds in the same way. That is, according to the definition of the operator $\mathcal{T}^\mathcal{M}_P$, we know that for each (possibly renamed) clause in $\{p(\overline{x}) :- \ell^i_1, \ldots, \ell^i_{n_i} \square d_i \mid 1 \leq i \leq m\} = Def_P(p(\overline{x})))$ there is a $J_i \subseteq \{1, \ldots n_i\}$ and $\mathcal{M}$-satisfiable constraints $\{c^i_j \mid 1 \leq i \leq m \wedge j \in J_i\}$ such that:

* $\mathcal{T}^\mathcal{M}_P \uparrow n((c^i_j \rightarrow \neg \ell_j)^\forall) = \underline{\mathtt{t}}$
* $\mathcal{M}((c \rightarrow \bigwedge_{1 \leq i \leq m} \forall \overline{y}_i(\bigvee_{j \in J_i} c^i_j \vee \neg d_i))^\forall) = \underline{\mathtt{t}}$

Again, by the induction hypothesis we have that for all $1 \leq i \leq m$ and $j \in J_i$, $\mathcal{OP}_P(\mathcal{M})((c^i_j \rightarrow \neg \ell^i_j)^\forall) = \underline{\mathtt{t}}$ so, for some $r^i_j > 0$

$$\mathcal{M}((c^i_j \rightarrow F^P_{r^i_j}(\ell^i_j))^\forall) = \underline{\mathtt{t}}$$

Let $r > 0$ be the largest number in $\{r^i_j \mid 1 \leq i \leq m \wedge j \in J_i\}$. Then, as a consequence of the monotonicity of the operator F^P_-, we know $\mathcal{M}((\bigvee_{j \in J_i} F^P_r(\ell^i_j))^\exists) \neq \underline{\mathtt{f}}$. And, since $F^P_r(\bigvee_{j \in J_i} \ell^i_j) = \bigvee_{j \in J_i} F^P_r(\ell^i_j)$ and $\mathcal{M}((\bigvee_{j \in J_i} c^i_j \rightarrow F^P_r(\bigvee_{j \in J_i} \ell^i_j))^\forall) = \underline{\mathtt{t}}$ we have that

$$\mathcal{M}((c \rightarrow F^P_{r+1}(p(\overline{x})))^\forall) = \underline{\mathtt{t}}$$

Therefore, we can guarantee that $p(\overline{x}) \square c$ is a $BCN(P, \mathcal{M})$-failure, so $\mathcal{OP}_P(\mathcal{M})((c \rightarrow \neg p(\overline{x}))^\forall) = \underline{\mathtt{t}}$.

(b) Finally, we prove that $\mathcal{OP}_P(\mathcal{M}) \preceq \mathcal{LOG}_P(\mathcal{M})$. Again we have two cases:

(i) Suppose that $\mathcal{OP}_P(\mathcal{M})((c \rightarrow \neg \overline{\ell})^\forall) = \underline{\mathtt{t}}$ so, $\overline{\ell} \square c$ is a $BCN(P, \mathcal{M})$-failed goal. Hence, $\mathcal{M}((c \rightarrow F^P_k(\overline{\ell}))^\forall) = \underline{\mathtt{t}}$, for some $k > 0$. Therefore, by Proposition 9, we can conclude that $P^* \cup Th(\mathcal{M}) \models (c \rightarrow \neg \overline{\ell})^\forall$.

(ii) Suppose now that $\mathcal{OP}_P(\mathcal{M})((c \rightarrow \overline{\ell})^\forall) = \underline{\mathtt{t}}$. Again we will prove the case $\overline{\ell} = p(\overline{x})$ since the general case will follow from the properties of T_k^P and the fact that BCN is independent of the selection rule. So we assume $p(\overline{x})\Box c$ has a $BCN(P, \mathcal{M})$-derivation

$$p(\overline{x})\Box\underline{\mathtt{t}} \ \leadsto_{(P,\mathcal{M})} \ \Box T_k^P(p(\overline{x}))$$

such that $\mathcal{M}((c \rightarrow T_k^P(p(\overline{x})))^\forall) = \underline{\mathtt{t}}$. Then, again as a consequence of Proposition 9, we can conclude that $P^* \cup Th(\mathcal{M}) \models (c \rightarrow p(\overline{x}))^\forall$.

■