Indexed Categories and Bottom-Up Semantics of Logic Programs

Gianluca Amato[1] and James Lipton[2]

[1] Dipartimento di Matematica e Informatica, Università di Udine
via delle Scienze 206, Udine, Italy
amato@dimi.uniud.it
[2] Mathematics Department, Wesleyan University
265 Church St., Middletown (CT), USA
lipton@wesleyan.edu

Abstract. We propose a categorical framework which formalizes and extends the syntax, operational semantics and declarative model theory of a broad range of logic programming languages. A program is interpreted in an indexed category in such a way that the base category contains all the possible states which can occur during the execution of the program (such as global constraints or type information), while each fiber encodes the logic at each state.
We define appropriate notions of categorical resolution and models, and we prove the related correctness and completeness properties.

1 Introduction

One of the greatest benefits of logic programming is that it is based upon the notion of *executable specifications*. The text of a logic program is endowed with both an operational (algorithmic) interpretation and an independent mathematical meaning which agree each other in several ways.

An operational interpretation is needed if we wish to specify programs which can be executed with some degree of efficiency, while a clear mathematical (declarative) meaning simplifies the work of the programmer, who can –to some extent– focus on "what to do" instead of "how". The problem is that operational expressiveness (i.e. the capability of directing the flow of execution of a program) tends to obscure declarative meaning. Research in logic programming strives to find a good balance between these opposite needs.

Horn logic programming was one of the first attempts in this area and surely the most famous. However it has limitations when it comes to real programming tasks. Its forms of control flow are too primitive: there are simple problems (such as computing the reflexive closure of a relation) which cannot be coded in the obvious way since the programs so obtained do not terminate. The expressive power of its logic is too weak, both for programming in the small and in the large: it lacks any mathematically precise notion of module, program composition, typing. Moreover, if we want to work with some data structure, we need to manually code the behavior and constraints of such a structure in the Horn logic, often obscuring the intended meaning of the code.

For these reasons, various extensions have been proposed to the original framework of Horn clause, often borrowing ideas from other paradigms. Some noteworthy extensions are the use of algebraic operators for modularization [7], the use of more powerful extensions to the logic [21], control operators like the "cut" of PROLOG and abstract data types [20]. The effect has been to expand the boundaries of the field and the notion itself of declarative content of a program. We lack criteria for good language design and models to evaluate the new features, or to formalize the very notion of declarative programming.

Moreover, semantic methods for Horn logic programming are increasingly similar in spirit to those for functional and imperative programming, under the stimulus of techniques such as abstract interpretation [12, 8]. This suggests looking for a sufficiently flexible new logic programming foundation using a framework in which all these paradigms can be well understood. Categorical logic seems an excellent candidate for such an undertaking.

Categorical approaches to logic programming go back to the treatment of unification given by Rydeheard and Burstall in [23]. Building on this work, in [3] the syntax of Horn clause logic is formalized using categorical tools and a topos-theoretic semantics. In [10], following some basic ideas already developed in [11], a categorical analysis of logic program transitions and models is given using indexed monoidal categories. The framework that we propose here builds on some of the ideas in that paper, which have proved seminal, but which fall short of formulating the kind of general blueprint we seek for declarative programming.

The approaches just cited focus on the operational or model theoretic side of logic programming, but lack any bottom-up denotational semantics like the T_P operator of van Emden and Kowalski [25]. For us, the immediate consequence operator seems to be a cornerstone of logic programming, since it appears, in one form or another, across several semantic treatments of logic programs [4]. Most of the studies in the semantics of logic programming are heavily based on the existence of some fixpoint construction: treatments of compositionality of semantics [8], modularity [6], static analysis [12], and debugging [9]. For this reason, it seems to us that further investigation of a categorical framework which includes a form of bottom-up semantics is advisable.

The first step in this direction was taken in [13], which uses categorical syntax over finite τ-categories [15]. It is the starting point for introducing both a notion of categorical SLD derivation and a denotational semantics which resembles the correct answer semantics for Horn logic programs. This semantics can be computed with a fixpoint construction and it can be shown to agree with a more general notion of categorical derivation.

1.1 The New Approach

Our framework starts from the work in [13] and [14]. However, we redefine the fundamental categorical structures with the hope of generalizing the methods in three main directions:

– in the ability to treat other kinds of semantics other than the "correct answers" one. The concept of interpretation must be far broader than that of [13], allowing for semantic domains different from $Set^{\mathbb{C}^o}$;
– in the ability to treat programs with constraints between goals. This means that we must provide a new bottom-up operator which works for a generic syntactic category and not only with $\mathbb{C}[X_1, \ldots, X_n]$;
– in the adaptability to different logic languages. In particular, we would like to treat languages such as CLP or add some control operators to the pure logic programming.

To pursue these goals, we move to a more general categorical interpretation of logic. Instead of taking goals to be monics in the category $\mathbb{C}$, we use an indexed category over $\mathbb{C}$. An object in the fiber $\sigma \in \mathrm{Obj}(\mathbb{C})$ will be the categorical counterpart of a goal of sort σ. It is the standard indexed/fibrational categorical interpretation of full first order logic, as can be found in, e.g. [24, 16].

To simplify the presentation, we begin without any kind of monoidal structure in the fibers. These means we are not able to handle conjunction in goals externally and compositionally: we are restricted to so-called binary clauses. However, adding monoidal structures is quite straightforward, and it has been done in [1].

1.2 Notation

We will assume the reader is familiar with the basic concepts of logic programming [2] and category theory [5]. Here, we only give some brief definitions and notation. Basic categorical background for logic programming can be found in e.g. [14, 18]

Given a category $\mathbb{C}$, we denote by $\mathrm{Obj}(\mathbb{C})$ and $\mathrm{Mor}(\mathbb{C})$ the corresponding classes of objects and morphisms (arrows). With id_A we denote the identity arrow for the object A, while 1 is the terminator, $\times$ the product and $\vee$ denotes coproducts. We use $\vee$ as a functor, applying it to objects and arrows as well. Given $f : A \to B$ and $g : B \to C$, we write $f \,;\, g$ for their composition. With $\mathrm{Hom}_{\mathbb{C}}(A, B)$ we denote the class of morphisms from A to B in $\mathbb{C}$. We omit the index $\mathbb{C}$ when it is clear from the context. Given a functor $F : \mathbb{C} \to \mathbb{C}$, a fixpoint for F is a pair (σ, t) such that $t : F\sigma \to \sigma$ is an isomorphism.

We denote sequences by writing one after the other its elements. We use λ to denote an empty sequence and $\cdot$ as the juxtaposition operator.

A (strict) indexed category over $\mathbb{C}$ is a functor $\mathcal{P} : \mathbb{C}^o \to \mathsf{Cat}$, where $\mathbb{C}^o$ is the opposite category of $\mathbb{C}$ and Cat is the category of all small categories. We refer to objects and arrows in $\mathbb{C}^o$ with the same symbols of their counterparts in $\mathbb{C}$. Given $\sigma \in \mathrm{Obj}(C^o)$, the category $\mathcal{P}\sigma$ is the *fiber* of $\mathcal{P}$ over σ. An indexed functor from $\mathcal{P} : \mathbb{C}^o \to \mathsf{Cat}$ to $\mathcal{Q} : \mathbb{D}^o \to \mathsf{Cat}$ is a pair (F, τ) such that $F : \mathbb{C} \to \mathbb{D}$ is the *change of base* functor and $\tau : \mathcal{P} \to F^o \,;\, \mathcal{Q}$ is a natural transformation. An indexed natural transformation $\eta : (F, \tau) \to (F', \tau') : \mathcal{P} \to \mathcal{Q}$ is given by a pair (ξ, δ) such that $\xi : F \to F'$ is a natural transformation and δ is a $\mathbb{C}$-indexed family of natural transformations

$$\delta_\sigma : \tau_\sigma \to \tau'_\sigma ; \mathcal{Q}(\xi_\sigma) \tag{1}$$

subject to some coherence conditions. A detailed treatment of indexed categories and fibrations can be found in [16].

2 Syntax

In the following, we introduce several kinds of indexed categories called doctrines [19]. We abuse terminology, since a doctrine is generally understood to be an indexed category where reindexing functors have left adjoints, and this property does not always holds for our doctrines. We have chosen this terminology to emphasize the relation between indexed categories used for the syntax and the semantics (which are actually Lawvere doctrines).

Definition 1 (Logic programming doctrine). *An* LP doctrine *(logic programming doctrine) is an indexed category* $\mathcal{P}$ *over a base category* $\mathbb{C}$*. For each* $\sigma \in \mathrm{Obj}(\mathbb{C})$*, objects and arrows in* $\mathcal{P}\sigma$ *are called* goals *and* proofs *(of sort* σ*) respectively. Given a goal* $\mathbf{G}$ *of sort* σ *and* $f : \rho \to \sigma$ *in* $\mathbb{C}$*,* $\mathcal{P}f(\mathbf{G})$ *is an* instance *of* $\mathbf{G}$*. We also denote it by* $f^\sharp\mathbf{G}$ *or* $\mathbf{G}(f)$*.*

We write $\mathbf{G} : \sigma$ and $f : \sigma$ as a short form for $\mathbf{G} \in \mathrm{Obj}(\mathcal{P}\sigma)$ and $f \in \mathrm{Mor}(\mathcal{P}\sigma)$. Given an LP doctrine $\mathcal{P}$, a clause (of sort σ) is a name cl, with an associated pair $(\mathbf{Tl}, \mathbf{Hd})$ of goals of sort σ, that we write as $\mathbf{Hd} \xleftarrow{cl} \mathbf{Tl}$.

Definition 2 (Logic program). *A* logic program *is a pair* $(P, \mathcal{P})$ *where* $\mathcal{P}$ *is an LP doctrine and* P *a set of clauses. We often say that* P *is a program over* $\mathcal{P}$*.*

It is possible to see a logic program as an indexed category P over $\mathrm{Obj}(\mathbb{C})$ such that $P\sigma$ is the category of goals of sort σ with arrows given by clauses of sort σ.

The idea underlying the framework is that the base category represents the world of all possible states to which program execution can lead. At each state, the corresponding fiber represents an underlying theory: a set of deductions which can always be performed, independently of the actual program. A clause is a new deduction, which we want to consider, freely adjoined to the proofs in the fibers.

The advantage of using categories is that we do not need to restrict our interest to syntactical terms and goals. We can choose any base category we desire and build binary logic programs over terms which are interpreted already at the syntactic level in the base.

Example 3 (Binary logic programs). Assume $\mathbb{C}$ is a finite product category. We can think of $\mathbb{C}$ as a not necessarily free model of an appropriate many-sorted signature. We can build a syntactic doctrine for binary logic programs where terms are arrows in this category. We need to fix a signature Π for predicates over $\mathbb{C}$, i.e. a set of predicate symbols with associated sort among $\mathrm{Obj}(\mathbb{C})$. We write $p : \sigma$ when p is a predicate symbol of sort σ. Then, we define an indexed category $\mathcal{P}_\Pi$ over $\mathbb{C}$:

- $\mathcal{P}_\Pi(\sigma)$ is the discrete category whose objects are pairs $\langle p, f\rangle$ such that $p : \rho$ in Π and $f : \sigma \to \rho$ is an arrow in $\mathbb{C}$. To ease notation, we write $p(f)$ instead of $\langle p, f\rangle$;
- $\mathcal{P}_\Pi(f)$, where $f : \rho \to \sigma$, is the functor mapping $p(t) \in \mathrm{Obj}(\mathcal{P}_\Pi(\sigma))$ to $p(f\,;t)$.

The interesting point here is that terms are treated semantically. For example, assume $\mathbb{C}$ is the full subcategory of *Set* whose objects are the sets $\mathbb{N}^i$ for every natural i and $p : \mathbb{N}$ is a predicate symbol. If succ and fact are the functions for the successor and factorial of a natural number, then $(\mathsf{succ}\,;\mathsf{succ}\,;\mathsf{succ})^\sharp(p(3)) = \mathsf{fact}^\sharp(p(3)) = p(6)$.

In the previous example, the fibers of the syntactic doctrine were discrete categories freely generated by a set of predicate symbols. When we define the concept of model for a program, below, it will be clear we are not imposing any constraints on the meaning of predicates. In general, we can use more complex categories for fibers.

Example 4 (Symmetric closure of predicates). In the hypotheses of Example 3, assume we have two predicate symbols p and symp of sort $\rho \times \rho$, and we want to encode in the syntactic doctrine the property that symp contains the symmetric closure of p. Then, we freely adjoin to $\mathcal{P}_\Pi$ the following two arrows in the fiber $\rho \times \rho$:

$$r_1 : p \to \mathit{symp} \enspace ,$$
$$r_2 : p \to \mathit{symp}(\langle \pi_2, \pi_1\rangle) \enspace ,$$

where π_1 and π_2 are the obvious cartesian projections. We call $\mathcal{P}_\Pi^{\mathit{symp}}$ the new LP doctrine we obtain. The intuitive meaning of the adjoined arrows is evident.

3 Models

A key goal of our treatment is to consider extensions to definite logic programs without losing the declarative point of view, namely by defining a corresponding straightforward extension of the notion of model for a program.

Functors (F, τ) of LP doctrines will be called *interpretations*. For every goal or proof x in the fiber σ, we write $\tau_\sigma(x)$ as $[\![x]\!]_\sigma$. We also use $[\![x]\!]$ when the fiber of x is clear from the context.

Definition 5 (Models). *Given a program P over the doctrine $\mathcal{P}$, a* model *of P is a pair $([\![\_]\!], \iota)$ where $[\![\_]\!] : \mathcal{P} \to \mathcal{Q}$ is an interpretation and ι is a function which maps a clause $\mathbf{Hd} \xleftarrow{cl} \mathbf{Tl} \in P$ to an arrow $[\![\mathbf{Hd}]\!] \xleftarrow{\iota(cl)} [\![\mathbf{Tl}]\!]$.*

In the following, a model $M = (\llbracket\_\rrbracket, \iota)$ will be used as an alias for its constituent parts. Hence, $M(cl)$ will denote $\iota(cl)$ and $M_\sigma(\mathbf{G})$ will denote $\llbracket\mathbf{G}\rrbracket_\sigma$. The composition of M with an interpretation N is given as the model $(\llbracket\_\rrbracket\,; N, \iota\,; N)$.

Example 6 (Ground answers for binary logic programs). Consider the LP doctrine $\mathcal{P}_\Pi$ defined in Example 3 and the indexed category $\mathcal{Q}$ over $\mathbb{C}$ such that

- for each $\sigma \in \mathrm{Obj}(\mathbb{C})$, $\mathcal{Q}(\sigma) = \wp(\mathrm{Hom}_\mathbb{C}(1, \sigma))$, which is an ordered set viewed as a category;
- for each $f \in \mathrm{Hom}_\mathbb{C}(\sigma, \rho)$, $\mathcal{Q}(f)(X) = \{r \in \mathrm{Hom}_\mathbb{C}(1, \sigma) \mid r\,; f \in X\}$.

An interpretation $\llbracket\_\rrbracket$ maps an atomic goal of sort σ to a set of arrows from the terminal object of $\mathbb{C}$ to σ. These arrows are indeed the categorical counterpart of ground terms.

Two significant models are given by the interpretations which map every goal $\mathbf{G}$ of sort σ to $\mathrm{Hom}(1, \sigma)$ or to $\emptyset$. Clauses and arrows are obviously mapped to identities. If we see $\mathrm{Hom}(1, \sigma)$ as the true value and $\emptyset$ as false, they correspond to the interpretations where everything is true or everything is false.

When the syntactic doctrine is discrete, as in the previous example, an interpretation from $\mathcal{P}$ to $\mathcal{Q}$ can map every object in $\mathcal{P}$ to every object in $\mathcal{Q}$, provided this mapping is well-behaved w.r.t. reindexing. However, in the general case, other restrictions are imposed.

Example 7. Assume the hypotheses of Example 4. Consider the LP doctrine $\mathcal{Q}$ as defined in Example 6. An interpretation $\llbracket\_\rrbracket$ from $\mathcal{P}_\Pi^{symp}$ to $\mathcal{Q}$ is forced to map the arrows r_1 and r_2 to arrows in $\mathcal{Q}$. This means that $\llbracket symp\rrbracket \supseteq \llbracket p\rrbracket$ and $\llbracket symp\rrbracket \supseteq \llbracket p(\langle\pi_2, \pi_1\rangle)\rrbracket$, i.e. $\llbracket symp\rrbracket \supseteq \llbracket p\rrbracket\,; \langle\pi_2, \pi_1\rangle$. In other words, the interpretation of *symp* must contain both the interpretation of p and its symmetric counterpart.

One of the way to obtain a model of a program P over $\mathcal{P}$ is *freely adjoining* the clauses of P to the fibers of $\mathcal{P}$. We obtain a *free model* of P over $\mathcal{P}$.

Definition 8 (Free model). *A model M of $(P, \mathcal{P})$ is said to be* free *when, for each model M' of $(P, \mathcal{P})$, there exists an unique interpretation N such that $M' = M\,; N$.*

It is easy to prove that, if M and M' are both free models for a program $(P, \mathcal{P})$ in two different logic doctrines $\mathcal{Q}$ and $\mathcal{R}$, then $\mathcal{Q}$ and $\mathcal{R}$ are isomorphic.

4 Operational Semantics

Our logic programs also have a quite straightforward operational interpretation. Given a goal $\mathbf{G}$ of sort σ in a program $(P, \mathcal{P})$, we want to reduce $\mathbf{G}$ using both arrows in the fibers of $\mathcal{P}$ and clauses. This means that, if $x : \mathbf{G} \leftarrow \mathbf{Tl}$ is a clause or a proof in $\mathcal{P}$, we want to perform a reduction step from $\mathbf{G}$ to $\mathbf{Tl}$.

In this way, the only rewritings we can immediately apply to $\mathbf{G}$ are given by rules (proofs or clauses) of sort σ. It is possible to rewrite using a clause cl of another sort ρ only if we find a common "ancestor" α of σ and ρ, i.e. a span

$$\begin{tikzcd} & \alpha \arrow[dl, "t_1"'] \arrow[dr, "t_2"] & \\ \sigma & & \rho \end{tikzcd} \tag{2}$$

in the base category such that $\mathbf{G}$ and the head of cl become equal once they are reindexed to the fiber α.

Definition 9 (Unifier). *Given two goals* $\mathbf{G}_1 : \sigma_1$ *and* $\mathbf{G}_2 : \sigma_2$ *in an LP doctrine* $\mathcal{P}$, *an* unifier *for them is a span* $\langle t_1, t_2 \rangle$ *of arrows of the base category such that* $t_1 : \alpha \to \sigma_1$, $t_2 : \alpha \to \sigma_2$ *and* ${t_1}^\sharp \mathbf{G}_1 = {t_2}^\sharp \mathbf{G}_2$

Unifiers for a pair of goals form a category $\mathit{Unif}_{\mathbf{G}_1,\mathbf{G}_2}$ where arrows from $\langle t_1, t_2 \rangle$ to $\langle r_1, r_2 \rangle$ are given by the common notion of arrow between spans, i.e. a morphism $f : \mathsf{dom}(t_1) \to \mathsf{dom}(r_1)$ such that $f ; r_1 = t_1$ and $f ; r_2 = t_2$.

Definition 10 (MGU). *A* most general unifier *(MGU) for goals* $\mathbf{G}_1 : \sigma_1$ *and* $\mathbf{G}_2 : \sigma_2$ *in an LP doctrine* $\mathcal{P}$ *is a maximal element of* $\mathit{Unif}_{\mathbf{G}_1,\mathbf{G}_2}$.

Example 11 (Standard mgu). Consider the indexed category $\mathcal{P}_\Pi$ as in Example 3. Given goals $p_1(t_1) : \sigma_1$ and $p_2(t_2) : \sigma_2$, an unifier is a pair of arrows $r_1 : \alpha \to \sigma_1$ and $r_2 : \alpha \to \sigma_2$ such that the following diagram commute:

$$\begin{tikzcd} \alpha \arrow[r, "r_1"] \arrow[d, "r_2"'] & \sigma_1 \arrow[d, "t_1"] \\ \sigma_2 \arrow[r, "t_2"'] & \rho \end{tikzcd} \tag{3}$$

This is exactly the definition of unifier for renamed apart terms t_1 and t_2 given in [3], which corresponds to unifiers in the standard syntactic sense. Moreover, the span $\langle r_1, r_2 \rangle$ is maximal when (3) is a pullback diagram, i.e. a most general unifier.

Note that in the standard syntactic categories with freely adjoined predicates there is no unifier between goals $p_1(t_1)$ and $p_2(t_2)$ if $p_1 \neq p_2$. However, this does not hold in our more general setting. Actually, in a completely general doctrine, we have a notion of logic program and execution without any notion of predicate symbol at all.

In the same way, it is possible to reduce a goal $\mathbf{G} : \sigma$ with a proof $f : \mathbf{Hd} \leftarrow \mathbf{Tl}$ in the fiber ρ iff there exists an arrow $r : \rho \to \sigma$ such that $r^\sharp \mathbf{G} = \mathbf{Hd}$. We call a pair $\langle r, f \rangle$ with such properties a *reduction pair*. Reduction pairs form a category such that $t \in \mathrm{Mor}(\mathbb{C})$ is an arrow from $\langle r_1, f_1 \rangle$ to $\langle r_2, f_2 \rangle$ if $r_1 = t ; r_2$ and $t^\sharp f_2 = f_1$. A *most general* reduction pair is a maximal reduction pair. Note that most general unifiers or reduction pairs do not necessarily exist. This is not a big problem since all the theory we develop works the same.

Following these ideas, it is possible to define a categorical form of SLD derivation.

Definition 12 (Categorical derivation). *Given a program* $(P, \mathcal{P})$*, we define a labeled transition system* $(\biguplus_{\sigma \in \mathrm{Obj}(\mathbb{C})} \mathrm{Obj}(\mathcal{P}\sigma), \leadsto)$ *with goals as objects, according to the following rules:*

backchain-clause $\mathbf{G} \xrightarrow{\langle r, t, cl \rangle} t^\sharp \mathbf{Tl}$ *if* cl *is a clause* $\mathbf{Hd} \xleftarrow{cl} \mathbf{Tl}$ *and* $\langle r, t \rangle$ *is an unifier for* $\mathbf{G}$ *and* $\mathbf{Hd}$ *(i.e.* $r^\sharp \mathbf{G} = t^\sharp \mathbf{Hd}$*);*

backchain-arrow $\mathbf{G} \xrightarrow{\langle r, f \rangle} \mathbf{Tl}$ *if* $\mathbf{G}$ *is a goal in the fiber* σ*,* $f : \mathbf{Hd} \leftarrow \mathbf{Tl}$ *is a proof in the fiber* ρ *and* $\langle r, f \rangle$ *is a reduction pair for* $\mathbf{G}$*.*

A categorical derivation *is a (possibly empty) derivation in this transition system.*

If we restrict SLD-steps to the use of most general unifiers and most general reduction pairs, we have a new transition system $(\biguplus_{\sigma \in \mathrm{Obj}(\mathbb{C})} \mathrm{Obj}(\mathcal{P}\sigma), \leadsto_g)$ and a corresponding notion of *most general* (m.g.) categorical derivation. In the following, when not otherwise stated, everything we say about derivations can be applied to m.g. ones.

If there are goals $\mathbf{G}_0, \ldots, \mathbf{G}_i$ and labels $l_0, \ldots, l_{i-1}$ with $i \geq 0$ such that

$$\mathbf{G}_0 \xrightarrow{l_0} \mathbf{G}_1 \cdots \mathbf{G}_{i-1} \xrightarrow{l_{i-1}} \mathbf{G}_i \tag{4}$$

we write $\mathbf{G}_0 \overset{d}{\leadsto}{}^* \mathbf{G}_i$ where $d = l_0 \cdots l_{i-1}$ is the string obtained by concatenating all the labels. Note that $d \neq \lambda$ uniquely induces the corresponding sequence of goals. We will write $\epsilon_{\mathbf{G}}$ for the empty derivation starting from goal $\mathbf{G}$.

Given a derivation d, we call *answer* of d (and we write $\mathsf{answer}(d)$) the arrow in $\mathbb{C}$ defined by induction on the length of d as follows

$$\begin{aligned} &\mathsf{answer}(\epsilon_{\mathbf{G}}) = id_\sigma && \text{if } \mathbf{G} : \sigma \\ &\mathsf{answer}(\langle r, f \rangle \cdot d) = \mathsf{answer}(d) \; ; r \\ &\mathsf{answer}(\langle r, t, a \rangle \cdot d) = \mathsf{answer}(d) \; ; r \end{aligned}$$

In particular, we call *most general* answers the answers corresponding to m.g. derivations.

Example 13 (Standard SLD derivations). Consider a program P in the syntactic doctrine $\mathcal{P}_\Pi$ and a goal $p(t_1)$ of sort σ. Given a clause $p(t_2) \xleftarrow{cl} q(t)$, and an mgu $\langle r_1, r_2 \rangle$ for $p(t_1)$ and $p(t_2)$, we have a most general derivation step

$$p(t_1) \xrightarrow{\langle r_1, r_2, cl \rangle}_g q(r_2; t) \ . \tag{5}$$

This strictly corresponds to a step of the standard SLD derivation procedure for binary clauses and atomic goals.

However, in the categorical derivation, it is possible to reduce w.r.t. one of the identity arrows of the fibers. Therefore, if $p(t) : \sigma$,

$$p(t) \xrightarrow{\langle id_\sigma, id_{p(t)} \rangle}_g p(t) \tag{6}$$

is an identity step which does not have a counterpart in the standard resolution procedure. However, these steps have an identity answer. Therefore, fixing a goal $p(t)$, the set

$$\mathsf{answer}\{d \mid d : p(t) \rightsquigarrow_g{}^* \mathbf{G}\} \tag{7}$$

is the set of partial answers for the goal $p(t)$.

We can use categorical derivations to build several interesting models for logic programs. In particular, with the answer function we can build models which are the general counterpart of partial computed answers, correct answer and ground answers.

Example 14 (Model for ground answers). Consider a program P in $\mathcal{P}_\Pi$ and an interpretation $[\![\_]\!]$ in the LP doctrine $\mathcal{Q}$ defined in Example 6, such that

$$[\![p(t)]\!] = \{\mathsf{answer}(d) \mid d : p(t) \rightsquigarrow_g{}^* \mathbf{G} \text{ is a m.g. ground derivation}\} \ , \tag{8}$$

where a *ground derivation* is a derivation whose last goal is in the fiber $\mathcal{P}(1)$. Now, for each clause $p_1(t_1) \xleftarrow{cl} p_2(t_2)$, if d is a m.g. ground derivation of $p_2(t_2)$, then

$$d' = p_1(t_1) \overset{\langle id, id, cl\rangle}{\rightsquigarrow} p_2(t_2) \cdot d \tag{9}$$

is a m.g. ground derivation for $p_1(t_1)$ with $\mathsf{answer}(d') = \mathsf{answer}(d)$. Therefore, $[\![p_1(t_1)]\!] \supseteq [\![p_2(t_2)]\!]$ and this gives an obvious mapping ι from clauses to arrows in the fibers of $\mathcal{Q}$. It turns out that $([\![\_]\!], \iota)$ is a model for P.

5 Completeness

Assume given a program P over the LP doctrine $\mathcal{P} : \mathbb{C}^o \to \mathsf{Cat}$. It is possible to use categorical derivations to obtain a free model of P. First of all, consider the following definitions:

Definition 15 (Flat derivations). *A derivation is called* flat *(on the fiber σ) when all the r fields in the labels of the two backchain rules are identities (on σ).*

Definition 16 (Simple derivations). *A derivation is called* simple *when*

- *there are no two consecutive* backchain-arrow *steps,*
- *there are no* backchain-arrow *steps with identity arrows f.*

Given a derivation $d : \mathbf{G}_1 \rightsquigarrow \mathbf{G}_2$ with $\mathsf{answer}(d) = \theta$, there is a canonical flat simple derivation $\bar{d} : \theta^\sharp \mathbf{G}_1 \rightsquigarrow \mathbf{G}_2$ obtained by collapsing consecutive backchain-arrow steps. If we define a congruence $\equiv$ on derivations such that

$$d_1 \equiv d_2 \iff \mathsf{answer}(d_1) = \mathsf{answer}(d_2) \text{ and } \bar{d_1} = \bar{d_2} \ , \tag{10}$$

it is possible to define an LP doctrine $\mathcal{F}_P$ over $\mathbb{C}$ such that $\mathcal{F}_P(\sigma)$ is the category of equivalence classes of flat simple derivations on the fiber σ.

Now, we can define an interpretation $[\![\_]\!] = (id_{\mathbb{C}}, \tau)$ from $\mathcal{P}$ to $\mathcal{F}_P$ and a function ι such that:

– $\tau_\sigma(\mathbf{G}) = \mathbf{G}$;
– $\tau_\sigma(f : \mathbf{G} \to \mathbf{G}') = \left[\mathbf{G}' \xrightarrow{\langle id_\sigma, f\rangle} \mathbf{G}\right]_\equiv$;
– $\iota(\mathbf{Hd} \xleftarrow{cl} \mathbf{Tl}) = \left[\mathbf{Hd} \xrightarrow{\langle id_\sigma, id_\sigma, cl\rangle} \mathbf{Tl}\right]_\equiv$

We obtain that $(\llbracket \_ \rrbracket, \iota)$ is a free model of P, which we will denote by F_P. Then, we have the following corollaries:

Theorem 17 (Soundness theorem). *Assume given a program P in $\mathcal{P}$, a goal $\mathbf{G}$ and a model $M = (\llbracket \_ \rrbracket, \iota) : \mathcal{P} \to \mathcal{Q}$. If d is a derivation from $\mathbf{G}$ to $\mathbf{G}'$ with computed answer θ, there exists an arrow $\theta^\sharp \llbracket \mathbf{G} \rrbracket \xleftarrow{p} \llbracket \mathbf{G}' \rrbracket$ in $\mathcal{Q}$, where $p = \mathsf{arrow}(d)$ is defined by induction:*

$$\mathsf{arrow}(\epsilon_{\mathbf{G}}) = id_{\mathbf{G}}$$
$$\mathsf{arrow}(d \cdot \langle r, f\rangle) = r^\sharp(\mathsf{arrow}(d)) \ ; f$$
$$\mathsf{arrow}(d \cdot \langle r, t, cl\rangle) = r^\sharp(\mathsf{arrow}(d)) \ ; t^\sharp(\iota(cl))$$

Theorem 18 (Completeness theorem). *Assume given a program P in $\mathcal{P}$, a free model $M : \mathcal{P} \to \mathcal{Q}$ and goals $\mathbf{G}$, $\mathbf{G}'$ of sort σ. If there is an arrow $f : M(\mathbf{G}) \to M(\mathbf{G}')$ in the fiber $M(\sigma)$ of $\mathcal{Q}$, then there is a simple flat derivation $\mathbf{G}' \overset{d}{\leadsto}{}^* \mathbf{G}$.*

6 Fixpoint Semantics

Assume we have a program $(P, \mathcal{P})$. We have just defined the notions of SLD derivations. Now, we look for a fixpoint operator, similar in spirit to the immediate consequence operator T_P of van Emden and Kowalski [25]. Starting from an interpretation $\llbracket \_ \rrbracket : \mathcal{P} \to \mathcal{Q}$, our version of T_P gives as a result a new interpretation $\llbracket \_ \rrbracket' : \mathcal{P} \to \mathcal{Q}$ which, in some way, can be extended to a model of P with more hopes of success than $\llbracket \_ \rrbracket$.

Our long term objective is the ability to give fixpoint semantics to all of the possible programs in our framework. However, in this paper we will restrict our attention to a subclass of programs which have particular freeness properties.

Definition 19 (Goal Free logic program). *A logic program $(P, \mathcal{P})$ is called* goal free *when there is a set $\{X_1 : \sigma_1, \ldots, X_n : \sigma_n\}$ of sorted* generic goals *with the following properties:*

– *$\mathcal{P}$ is obtained from an LP doctrine $\bar{\mathcal{P}}$ by freely adjoining the generic goals to the appropriate fibers of $\bar{\mathcal{P}}$;*
– *there are no clauses targeted at a goal in $\bar{\mathcal{P}}$.*

An instance of a generic goal is called *dynamic goal*. We want to stress here that only the meaning of dynamic goals is modified step after step by the fixpoint construction, while all the goals in $\bar{\mathcal{P}}$ have a fixed meaning. Note that, given $\llbracket \_ \rrbracket : \mathcal{P} \to \mathcal{Q}$, the interpretation of all the dynamic goals only depends from the interpretation of generic goals. Intuitively, dynamic goals are those defined by the program P, and which are modified incrementally by bottom-up approximation. Fixed goals are the built-in predicates contributed by the ambient theory.

Example 20. If P is a program over the syntactic doctrine $\mathcal{P}_\Pi$ of Example 3, then it is goal free. Actually, we can define $\bar{\mathcal{P}} : \mathbb{C}^o \to \mathsf{Cat}$ such that

- for each $\sigma \in \mathrm{Obj}(\mathbb{C})$, $\bar{\mathcal{P}}(\sigma) = 0$, i.e. the empty category;
- for each $t \in \mathrm{Mor}(\mathbb{C})$, $\bar{\mathcal{P}}(t) = id_0$.

Then $\mathcal{P}_\Pi$ is obtained by $\bar{P}$ freely adjoining a goal $p(id_\sigma)$ for each $p : \sigma \in \Pi$.

However, if we consider the syntactic doctrine in Example 4, then a program P must not have any arrow targeted at p or *symp* if it wants to be goal free.

In order to define a fixpoint operator with reasonable properties, we require a more complex categorical structure in the target doctrine $\mathcal{Q}$ than in $\mathcal{P}$.

Definition 21 (Semantic doctrine). *A semantic LP doctrine $\mathcal{Q}$ is an LP doctrine where*

- *fibers have coproducts and canonical colimits of ω-chains;*
- *each reindexing functor $\mathcal{Q}t$ has a left-adjoint $\exists_t^{\mathcal{Q}}$ and preserves on the nose canonical colimits of ω-chains.*

We will drop the superscript $\mathcal{Q}$ from $\exists_t^{\mathcal{Q}}$ when it is clear from the context. If we only work with finite programs, it is enough for fibers to have only finite coproducts.

Example 22. Given a finite product category $\mathbb{C}$, consider the indexed category $\mathcal{Q}$ as defined in Example 6. It is possible to turn $\mathcal{Q}$ into a semantic LP doctrine. Actually:

- each fiber is a complete lattice, hence it has coproducts given by intersection and canonical colimits of ω-chains given by union;
- we can define $\exists_f^{\mathcal{Q}}$, with $f : \rho \to \sigma$ as the function which maps an $X \subseteq \mathrm{Hom}_{\mathbb{C}}(1, \rho)$ to
$$\exists_f^{\mathcal{Q}}(X) = \{t \,;\, f \mid t \in X\} \tag{11}$$
which is a subset of $\mathrm{Hom}_{\mathbb{C}}(1, \sigma)$.

We can prove that all the conditions for semantic doctrines are satisfied.

Now, assume we have an interpretation $[\![\_]\!] = (F, \tau)$ from $\mathcal{P}$ to $\mathcal{Q}$, where $\mathcal{Q}$ is a semantic LP doctrine. We want to build step after step a modified interpretation which is also a model of P. With a single step we move from $[\![\_]\!]$ to $\mathbf{E}_P([\![\_]\!]) = (F, \tau')$ where

$$\begin{aligned} \tau'_{\sigma_i}(X_i) &= [\![X_i]\!] \vee \bigvee_{X_i(t) \xleftarrow{\mathbf{cl}} \mathbf{T} \in P} \exists_{Ft} [\![\mathbf{T}]\!] \ , \\ \tau'_\sigma(X_i(t)) &= t^\sharp\big(\tau'_{\sigma_i}(X_i)\big) \ , \end{aligned} \tag{12}$$

while $\tau' = \tau$ restricted to $\bar{\mathcal{P}}$. We should define τ' on arrows but, since there are only identities between dynamic goals, the result is obvious.

In the same way, if δ is an arrow between interpretations, we have $\mathbf{E}_P(\delta) = \delta'$, where

$$\begin{aligned}
&\delta'_{\sigma_i,X_i} = \delta_{\sigma_i,X_i} \vee \bigvee_{X_i(t:\rho\to\sigma_i)\leftarrow\mathbf{Tl}\in P} \exists_{Ft}\delta_{\rho,\mathbf{Tl}} \ , \\
&\delta'_{\sigma,X_i(t)} = t^\sharp(\delta'_{\sigma_i,X_i}) \ , \\
&\delta'_{\sigma,\mathbf{G}} = \delta_{\sigma,\mathbf{G}} \text{ if } \mathbf{G} \in \mathrm{Obj}(\bar{\mathcal{P}}) \ .
\end{aligned} \tag{13}$$

Since the only non-trivial arrows are in $\bar{\mathcal{P}}$ and δ_σ is a natural transformation for each σ, the same can be said of δ'_σ. It follows that $\mathbf{E}_P$ is well defined.

It is interesting to observe that there is a canonical transformation ν between $[\![\_]\!]$ and $\mathbf{E}_P([\![\_]\!])$ given by:

$$\nu_{\sigma_i,X_i} = [\![X_i]\!]_{\sigma_i} \xrightarrow{in} \mathbf{E}([\![\_]\!])_{\sigma_i}(X_i) \ , \tag{14}$$

$$\nu_{\sigma,X_i(t)} = t^\sharp(\iota_{\nu_i,X_i}) \ , \tag{15}$$

$$\nu_{\sigma,\mathbf{G}} = id_{\mathbf{G}} \text{ if } \mathbf{G} \in \mathrm{Obj}(\bar{\mathcal{P}}) \ , \tag{16}$$

where *in* is the appropriate injection. Therefore, we can build an ω-chain

$$[\![\_]\!] \xrightarrow{\nu} \mathbf{E}_P([\![\_]\!]) \xrightarrow{\mathbf{E}_P(\nu)} \mathbf{E}^2_P([\![\_]\!]) \to \ldots \mathbf{E}^n_P([\![\_]\!]) \xrightarrow{\mathbf{E}^n_P(\nu)} \ldots$$

and we can prove that the colimit $\mathbf{E}^\omega_P([\![\_]\!])$ is a fixpoint for $\mathbf{E}_P$. Finally, we have the following:

Theorem 23. *Given a program $(P, \mathcal{P})$, a semantic LP doctrine $\mathcal{Q}$ and an interpretation $[\![\_]\!] : \mathcal{P} \to \mathcal{Q}$, then $\mathbf{E}_P$ has a least fixpoint greater than $[\![\_]\!]$. Such a fixpoint can be extended to a model of P in $\mathcal{Q}$.*

Example 24. If we write the definition of $\mathbf{E}_P$ in all the details for the syntactic doctrine in Example 22, we obtain

$$\mathbf{E}_P([\![\_]\!])_{\sigma_i}(X_i) = [\![X_i]\!] \cup \bigcup_{X_i(t)\leftarrow X_j(r)} \{f\,;t \mid f\,;r \in [\![X_j]\!], \mathsf{dom}(f) = 1\} \ , \tag{17}$$

$$\mathbf{E}_P([\![\_]\!])_{\sigma_i}(X_i(t)) = \{f \mid f;t \in [\![\mathbf{Tl}]\!]\} \ . \tag{18}$$

If we work with $\mathbb{C}$ defined by the free algebraic category for a signature Σ, then $\mathbf{E}_P([\![\_]\!])$ becomes equivalent to the standard T_P semantics for logic programs.

Assume $\mathbb{C} = \mathit{Set}$. Moreover, assume we have two predicate symbols $p : \mathbb{N}$ and $\mathsf{true} : 1$ and two clauses $p(\mathsf{succ}\,;\mathsf{succ}) \leftarrow p(id_\mathbb{N})$ and $p(0) \leftarrow \mathsf{true}$. Let $[\![\_]\!]$ be the unique interpretation which maps true to $\mathrm{Hom}(1,1)$ and p to $\emptyset$. Then, we can compute successive steps of the $\mathbf{E}_P$ operator starting from $[\![\_]\!]$, obtaining

$$\begin{aligned}
&\mathbf{E}^0_P([\![_]\!])(p) = \emptyset \\
&\mathbf{E}^1_P([\![_]\!])(p) = \{0\} \\
&\mathbf{E}^2_P([\![_]\!])(p) = \{0, 2\} \\
&\qquad\vdots \\
&\mathbf{E}^n_P([\![_]\!])(p) = \{0, 2, \ldots, 2(n-1)\}
\end{aligned} \tag{19}$$

where we identify an arrow $f : 1 \to \mathbb{N}$ with $f(\cdot)$, i.e. f applied to the only element of its domain. If we take the colimit of this ω-chain, we have

$$\mathbf{E}_P^{\omega}(\llbracket \_ \rrbracket)(p) = \{ f : 1 \to \mathbb{N} \mid f(\cdot) \text{ is even } \} \ , \tag{20}$$

which is what we would expect from the intuitive meaning of the program P.

7 An Example: Binary CLP

We are going to show that our categorical framework can handle quite easily the broad class of languages known with the name of *constraint logic programming* [17]. It is evident we need a categorical counterpart of a *constraint system*. We refer to the definition which appears in [22].

Definition 25 (Constraint system). *A* constraint system *over a finite product category $\mathbb{C}$ is an indexed category over $\mathbb{C}$ such that each fiber is a meet semilattice and reindexing functors have left adjoints.*

Now, given a constraint system $\mathcal{D}$ over $\mathbb{C}$, let us denote by $\mathbb{D}$ the corresponding category we obtain by the Grothendieck construction [16]. To be more precise:

- objects of $\mathbb{D}$ are pairs $\langle \sigma, c \rangle$ where $\sigma \in \mathrm{Obj}(\mathbb{C})$ and $c \in \mathrm{Obj}(\mathcal{D}(\sigma))$;
- arrows in $\mathbb{D}$ from $\langle \sigma_1, c_1 \rangle$ to $\langle \sigma_2, c_2 \rangle$ are given by arrows $f : \sigma_1 \to \sigma_2$ in $\mathbb{C}$ such that $c_1 \leq f^\sharp c_2$. We denote such an arrow with $(f, c_1 \leq c_2)$.

Given a predicate signature Π over $\mathbb{C}$, we define a new LP doctrine $\mathcal{P}_\Pi^{\mathcal{D}}$ over $\mathbb{D}$. For each $\langle \sigma, c \rangle$ in $\mathbb{D}$, the corresponding fiber is the discrete category whose objects are of the form $c \,\Box\, p(t)$ with $p : \rho$ in Π and $t : \sigma \to \rho$. For each arrow $(f, c_1 \leq c_2)$, the reindexing functor maps $c_2 \,\Box\, p(t)$ to $c_1 \,\Box\, p(f\,;t)$.

Now, we fix a program P. For each goal $c \,\Box\, p_1(f)$ of sort $\langle \sigma, c \rangle$ and clause $c' \,\Box\, p_1(f_1) \xleftarrow{cl} c' \,\Box\, p_2(f_2)$ of sort $\langle \sigma', c' \rangle$, let $\langle r, t \rangle$ be the mgu of f and f_1 in $\mathbb{C}$, and $c'' = r^\sharp c \wedge t^\sharp c'$. Then $\langle (r, c'' \leq c), (t, c'' \leq c_1) \rangle$ is an mgu of $\mathbf{G}$ and cl in $\mathcal{P}_\Pi^{\mathcal{D}}$. We can perform a m.g. SLD step

$$\mathbf{G} \xrightarrow{\langle (r, c'' \leq c), (t, c'' \leq c'), cl \rangle} c'' \,\Box\, p_2(t\,; f_2) \ . \tag{21}$$

As a result, a clause that we typically write as $p_1(t_1) \leftarrow c \,\Box\, p_2(t_2)$, with $p_1(t_1)$, $p_2(t_2)$ and c of sort σ, behaves like a clause $c \,\Box\, p_1(t_1) \leftarrow c \,\Box\, p_2(t_2)$ of sort $\langle \sigma, c \rangle$ in our framework.

We can also build a semantic doctrine $\mathcal{Q}$ over $\mathbb{D}$ such that the fiber corresponding to $\langle \sigma, c \rangle$ is the lattice of downward closed subsets of constraints of sort σ less than c, i.e.

$$\mathcal{Q}(\sigma, c) = \wp_{\downarrow}\{ c' \in \mathrm{Obj}(\mathcal{D}\sigma) \mid c' \leq c \}. \tag{22}$$

Moreover,

$$\mathcal{Q}(f, c_1 \leq c_2)(X) = \downarrow \{ c_1 \wedge f^\sharp c \mid c \in X \} \ , \tag{23}$$

$$\exists^{\mathcal{Q}}_{(f, c_1 \leq c_2)}(X) = \downarrow \{ \exists^{\mathcal{D}}_f c \mid c \in X \} \ . \tag{24}$$

where $\downarrow Y$ is the downward closure of the set of constraints Y. It is easy to check that these form a pair of adjoint functors and that Ω is indeed a semantic LP doctrine. Therefore, we can compute a fixpoint semantics.

If $\mathbb{C}$ is the algebraic category freely generated by an empty set of function symbols and $\mathcal{D}$ is a constraint system on real numbers, consider the program

$$
\begin{aligned}
&x^2 = y \,\Box\, p(x) \leftarrow x^2 = y \,\Box\, q(x,y)\\
&x = 2y \,\Box\, q(x,y) \leftarrow x = 2y \,\Box\, \mathsf{true}
\end{aligned}
$$

where p, q and true are predicate symbols of sort $\langle 1, \top_1\rangle$, $\langle 2, \top_2\rangle$ and $\langle 0, \top_0\rangle$ respectively. Here, $0, 1$ and 2 denote the sorts of all the goals of the corresponding arity. Moreover, we assume there is a constraint $\top_i$ for each arity i which is the top of $\mathcal{D}(i)$, preserved by reindexing functors. Assume we start with the interpretation $[\![\_]\!]$ mapping p and q to $\emptyset$ and true to $\mathcal{D}(0)$. Then, by applying the $\mathbf{E}_P$ operator:

$$
\begin{aligned}
\mathbf{E}_P([\![\_]\!])(q) &= [\![q]\!] \cup \exists_{(id_2, x=2y \le \top_2)} [\![x = 2y \,\Box\, \mathsf{true}]\!]\\
&= (!_2, x = 2y)^\sharp\, [\![\mathsf{true}]\!]\\
&= \downarrow \{x = 2y\}
\end{aligned}
\tag{25}
$$

where $!_2$ is the unique arrow from 2 to 0. We also have $\mathbf{E}_P([\![\_]\!])(p) = \emptyset$. At the second step

$$
\begin{aligned}
\mathbf{E}_P^2([\![\_]\!])(p) &= \mathbf{E}_P([\![\_]\!])(p) \cup \exists_{(\pi_1^2, x^2=y \le \top_1)} \mathbf{E}_P([\![\_]\!])(x^2 = y \,\Box\, q(x,y))\\
&= \exists_{(\pi_1^2, x^2=y \le \top_1)} (id_2, x^2 = y \le \top_2)^\sharp\, \mathbf{E}_P([\![\_]\!])([\![q]\!])\\
&= \exists_{(\pi_1^2, x^2=y \le \top_1)} \downarrow \{x^2 = y \text{ and } x = 2y\}\\
&= \downarrow \{x = 0 \text{ or } x = 1/2\}
\end{aligned}
\tag{26}
$$

where π_1^2 is the projection arrow from 2 to 1. Moreover, $\mathbf{E}_P^2([\![\_]\!])(q) = \mathbf{E}_P([\![\_]\!])(q)$ and we have reached the fixpoint.

8 Conclusions

We have introduced a categorical framework to handle several extensions of logic programming, based on Horn clauses, but interpreted in a context which is not necessarily the Herbrand universe. Typical examples of these languages are CLP [17] and logic programs with built-in or embedded data types [20].

With respect to the stated intentions in Section 1.1, we have not tackled the problem of programs with constraints on goals when it comes to fixpoint semantics. From this point of view, the only advantage offered by our framework is the ability to treat builtins. Goals whose semantics can be modified by clauses (i.e. dynamic goals) must be freely generated as in [13].

Our categorical structures capture pure clausal programs but require the addition of monoidal structure to handle conjunctions of categorical predicates externally. But the addition of monoidal structure is straightforward, and is described in detail in [1].

We briefly sketch the main ideas. We use monoidal LP doctrines, i.e. LP doctrine endowed with a monoidal structure for each fiber which is preserved on the nose by reindexing functors. Given monoidal LP doctrines $\mathcal{P}$ and $\mathcal{Q}$, a monoidal interpretation is an interpretation $(F,\tau) : \mathcal{P} \to \mathcal{Q}$ such that τ_σ preserves on the nose the monoidal structure. This condition means that the semantics of the conjunction is given compositionally. A monoidal model for $(P, \mathcal{P})$ is a monoidal interpretation together with a choice function ι for the clauses in P. We also define a monoidal derivation in the same way as we have done in section 4, but the backchain-clause rule is replaced with the following:

$$\mathbf{G} \xrightarrow{\langle r, t, \mathbf{G}_1, \mathbf{G}_2, cl\rangle} \mathbf{G}_1 \otimes t^\sharp \mathbf{Tl} \otimes \mathbf{G}_2 \tag{27}$$

if $\mathbf{Hd} \xleftarrow{cl} \mathbf{Tl}$ is a clause, $\otimes$ is the monoidal tensor and $r^\sharp \mathbf{G} = \mathbf{G}_1 \otimes t^\sharp \mathbf{Hd} \otimes \mathbf{G}_2$. Again, if we define an appropriate equivalence relation on monoidal derivations, we can build a free monoidal model. Finally, for the fixpoint semantics, everything proceeds as for Section 6, provided that we use monoidal semantic LP doctrines, i.e. a monoidal LP doctrines with the same properties which hold for semantic LP doctrines. We just need to add a pair of conditions to the definition of τ'_σ and δ'_σ in (12) and (13), namely,

$$\tau'_\sigma(\mathbf{G}_1 \otimes_\sigma \mathbf{G}_2) = \tau'_\sigma(\mathbf{G}_1) \otimes \tau'_\sigma(\mathbf{G}_2) \ , \tag{28}$$

$$\delta'_{\sigma,\mathbf{G}_1\otimes\mathbf{G}_2} = \delta'_{\sigma,\mathbf{G}_1} \otimes \delta'_{\sigma,\mathbf{G}_2} \ . \tag{29}$$

Together with the goal-free condition, we also require that clauses only have atomic goals as heads. Then, all the results we have shown in this paper also hold for the monoidal case.

Finally, note that we can use weaker structure on the fibers, like premonoidal structures, to give an account of selection rules [1].

References

1. G. Amato. *Sequent Calculi and Indexed Categories as a Foundation for Logic Programming.* PhD thesis, Dipartimento di Informatica, Università di Pisa, 2000.
2. K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 495–574. Elsevier and The MIT Press, 1990.
3. A. Asperti and S. Martini. Projections instead of variables. In G. Levi and M. Martelli, editors, *Proc. Sixth Int'l Conf. on Logic Programming*, pages 337–352. The MIT Press, 1989.
4. R. Barbuti, R. Giacobazzi, and G. Levi. A General Framework for Semantics-based Bottom-up Abstract Interpretation of Logic Programs. *ACM Transactions on Programming Languages and Systems*, 15(1):133–181, 1993.
5. M. Barr and C. Wells. *Category Theory for Computing Science.* Prentice Hall, New York, 1990.

6. A. Bossi, M. Gabbrielli, G. Levi, and M. C. Meo. A Compositional Semantics for Logic Programs. *Theoretical Computer Science*, 122(1–2):3–47, 1994.
7. A. Brogi. *Program Construction in Computational Logic*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1993.
8. M. Comini, G. Levi, and M. C. Meo. A Theory of Observables for Logic Programs. *Information and Computation*, 169:23–80, 2001.
9. M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Abstract diagnosis. *Journal of Logic Programming*, 39(1-3):43–93, 1999.
10. A. Corradini and A. Asperti. A categorical model for logic programs: Indexed monoidal categories. In *Proceedings REX Workshop '92*. Springer Lectures Notes in Computer Science, 1992.
11. A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *Theoretical Computer Science*, 103(1):51–106, August 1992.
12. P. Cousot and R. Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2 & 3):103–179, 1992.
13. S. Finkelstein, P. Freyd, and J. Lipton. Logic programming in tau categories. In *Computer Science Logic '94*, volume 933 of *Lecture Notes in Computer Science*, pages 249–263. Springer Verlag, Berlin, 1995.
14. S. Finkelstein, P. Freyd, and J. Lipton. A new framework for declarative programming. To appear in *Theoretical Computer Science*, 2001.
15. P. J. Freyd and A. Scedrov. *Categories, Allegories*. North-Holland, Elsevier Publishers, Amsterdam, 1990.
16. B. Jacobs. *Categorical Logic and Type Theory*. Studies in Logic and the Foundations of Mathematics. North Holland, Elsevier, 1999.
17. J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19–20:503–581, 1994.
18. Y. Kinoshita and A. J. Power. A fibrational semantics for logic programs. In R. Dyckhoff, H. Herre, and P. Schroeder-Heister, editors, *Proceedings of the Fifth International Workshop on Extensions of Logic Programming*, volume 1050 of *LNAI*, pages 177–192, Berlin, Mar. 28–30 1996. Springer.
19. A. Kock and G. E. Reyes. Doctrines in categorical logic. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 283–313. North Holland, 1977.
20. J. Lipton and R. McGrail. Encapsulating data in logic programming via categorical constraints. In C. Palamidessi, H.Glaser, and K. Meinke, editors, *Principles of Declarative Programming*, volume 1490 of *Lecture Notes in Computer Science*, pages 391–410. Springer Verlag, Berlin, 1998.
21. G. Nadathur and D. Miller. An overview of λProlog. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 810–827. The MIT Press, 1988.
22. P. Panangaden, V. J. Saraswat, P. J. Scott, and R. A. G. Seely. A Hyperdoctrinal View of Concurrent Constraint Programming. In J. W. de Bakker et al, editor, *Semantics: Foundations and Applications*, volume 666 of *Lecture Notes in Computer Science*, pages 457–475. Springer-Verlag, 1993.
23. D. Rydeheard and R. Burstall. A categorical unification algorithm. In *Category Theory and Computer Programming, LNCS 240*, pages 493–505, Guildford, 1985. Springer Verlag.
24. R. Seely. Hyperdoctrines, Natural Deduction and the Beck Condition. *Zeitschrift für Math. Logik Grundlagen der Math.*, 29(6):505–542, 1983.
25. M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.